BIBLIOMETRÍA IPBLN: GUÍA TÉCNICA

Herramienta web interactiva para el análisis de producción científica del IPBLN-CSIC

GRANADA

INSTITUTO DE PARASITOLOGÍA Y BIOMEDICINA LÓPEZ-NEYRA (IPBLN-CSIC)

Pablo Gradolph Oliva

CONTENIDO

1. INTRODUCCIÓN

1.1. EXPLICACIÓN Y ESTRUCTURA DEL DOCUMENTO

El presente documento constituye la **Guía Técnica** de la aplicación *Bibliometría IPBLN*, desarrollada para el análisis, visualización y gestión de datos bibliométricos del Instituto de Parasitología y Biomedicina López-Neyra (IPBLN-CSIC).

El propósito principal de esta guía es servir como referencia a desarrolladores, administradores del sistema y personal técnico encargado del mantenimiento y evolución de la aplicación. A diferencia de la guía de usuario, este documento se centra en aspectos internos del software: tecnologías empleadas, estructura del proyecto, procesos de actualización, despliegue e indicaciones de soporte técnico.

Para facilitar la consulta, el documento se organiza en diferentes secciones que siguen una progresión lógica:

- **Introducción**, donde se contextualiza la aplicación, se explica el alcance de la guía y se ofrece una descripción general del proyecto.
- **Tecnologías utilizadas**, apartado en el que se detallan los lenguajes, *frameworks*, librerías y herramientas empleados en el desarrollo.
- **Estructura del proyecto (Django)**, que describe la organización interna de la aplicación, sus módulos principales y la lógica de interacción entre *backend* y *frontend*.
- **Base de datos,** que incluye una explicación detallada de cómo se estructura y organiza la base de datos del proyecto para facilitar su entendimiento y uso.
- **Proceso de actualización de la aplicación**, sección clave en la que se documentan los pasos necesarios para mantener el sistema actualizado.
- **Despliegue**, donde se especifican los procedimientos para instalar y poner en marcha la aplicación, incluyendo dependencias y configuraciones necesarias.
- **Soporte y contacto**, con la información relativa a los canales disponibles para resolver incidencias técnicas o plantear consultas específicas.

Con esta estructura, la guía busca asegurar la continuidad y mantenibilidad del proyecto, ofreciendo un recurso técnico completo tanto para la incorporación de nuevos desarrolladores como para el trabajo cotidiano de administración y soporte.

1.2. DESCRIPCIÓN GENERAL DEL PROYECTO

El proyecto *Bibliometría IPBLN* consiste en el desarrollo de una **aplicación web interactiva** diseñada para analizar, organizar y visualizar la producción científica del Instituto de Parasitología y Biomedicina López-Neyra (IPBLN-CSIC). Su finalidad principal es proporcionar una plataforma que transforme los datos bibliográficos exportados desde el

sistema institucional GesBIB en información procesada, estructurada y fácilmente interpretable tanto por investigadores como por gestores.

Desde un punto de vista técnico, la aplicación se ha construido sobre una arquitectura basada en Django (Python) para el *backend* y tecnologías web modernas para el *frontend*, combinando **HTML, CSS, JavaScript, Vite** y librerías de visualización como **D3.js y Sigma.js**. La base de datos relacional empleada es **SQLite**, que actúa como repositorio de los registros importados desde GesBIB y de los resultados de análisis posteriores.

El sistema integra diferentes módulos que trabajan de manera coordinada:

- Un **módulo de ingesta y procesamiento de datos**, encargado de importar ficheros JSON desde GesBIB, limpiarlos, enriquecerlos y transformarlos en estructuras relacionales.
- Un **módulo de configuración**, que reúne todos los parámetros, dependencias y requisitos necesarios para garantizar el correcto funcionamiento de la aplicación. Incluye las variables de entorno, la configuración de bases de datos, el ajuste de librerías externas y otros elementos clave para la inicialización del proyecto.
- Un **módulo central**, que constituye la base de la aplicación, donde se encuentra la estructura principal de la plataforma web. En él se definen, la lógica de negocio y los controladores que permiten la interacción entre el *backend* y el *frontend*.
- Un **módulo de interfaz de usuario (frontend),** que concentra todos los componentes visibles para el usuario final, incluyendo las plantillas, hojas de estilo y elementos gráficos. Este módulo garantiza una experiencia de uso clara y atractiva, integrando las visualizaciones dinámicas e interactivas que caracterizan a la herramienta.
- Un **módulo de inteligencia artificial y análisis avanzado**, que aplica técnicas de procesamiento de lenguaje natural (NLP) para clasificar publicaciones por temáticas, detectar comunidades mediante algoritmos de clustering y de detección de comunidades (Louvain, Leiden, KMeans, Agglomerative, entre otros) y predecir áreas temáticas en publicaciones sin clasificar.
- Un **módulo de gestión de cuentas**, que permite administrar a los distintos usuarios de la aplicación. Abarca funcionalidades como la creación y edición de perfiles, la autenticación y autorización de accesos, así como la asignación de permisos específicos según el rol del usuario.

La herramienta ha sido diseñada siguiendo principios de **interactividad, escalabilidad y mantenibilidad**. Esto implica que cada componente puede actualizarse de forma independiente, que las visualizaciones responden en tiempo real a las acciones del usuario y que se han incorporado mecanismos para asegurar la continuidad del proyecto a largo plazo.

2. TECNOLOGÍAS UTILIZADAS

El correcto funcionamiento y mantenimiento de la aplicación *Bibliometría IPBLN* requiere un conjunto de tecnologías que abarcan tanto el *backend* como el *frontend*, así como herramientas auxiliares para el procesamiento de datos y la gestión del proyecto. A continuación, se resumen las más relevantes:

- **Python:** lenguaje principal en el que está implementada la lógica de servidor y el procesamiento de datos. Todas las dependencias necesarias se encuentran especificadas en el fichero requirements.txt, que debe instalarse en el entorno de ejecución para garantizar la compatibilidad. Entre estas librerías destacan **Django** (para la estructura del proyecto web), **pandas** y **scikit-learn** (para preprocesamiento y análisis de datos), así como librerías específicas de clustering y NLP.

- **Django:** *framework* de alto nivel en Python que organiza la aplicación según el patrón Modelo-Vista-Plantilla (MVT). Permite la gestión de la base de datos, el control de usuarios y la generación de contenido dinámico. Es la base del módulo central del sistema.

- **JavaScript y librerías de visualización:** el *frontend* incorpora JavaScript moderno junto con librerías especializadas para gráficos interactivos:

    - **D3.js**, para visualizaciones de datos dinámicas.

    - **Graphology.js**, para manipulación y análisis de grafos.

    - **Sigma.js**, para la representación interactiva de redes de coautoría.

- **Vite:** herramienta de construcción moderna utilizada en el *frontend*, que permite empaquetar y optimizar el código JavaScript/ESM, facilitando además un refresco rápido en el entorno de desarrollo.

- **Bootstrap y HTML:** la capa visual de la aplicación se apoya en plantillas HTML generadas desde Django y en el *framework* Bootstrap para el diseño responsivo y la coherencia estética.

- **SQLite:** base de datos relacional utilizada para almacenar publicaciones, autores, métricas y resultados derivados del análisis.

- **Git y GitHub:** sistema de control de versiones empleado para gestionar el código fuente del proyecto. El repositorio permite mantener el historial de cambios y facilita la colaboración en equipo.

- **Entornos de desarrollo:** aunque puede utilizarse cualquier editor de código, se recomienda trabajar con un entorno ligero y extensible (como Visual Studio Code o equivalente) para integrar fácilmente el control de versiones, depuración y ejecución local de Django.

En conjunto, estas tecnologías conforman un ecosistema robusto que facilita tanto el despliegue de la aplicación como su actualización y evolución futura.

3. ESTRUCTURA DEL PROYECTO (DJANGO)

La aplicación *Bibliometría IPBLN* está desarrollada sobre el *framework* **Django** y organizada en una arquitectura modular basada en múltiples aplicaciones. Esta división en módulos facilita la escalabilidad, el mantenimiento y la comprensión del sistema, ya que cada aplicación se encarga de una responsabilidad específica.

Cada aplicación Django mantiene la estructura estándar del *framework*, incluyendo:

- **models.py:** definición de los modelos de datos y sus relaciones en la base de datos.
- **views.py:** vistas que gestionan las peticiones y generan las respuestas.
- **urls.py:** rutas específicas de la aplicación, mapeadas a las vistas correspondientes.
- **forms.py (opcional):** formularios utilizados para la entrada y validación de datos.
- **admin.py:** configuración de la interfaz de administración de Django.
- **apps.py:** configuración básica de la aplicación.

A continuación, se detallan las principales aplicaciones y carpetas del proyecto:

- ***accounts*:** aplicación responsable de la gestión de usuarios. Incluye el registro, autenticación (*login*/*logout*), recuperación de contraseñas y validación de cuentas mediante correo electrónico. Define también un modelo de usuario personalizado adaptado a las necesidades del sistema.
- ***analysis*:** aplicación orientada a las funcionalidades de análisis bibliométrico avanzado e inteligencia artificial. Contiene los módulos encargados del procesamiento semántico, clasificadores de áreas temáticas, algoritmos de agrupamiento de investigadores y análisis de redes de coautoría mediante detección de comunidades y embeddings.
- ***bibliometrics*:** carpeta principal del proyecto Django. Contiene la configuración global (settings.py, urls.py, wsgi.py y asgi.py), así como los elementos necesarios para la orquestación del resto de aplicaciones.
- ***bibliodata*:** aplicación dedicada a la gestión de los modelos de datos importados desde GesBIB. Define autores, publicaciones, métricas, instituciones y otros elementos clave. Además, incluye funciones para cargar y transformar los datos desde archivos JSON, JSONL y CSV.
- ***core*:** aplicación central que contiene la lógica de integración. Gestiona la renderización de páginas, el procesamiento de peticiones, la aplicación de filtros, las búsquedas y la comunicación entre el *backend* y el *frontend*.

- ***data* (carpeta):** directorio auxiliar que almacena los archivos de datos empleados durante el desarrollo. No es una aplicación Django, pero sirve como repositorio de información procesada y datos de prueba. Es necesaria para todo el proceso de actualización también.
- ***frontend* (carpeta):** directorio externo a Django que contiene el código JavaScript moderno gestionado con Vite. Aquí se implementan las visualizaciones interactivas, la lógica de filtros, las redes de coautoría, los gráficos con D3.js y los componentes dinámicos de la interfaz.

Esta organización modular asegura una clara separación entre la lógica del *backend*, las funciones analíticas y la capa de presentación. Gracias a esta estructura, cada parte del sistema puede evolucionar de forma independiente, simplificando la incorporación de nuevas funcionalidades, la actualización de algoritmos y el rediseño del *frontend*.

3.1. COMUNICACIÓN ENTRE EL *FRONTEND* Y EL *BACKEND*

La comunicación entre el *backend* de Django y el *frontend* desarrollado con JavaScript y Vite combina dos enfoques complementarios: el renderizado de plantillas y el intercambio de datos en formato JSON.

1. **Renderizado con plantillas Django**
    - Se utiliza para las páginas estructurales y de acceso general (inicio, autenticación, panel principal, etc.).
    - Django genera contenido HTML dinámico en el servidor, integrando variables y datos en las plantillas antes de enviarlos al navegador.
2. **Intercambio dinámico de datos con JSON**
    - Para funcionalidades interactivas y visualizaciones en tiempo real, el *backend* expone *endpoints* específicos que devuelven datos en formato JSON.
    - El *frontend*, mediante JavaScript (y en particular con D3.js, Sigma.js o Graphology.js), consume estos datos y los transforma en gráficos, tablas dinámicas y redes interactivas.
    - Ejemplos típicos de este mecanismo incluyen:
        - La actualización de filtros (años, áreas temáticas, instituciones).
        - La recalculación de gráficas como la línea temporal o la distribución de áreas.
        - La construcción de redes de coautoría y comunidades detectadas mediante algoritmos de clustering.
3. **Gestión híbrida**

- Ambos enfoques conviven en la aplicación: las páginas estáticas y estructurales se sirven mediante plantillas de Django, mientras que las secciones analíticas y visuales se nutren de peticiones asíncronas (AJAX/fetch) para mejorar la experiencia del usuario.

En conjunto, esta arquitectura asegura que el sistema pueda manejar grandes volúmenes de información de manera eficiente y con una experiencia de usuario dinámica, manteniendo la flexibilidad necesaria para integrar nuevas visualizaciones o algoritmos en el futuro.

4. LA BASE DE DATOS

La aplicación *Bibliometría IPBLN* utiliza una **base de datos relacional SQLite** como sistema de almacenamiento. Su diseño sigue un modelo normalizado que permite representar de manera estructurada toda la información bibliométrica procedente de GesBIB y facilita su explotación a través del ORM de Django.

El uso de SQLite responde a la necesidad de contar con un sistema ligero y fácilmente desplegable en distintos entornos, sin requerir la configuración de un servidor de base de datos externo. Para entornos más exigentes (producción o integraciones futuras), la estructura definida puede migrarse a sistemas más robustos como PostgreSQL sin necesidad de modificar la lógica del proyecto, gracias a la abstracción que proporciona el ORM de Django.

4.1. DISEÑO Y ENTIDADES PRINCIPALES

La base de datos está organizada en múltiples tablas interrelacionadas que cubren las distintas dimensiones de la producción científica. A continuación, se describen las entidades principales:

- **Publication**
  Tabla central que almacena todas las publicaciones científicas. Incluye metadatos esenciales como título, año, tipo de publicación, resumen, enlaces externos, identificadores únicos (DOI, Scopus, WoS), palabras clave, idioma y marcadores de colaboración internacional. Cada publicación se asocia con uno o varios autores y puede pertenecer a múltiples áreas temáticas.

- **PublicationMetric**
  Almacena métricas bibliométricas asociadas a cada publicación, desglosadas por fuente (WoS, Scopus, Dimensions), tipo de métrica (citas, JIF, SJR, CiteScore, etc.) y año. El modelo sigue un esquema normalizado donde cada fila representa una métrica única, evitando redundancias y estructuras anidadas.

- **Author**
  Representa a los autores identificados en GesBIB. Incluye información como ORCID, Scopus ID, métricas personales (citas totales, índice-h, colaboración internacional), afiliaciones institucionales y materias asociadas. Además,

incorpora campos adicionales para reflejar clasificaciones obtenidas mediante algoritmos de clustering.

- **Collaboration**
  Define las relaciones de coautoría entre autores. Cada fila representa un par de investigadores con el número de publicaciones conjuntas, constituyendo la base para la construcción de la red de colaboraciones científicas.

- **AuthorClustering**
  Registra los resultados de los algoritmos de clustering aplicados sobre los autores (KMeans, DBSCAN, Leiden, Louvain, etc.). Incluye el número de grupos, parámetros de configuración (como reducciones dimensionales) y métricas de validación (Silhouette, Davies-Bouldin).

- **Institution**
  Contiene información de los centros del CSIC, incluyendo nombre, área principal de investigación, localización geográfica, índice de colaboración internacional y vínculos con áreas temáticas.

- **InstitutionMetric**
  Tabla que recoge métricas institucionales por año y fuente bibliográfica, como número de publicaciones, citas acumuladas e índice-h en distintas bases de datos (WoS, Scopus).

- **ThematicArea**
  Representa las áreas temáticas que pueden asociarse tanto a publicaciones como a instituciones. Permite clasificar los registros por campos de conocimiento y es clave para las visualizaciones de distribución temática.

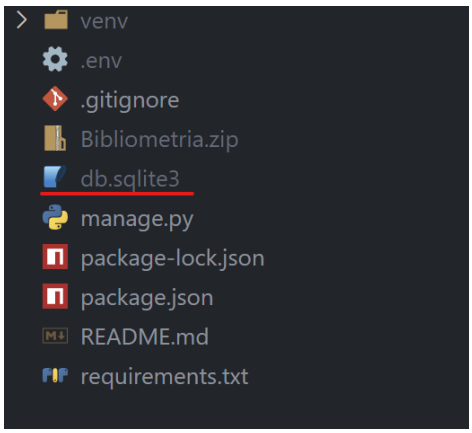
- **CustomUser**
  Modelo de usuario extendido a partir del sistema de autenticación de Django. Sustituye al modelo por defecto e incorpora el inicio de sesión mediante correo electrónico, junto con el control de permisos y acceso a funcionalidades restringidas de la aplicación.

4.2. CONSIDERACIONES TÉCNICAS

- La base de datos puede inicializarse automáticamente mediante las **migraciones de Django**, ejecutando:
  - *python manage.py makemigrations*
  - *python manage.py migrate*
- Todos los modelos están versionados en la carpeta correspondiente del proyecto (bibliodata/models.py, accounts/models.py, etc.), lo que facilita su extensión y modificación.

NOTA IMPORTANTE: Antes de iniciar el proceso de actualización de la aplicación, es recomendable descargar, desde el servidor, la base de datos SQLite que se esté usando

allí en producción y actualizar directamente esa base de datos. Lo que debemos hacer entonces es descargar esa base de datos y añadirla a nuestro proyecto:



Ese archivo db.sqlite3 debe ser el descargado desde el servidor. La razón es que esa base de datos contendrá toda la información de los usuarios que se han registrado y que forman parte de la aplicación y, para asegurar la consistencia en los datos y que los usuarios puedan seguir accediendo sin problema, debemos actualizar dicha base de datos y no crear una nueva ni actualizar una base de datos con la que trabajemos en local.

5. PROCESO DE ACTUALIZACIÓN DE LA APLICACIÓN

Esta sección documenta el procedimiento oficial para actualizar la aplicación *Bibliometría IPBLN* de forma segura y reproducible. El objetivo es que cualquier miembro del equipo técnico pueda desplegar nuevas versiones del código, incorporar mejoras de datos, o actualizarlos desde GesBIB y mantener alineadas las dependencias del proyecto sin interrumpir el servicio. Paso a paso llevaremos a cabo este proceso.

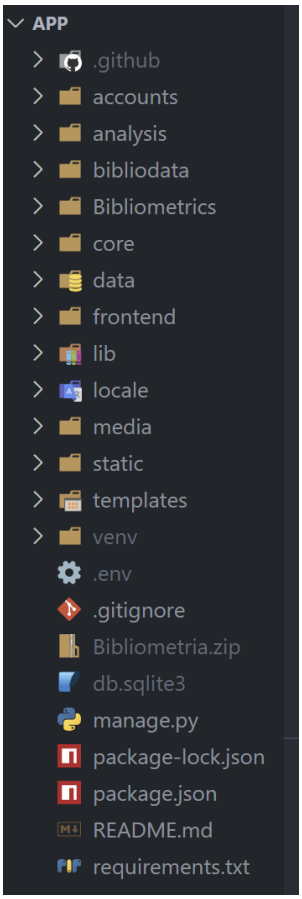
5.1. REQUISITOS Y DEPENDENCIAS PREVIAS

A continuación, se detalla, paso a paso, lo necesario para preparar el entorno local de actualización. Estos pasos deben ejecutarse antes de realizar cualquier operación de migración de base de datos o construcción de activos del *frontend*.

5.1.1. OBTENCIÓN DEL CÓDIGO Y PREPARACIÓN DE LA CARPETA DE TRABAJO

Se detallan los pasos para obtener el código y preparar el entorno de trabajo:

1. **Solicitar acceso al código:** Pide el repositorio o el paquete de código al creador o al equipo de bioinformática del IPBLN. El acceso puede proporcionarse mediante Git (recomendado) o mediante un archivo comprimido.
2. **Crear carpeta de proyecto:** Crea un directorio de trabajo y coloca dentro todo el contenido del proyecto (estructura Django + carpeta *frontend*).
3. **Abrir la carpeta con un editor de código:** Puedes usar cualquier editor, aunque recomiendo VSCode o similar. Todo debe estar incluido en lo que se te ha proporcionado y te debería quedar una estructura como la que ves en la siguiente captura de pantalla:



Lo más probable es que las carpetas de *.github, venv* y *Bibliometria.zip* no las tengas, pero es normal.

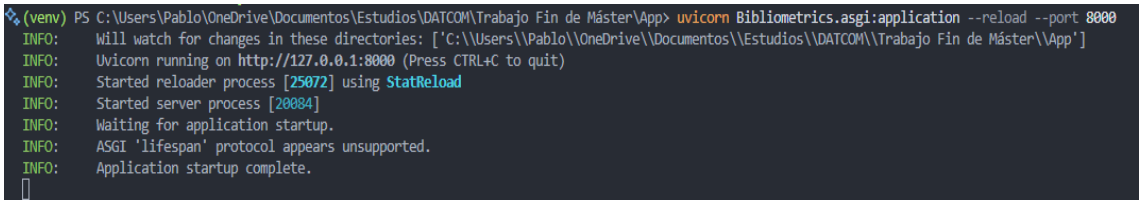
5.1.2. PREPARACIÓN DEL ENTORNO

Debes tener instalado Python con pip. La versión que se ha utilizado para el desarrollo del proyecto es la versión python 3.12.8. Se recomienda tener una versión de python igual o superior a esta para evitar posibles conflictos. Los siguientes pasos son:

1. Crear y activar un entorno virtual (venv): Ejecuta estos comandos en tu terminal dentro de la carpeta del proyecto.
    a. *python -m venv venv*
    b. *cd venv/scripts*.
    c. *.\activate*
    d. *cd ../..*
2. Instalar dependencias del proyecto: Todas las librerías del *backend* están declaradas en *requirements.txt*. Para instalar todo ejecuta:
    a. *pip install -r requirements.txt*
3. Revisar las variables de entorno: Revisa que tienes el archivo .env en la raíz del proyecto y de que están configuradas las siguientes variables de entorno:
    a. MY_API_KEY
    b. MY_PASSWORD
    c. SECRET_KEY

4. Preparación del *frontend*: Es necesario tener instalado Node.js y npm. Las versiones que yo he usado para el proyecto son:
    a. Node.js: v22.16.0
    b. npm: v10.9.2
5. Comprobación final: Ejecuta los siguientes comandos para ver que todo funciona correctamente:
    a. *python manage.py makemigrations*
    b. *python manage.py migrate*
    c. *python manage.py build frontend*
    d. *uvicorn Bibliometrics.asgi:application --reload --port 8000*

Este último paso debe lanzar un servidor en local al cuál puedes acceder y revisar que todo está funcionando correctamente. Deberías ver lo siguiente:



Y accediendo a http://127.0.0.1:8000/BiblioMetrics/ podrás ver la aplicación en funcionamiento. Para cerrar el servidor lanzado en local basta con hacer Ctrl+C sobre la terminal (Windows), CTRL-BREAK en general.

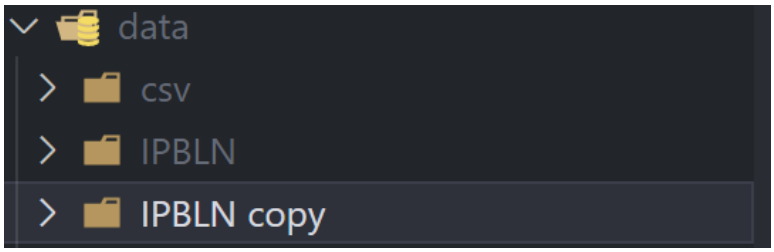
Con este servidor de desarrollo, podemos trabajar en el desarrollo de nuevas funcionalidades en local. En secciones posteriores, se explicará tanto el proceso de actualización de la herramienta (en cuanto a datos se refiere) como el proceso de despliegue de la misma. Querremos desplegar una nueva versión de la aplicación por dos motivos:

1. Hemos actualizado los datos provenientes de GesBIB y queremos que se muestren.
2. Hemos desarrollado nuevas funcionalidades y están listas para el paso a producción.

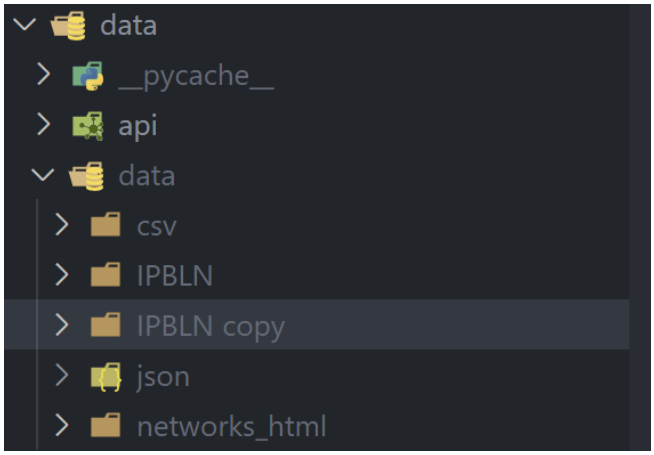
5.2. PROCESO DE ACTUALIZACIÓN DE LA HERRAMIENTA COMPLETO

Esta sección se centra en la explicación, paso a paso, de todo el proceso necesario para actualizar la herramienta con los nuevos datos de GesBIB. Los datos allí se actualizan diariamente así que, de vez en cuando, es necesario actualizar también esta herramienta. Por esa razón, se documentan aquí todos los pasos para llevarlo a cabo.

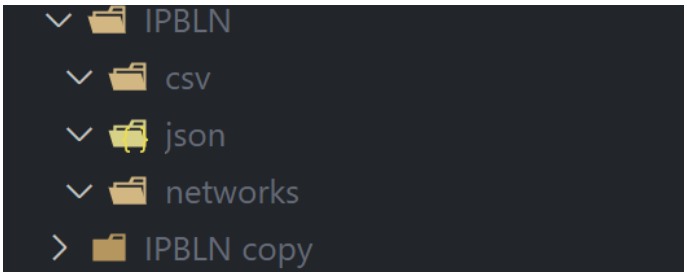
El primer paso que recomiendo hacer es una copia de seguridad de los datos actuales. Para ellos vamos a coger la carpeta que se encuentra en la ruta: data/data/IPBLN y hacer una copia de seguridad de la misma. Debería quedarte algo así:



Para más contexto:



Bien, ahora puedes borrar todo lo que hay en las carpetas csv, json y networks de dentro de la carpeta IPBLN original (en caso de fallos siempre puedes restablecer con la copia que hemos creado). Debe quedar así:



5.2.1. EXTRACCIÓN DE DATOS DE GESBIB

Este paso es crucial y consiste en el sistema de web scraping que extrae directamente parte de los datos de la aplicación, desde la propia web de GesBIB. Para ello, es necesario tener una **cuenta institucional del CSIC con acceso a GesBIB** y disponer de un **navegador compatible** y su correspondiente **WebDriver**:

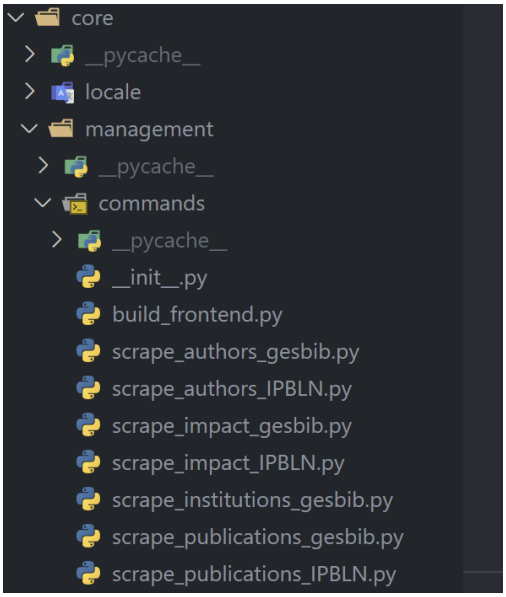
- Para este proyecto se emplea Google Chrome/Chromium. En principio, con la propia instalación de selenium (hecho antes al ejecutar pip install -r requirements.txt) todo debería funcionar correctamente. Pero, sino es así, puede ser necesario instalar el WebDriver adecuado, aquí tienes el enlace donde puedes descargar ChromeDriver: https://googlechromelabs.github.io/chrome-for-testing/

Dicho esto, el procedimiento para ejecutar los *scrapers* es sencillo y consiste en ejecutar, en orden, los siguientes comandos:

1. *python manage.py scrape_institutions_gesbib --user USUARIO --password CONTRASEÑA*
2. *python manage.py scrape_authors_IPBLN --user USUARIO --password CONTRASEÑA*
3. *python manage.py scrape_publications_IPBLN --user USUARIO --password CONTRASEÑA*
4. *python manage.py scrape_impact_IPBLN --user USUARIO --password CONTRASEÑA*

En los apartados de USUARIO y CONTRASEÑA, deberás incluir los datos de acceso a GesBIB, que consisten en el DNI y la contraseña que has creado para ese propósito. Si no tienes una cuenta con acceso a GesBIB, será necesario, como paso previo, solicitar esta cuenta.

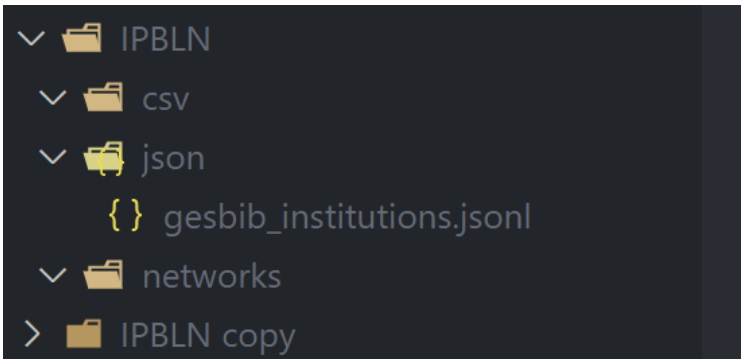
NOTA IMPORTANTE: Es posible que la página de GesBIB o la web de acceso vía CSIC, vayan cambiando con el tiempo y, entonces, es necesario meterse en cada uno de estos comandos y adaptar el código para que funcionen atendiendo a las nuevas webs. Si no te quieres meter en este proceso y/o necesitas ayuda por favor ve a la sección "6. Soporte y Contacto" para solicitar ayuda. Estos 4 comandos se encuentran en la aplicación core de Django, en la siguiente ruta: *core/management/commands/scrape_...*



Ahora sí, vamos paso a paso a explicar cómo funciona y qué hace cada uno de estos comandos:

- *python manage.py scrape_institutions_gesbib --user USUARIO --password CONTRASEÑA*

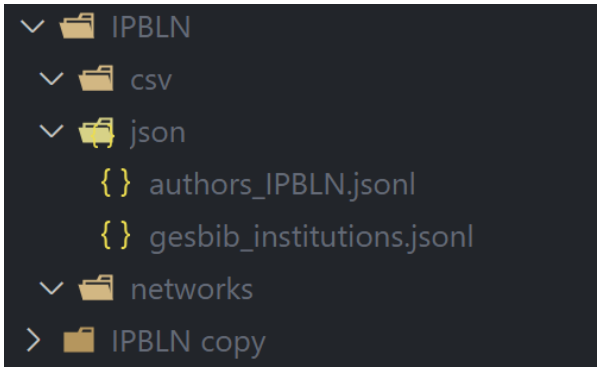
Este comando accede GesBIB y extrae todas las instituciones y las guarda en un archivo jsonl (JSON Lines). Tras su ejecución deberías ver en la carpeta IPBLN el siguiente documento:



Llegados a este punto toca ejecutar:

- *python manage.py scrape_authors_IPBLN --user USUARIO --password CONTRASEÑA*

Este comando accede de la misma manera a GesBIB, a la página de autores del IPBLN y va descargando los datos. Tarda algo más y va guardando páginas de una en una (en el momento que estoy escribiendo esto, sólo hay 3 páginas de autores) en un fichero. Al finalizar el proceso deberías ver el nuevo fichero .jsonl generado:



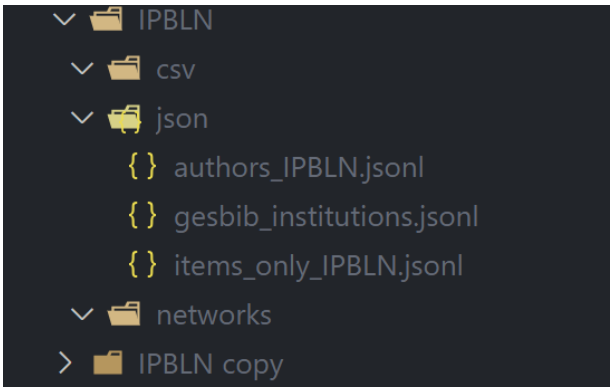
El siguiente comando es:

- *python manage.py scrape_publications_IPBLN --user USUARIO --password CONTRASEÑA*

Este proceso hace lo mismo, pero va extrayendo las publicaciones. Esto tarda algo más que los comandos anteriores y durante su ejecución deberías ver algo así hasta que llega al final del proceso:

```
✅ Página 29 guardada en data/data/IPBLN/json/items_only_IPBLN.jsonl.
[6424:27032:0902/095431.382:ERROR:google_apis\gcm\engine\registration_
✅ Página 30 guardada en data/data/IPBLN/json/items_only_IPBLN.jsonl.
✅ Página 31 guardada en data/data/IPBLN/json/items_only_IPBLN.jsonl.
✅ Página 32 guardada en data/data/IPBLN/json/items_only_IPBLN.jsonl.
✅ Página 33 guardada en data/data/IPBLN/json/items_only_IPBLN.jsonl.
✅ Página 34 guardada en data/data/IPBLN/json/items_only_IPBLN.jsonl.
✅ Página 35 guardada en data/data/IPBLN/json/items_only_IPBLN.jsonl.
✅ Página 36 guardada en data/data/IPBLN/json/items_only_IPBLN.jsonl.
✅ Scraping finalizado.
```

Una vez finalizado, deberías ver el siguiente archivo y tener la siguiente estructura:



Por último, vamos a ejecutar el último de los comandos de extracción vía scrapping:

- *python manage.py scrape_impact_IPBLN --user USUARIO --password CONTRASEÑA*

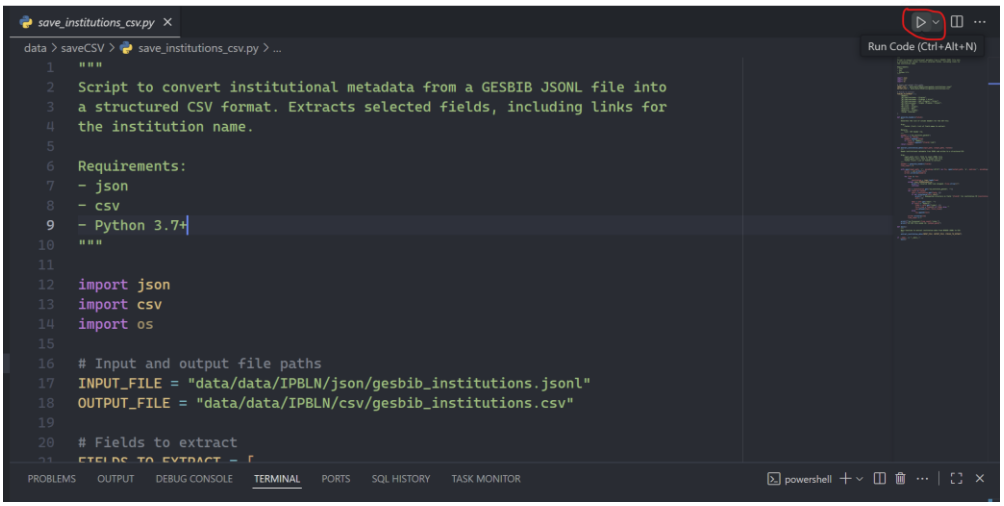
Este último comando funciona de la misma manera y extrae las métricas de impacto de cada una de las publicaciones, el proceso debe tardar algo más que el comando anterior y el número de registros extraídos debe ser el mismo que en el comando anterior. Una vez finalizado el proceso debes tener lo siguiente:

En general todos los comandos, pero en especial este último comando me ha tocado actualizarlo con frecuencia puesto que la web, concretamente los checkbox que se van activando, cambian ligeramente. Si no te funciona o encuentras algún problema puedes intentar actualizar el comando en el código o ponerte en contacto con alguien que te ayude.

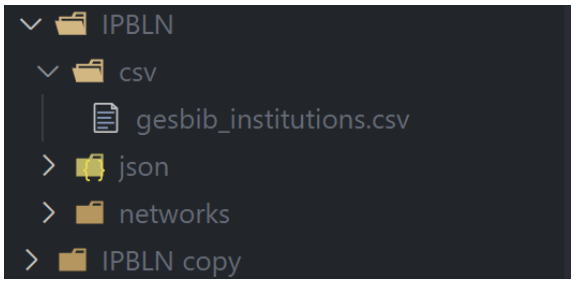
5.2.2. PASAR LOS DATOS A FORMATO CSV

Una vez extraídos los ficheros .jsonl, toca convertirlos a .csv. Para ello seguiremos los siguientes pasos. Debemos acceder a *data/saveCSV/,* en esta ruta encontrarás los archivos de conversión de .jsonl a .csv. Vamos paso a paso a convertir cada uno de los archivos:

Lo primero que vamos a hacer es abrir el archivo *data/saveCSV/save_institutions_csv.py* y ejecutarlo, tal y como se muestra en la siguiente imagen:

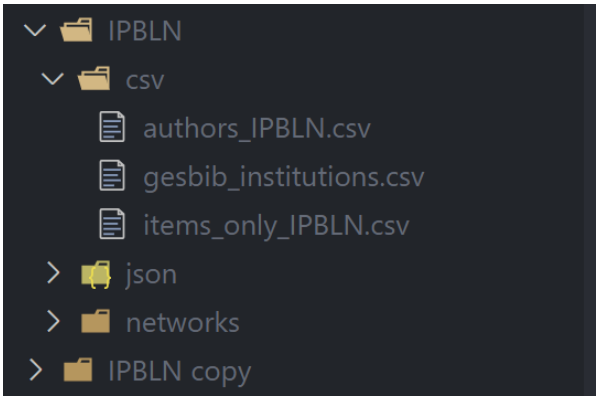


Una vez ejecutado veremos:

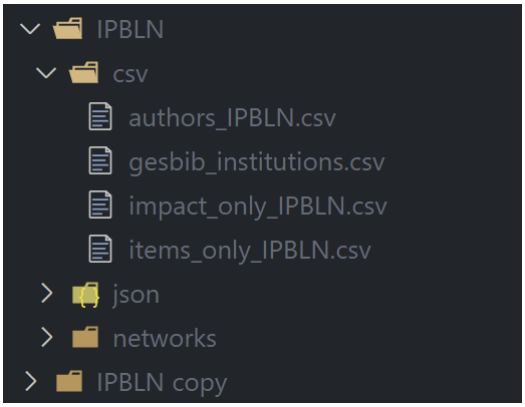


Continuaremos haciendo exactamente lo mismo con el archivo *data/saveCSV/save_authors_csv.py*. Tras ejecutar debemos ver:

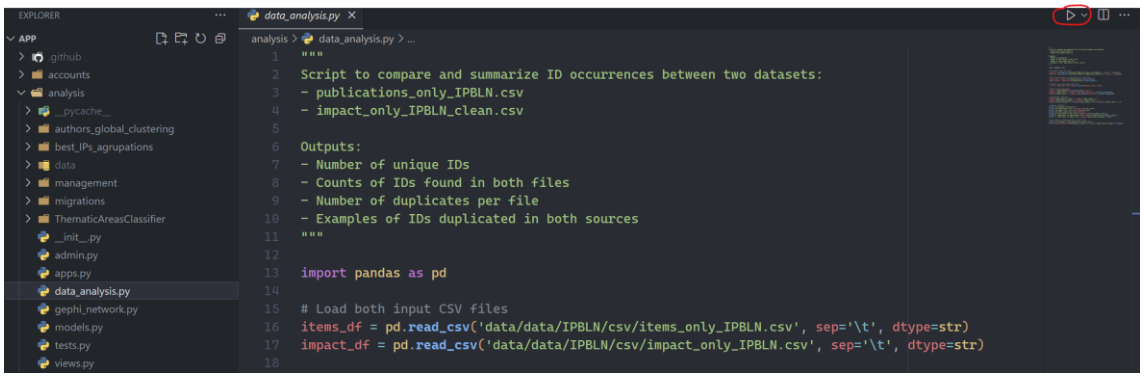
A partir de aquí podemos continuar con el archivo *data/saveCSV/save_publications_csv.py*. Al ejecutar debemos tener lo siguiente:



Repetimos el proceso con: *data/saveCSV/save_impact_csv.py*. Ya hemos convertido todos los archivos a .csv y debemos tener lo siguiente:



Por último, aunque no es necesario, recomiendo ejecutar el siguiente archivo *análisis/data_analysis.py* exactamente como hemos hecho con los archivos anteriores:



Si obtenemos el siguiente resultado por consola, es que todo está correcto en cuanto a los datos se refiere:

```
📊 Summary statistics:
📁 Publications file: 3592 rows
📁 Impact file: 3592 rows
🔢 Total unique IDs: 3592
✅ IDs present in both files: 3592
♻️Duplicates in publications file: 0
♻️Duplicates in impact file: 0

📌 Example of IDs duplicated in both files:
Empty DataFrame
Columns: [items_count, impact_count, duplicated_in_items, duplicated_in_impact, exists_in_both]
Index: []
```

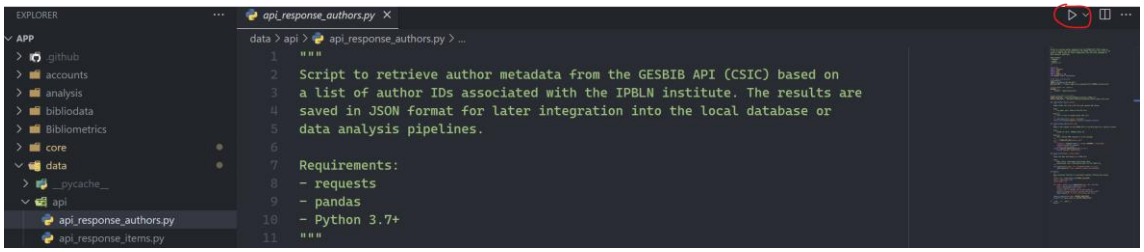
Si no tenemos este resultado, puede deberse a una inconsistencia en los datos y se recomienda repetir el proceso de extracción y scrapping, significaría que las publicaciones no se corresponden correctamente con el impacto de las mismas.

5.2.3. EXTRAER MÁS DATOS DETALLADOS A TRAVÉS DE LA API DE GESBIB

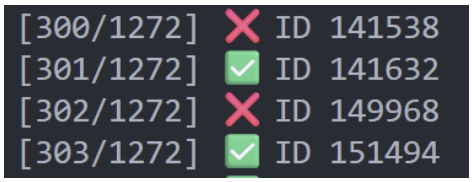
El siguiente paso es extraer más datos disponibles en la API de GesBIB pero no accesibles a través de scrapping. Por esta razón, necesitamos esta sección en la que se ejecutarán

dos archivos que acceden a la API de GesBIB y extraen más datos de los autores y de las publicaciones.

¿Y no podríamos haber hecho esto directamente? La respuesta es no, necesitamos los pasos anteriores porque necesitamos todos los IDs que guarda GesBIB, tanto de publicaciones como de autores, para ir haciendo consultas por ID a la API. Dicho esto, lo primero que tenemos que hacer es acceder a *data/api/api_response_authors.py* y ejecutar al igual que hemos estado haciendo hasta ahora:

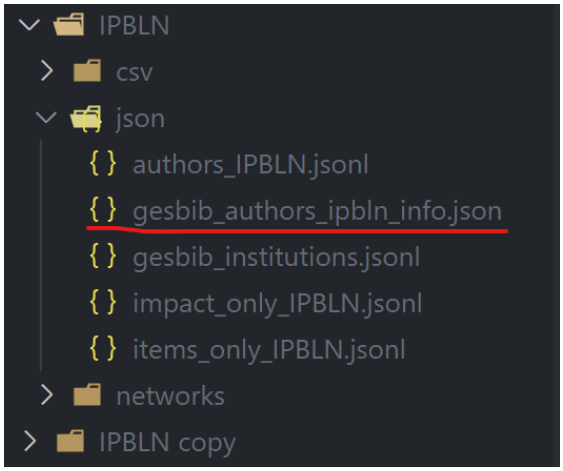


Este proceso lo que hace es recorrer todos los autores que tenemos guardados y pedirle a la API más datos sobre ellos. Esto tarda un rato, y no te preocupes si durante la ejecución ves algún error del tipo:



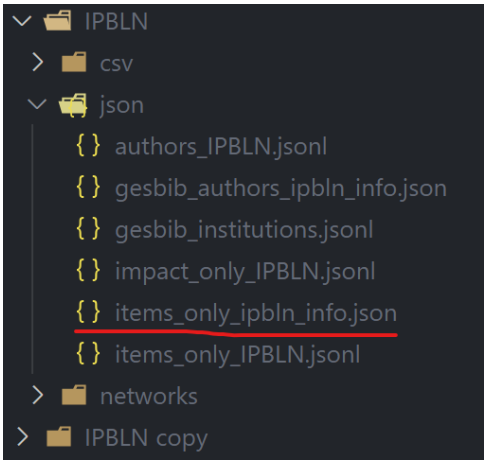
Esto solo significa que el autor no es accesible a través de la API y de nuestra API KEY por lo que no tendremos más datos sobre ellos. La aplicación gestiona esto automáticamente y lo único que ocurrirá es que para esos autores habrá menos información disponible dentro de la aplicación web. De cara al final suele haber más autores con este tipo de error, puesto que son los últimos añadidos y a lo mejor no tienen su información en la API todavía o no son accesibles con nuestra API KEY, probablemente cuando volvamos a actualizar más adelante, muchos de estos autores estarán disponibles y habrá muchos nuevos que no.

Cuando finalice todo el proceso, deberíamos ver el siguiente archivo:



Repetiremos este mismo proceso con el archivo *data/api/api_response_items.py*.

Este proceso es más lento, puesto que hay más publicaciones. Pero es exactamente lo mismo que para los autores. Una vez finalizado el proceso, deberíamos ver también:

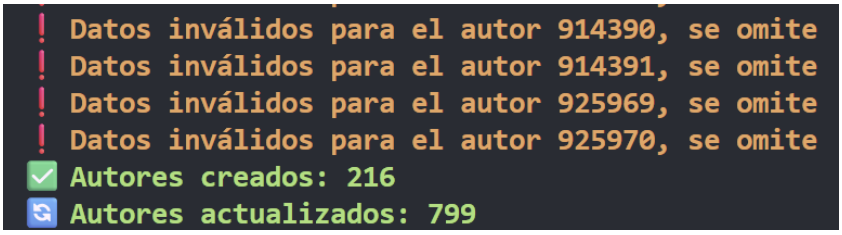


Con esto finaliza el proceso de extracción de datos extra a través de la API.

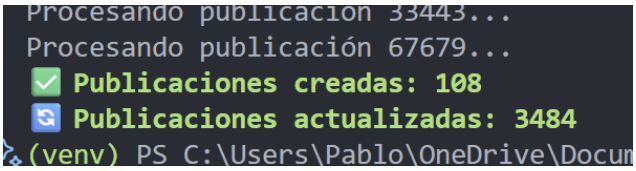
5.2.4. CARGAR LOS DATOS EXTRAÍDOS HASTA EL MOMENTO EN LA BASE DE DATOS

Una vez hemos llegado a este punto, podemos cargar todos los datos que tenemos en nuestra base de datos (recomendablemente la base de datos descargada desde producción). Para hacerlo es muy sencillo, simplemente ejecutaremos en orden los siguientes comandos por terminal (estos comandos se encuentran en *bibliodata/management/commands/load_...*:

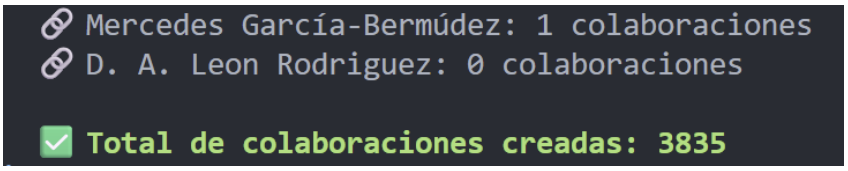
- *python manage.py load_authors_db:* Esto carga en base de datos todos los datos sobre los investigadores del IPBLN. Los investigadores a los cuales no hemos tenido acceso a través de la API se omiten. Tras la ejecución deberías ver algo similar a:



- *python manage.py load_institutions_db:* Este comando hace lo mismo con las instituciones. Al finalizar muestra un mensaje de que el proceso ha finalizado con éxito.
- *python manage.py load_publications:* Carga en base de datos todos los datos sobre las publicaciones. Tarda un rato y al final deberías ver:



- *python manage.py load_collaborations:* Este comando crea una colaboración por cada dos autores que participan juntos en una publicación, recorre todos los autores y las publicaciones de cada uno y va guardando para cada autor sus colaboraciones. Al finalizar deberías ver:

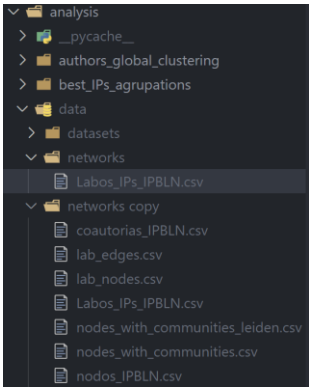


El número de publicaciones, colaboraciones y resto de datos dependerá de cuántos datos nuevos hayan actualizado en GesBIB y cuántos tuvieras antes, es normal que no te aparezcan el mismo número de datos actualizados por comando que a mí. Una vez hecho esto ya tenemos todos nuestros datos guardados. Pero hace falta actualizar alguna cosa más, antes de poder volver a desplegar nuestra aplicación. Se explica en las siguientes secciones.

5.2.5. PREPARACIÓN DE TODAS LAS REDES DE COLABORACIÓN

Este proceso es largo y, consiste en preparar todas las redes de colaboración que se van a ver en la sección correspondiente de la web. Se montan las redes (nodos y enlaces), y se ejecutan todos los algoritmos de clustering y detección de comunidades. El proceso es el siguiente:

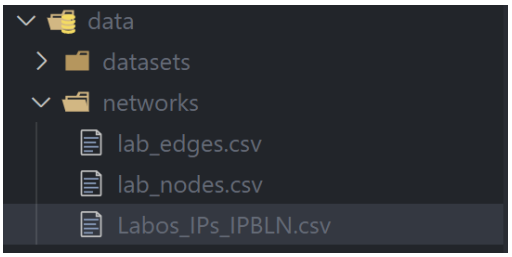
Lo primero que vamos a hacer es una copia de seguridad de la carpeta *analysis/data/networks/* y, luego, borrar todos los archivos que haya en dentro de esa carpeta, menos el archivo *Labos_IPs_IPBLN.csv*. Nos queda la siguiente estructura:



Lo siguiente, es ejecutar dos comandos que se sitúan en *analysis/management/commands/...*:

- *python manage.py generate_IPs_labs_network*

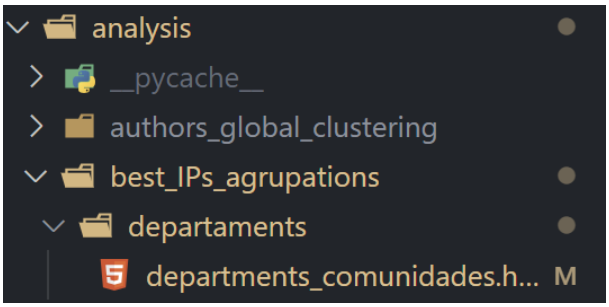
Este comando crea los documentos necesarios con la información de la red (nodos y aristas) de investigadores principales del centro. Puede que, al ejecutar, salga que no ha encontrado a alguno de los IPs del centro en la base de datos, esto no es problema, simplemente en GesBIB son investigadores que acaban de llegar y todavía no hay información de ellos en la API. Tras la ejecución deberíamos ver dos archivos nuevos:



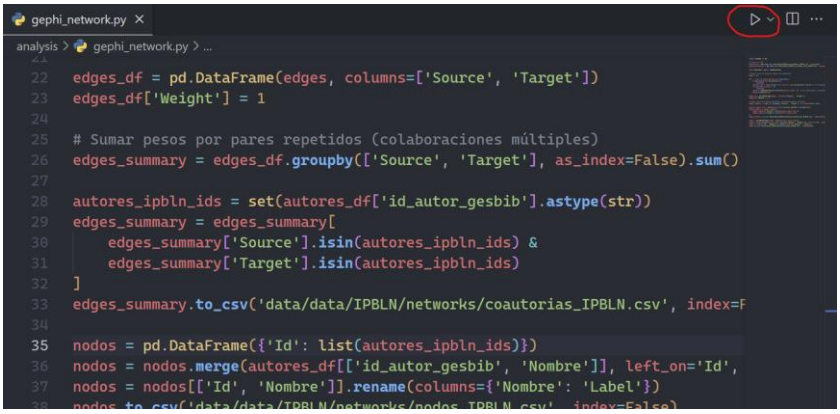
El siguiente paso es ejecutar:

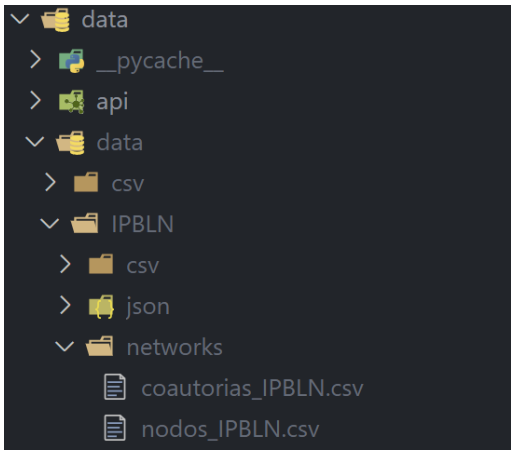
- *python manage.py generate_IPs_departments_HTML*

Esto genera un HTML interactivo de la red de IPs agrupados por departamentos. Este HTML es el que se exporta en los informes de la aplicación. El comando guarda un archivo tal y como se ve en la imagen:

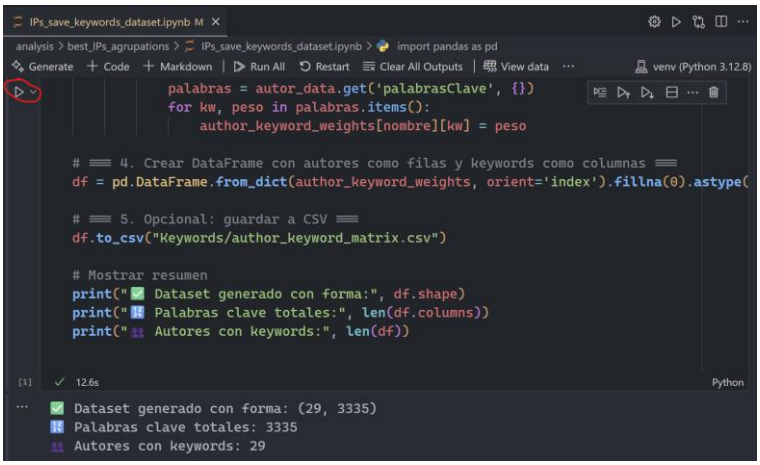


Lo siguiente que vamos a hacer es abrir el archivo *analysis/gephi_network.py* y ejecutarlo, esto crea dos archivos en la ruta *data/data/IPBLN/networks* y otros dos en *analysis/data/networks*:





Una vez hecho todo esto, toca ejecutar los notebooks donde están los algoritmos de clustering y detección de comunidades. Para ello, vamos a trabajar en dos carpetas, *analysis/authors_global_clustering* y *analysis/best_IPs_agrupations*. Lo primero, vamos a ejecutar el notebook analysis/*best_IPs_agrupations*/IPs_save_keywords_dataset.ipynb. Abrimos el archivo y ejecutamos la única celda que hay en él:



Esto debe reescribir el archivo *analysis/best_IPs_agrupations_author_keyword_matrix.csv,* que es importante para todos los algoritmos posteriores. Lo siguiente que vamos a hacer es, por orden, ejecutar todas las celdas que hay en los siguientes 3 notebooks. Abrimos el primero y ejecutamos todas sus celdas, luego lo mismo con el segundo y luego con el tercero. Los archivos son:

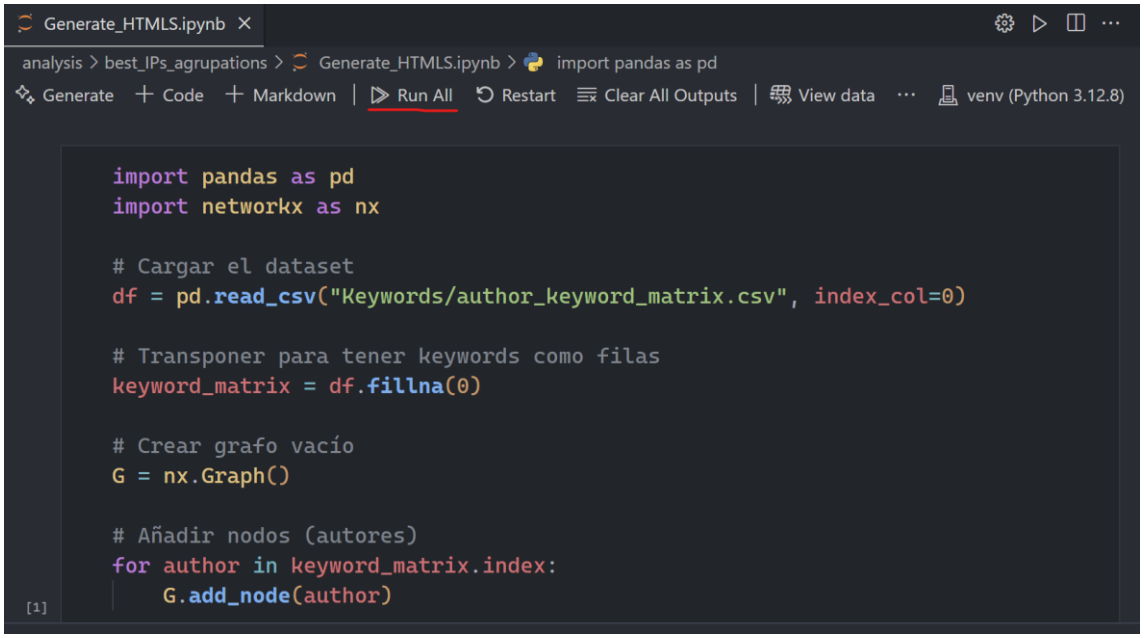
1. *analysis/authors_global_clustering/networks/authors_leiden.ipynb*
2. *analysis/authors_global_clustering/networks/authors_lovaina.ipynb*
3. *analysis/authors_global_clustering/networks/authors_semisupervised.ipynb*

Los dos primeros notebooks ejecutan los algoritmos de Leiden y Lovaina respectivamente y guardan las particiones para mostrar en la red. Esto lo hace a nivel global, para las redes completas. El tercero, de una manera semisupervisada, a partir de los nodos semilla (son los IPs, ya que tenemos los datos de los departamentos a los que pertenecen), clasifica todo el resto de los investigadores del centro (red completa) en los 3 departamentos. Este último notebook es más largo y tarda algo más. No deberías encontrar problemas en la ejecución de las celdas y, cuando haya finalizado todo el proceso, podemos pasar con el

siguiente. El siguiente paso es hacer exactamente lo mismo, pero con otros notebooks, esta vez se encuentran en analysis/best_IPs_agrupations y hay que ejecutar, en orden, todas las celdas, de todos estos notebooks, también en orden:

- *analysis/best_IPs_agrupations/Generate_HTMLS*
- *analysis/best_IPs_agrupations/IPs_Agglomerative*
- *analysis/best_IPs_agrupations/IPs_DBSCAN*
- *analysis/best_IPs_agrupations/IPs_GMM*
- *analysis/best_IPs_agrupations/IPs_HDBSCAN*
- *analysis/best_IPs_agrupations/IPs_Hierarchical*
- *analysis/best_IPs_agrupations/IPs_KMeans*
- *analysis/best_IPs_agrupations/IPs_leiden*
- *analysis/best_IPs_agrupations/IPs_lovaina*
- *analysis/best_IPs_agrupations/IPs_Spectral_clustering*
- *analysis/best_IPs_agrupations/IPs_graph_by_keywords*

Estos notebooks, ejecutan todos los algoritmos de clustering y detección de comunidades sobre la red de IPs del centro, es todo lo que veremos luego en la sección “Por palabras clave” de las redes de coautoría de la aplicación web. Ejecuta todas las celdas en orden y no deberías encontrar problemas. Para ejecutar todas las celdas de un notebook puedes hacer lo siguiente, por ejemplo con el primero:



Una vez hecho este proceso, los notebooks no tardan mucho en ejecutarse, podemos guardar todo lo generado en base de datos, para que la aplicación web pueda mostrarlos correctamente.

5.2.6. CARGAR EN LA BASE DE DATOS LAS REDES DE COLABORACIÓN

Para guardar todos los datos, tenemos que ejecutar por terminal un montón de comandos en orden. Simplemente debes copiar de aquí los comandos y ejecutarlos por tu terminal.

Los comandos son los siguientes, es importante ejecutarlos en orden:

1. *python manage.py load_departments_db*
2. *python manage.py load_departments_global_db*
3. *python manage.py load_leiden_community*
4. *python manage.py load_leiden_global_db*
5. *python manage.py load_lovaina_community*
6. *python manage.py load_lovaina_global_db*
7. *python manage.py load_IPs_clustering --file analysis/best_IPs_agrupations/Keywords/KMeans/kmeans_best_configs.csv --model kmeans*
8. *python manage.py load_IPs_clustering --file analysis/best_IPs_agrupations/Keywords/Agglomerative/agglomerative_best_configs.csv --model agglomerative*
9. *python manage.py load_IPs_clustering --file analysis/best_IPs_agrupations/Keywords/GMM/gmm_best_configs.csv --model gmm*
10. *python manage.py load_IPs_clustering --file analysis/best_IPs_agrupations/Keywords/Hierarchical/hierarchical_best_configs.csv --model hierarchical*
11. *python manage.py load_IPs_clustering --file analysis/best_IPs_agrupations/Keywords/HDBSCAN/hdbscan_best_configs.csv --model hdbscan*
12. *python manage.py load_IPs_clustering --file analysis/best_IPs_agrupations/Keywords/DBSCAN/dbscan_best_configs.csv --model dbscan*
13. *python manage.py load_IPs_clustering --file analysis/best_IPs_agrupations/Keywords/Lovaina/lovain_best_configs.csv --model Lovaina*
14. *python manage.py load_IPs_clustering --file analysis/best_IPs_agrupations/Keywords/Spectral/spectral_best_configs.csv --model spectral*

Si encontraras algún problema, probablemente sea porque no has ejecutado todas las celdas de alguno de los notebooks anteriores. Por favor inténtalo de nuevo y si el problema persiste puedes intentar solucionarlo en el código o poniéndote en contacto con alguien. Una vez finalizado el proceso, ya tenemos toda la sección de redes de coautoría correctamente actualizada y ya sólo falta una sección.

5.2.7. PREDICCIÓN DE LAS ÁREAS TEMÁTICAS

Lo único que falta por hacer y actualizar es la predicción de las áreas temáticas de aquellas publicaciones que no tengan un área temática definida. El proceso para hacerlo es el siguiente:

Lo primero es ejecutar los dos comandos que crean los dos datasets para los modelos. El dataset de training y el de predicción. Los dos comandos que tienes que ejecutar por terminal son:

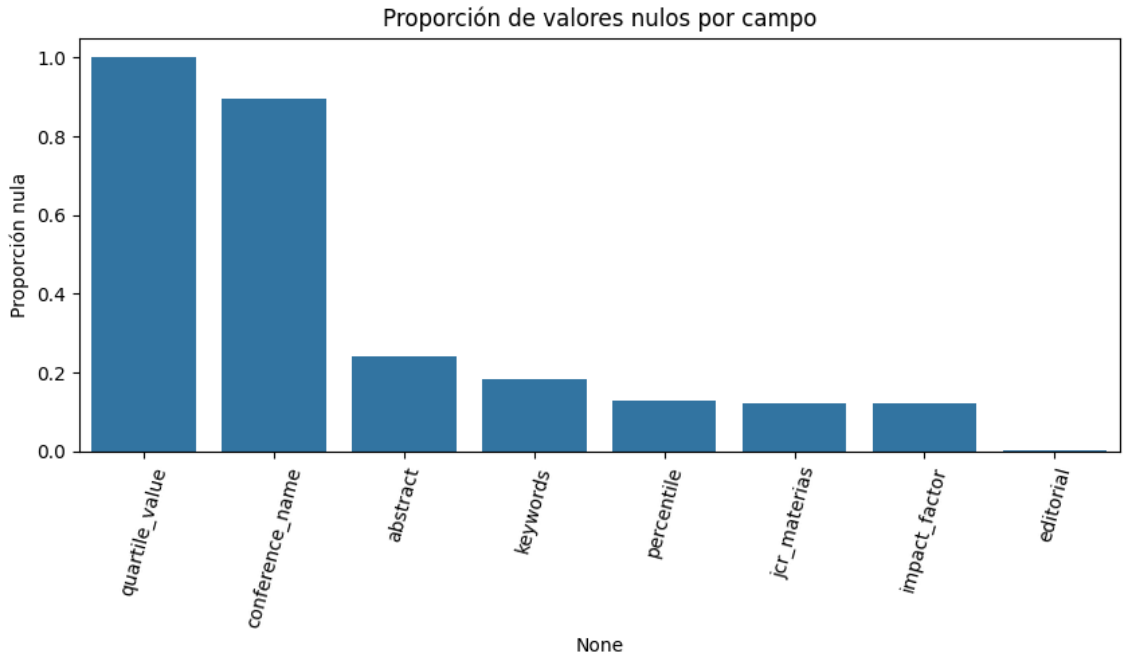
1. *python manage.py generate_publication_training_data*
2. *python manage.py generate_publication_to_predict_data*

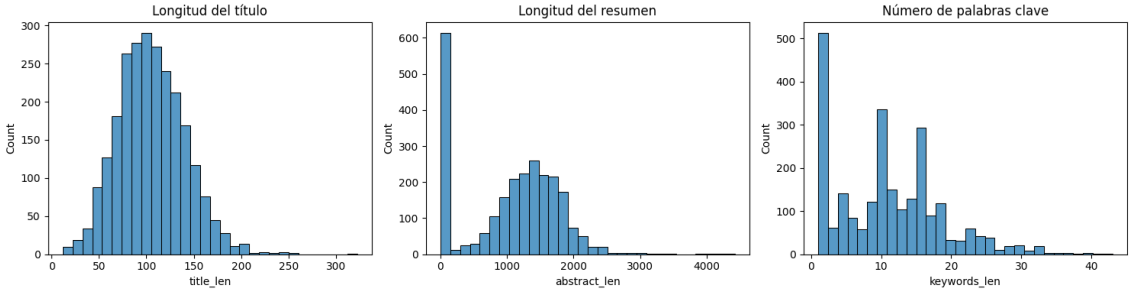
Tras ejecutar ambos comandos deberías ver algo así (el número de filas puede variar):

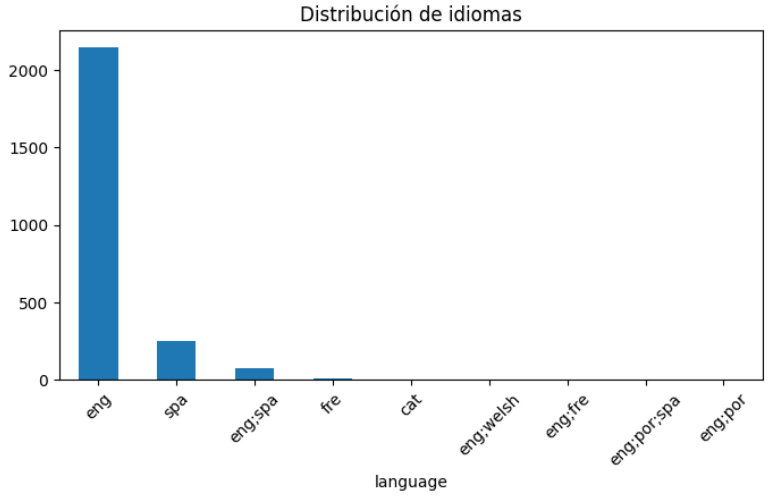
```
(venv) PS C:\Users\Pablo\OneDrive\Documentos\Estudios\DATCOM\Trabajo Fin de Máster\App> python manage
.py generate_publication_training_data
Procesando publicaciones: 100%|                                        | 3649/3649 [00:04<00:00, 887.84it/s]
Dataset exportado en analysis/data/datasets/publications_training_dataset.csv (2422 filas)
(venv) PS C:\Users\Pablo\OneDrive\Documentos\Estudios\DATCOM\Trabajo Fin de Máster\App> python manage
.py generate_publication_to_predict_data
Procesando publicaciones: 100%|                                        | 3649/3649 [00:02<00:00, 1431.09it/s]
Dataset exportado en analysis/data/datasets/publications_to_predict_dataset.csv (1227 filas)
(venv) PS C:\Users\Pablo\OneDrive\Documentos\Estudios\DATCOM\Trabajo Fin de Máster\App>
```

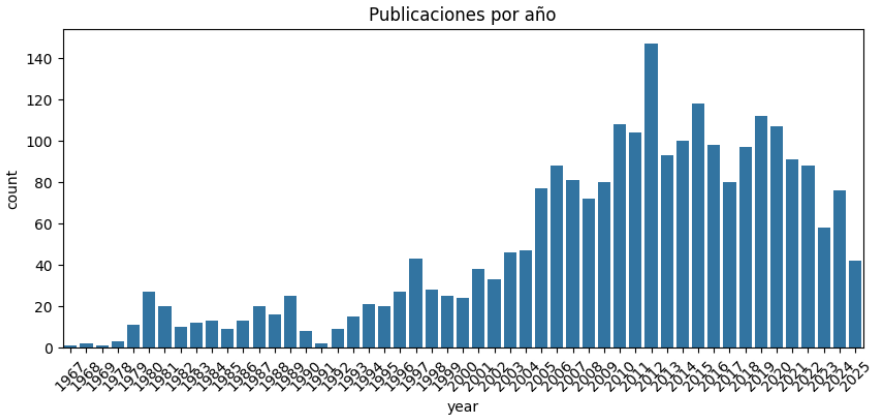
Comprueba que en la ruta especificada se encuentran realmente los archivos. El siguiente paso es ejecutar todas las celdas de 4 notebooks, como ya hemos hecho anteriormente, en orden. Los notebooks son los siguientes:

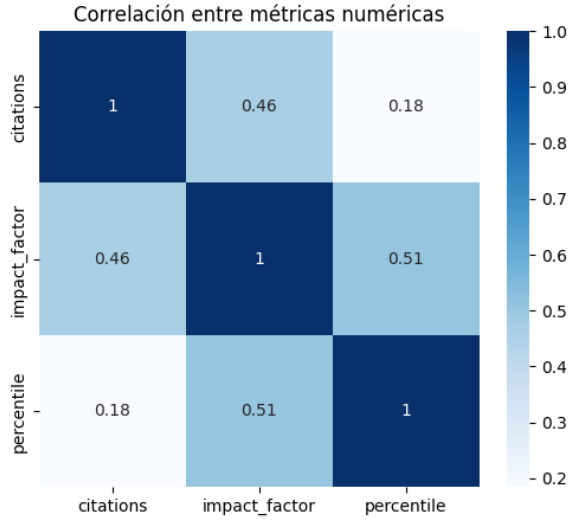
1. *analysis/ThematicAreasClassifier/ThematicAareasEDA.ipynb:* No es estrictamente necesario ejecutar este Notebook, pero si que es conveniente para hacerse una idea de los datos con los que va a trabajar en modelo y, sobre todo, ver si lo que se dice en él tiene sentido con los gráficos y datos que tu tienes en ese momento, yo me he dado cuenta de errores en la carga de datos de publicaciones gracias a este Notebook que he podido corregir gracias a ello. Te adjunto algunas capturas de pantalla para que compruebes que tus datos tienen buena pinta y puedes proseguir con el preprocesamiento, debes obtener cosas parecidas, ya que el preprocesamiento se hace en función de esto:




Longitud del título
Count
title_len
Longitud del resumen
Count
abstract_len
Número de palabras clave
Count
keywords_len


Distribución de idiomas
language
eng
spa
eng;spa
fre
cat
eng;welsh
eng;fre
eng;por;spa
eng;por

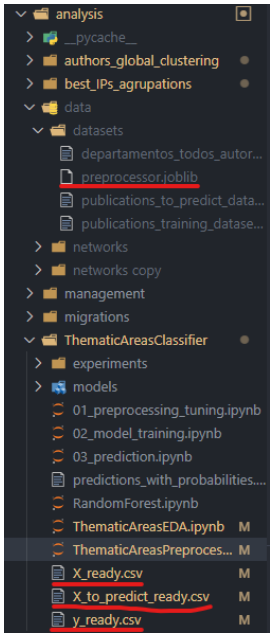

Publicaciones por año
count
year


Correlación entre métricas numéricas
citations
impact_factor
percentile
1
0.46
0.18
0.46
1
0.51
0.18
0.51
1

Bien, el siguiente notebook ya si que es imprescindible y es el de preprocesamiento. Ejecuta todas las celdas de este notebook:

2. *analysis/ThematicAreasClassifier/ThematicAareasPreprocessing.ipynb*

Tarda un rato, pero lo más importante es que veas 3 archivos generados. Te los remarco en la siguiente imagen:



Con esto, ya tenemos todos los datos correctamente procesados para empezar con el fine tunning y el entrenamiento en sí del modelo. Estos son los dos siguientes notebooks:

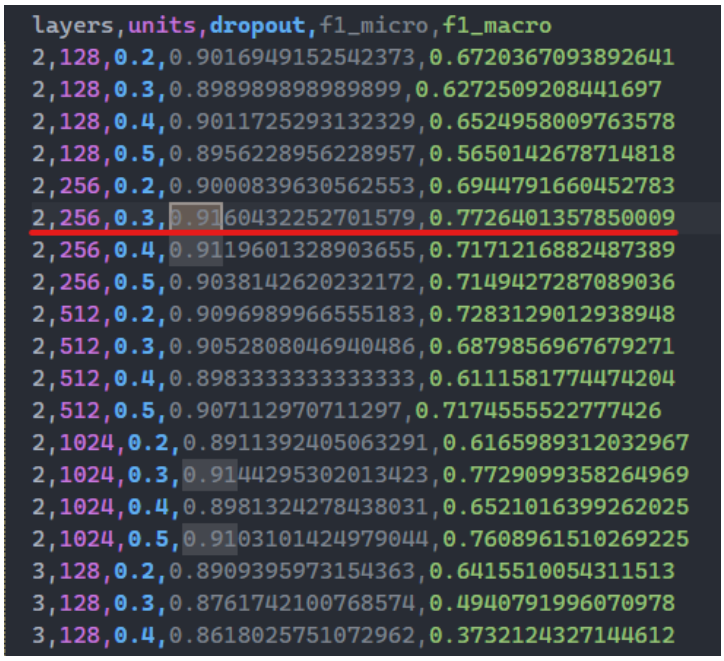
3. *analysis/ThematicAreasClassifier/01_preprocessing_tuning.ipynb*

Este notebook tarda bastante, alrededor de 5-6 minutos, y lo importante, es que veas actualizado todas las subcarpetas que hay en la carpeta experiments:

Ahora pasamos con el entrenamiento del modelo:

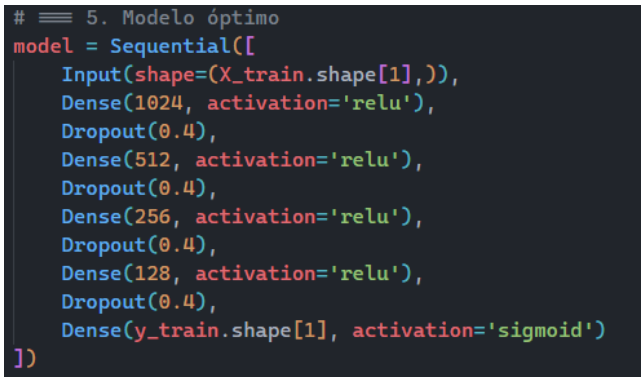
4. *analysis/ThematicAreasClassifier/02_model_training.ipynb*

Puedes ejecutarlo tal y como está. Lo más probable es que el modelo definido ahí sea bastante adecuado para tus datos también. Sin embargo, lo conveniente es que, tras ejecutar la tercera celda del notebook (tarda un rato), mires para tu caso, dentro del archivo generado *nn_gridsearch_results.csv*, cuál ha sido el mejor modelo para tus datos. Es importante que tengan una buena combinación de F1_micro y de F1_macro:



```
layers,units,dropout,f1_micro,f1_macro
2,128,0.2,0.9016949152542373,0.6720367093892641
2,128,0.3,0.898989898989899,0.6272509208441697
2,128,0.4,0.9011725293132329,0.6524958009763578
2,128,0.5,0.8956228956228957,0.5650142678714818
2,256,0.2,0.9000839630562553,0.6944791660452783
2,256,0.3,0.9160432252701579,0.7726401357850009
2,256,0.4,0.9119601328903655,0.7171216882487389
2,256,0.5,0.9038142620232172,0.7149427287089036
2,512,0.2,0.9096989966555183,0.7283129012938948
2,512,0.3,0.9052808046940486,0.6879856967679271
2,512,0.4,0.8983333333333333,0.6111581774474204
2,512,0.5,0.907112970711297,0.7174555522777426
2,1024,0.2,0.8911392405063291,0.6165989312032967
2,1024,0.3,0.9144295302013423,0.7729099358264969
2,1024,0.4,0.8981324278438031,0.6521016399262025
2,1024,0.5,0.9103101424979044,0.7608961510269225
3,128,0.2,0.8909395973154363,0.6415510054311513
3,128,0.3,0.8761742100768574,0.4940791996070978
3,128,0.4,0.8618025751072962,0.3732124327144612
```

En mi caso ha sido el modelo con 2 capas, 256 unidades y 0.3 de dropout, pero esto puede variar. Ahora, en la celda 4, debemos construir este modelo. Para ello, en la sección 5.Modelo óptimo construimos la red de la siguiente manera. Primero os enseño el modelo que tengo yo de la vez anterior:



```
# ═══ 5. Modelo óptimo
model = Sequential([
    Input(shape=(X_train.shape[1],)),
    Dense(1024, activation='relu'),
    Dropout(0.4),
    Dense(512, activation='relu'),
    Dropout(0.4),
    Dense(256, activation='relu'),
    Dropout(0.4),
    Dense(128, activation='relu'),
    Dropout(0.4),
    Dense(y_train.shape[1], activation='sigmoid')
])
```

Es un modelo de 4 capas con 1024 unidades y 0.4 de dropout. Entonces, lo primero que voy a hacer es quitar las últimas dos capas, ya que mi nuevo modelo óptimo es solo de dos:

```python
# ═══ 5. Modelo óptimo
model = Sequential([
    Input(shape=(X_train.shape[1],)),
    Dense(1024, activation='relu'),
    Dropout(0.4),
    Dense(512, activation='relu'),
    Dropout(0.4),
    Dense(y_train.shape[1], activation='sigmoid')
])
```

Ahora, ajusto las unidades, que empiezan en 256 (y siempre va dividiendo entre 2 el número de unidades en la siguiente capa). Y también ajusto el dropout, en este caso a 0.3:

```python
# ═══ 5. Modelo óptimo
model = Sequential([
    Input(shape=(X_train.shape[1],)),
    Dense(256, activation='relu'),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(y_train.shape[1], activation='sigmoid')
])
```

Y así quedaría mi nuevo modelo construido. Puedes probar con varios por si algún otro modelo obtiene buenas métricas tras probarlo en test, normalmente modelos con más capas 3 o 4 funcionan mejor en test, te recomiendo probarlos. El mío ha obtenido las siguientes métricas:

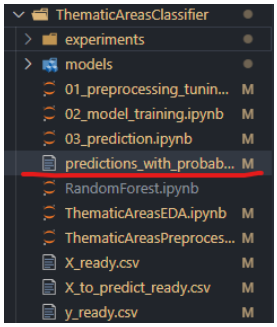
```
...
                    weighted avg       0.91      0.91      0.90       620
                     samples avg       0.93      0.93      0.92       620

Multilabel exact match accuracy: 0.8246
```

Deberías conseguir más de 0.9 en todo y en exact accuracy más de 0.8 (prueba otros modelos con los que hayas obtenido buenas métricas en *nn_gridsearch_results.csv* si no estás conforme con tus resultados). A continuación, vamos a ejecutar la única celda del notebook:

5. *analysis/ThematicAreasClassifier/03_prediction.ipynb*

Este último notebook guarda las predicciones hechas por el modelo entrenado de aquellas publicaciones que no tienen un área temática asignada. Debes ver modificado el siguiente archivo:



Una vez hecho guardamos las predicciones en la base de datos con el siguiente comando por consola:

- *python manage.py load_predicted_thematic_areas*

Deberías ver algo así:

```
Publicaciones actualizadas: 1163
```

4.5.1. CONSTRUCCIÓN DEL FRONTEND CON LOS NUEVOS DATOS

Por último, para asegurarnos de que todo funciona correctamente ejecutaremos estos dos últimos comandos:

- *python manage.py build_frontend*
- *uvicorn Bibliometrics.asgi:application --reload --port 8000*

Esto se asegura de que toda la parte visual está actualizada y ejecuta el servidor de pruebas en local. Ahora te recomiendo navegar por la aplicación y hacer tus pruebas para ver que todo funciona correctamente. Debes ver algo así en la consola:

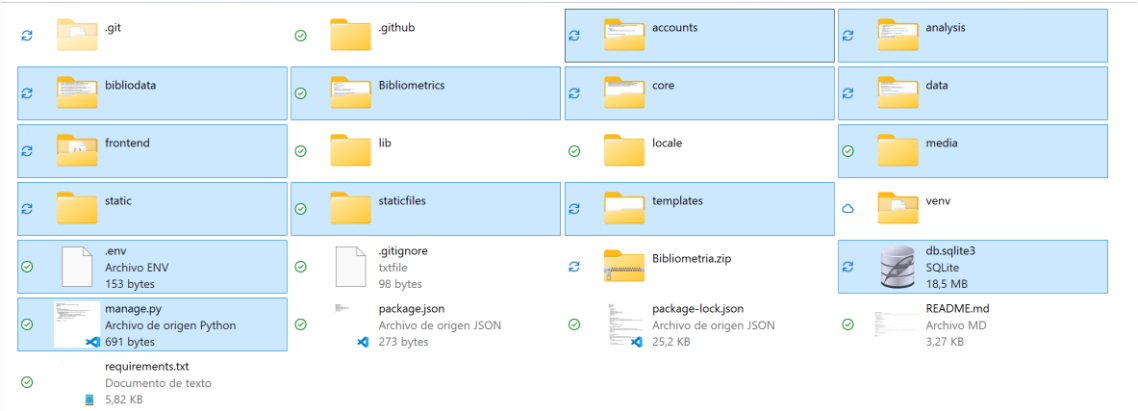
```
(venv) PS C:\Users\Pablo\OneDrive\Documentos\Estudios\DATCOM\Trabajo Fin de Máster\App> uvicorn Bibliometrics.asgi:application --reload --port 8000
INFO:     Will watch for changes in these directories: ['C:\\Users\\Pablo\\OneDrive\\Documentos\\Estudios\\DATCOM\\Trabajo Fin de Máster\\App']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [25072] using StatReload
INFO:     Started server process [20084]
INFO:     Waiting for application startup.
INFO:     ASGI 'lifespan' protocol appears unsupported.
INFO:     Application startup complete.
```

Y puedes acceder a este enlace en tu navegador para navegar por la aplicación actualizada: http://127.0.0.1:8000/BiblioMetrics/

5. DESPLIEGUE

Tanto si se ha actualizado la aplicación web con nuevos datos, como si se quieren mostrar nuevas funcionalidades desarrolladas, es importante entender y tener claro el procedimiento de despliegue de la herramienta. En esta sección lo detallaremos paso a paso:

El primero es que debes comprimir todas las carpetas seleccionadas en un archivo .zip llamado: Bibliometria.zip:



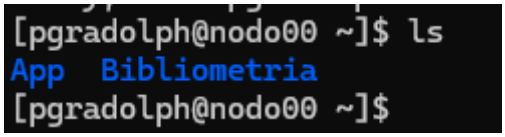
Lo que debes comprimir es:

- *accounts*.
- *analysis*.
- *bibliodata*.
- *Bibliometrics*.
- *core*.
- *data*.
- *frontend*.
- *media*.
- *static*.
- *staticfiles*.
- *templates*.
- *.env*
- *db.sqlite3*
- *manage.py*

El siguiente paso es acceder a Halowan y a Hansolo para eliminar o mover la versión antigua de la aplicación y así poder actualizar con nuestra nueva carpeta .zip. Para ello, necesitas tener acceso a estos servidores con tu usuario y contraseña. Vamos a abrir una terminal desde donde hemos guardado la carpeta comprimida y a ejecutar los siguientes comandos (necesitas estar conectado a un WiFi externo al CSIC o tener una VPN):

- *ssh halowan.ipb.csic.es -l nombre_usuario*

Esto nos pedirá una contraseña y al introducirla accedemos, en mi caso:



Como seguramente no tengamos permisos para eliminar, podemos renombrar la carpeta de Bibliometria a VersionAntigua, por ejemplo, con el comando:

- *mv Bibliometria VersionAntigua*

Si tienes permisos para eliminar o si puedes hablar con alguien que los tenga, entonces es mejor ir eliminando versiones antiguas o, quizá justo la versión anterior no, por si acaso, pero otras que se acumulen sí. Cuando ya tenemos esto hecho, vamos a hacer lo mismo en Hansolo, para ello, lo primero es acceder:

- *ssh nombre_usuario@hansolo.ipb.csic.es*

E introducimos la contraseña. Dentro, ejecutamos los mismos comandos:

```
pgradolph@hansolo:~$ ls
Bibliometria
pgradolph@hansolo:~$ mv Bibliometria VersionAntigua
pgradolph@hansolo:~$ ls
VersionAntigua
```

Ya que estamos aquí, y, como vamos a actualizar, vamos a aprovechar para matar el proceso que levanta la web en este momento. Para ello:

- *ss -ltnp | grep 8001*

```
pgradolph@hansolo:~$ ss -ltnp | grep 8001
LISTEN     0      128    127.0.0.1:8001                 *:*                   users:(("gunicorn",pid=6194,fd=5),("gu
nicorn",pid=5815,fd=5))
pgradolph@hansolo:~$
```

Nos fijamos en ese pid y matamos el proceso:

- *kill -9 pid*

```
pgradolph@hansolo:~$ kill -9 5815
pgradolph@hansolo:~$ ss -ltnp | grep 8001
pgradolph@hansolo:~$
```

Una vez hecho esto podemos salir para subir nuestro nuevo archivo zip. Para ello ejecutamos exit dos veces, con el primero salimos a Halowan y con el segundo volvemos a nuestra terminal. Cuando volvemos a nuestra terminal ejecutamos, desde la carpeta donde hemos creado el zip:

- *scp -rp Bibliometria.zip nombre_usuario@halowan.ipb.csic.es:*

Este proceso tarda un rato, cuando termina vemos lo siguiente, y tenemos que acceder de nuevo a halowan con nuestro usuario:

```
PS C:\Users\Pablo\OneDrive\Documentos\Estudios\DATCOM\Trabajo Fin de Máster\App> scp -rp Bibliometria.zip pgradolph@halo
wan.ipb.csic.es:
pgradolph@halowan.ipb.csic.es's password:
Bibliometria.zip                                                              100% 1023MB   1.0MB/s   16:47
PS C:\Users\Pablo\OneDrive\Documentos\Estudios\DATCOM\Trabajo Fin de Máster\App> ssh halowan.ipb.csic.es -l pgradolph
pgradolph@halowan.ipb.csic.es's password:
Last login: Thu Sep 11 08:46:16 2025 from 212-73-34-48.red-acceso.airtel.net
[pgradolph@nodo00 ~]$ ls
App  Bibliometria.zip  VersionAntigua
[pgradolph@nodo00 ~]$
```

Ahora tenemos que hacer lo mismo y mandar el archivo .zip a Hansolo. Para ello:

- *scp -rp Bibliometria.zip nombre_usuario@hansolo.ipb.csic.es:*

Cuando acabe, accedemos de nuevo a hansolo:

```
[pgradolph@nodo00 ~]$ scp -rp Bibliometria.zip pgradolph@hansolo.ipb.csic.es:
pgradolph@hansolo.ipb.csic.es's password:
Bibliometria.zip                                              100% 1023MB 104.3MB/s   00:09
[pgradolph@nodo00 ~]$ ssh pgradolph@hansolo.ipb.csic.es
pgradolph@hansolo.ipb.csic.es's password:
Welcome to Ubuntu 16.04.7 LTS (GNU/Linux 4.4.0-210-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro
Last login: Thu Sep 11 08:52:41 2025 from 161.111.250.88
pgradolph@hansolo:~$
```

Aquí, tenemos que descomprimir el archivo zip en la carpeta Bibliometria con:

- *unzip Bibliometria.zip -d Bibliometria*

Cuando termina:

```
pgradolph@hansolo:~$ ls
Bibliometria  Bibliometria.zip  VersionAntigua
pgradolph@hansolo:~$
```

Ahora vamos a mover el entorno virtual que había en la VersionAntigua a Bibliometria. Para ello:

- *mv VersionAntigua/venv Bibliometria/*

```
pgradolph@hansolo:~$ mv VersionAntigua/venv Bibliometria/
pgradolph@hansolo:~$ cd Bibliometria
pgradolph@hansolo:~/Bibliometria$ ls
accounts  bibliodata     core  db.sqlite3  manage.py  static       templates
analysis  Bibliometrics  data  frontend    media      staticfiles  venv
pgradolph@hansolo:~/Bibliometria$
```

Ahora, antes de lanzar el comando que levanta la web, debemos editar el settings.py. Para ello:

- *cd Bibliometrics*
- *vim settings.py*

Hay que editar el archivo para que quede así:

```
# Configuración para subruta /BiblioMetrics
FORCE_SCRIPT_NAME = '/BiblioMetrics'
STATIC_URL = '/BiblioMetrics/static/'
# STATIC_ROOT = BASE_DIR / 'staticfiles'

# STATICFILES_DIRS = [
#     BASE_DIR / 'static',
# ]
# MEDIA_URL = '/BiblioMetrics/media/'
# MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

# EN PRODUCCIÓN CAMBIAR ESTO:
STATIC_ROOT = '/home/pgradolph/Bibliometria/static'
STATICFILES_DIRS = [
    "/home/pgradolph/Bibliometria/Bibliometrics/staticfiles",  # si tienes una carpeta de estáticos dentro del proyecto
]
MEDIA_ROOT = '/home/pgradolph/Bibliometria/media'
```

En comentarios está especificado lo que tienes que cambiar. Una vez hecho eso, podemos lanzar el servidor y probar el funcionamiento. Para ello ejecutamos, desde dentro de la carpeta Bibliometria:

- *source venv/bin/actívate*
- *gunicorn Bibliometrics.wsgi:application --bind 127.0.0.1:8001*

```
pgradolph@hansolo:~/Bibliometria$ source venv/bin/activate
(venv) pgradolph@hansolo:~/Bibliometria$ gunicorn Bibliometrics.wsgi:application --bind 127.0.0.1:8001
[2025-09-11 09:50:06 +0200] [20020] [INFO] Starting gunicorn 23.0.0
[2025-09-11 09:50:06 +0200] [20020] [INFO] Listening at: http://127.0.0.1:8001 (20020)
[2025-09-11 09:50:06 +0200] [20020] [INFO] Using worker: sync
[2025-09-11 09:50:06 +0200] [20021] [INFO] Booting worker with pid: 20021
```

Esto levanta la página web y podemos acceder a http://bioinfo.ipb.csic.es/BiblioMetrics/es/ para probar el correcto funcionamiento de la misma:

6. SOPORTE Y CONTACTO

Para cualquier consulta relacionada con el uso de la aplicación Bibliometría IPBLN, el primer punto de contacto es el desarrollador del sistema. Todas las dudas, comentarios o sugerencias de mejora deben dirigirse al siguiente correo electrónico:

- Correo de contacto principal: *pablo.gradolph@gmail.com*

Se recomienda emplear este canal tanto para cuestiones de uso general de la herramienta como para notificar posibles errores de funcionamiento, también para cualquier duda relacionada con el proceso de actualización de la herramienta.

En caso de que la consulta no pueda ser atendida en un plazo razonable o si se trata de una incidencia de especial importancia que afecte al acceso o al funcionamiento general de la aplicación, el usuario podrá ponerse en contacto con el equipo de bioinformática del IPBLN-CSIC, que también ofrece soporte técnico sobre la aplicación y puede dar continuidad al mantenimiento.