# Análisis de librerías Python en repositorios GitHub con minería de patrones

Pedro P. García-Pozo y Aurora Ramírez

Universidad de Córdoba, Campus de Rabanales, 14071 Córdoba
{i82gapop, aramirez}@uco.es

**Resumen** Python se ha erigido como uno de los lenguajes de programación más populares hoy en día, en parte gracias a la cantidad de recursos a disposición de los programadores. Incluir desarrollos de terceros, en forma de librerías, en programas Python es una práctica muy habitual pues aligera el esfuerzo de codificación y permite una alta reutilización. En este trabajo se presenta una visión del ecosistema de librerías Python y sus singularidades. Mediante herramientas y técnicas propias de la minería de repositorios software, se ha extraído y analizado un conjunto de más 3.000 librerías Python utilizadas por más de 800 repositorios públicos de GitHub. Además de mostrar estadísticas sobre su uso, hemos aplicado minería de patrones para descubrir relaciones aparentemente ocultas entre librerías. Nuestro estudio revela que aunque el ecosistema de librerías es vasto, los repositorios tienden a utilizar un número reducido de librerías (15 o menos). No obstante, existen librerías que destacan por su amplio uso, como son `requests` para desarrollos web y `numpy` para proyectos de carácter más general o científico.

**Palabras clave:** Minería de repositorios software · librerías software · analítica software · minería de patrones · Python · GitHub

## 1. Introducción

El desarrollo de software ha experimentado en las últimas décadas una revolución en lo que a metodologías y herramientas de soporte se refiere. Conforme la programación se ha ido profesionalizando, es cada vez más habitual reutilizar desarrollos de terceros, integrándolos mediante sus correspondientes APIs [6]. En este aspecto ha influido notablemente la aparición de herramientas que facilitan la importación y gestión de dependencias entre proyectos. La disponibilidad de librerías externas es sin duda una de las características de la programación en Python. Cabe destacar que PyPi, uno de los gestores de paquetes para Python, aloja actualmente más de 370.000 proyectos[1]. Un estudio de 2018 muestra que el número de nuevos paquetes añadidos crece a un ritmo exponencial, y que la mayoría de paquetes se alojan en GitHub [15].

En un ecosistema tan vasto, conocer qué librerías están disponibles y cómo podemos hacer un uso eficaz de ellas se convierte en una ardua tarea. Un

[1] PyPi: `https://pypi.org/` (Acceso: 05/05/2022)

desarrollador posiblemente se fijaría primero en aquellas librerías más populares, pero también debe tener en cuenta cuáles son las que mejor se adaptan a sus necesidades y cómo pueden combinarse entre sí. Es en escenarios como este donde la minería de repositorios software (MSR, *Mining Software Repositories*) puede resultar de utilidad. Desde su aparición en 2004, el análisis de repositorios software mediante técnicas estadísticas y de aprendizaje automático ha experimentado un creciente interés [16]. MSR permite analizar la evolución de los sistemas, predecir posibles fallos y estudiar la interacción entre desarrolladores, entre otros [5]. Para ello extrae conocimiento no solo del código, sino también de los metadatos del repositorio donde se aloja y de mensajes entre desarrolladores.

En este trabajo realizamos un estudio del ecosistema de librerías Python, analizando su uso en repositorios públicos alojados en GitHub. Nos centramos en el lenguaje Python por su relevancia hoy en día, pues ha pasado a ser el primer lenguaje de programación de acuerdo al informe TIOBE de abril de 2022[2], y es también uno de los lenguajes que experimenta más actividad en GitHub[3]. Con este estudio pretendemos conocer cuáles son las librerías más destacadas en cuanto a su uso, y establecer relaciones entre ellas en base a los repositorios que las utilizan. Para ello proponemos el uso de la minería de patrones [3], pues nos va a permitir descubrir tendencias a priori ocultas bajo una gran cantidad de datos. Del análisis de más de 800 repositorios y más de 3.000 librerías, podemos destacar que la mayoría de los repositorios utiliza menos de 15 librerías, si bien existen librerías como `requests` y `numpy` cuyo uso es muy superior al resto. Además, el análisis de patrones revela que estas dos librerías aparecen con cierta frecuencia en combinación con otras, lo cual podría ser de gran utilidad para construir sistemas de recomendación inteligentes.

El resto del artículo se estructura como sigue. La Sección 2 explica los fundamentos de la minería de patrones. La Sección 3 resume el trabajo relacionado. A continuación, la Sección 4 detalla la metodología experimental para la extracción y análisis de datos. El estudio de las librerías se presenta en la Sección 5, seguido de una discusión de retos abiertos en la Sección 6. Las conclusiones del trabajo y las líneas de investigación futuras se presentan en la Sección 7.

## 2. Fundamentos de la minería de patrones

Dentro de la minería de datos, la minería de patrones busca descubrir la co-ocurrencia de ciertos elementos (denominados ítems) en un conjunto de transacciones [3]. Este tipo de análisis es muy útil para estudiar tendencias y hacer recomendaciones. Habitualmente, la minería de patrones se centra en encontrar conjuntos de ítems (patrones o *itemsets*) frecuentes, esto es, que aparecen en las transacciones por encima de un umbral mínimo. A dicho umbral se le denomina soporte mínimo, el cual puede expresarse de forma absoluta (número de transacciones) o relativo (porcentaje de transacciones respecto al total). La longitud o

[2] Índice TIOBE: `https://www.tiobe.com/tiobe-index/` (Acceso: 05/05/2022)

[3] Estadísticas en GitHub: `https://madnight.github.io/githut/#/pull_requests/2021/4` (Acceso: 05/05/2022)

tamaño del patrón es el número de ítems que lo compone. Un conjunto especial de patrones son los patrones cerrados (*closed*). Representan el subconjunto de los patrones frecuentes para los cuales no es posible encontrar un superconjunto con el mismo soporte. Obtener este subconjunto supone una reducción respecto al número de patrones a manejar, sin implicar por ello una pérdida de información ya que los patrones frecuentes se pueden recuperar a partir de ellos [11].

Tabla 1: Conjunto de transacciones de ejemplo

| **Repositorio** | **Librerías** |
|---|---|
| $r_1$ | $l_1$, $l_2$, $l_3$ |
| $r_2$ | $l_1$, $l_2$, $l_3$, $l_4$ |
| $r_3$ | $l_2$, $l_3$ |
| $r_4$ | $l_1$, $l_3$, $l_4$ |
| $r_5$ | $l_4$, $l_5$ |

Para ilustrar estos conceptos, la Tabla 1 muestra un conjunto de repositorios (transacciones) y, para cada uno de ellos, las librerías (ítems) que utiliza. Con un soporte mínimo igual a 2, los patrones frecuentes y cerrados son los siguientes:

- Patrones frecuentes: $l_1$ (3 repositorios), $l_2$ (3), $l_3$ (4), $l_4$ (3), $\{l_1, l_2\}$ (2), $\{l_1, l_3\}$ (3), $\{l_1, l_4\}$ (2), $\{l_2, l_3\}$ (3), $\{l_3, l_4\}$ (2), $\{l_1, l_2, l_3\}$ (2), $\{l_1, l_3, l_4\}$ (2)
- Patrones frecuentes cerrados: $l_3$ (4), $l_4$ (3), $\{l_1, l_3\}$ (3), $\{l_2, l_3\}$ (3), $\{l_1, l_2, l_3\}$ (2), $\{l_1, l_3, l_4\}$ (2)

Todas las librerías, excepto $l_5$, aparecen en dos o más repositorios, por lo que se añaden al conjunto de patrones frecuentes de longitud 1. A partir de ellas, podemos buscar las combinaciones de 2, 3 y 4 librerías que también son frecuentes. En el caso de los patrones cerrados, vemos que $l_1$ y $l_2$ no lo son, pues existen patrones de mayor longitud (en concreto, $\{l_1, l_3\}$ y $\{l_2, l_3\}$) que las contienen y cumplen el criterio de tener el mismo soporte (3).

Existe una amplia colección de algoritmos para la extracción de patrones, tanto frecuentes como cerrados [3]. Uno de los aspectos fundamentales de estos algoritmos debe ser su eficiencia, pues el número de *itemsets* aumenta considerablemente cuando se tienen conjuntos de datos con alta dimensionalidad. Entre los algoritmos para extraer patrones cerrados podemos destacar variantes de algoritmos para la extracción de patrones frecuentes, como Apriori-Close y FPClose (basado en FP-Growth), o propuestas específicas como CHARM y DCI_Closed. Este último destaca por su eficiencia [8]. Siguiendo un proceso recursivo, extiende cada posible *itemset* generador de patrones cerrados mediante una búsqueda en profundidad, descartando *itemsets* que no garantizan la propiedad de cierre en las sucesivas llamadas. El único parámetro que necesita es el umbral de soporte mínimo, al que denominaremos *min_supp*.

## 3. Trabajo relacionado

La minería de repositorios software permite la extracción de conocimiento a partir de artefactos software y metadatos alojados en plataformas colabora-

tivas vinculadas al desarrollo de software, como GitHub o Stack Overflow. Los estudios en MSR arrojan valiosa información para comprender mejor cuáles son las prácticas de desarrollo habituales en ciertas comunidades de desarrolladores, predecir la evolución del software, o apoyar tareas de detección de errores y refactorización de código, entre otros propósitos [2].

Dentro de MSR podemos encontrar estudios centrados en librerías software que analizan cómo suelen combinarse entre ellas [18] o si los desarrolladores las actualizan en sus proyectos a medida que estas evolucionan [13]. No obstante, el principal propósito de los estudios MSR relacionados con librerías software ha consistido en proporcionar recomendaciones sobre qué librerías pueden ser de interés para un proyecto software determinado [14]. Este problema se basa en analizar las similitudes, en cuanto al uso de librerías se refiere, de un conjunto amplio de proyectos software. Este tipo de análisis ha dado lugar a varios sistemas de recomendación especializados en librerías Java [14,12,10]. Este tipo de sistemas basan la recomendación en la mera aparición conjunta de las librerías, pero no tienen la capacidad de analizar en profundidad el ecosistema de librerías para ver dónde y cómo pueden ser integradas en el proyecto para el cual se recomiendan.

Hasta la fecha, no se conocen sistemas de recomendación similares a los anteriores para Python. Sin embargo, este lenguaje es de gran uso en la actualidad y, además, se fundamenta ampliamente en el uso de librerías externas. De hecho, el ecosistema de librerías Python ya era considerado uno de los mayores y con más perspectivas de crecimiento en el año 2016 [9]. Recientemente, ha sido identificado también como el lenguaje de programación predilecto en GitHub para proyectos de inteligencia artificial [4]. A pesar de ello, no es frecuente encontrar estudios en MSR centrados en este lenguaje y, más concretamente, en el uso de librerías. En 2018, un primer trabajo analizó el ecosistema de librerías disponibles en PyPi [15] con el objetivo de estudiar su ciclo de vida y las dependencias entre ellas, pero no su uso por parte de repositorios en GitHub. Un segundo trabajo más reciente propone Py2Src, un método que permite la identificación del repositorio GitHub vinculado a una librería disponible en PyPi [17].

## 4. Metodología

Para describir la metodología experimental seguida en nuestro trabajo, nos fundamentaremos en las directrices para la elaboración de estudios en el área MSR publicadas recientemente [16]. En dicho artículo se propone seguir un proceso sistemático que consta de tres fases: 1) planificación, en la que determina el alcance del estudio y las fuentes de datos; 2) ejecución, donde se establecen los pasos a seguir para seleccionar los repositorios y extraer los datos; y 3) análisis de resultados, donde se plantea cómo será el estudio de la información obtenida. A continuación, se detalla cómo hemos abordado cada una de estas tres fases.

***Planificación.*** El primer paso en la fase de planificación es plantear los objetivos de la investigación, definiendo para ello las preguntas de investigación (RQ, *research question*). En este estudio, nos planteamos las siguientes RQ:

**RQ1:** ¿Qué cantidad de librerías Python se suelen utilizar en repositorios de GitHub?
**RQ2:** ¿Cuáles son las librerías más utilizadas en dichos repositorios?
**RQ3:** ¿Existen librerías que se utilizan frecuentemente de forma combinada?

Para responder a estas preguntas, se realizará un proceso de extracción y análisis de repositorios software alojados en GitHub. Aunque no puede considerarse la única fuente posible, GitHub es la más utilizada para estudios en MSR [7] ya que dispone de un alto número de repositorios[4] y existen herramientas —tanto propias como de terceros— que facilitan la extracción de datos. Además de la fuente de datos, deben establecerse los criterios de selección y filtrado que garanticen que el conjunto de repositorios es representativo. Algunos de los criterios habituales son el lenguaje de programación, número de contribuidores, tamaño, popularidad y actualización de los repositorios, entre otros [16]. En nuestro estudio, nos limitamos a desarrollos en lenguaje Python, ya que la existencia y prácticas de uso de las librerías software es particular para cada lenguaje de programación. Para garantizar que los proyectos utilizan un número alto de librerías, necesitamos asegurar que los repositorios seleccionados son de cierta envergadura. Por ello, se establecieron tres criterios respecto a los repositorios habituales en MSR: número mínimo de *commits* (1.000), número mínimo de estrellas (100) y actualización (último *commit* posterior al 1 de enero de 2021).

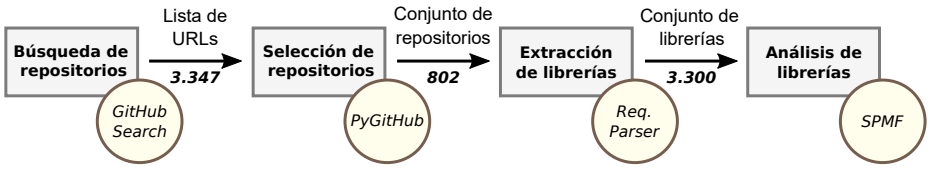


Figura 1: Pasos para realizar el análisis de repositorios y herramientas empleadas.

***Ejecución.*** La Figura 1 resume los pasos seguidos y las herramientas utilizadas para la búsqueda de repositorios, la extracción de datos y el análisis de resultados. A continuación nos centramos en los tres primeros pasos, los que conciernen a la preparación del conjunto de datos. Para la búsqueda de repositorios hemos utilizado GitHub Search[5] [1], que explora los metadatos de los repositorios públicos de GitHub en base a un amplio catálogo de filtros (incluido el número de *commits*, no disponible en otros buscadores). Tras esta búsqueda[6], se obtuvo un listado con las URLs de 3.347 repositorios que cumplían los requisitos.

A continuación, se ha procedido a la descarga del contenido de los repositorios utilizando para ello la herramienta PyGitHub[7]. Con el objetivo de localizar las

[4] Estadísticas sobre GitHub: `https://octoverse.github.com/` (Acceso: 05/05/2022)
[5] GitHub Search: `https://seart-ghs.si.usi.ch/` (Acceso: 05/05/2022)
[6] La búsqueda se realizó en la fecha 02/03/2022.
[7] PyGitHub: `https://pygithub.readthedocs.io/` (Acceso: 05/05/2022)

librerías Python utilizadas en los repositorios, nos hemos centrado en los repositorios que contienen el fichero `requirements.txt`. De forma similar al fichero `pom.xml` de Maven, el fichero `requirements.txt` enumera las librerías (y sus versiones) de las que depende un proyecto Python, facilitando su identificación y extracción. No obstante, este fichero debe ser creado por los contribuidores al repositorio explícitamente con el comando `pip`. De los 3.347 repositorios iniciales, 802 contenían un fichero `requirements.txt` válido. Utilizando la herramienta Requirements Parser[8], se ha obtenido la lista de librerías asociada a cada repositorio. En total, los 802 repositorios utilizan 3.300 librerías diferentes.

***Análisis de resultados.*** Para el análisis de resultados nos hemos guiado por las RQ planteadas, eligiendo las técnicas y métricas más adecuadas para darles respuesta. Respecto a RQ1 y RQ2, se han utilizado técnicas de visualización y estadísticas descriptivas para analizar las características del conjunto de datos resultante del proceso de extracción.

El estudio de librerías frecuentes (RQ3) se ha abordado como un problema de minería de patrones, donde el objetivo es encontrar combinaciones de librerías (*itemsets*) que aparecen con frecuencia en los repositorios (transacciones). Para ello se ha utilizando el algoritmo DCI_Closed [8], implementado en la librería SPMF[9]. El soporte mínimo se ha configurado a los valores siguientes: $min\_supp = \{8, 20, 40, 80\}$. Estos valores coinciden, aproximadamente con soportes relativos del 1 %, 2.5 %, 5 % y 10 % respecto al número de repositorios. Utilizar un soporte más alto limitaría mucho la posibilidad de encontrar combinaciones frecuentes, pues tenemos un número elevado de ítems (librerías) con respecto a transacciones (repositorios), y baja frecuencia de aparición.

## 5. Resultados

A continuación se presentan los resultados, divididos entre las estadísticas de uso de librerías (Sección 5.1) y el análisis de patrones frecuentes (Sección 5.2).

### 5.1. Análisis del uso de librerías en repositorios

Realizamos aquí un primer análisis de carácter cuantitativo sobre la relación entre repositorios y librerías. Como se mencionó anteriormente, se han identificado 3.300 librerías diferentes en los 802 repositorios considerados. El número de librerías por repositorio varía ente un mínimo de 1 y un máximo de 232. La mediana es 8, mientras que la media es 15.5 con una desviación estándar igual a 24.6. Estos valores nos indican que la mayoría de repositorios depende de un número bajo de librerías, mientras que solo unos pocos repositorios utilizan un número muy alto de ellas en comparación con el resto. La Figura 2 lo ilustra claramente, pues se puede observar que lo más habitual es que los repositorios

[8] Requirements Parser: `https://requirements-parser.readthedocs.io/` (Acceso: 05/05/2022)
[9] SPMF: `https://www.philippe-fournier-viger.com/spmf` (Acceso: 05/05/2022)

utilicen entre 2 y 5 librerías. Tan solo 18 repositorios (2 % del total) requieren más de 100. En concreto, los 5 proyectos que más librerías utilizan (160 o más) son: *mozilla/telemetry-airflow*[10], para la planificación y monitorización de flujos de trabajos con Apache Airflow; *certbot/certbot*[11], un cliente web para la gestión de certificados; *dimagi/commcare-hq*[12], un servidor para el diseño, gestión y despliegue de aplicaciones móviles; *epiphany-platform/epiphany*[13], una solución para la automatización de sistemas basados en Docker y Kubernetes; y *macroconnections/dive-backend*[14], una plataforma para la visualización interactiva de datos estructurados.

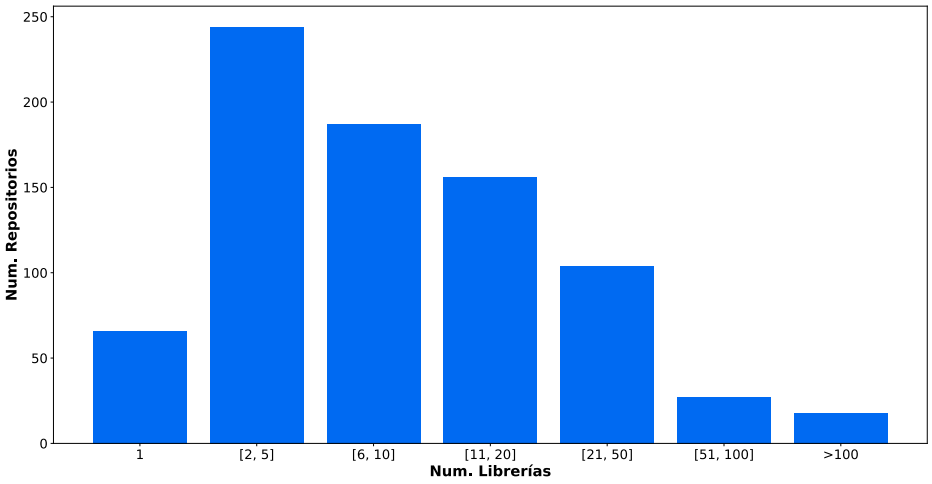


Figura 2: Distribución de repositorios según el número de librerías que utilizan.

Para profundizar en el uso de las librerías, la Figura 3 clasifica las 3.300 librerías según el rango de repositorios en los que se utilizan. Vemos que más de 2.000 librerías aparecen en un único repositorio, lo que nos hace pensar que se trata de librerías de propósito muy específico. Inspeccionándolas en más detalle, algunas de ellas son desarrollos de los propios contribuidores al repositorio. Un número considerable de librerías (848) aparece en entre 2 y 5 repositorios, cifra que desciende a 173 si buscamos librerías utilizadas en entre 6 y 10 repositorios. Solo 32 de las 3.300 librerías (0.97 % del total) aparece en 50 o más repositorios, aunque cabe destacar que 10 de ellas (`requests`, `numpy`, `sphinx`, `six`, `pyyaml`, `scipy`, `python-dateutil`, `jinja2`, `matplotlib` y `pytz`) lo hacen en 100 reposi-

[10] `https://github.com/mozilla/telemetry-airflow` (Acceso: 05/05/2022)
[11] `https://github.com/certbot/certbot` (Acceso: 05/05/2022)
[12] `https://github.com/dimagi/commcare-hq` (Acceso: 05/05/2022)
[13] `https://github.com/epiphany-platform/epiphany` (Acceso: 05/05/2022)
[14] `https://github.com/CenterForCollectiveLearning/DIVE-backend` (Acceso: 05/05/2022)

torios o más. Podemos concluir por tanto que, aunque el número de librerías es elevado respecto al número de repositorios, no existe un amplio uso de la mayoría de ellas.

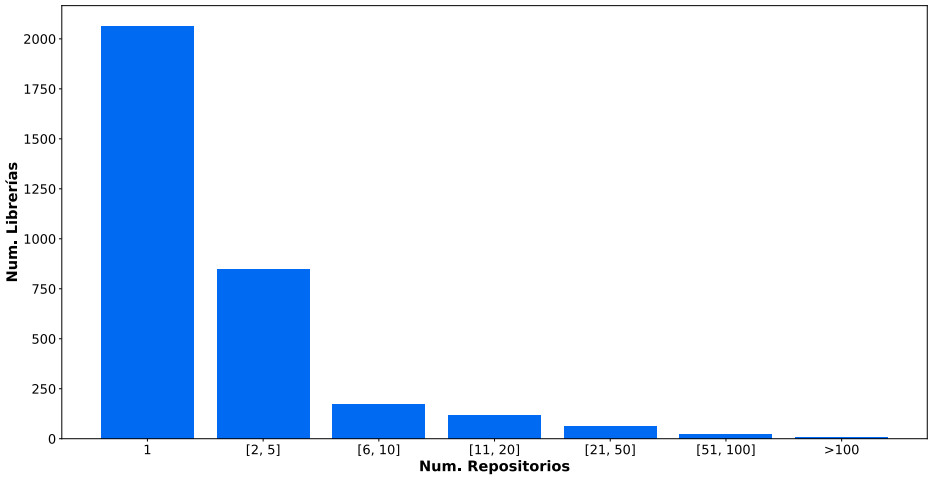


Figura 3: Distribución de las 3.300 librerías según el número de repositorios en las que aparecen.

La Figura 4 enumera las 25 librerías más utilizadas en los repositorios analizados, ordenadas por número de ocurrencias. Todas ellas aparecen en más de 50 repositorios, aunque destacan especialmente `requests` y `numpy` que superan los 200 (244 y 209, respectivamente). La Tabla 2 resume las diferentes categorías en las que se enmarcan estas librerías según su propósito, para lo cual nos hemos basado en su documentación y en la categorización que hace PyPi de ellas. Las categorías se presentan ordenadas alfabéticamente, mientras que las librerías se enumeran en orden decreciente de ocurrencias. Las categorías a las que se asocian más librerías son utilidades (6 librerías), documentación y soporte al desarrollo (5), procesamiento de datos (5) y desarrollo web (4). La aparición de librerías dedicadas a facilitar el desarrollo web puede parecer sorprendente a priori, pues no es un lenguaje tradicionalmente asociado a este tipo de desarrollo. No obstante, lo cierto es que Python está ganando terreno a otros lenguajes como PHP y Java para el desarrollo de aplicaciones web[15]. También es reseñable la aparición de la librería `six`, destinada a proporcionar compatibilidad entre las versiones 2 y 3 de Python. Esto nos indica que muchas aplicaciones garantizan la retrocompatibilidad conforme el lenguaje evoluciona.

[15] Estadísticas sobre lenguajes en GitHub: `https://www.analyticsinsight.net/top-10-most-popular-programming-languages-on-github-for-developers/` (Acceso: 05/05/2022).

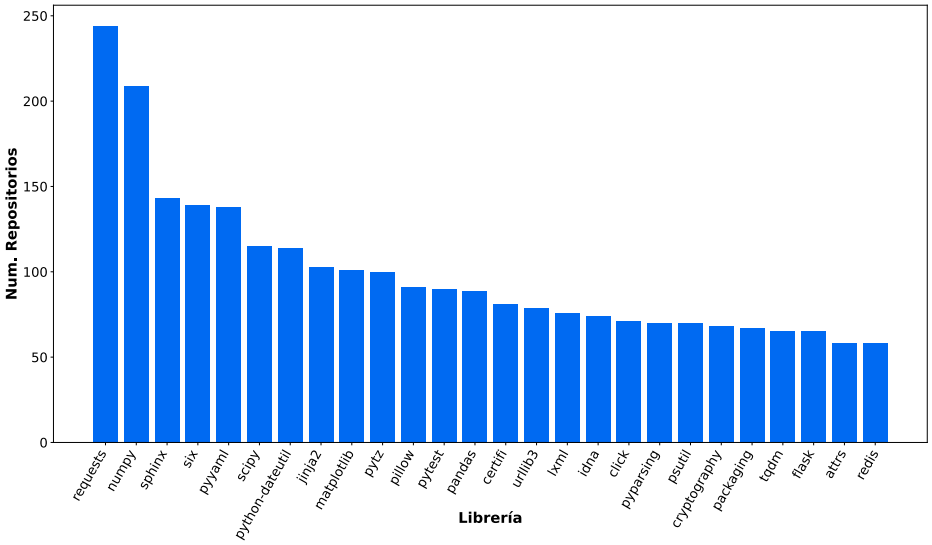


Figura 4: Las 25 librerías Python más utilizadas en los repositorios de GitHub analizados. Las librerías `attrs` y `redis` tienen igual número de ocurrencias.

En base al análisis realizado en esta sección, podemos responder las dos primeras RQ planteadas:

- **RQ1 -** A pesar de haber encontrado más de 3.000 librerías en uso por parte de 802 repositorios, el 75 % de los repositorios solo utiliza entre 1 y 15 librerías.
- **RQ2 -** Existen 10 librerías Python que aparecen en 100 o más repositorios, siendo `requests` y `numpy` las más utilizadas (en más de 200 repositorios).

### 5.2. Estudio de librerías frecuentes

Tras analizar las características más generales del conjunto de repositorios y librerías, esta sección se centra en el descubrimiento de relaciones entre librerías

Tabla 2: Categorización de las librerías más utilizadas según su propósito.

| **Categoría** | **Librerías** |
|---|---|
| Cómputo científico | `numpy`, `scipy` |
| Desarrollo web | `requests`, `jinja2`, `urllib3`, `flask` |
| Documentación y soporte al desarrollo | `sphinx`, `six`, `pytest`, `packaging`, `attrs` |
| Gráficos | `matplotlib`, `pillow` |
| Procesamiento de datos | `pyyaml`, `pytz`, `pandas`, `lxml`, `redis` |
| Seguridad y encriptación | `certifi`, `cryptography` |
| Utilidades | `python-dateutil`, `idna`, `click`, `pyparsing`, `psutil`, `tqdm` |

mediante minería de patrones. Cabe mencionar que para este análisis se han excluido las 2.063 librerías que solo aparecen en un repositorio, pues no van a contribuir a la generación de combinaciones de librerías frecuentes. A su vez, esto reduce el número de repositorios hasta los 787, pues encontramos 15 repositorios cuya única librería solo aparecía en dicho repositorio.

Como se indicó en la Sección 4, la extracción de patrones se ha realizado con varios umbrales de soporte. La Tabla 3 recoge el número de patrones encontrados en cada caso, desglosado por la longitud del *itemset* (entre 1 y 22). Esto es, la primera fila indica cuántas librerías superan el umbral mostrado en la columna correspondiente. La segunda fila indica cuántas parejas de librerías superan el umbral de soporte, y así sucesivamente hasta alcanzar patrones compuestos por la aparición conjunta de hasta 22 librerías. En la última fila se recoge el tiempo de ejecución del algoritmo DCI_Closed en cada caso. Como es lógico, utilizar un soporte bajo (8) nos permite encontrar un número mayor de patrones donde es más fácil, además, que aparezcan un número más amplio de librerías. A partir de las 235 librerías que se utilizan en 8 o más repositorios (primera columna), el algoritmo encuentra más de 16.000 combinaciones de ellas que también están en 8 o más repositorios. Aunque no podamos considerar dichas combinaciones como "frecuentes" con respecto al total de repositorios analizado (787), es interesante observar que se pueden encontrar algunos repositorios que presentan más de 20 librerías en común.

Conforme se incrementa el soporte mínimo, esto es, aumenta el número de repositorios para considerar al *itemset* como frecuente, disminuye significativamente tanto el número de combinaciones como la longitud de los *itemsets*. Para los valores de soporte intermedios (20 y 40), el patrón de mayor longitud encontrado contiene 11 y 6 librerías, respectivamente. En el caso del patrón de 11 librerías (que aparece en 20 repositorios), encontramos 8 librerías del "top-25" (`requests`, `six`, `python-dateutil`, `jinja2`, `pytz`, `certifi`, `urllib3` e `idna`) junto a otras 3 relativamente menos frecuentes: `markupsafe` (55 ocurrencias), `cffi` (49) y `pycparser` (42). Las dos últimas están relacionadas con la compatibilidad entre Python y C, mientras que la primera está vinculada al desarrollo web como la mayoría de las que conforman este patrón.

Respecto a patrones de menor longitud pero mayor soporte, la Figura 5 recoge algunos ejemplos. Por cuestiones de espacio, se muestran los cuatros patrones más frecuentes para longitudes entre 2 y 6. Podemos observar que predominan las librerías del "top-25" (ver Figura 4), siendo las relacionadas con desarrollo web las que aparecen en combinaciones de mayor longitud. Esto nos sugiere que para el desarrollo de aplicaciones web es habitual utilizar un conjunto de librerías "básicas", compuesto por librerías especializadas, como son `requests`, `jinja2` y `urllib3`, así como otras librerías de carácter más general, como son `idna`, `pytz` y `six`. Por otro lado, también es interesante destacar la aparición conjunta de `numpy`, `scipy` y `matplotlib` en 57 repositorios. Son tres librerías de amplio uso en desarrollos científicos, y además tanto `scipy` como `matplotlib` están construidas sobre la base de `numpy`. Por lo tanto, no es extraño que las dos parejas de librerías con mayor soporte sean las formadas por `numpy` con cada

Tabla 3: Número de combinaciones de librerías frecuentes encontradas.

| **Longitud** | $min\_supp = 8$ | $min\_supp = 20$ | $min\_supp = 40$ | $min\_supp = 80$ |
|---|---|---|---|---|
| 1 | 235 | 96 | 43 | 14 |
| 2 | 1.172 | 318 | 65 | 4 |
| 3 | 1.895 | 420 | 39 | - |
| 4 | 1.914 | 371 | 21 | - |
| 5 | 1.677 | 290 | 4 | - |
| 6 | 1.495 | 197 | 1 | - |
| 7 | 1.376 | 129 | - | - |
| 8 | 1.248 | 75 | - | - |
| 9 | 1.200 | 29 | - | - |
| 10 | 1.107 | 7 | - | - |
| 11 | 992 | 1 | - | - |
| 12 | 755 | - | - | - |
| 13 | 575 | - | - | - |
| 14 | 417 | - | - | - |
| 15 | 245 | - | - | - |
| 16 | 120 | - | - | - |
| 17 | 73 | - | - | - |
| 18 | 33 | - | - | - |
| 19 | 18 | - | - | - |
| 20 | 9 | - | - | - |
| 21 | 5 | - | - | - |
| 22 | 3 | - | - | - |
| **Tiempo (ms)** | 664 | 404 | 281 | 194 |

una de las otras dos por separado: en 111 repositorios con `scipy` y en 85 con `matplotlib`. Es más, considerando que `scipy` y `matplotlib` aparecían en 115 y 101 repositorios, respectivamente, el hecho de que un alto porcentaje de esos repositorios (97 % y 84 %) incorpore también `numpy` es significativo y podría ser útil para la construcción de sistemas de recomendación de librerías.

En base al conocimiento extraído con la minería de patrones, podemos responder a la tercera RQ:

- **RQ3 -** Aunque la densidad de librerías es algo baja, encontramos pares de librerías que aparecen conjuntamente en entre un 10 % y un 14 % de los repositorios analizados (787). En menor medida, se encuentran también combinaciones de librerías de mayor longitud vinculadas a desarrollos web.

## 6. Discusión y retos abiertos

La extracción de datos de GitHub no siempre es tan efectiva y concluyente como podemos pensar, y puede condicionar el tipo de análisis a realizar y las conclusiones que se pueden extraer [7]. En este sentido, nuestro proceso de extracción de los datos ha arrojado ciertos aspectos que es importante tener en cuenta de cara a estudios futuros. A pesar de disponer de un número alto de

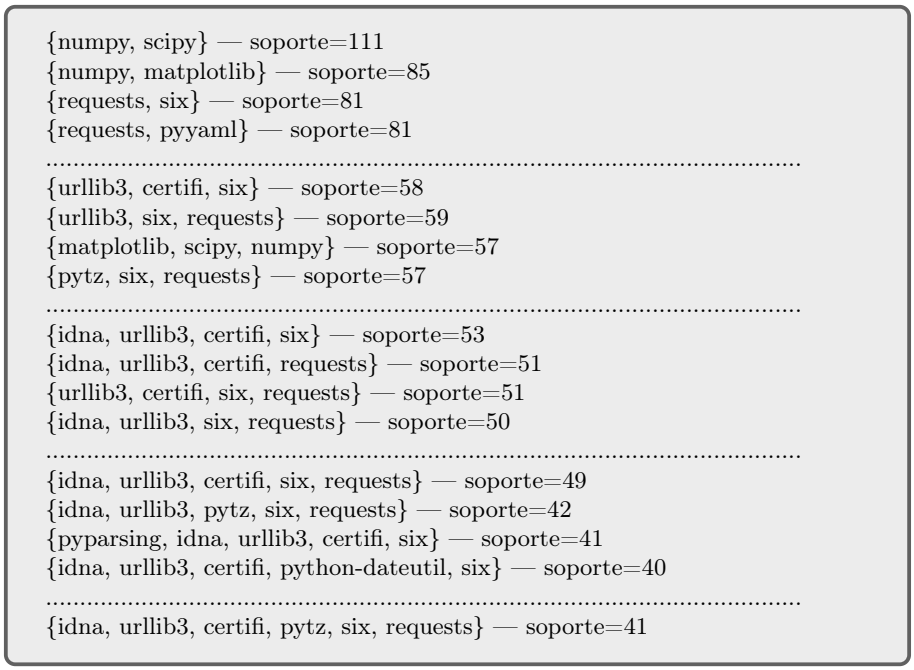


Figura 5: Patrones frecuentes encontrados en el ecosistema de librerías Python.

repositorios que cumplían los criterios de selección impuestos, la presencia del fichero `requirements.txt` ha sido menor de lo esperada. Este hecho nos indica que generar este fichero no es una práctica común, a pesar de que ayuda a hacer explícitas las dependencias del código, lo cual es deseable en proyectos de envergadura. Además, en los casos en los que el fichero estaba presente, no siempre tenía un formato válido, o el repositorio contenía más de un fichero, lo que no permite vincular claramente el código —pues el repositorio parece contener varios subproyectos— y las librerías. De cara a construir sistemas de recomendación, puede ser necesario un rastreo más preciso de las librerías a nivel del código fuente, así como estudiar el historial de *commits* para garantizar además que el fichero está actualizado respecto al código.

El análisis aquí presentado constituye un primer paso para tratar de comprender el complejo ecosistema de librerías Python y las prácticas que rodean a su uso en GitHub. La información extraída y las tendencias encontradas ayudarían al desarrollo de sistemas de recomendación similares a los propuestos para otros lenguajes como Java [14]. Sin embargo, consideramos que dichos sistemas aún podrían mejorarse enormemente para ser más precisos en sus recomendaciones si se analizan otros aspectos de las librerías, como puede ser su evolución. Un claro ejemplo sería la detección de incompatibilidades entre versiones de librerías, o evitar recomendar librerías cuya tendencia de uso está decayendo en

favor de otras. De manera similar, la propia evolución de los repositorios donde se utilizan las librerías también puede arrojar información valiosa. En base al historial de *commits* se podría estudiar por qué ciertas librerías comenzaron o dejaron de utilizarse, o si se sustituyeron por otras en algún momento del desarrollo del proyecto. La actividad de los programadores también puede tener influencia en el uso de librerías, pudiendo estudiar si los programadores tienden a integrar librerías que ya conocen en nuevos proyectos, incluso si no son las más recientes o más apropiadas en el nuevo contexto. Además de mejorar la precisión de las recomendaciones teniendo en cuenta todos estos factores, el análisis del propio proyecto objetivo de la recomendación también puede ayudar a determinar dónde deben utilizarse esas librerías, estableciendo similitudes en cuanto a la funcionalidad del código y la que ofrece la librería. También sería interesante analizar en qué situaciones es preferible incorporar una única librería o combinar varias de ellas para lograr un mismo propósito, atendiendo a razones de eficiencia, seguridad, compatibilidad del código o facilidad de uso.

## 7. Conclusiones

La integración de librerías desarrolladas por terceros es una práctica muy habitual cuando se programa en Python. El amplio número de librerías disponibles en gestores de paquetes como PyPi abre un gran abanico de posibilidades no solo en cuanto a qué librerías instalar, sino a cómo es mejor combinarlas. Nuestro estudio, focalizado en más de 800 repositorios públicos de GitHub, supone un primer paso para comprender el ecosistema de librerías Python. Con este estudio hemos detectado no solo las librerías más empleadas, sino también algunas combinaciones que aparecen con relativa frecuencia, utilizando para ello conceptos de la minería de patrones.

Con el conocimiento extraído, nos podremos encaminar hacia la propuesta de mejores sistemas de recomendación de librerías. En este sentido, el trabajo futuro se centrará en incorporar información temporal al conjunto de datos a partir del historial de *commits*, y estudiar la complejidad estructural de los proyectos que las utilizan. También queremos abordar la construcción de sistemas inteligentes de recomendación que no solo sugieran con precisión qué librerías pueden ser de interés para el proyecto, sino dónde y cómo integrarlas.

**Agradecimientos** Trabajo parcialmente financiado por el Ministerio de Ciencia e Innovación (proyectos PID2020-115832GB-I00, RED2018-102472-T), y la Junta de Andalucía (contrato postdoctoral DOC_00944).

## Referencias

1. Dabic, O., Aghajani, E., Bavota, G.: Sampling Projects in GitHub for MSR Studies. In: 18th Int. Conf. Mining Software Repositories. pp. 560–564 (2021)

2. de F. Farias, M.A., Novais, R., Júnior, M.C., da Silva Carvalho, L.P., Mendonça, M., Spínola, R.O.: A Systematic Mapping Study on Mining Software Repositories. In: 31st Symposium on Applied Computing. p. 1472–1479 (2016)
3. Fournier-Viger, P., Lin, J.C.W., Vo, B., Chi, T.T., Zhang, J., Le, H.B.: A survey of itemset mining. WIREs Data Mining and Knowledge Discovery **7**(4), e1207 (2017)
4. Gonzalez, D., Zimmermann, T., Nagappan, N.: The State of the ML-Universe: 10 Years of Artificial Intelligence and Machine Learning Software Development on GitHub. In: 17th Int. Conf. Mining Software Repositories. p. 431–442 (2020)
5. Güemes-Peña, D., López-Nozal, C., Marticorena-Sánchez, R., Maudes-Raedo, J.: Emerging topics in mining software repositories. Progress in Artificial Intelligence **7**(3), 237–247 (2018)
6. Huang, K., Chen, B., Xu, C., Wang, Y., Shi, B., Peng, X., Wu, Y., Liu, Y.: Characterizing usages, updates and risks of third-party libraries in java projects. Empirical Software Engineering **27**(4) (2022)
7. Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M., Damian, D.: The Promises and Perils of Mining GitHub. In: 11th Working Conference on Mining Software Repositories (MSR). p. 92–101 (2014)
8. Lucchese, C., Orlando, S., Perego, R.: DCI_Closed: A Fast and Memory Efficient Algorithm to Mine Frequent Closed Itemsets. In: ICDM Workshop on Frequent Itemset Mining Implementations. pp. 1–9 (2004)
9. Ma, W., Chen, L., Zhou, Y., Xu, B.: What Are the Dominant Projects in the GitHub Python Ecosystem? In: 3rd Int. Conf. Trustworthy Systems and their Applications. pp. 87–95 (2016)
10. Nguyen, P.T., Di Rocco, J., Di Ruscio, D., Di Penta, M.: CrossRec: Supporting software developers by recommending third-party libraries. Journal of Systems and Software **161**, 110460 (2020)
11. Rodríguez-González, A.Y., Lezama, F., Iglesias-Alvarez, C.A., Martínez-Trinidad, J.F., Carrasco-Ochoa, J.A., de Cote, E.M.: Closed frequent similar pattern mining: Reducing the number of frequent similar patterns without information loss. Expert Systems with Applications **96**, 271–283 (2018)
12. Saied, M.A., Ouni, A., Sahraoui, H., Kula, R.G., Inoue, K., Lo, D.: Improving reusability of software libraries through usage pattern mining. Journal of Systems and Software **145**, 164–179 (2018)
13. Salza, P., Palomba, F., Di Nucci, D., De Lucia, A., Ferrucci, F.: Third-party libraries in mobile apps: When, how, and why developers update them. Empirical Software Engineering **25**(3), 2341–2377 (2020)
14. Thung, F., Lo, D., Lawall, J.: Automated library recommendation. In: 20th Working Conf. Reverse Engineering. pp. 182–191 (2013)
15. Valiev, M., Vasilescu, B., Herbsleb, J.: Ecosystem-Level Determinants of Sustained Activity in Open-Source Projects: A Case Study of the PyPI Ecosystem. In: 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. p. 644–655 (2018)
16. Vidoni, M.: A systematic process for mining software repositories: Results from a systematic literature review. Information and Software Technology **144**, 106791 (2022)
17. Vu, D.L.: py2src: Towards the Automatic (and Reliable) Identification of Sources for PyPI Package. In: 36th Int. Conf. Automated Software Engineering. pp. 1394–1396 (2021)
18. Yano, Y., Kula, R.G., Ishio, T., Inoue, K.: VerXCombo: An Interactive Data Visualization of Popular Library Version Combinations. In: 23rd Int. Conf. Program Comprehension. pp. 291–294 (2015)