



DEPARTAMENTO  
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

# Trabajo Práctico Final

## Ciencia de Redes Humanas y Sociales

---

10 de Diciembre de 2025

Integrante	LU	Correo electrónico
Martina Rosario Pérez	79/23	<a href="mailto:martinarosarioperezgonza@gmail.com">martinarosarioperezgonza@gmail.com</a>
Juan Ignacio Catania	840/22	<a href="mailto:juancatania.fceyn@gmail.com">juancatania.fceyn@gmail.com</a>
Mateo Guerrero Schmidt	688/22	<a href="mailto:mateogs01@gmail.com">mateogs01@gmail.com</a>
Sofia Gutierrez	410/22	<a href="mailto:sogutierrez02@gmail.com">sogutierrez02@gmail.com</a>



**Facultad de Ciencias Exactas y Naturales**

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón Cero + Infinito)

Intendente Güiraldes 2610 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Conmutador: (+54 11) 5285-9721 / 5285-7400

<https://dc.uba.ar>

# Índice

<b>1. Introducción al problema</b>	<b>2</b>
<b>2. Obtención de datos</b>	<b>2</b>
2.1. Origen de los datos . . . . .	2
2.2. Limpieza de los datos . . . . .	2
<b>3. Análisis exploratorio</b>	<b>3</b>
3.1. Dataset 2016 . . . . .	3
3.1.1. Generalidades del grafo . . . . .	3
3.1.2. Conectividad . . . . .	3
3.1.3. Análisis de centralidad . . . . .	4
3.1.4. Subgrafos . . . . .	5
3.1.5. Comunidades . . . . .	6
3.1.6. Wordcloud . . . . .	7
3.2. Dataset 2025 . . . . .	7
3.2.1. Generalidades del grafo . . . . .	7
3.2.2. Conectividad . . . . .	7
3.2.3. Análisis de centralidad . . . . .	8
3.2.4. Subgrafos . . . . .	9
3.2.5. Comunidades . . . . .	9
3.2.6. Wordcloud . . . . .	10
<b>4. Conclusiones</b>	<b>10</b>
<b>5. Bibliografía</b>	<b>11</b>

# 1. Introducción al problema

El ecosistema de librerías de Python ha experimentado una expansión significativa durante la última década, impulsado por el crecimiento de disciplinas como el análisis de datos, el aprendizaje automático, la ingeniería de software y la computación científica. A medida que la cantidad de paquetes disponibles aumenta, también lo hace la complejidad de las relaciones entre ellos, especialmente en lo referente a sus dependencias. Estas dependencias forman redes que pueden analizarse mediante herramientas de teoría de grafos con el fin de comprender su estructura, detectar patrones organizacionales y evaluar su evolución en el tiempo.

El presente informe analiza la Red de Dependencias de librerías de Python en dos momentos temporales: 2016 y 2025. Para cada dataset se construye un grafo dirigido donde los nodos representan paquetes y las aristas representan relaciones de dependencia. Sobre cada red se aplican métricas estructurales (grado, centralidades, componentes, conectividad), análisis de subgrafos, y detección de comunidades utilizando algoritmos de particionado como Louvain. Esto permite caracterizar el ecosistema en términos de concentración, modularidad, robustez y vulnerabilidad.

La comparación entre ambas redes revela cómo ha evolucionado la arquitectura del ecosistema Python: si se ha vuelto más denso o más modular, si dependen más de ciertos paquetes centrales, etc. Este análisis no solo aporta una comparación temporal, sino que también permite identificar tendencias relevantes del ecosistema de paquetes en Python.

## 2. Obtención de datos

### 2.1. Origen de los datos

Para llevar a cabo el análisis de la Red de Dependencias de librerías de Python, se construyeron dos conjuntos de datos correspondientes a los años 2016 y 2025. El dataset de 2016 se adquirió de un análisis anterior hecho por Kevin Gullikson [1] (estudiante de Texas University). El de 2025 fue obtenido directamente desde Python Package Index (PyPI), la principal fuente pública y oficial de distribución de paquetes Python.

### 2.2. Limpieza de los datos

La descarga de información se realizó a partir de los metadatos publicados por cada paquete en PyPI, los cuales incluyen, entre otros campos, el nombre del paquete, su versión y la lista de dependencias declaradas, además de una pequeña descripción del paquete. Una vez recopilados los datos brutos, se aplicó un proceso de limpieza y estandarización. Esto incluyó:

1. Eliminación de paquetes duplicados o versiones redundantes, conservando únicamente la versión relevante para cada año.

2. Normalización de nombres de paquetes, debido a variaciones de estilo o inconsistencias en la forma en que algunos proyectos declaran dependencias
3. Filtrado de dependencias inválidas o huérfanas, es decir, aquellas que refieren a paquetes que no existen en el índice.
4. Construcción final de una lista consistente de pares (paquete  $\rightarrow$  dependencia), lista para su representación como grafo dirigido.

Este proceso garantizó que ambos datasets fueran comparables entre sí y que la red obtenida reflejara con la mayor fidelidad posible la estructura real del ecosistema de librerías de Python para cada año analizado.

## 3. Análisis exploratorio

### 3.1. Dataset 2016

#### 3.1.1. Generalidades del grafo

Para comenzar el análisis del grafo, se exploraron primero algunas de sus características estructurales fundamentales, tales como la cantidad de nodos, la cantidad de aristas y el grado promedio. El grafo, al que se hará referencia como  $G$ , fue construido utilizando la biblioteca NetworkX y cuenta con 26.234 nodos y 72.252 aristas dirigidas. El grado de entrada y de salida promedio son idénticos, ambos con un valor aproximado de 2.75, lo que deja un grado promedio total cercano a 5.51.

#### 3.1.2. Conectividad

Para profundizar en la estructura interna de  $G$ , se procedió a analizar su conectividad. Se observó la presencia de una cantidad muy elevada de componentes fuertemente conexas, alcanzando un total de 26.186. Sin embargo, a pesar de este número, sus tamaños individuales son reducidos: la componente fuertemente conexa de mayor tamaño posee únicamente 17 nodos. Esto sugiere que, si bien muchas regiones del grafo presentan conectividad bidireccional local, estas zonas son chicas y dispersas.

Dado que se trata de un grafo dirigido, también resulta relevante examinar las componentes débilmente conexas. En este caso, el panorama es opuesto en algún sentido: se encontraron 411 componentes débilmente conexas, pero ahora sí con tamaños considerablemente mayores, destacándose una componente gigante con 25.169 nodos. Esto indica que, aunque el grafo no posee grandes regiones donde todos los nodos sean alcanzables mutuamente en ambos sentidos, sí presenta un núcleo de conectividad unidireccional muy extenso. Dentro de esta gran componente, se encuentran “islas” pequeñas que sí cumplen la condición de conectividad fuerte

### 3.1.3. Análisis de centralidad

Una vez realizado este análisis preliminar, se procedió a examinar G con mayor detalle mediante la aplicación de diversas métricas de centralidad, ampliamente utilizadas en teoría de redes:

1. Degree centrality: mide la cantidad de conexiones directas de un nodo.
2. Closeness centrality: cuantifica cuán cerca se encuentra un nodo del resto, en términos de distancias mínimas.
3. Betweenness centrality: refleja la proporción de caminos mínimos que atraviesan un nodo.
4. PageRank: estima la importancia de un nodo considerando tanto la cantidad como la relevancia de quienes lo apuntan.

Los resultados de los cinco nodos más destacados en cada métrica se muestran en la Figura 1.

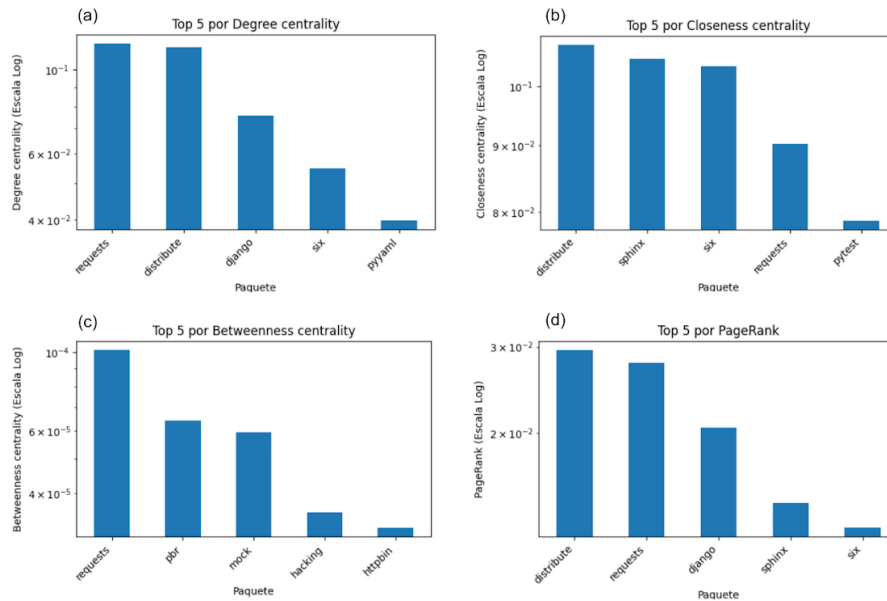


Figura 1: Análisis de centralidad para el dataset de 2016. Se reportan los top 5 nodos con valor más elevado en (a) Degree centrality, (b) Closeness centrality, (c) Betweenness centrality y (d) PageRank

Como primera observación, se identifica que la librería Requests aparece de forma recurrente en el top 5 de todas las medidas, lo que la posiciona como un nodo altamente relevante desde múltiples perspectivas. Requests es una biblioteca que permite enviar solicitudes HTTP/1.1 de manera sencilla, y se encuentra entre los paquetes más descargados del ecosistema Python, alcanzando alrededor de 30 millones de descargas semanales (<https://www.python.org/downloads/>).

Que Requests ocupe posiciones tan altas en degree centrality, closeness y PageRank indica que es apuntada por numerosos nodos, que se ubica estructuralmente cerca de gran parte del grafo y que recibe enlaces desde otros nodos “importantes”. Además, su presencia en PageRank en el puesto número 2 refuerza la idea de que su rol no se debe únicamente a la cantidad de dependencias: sino que también es central dentro de la “jerarquía” del sistema.

Se puede ver que tanto en Degree centrality, PageRank y Closeness centrality, hay varios de los paquetes que se repiten en el top. Sin embargo, al observar betweenness centrality, el panorama cambia: muchas de las librerías destacadas en esta métrica no aparecen en los rankings previos. Esto es esperable, dado que la métrica captura nodos que, sin tener grandes grados o posiciones privilegiadas en términos de cercanía, resultan indispensables para conectar distintas regiones del grafo.

Tres casos representativos son mock, hacking y PBR, todos presentes en el top de betweenness:

1. mock es una biblioteca empleada en entornos de prueba, utilizada para reemplazar componentes del sistema bajo evaluación.
2. hacking es un conjunto de extensiones destinadas a hacer cumplir la guía de estilo de OpenStack, promoviendo uniformidad y evitando patrones peligrosos.
3. PBR proporciona comportamientos predeterminados útiles durante los procesos de empaquetado con setuptools.

Estos paquetes no poseen valores particularmente altos en las otras métricas, pero sí ocupan posiciones clave en los caminos mínimos del grafo. Esto sugiere que funcionan como puentes estructurales entre diferentes regiones o comunidades del ecosistema.

En conjunto, este análisis revela que el ecosistema presenta una estructura altamente fragmentada en términos de conectividad fuerte, pero sostenida por una gran componente débilmente conexa. A nivel centralidad, se distinguen nodos que son importantes por su visibilidad global (como Requests) y otros cuya relevancia se encuentra relacionada con su función de “articulación” entre grupos (como mock, hacking y PBR). Esta diversidad de roles estructurales es consistente con las dinámicas típicas de redes tecnológicas amplias y descentralizadas como la del ecosistema Python.

#### **3.1.4. Subgrafos**

Antes de realizar la detección de comunidades, se construyeron distintos subgrafos núcleo a partir del filtrado de nodos según medidas de centralidad. En particular, se consideraron subgrafos inducidos por umbrales de grado de entrada y salida, así como un subgrafo formado por el 10 % de los nodos con mayor PageRank, con el objetivo de reducir el tamaño de la red y quedarnos con los nodos más importantes.

Estos subgrafos permiten eliminar nodos periféricos y concentrar el análisis en aquellos paquetes que cumplen un rol central dentro del ecosistema de dependencias. El grado de entrada refleja cuántos paquetes dependen directamente de una librería, mientras que el

PageRank aporta una medida más global de importancia, teniendo en cuenta también la relevancia de los nodos conectados.

Al comparar los distintos subgrafos obtenidos, se observa que algunas librerías aparecen de manera consistente en todos los recortes. En particular, Requests, Django, Distribute y Sphinx se destacan como nodos centrales, lo que sugiere que funcionan como componentes básicos compartidos por una gran cantidad de paquetes y justifican su análisis en etapas posteriores, como la detección de comunidades.

### 3.1.5. Comunidades

Primero, se analizó la red completa para identificar comunidades utilizando el algoritmo Louvain buscando maximizar la modularidad. Este paso permite conocer cuántas comunidades existen, el tamaño de cada una y la importancia relativa de cada nodo mediante el cálculo de grado y PageRank. Aquí se muestra una cantidad de 639 comunidades teniendo la comunidad más grande 5840 nodos. La modularidad de esa división es fué de 0.538.

Para poder graficar y hacer análisis más a detalle, se crearon subgrafos filtrando nodos según criterios de relevancia. Por un lado, se seleccionaron nodos que tengan un grado de entrada mínimo, con el objetivo de concentrarse en los nodos más conectados dentro de la red. Por otro lado, se crearon subgrafos con los nodos más importantes según PageRank, permitiendo analizar la estructura de la red focalizándose en los nodos centrales. Algunos de estos gráficos los podemos ver en la figura 2.

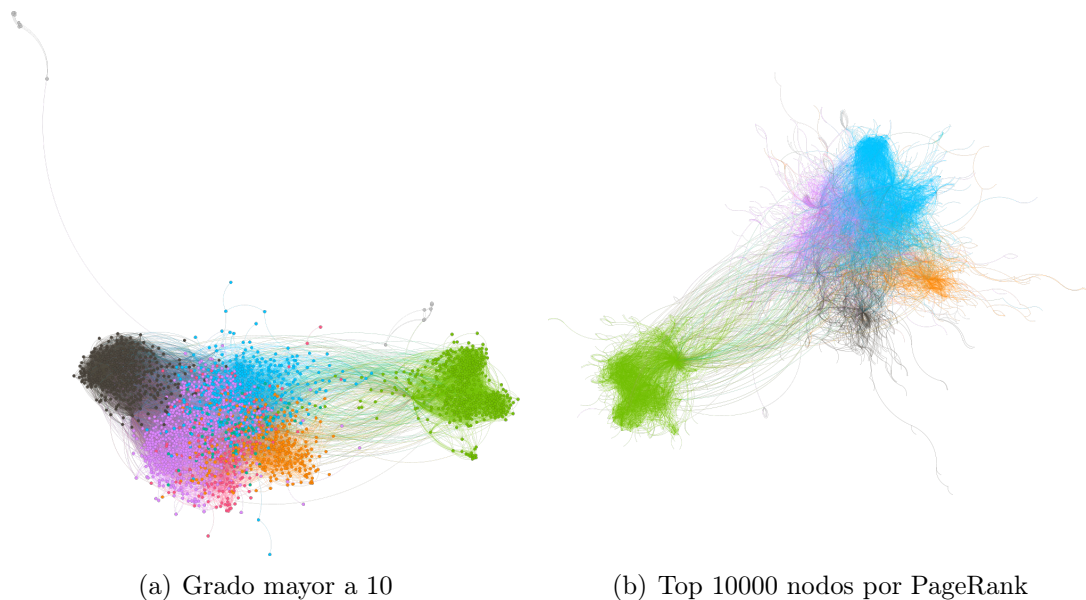


Figura 2: Visualizaciones de la red separada por comunidades y filtrados por distintos criterios.

De estos gráficos se puede ver que hay algunas comunidades que están bien distinguidas del resto de la red, mientras que hay otras cuya línea divisoria es más difusa.

Además, se realizó un análisis específico de las tres comunidades más grandes de la red. Se seleccionan todos los nodos que pertenecen a estas comunidades y se construye un

subgrafo, en él se ve como cada comunidad se centra en una de las siguientes librerías: Distribute, Request y Django. Cada una representa 22.3 %, 20.3 % y 12.1 % de la cantidad total de nodos.

### **3.1.6. Wordcloud**

Para poder comprender mejor la composición de cada comunidad y que las representa, se decidió estudiar las descripciones textuales de los paquetes en cada comunidad. El texto se procesó eliminando palabras comunes o irrelevantes y separando correctamente los términos, con el fin de identificar las palabras más frecuentes. Paralelamente, se evalúa la relevancia de cada nodo mediante su valor de PageRank, que indica su centralidad en la red. Esta combinación permite comprender los temas predominantes en las descripciones de los nodos y, al mismo tiempo, identificar los nodos más influyentes dentro de la red.

Analizando las 10 comunidades más grandes, podemos ver que la mayoría de comunidades tienen un enfoque en herramientas web. La primer comunidad es bastante variada usando paquetes como sphinx, six y pyyaml. La segunda librería está liderada por el paquete requests y son herramientas de parseado y apis. La tercer comunidad es liderada por distribute y contiene muchos paquetes de bajo pagerank. Es interesante ver que las librerías científicas recién aparecen como quinta comunidad más grande.

## **3.2. Dataset 2025**

Para el dataset correspondiente al año 2025 se repitió la estrategia inicial de exploración, aunque con ciertos ajustes necesarios debido al tamaño significativamente mayor del conjunto de datos.

### **3.2.1. Generalidades del grafo**

El grafo construido, al que se lo denominará H, es un grafo dirigido elaborado mediante la biblioteca NetworkX y está compuesto por 295.898 nodos y 1.606.337 aristas.

El análisis de sus grados revela que tanto el in-degree como el out-degree presentan un promedio cercano a 5,43, lo que sitúa el grado total medio alrededor de 10,86. Estos valores muestran un incremento notable en la densidad de conexiones respecto al grafo de 2016, consistente con la expansión del ecosistema Python en los últimos años.

### **3.2.2. Conectividad**

Con el objetivo de comprender cómo se organiza internamente H, se examinó su conectividad. El grafo exhibe 294.935 componentes fuertemente conexas, casi todas de un tamaño extremadamente reducido; la mayor de ellas reúne apenas 62 nodos. Este panorama confirma que la posibilidad de ir y volver entre ellos continúa siendo un fenómeno excepcional y localizado en grupos pequeños dentro de la red.

La situación cambia al observar las componentes débilmente conexas: se identificaron 1.386, de las cuales una sobresale ampliamente al contener 291.204 nodos (prácticamente englobando a toda la red). Este comportamiento, el cual ya se notaba en el análisis del dataset 2016, se manifiesta aquí de manera aún más marcada: existe un gran conglomerado de nodos enlazados de manera unidireccional, que constituye el núcleo estructural del grafo, acompañado por numerosos subconjuntos más pequeños donde también existe conectividad fuerte.

### 3.2.3. Análisis de centralidad

Se calcularon tres métricas de centralidad: degree centrality, closeness centrality y PageRank, todas implementadas mediante NetworkX. No fue posible obtener la betweenness centrality debido al tiempo de procesamiento requerido: dado que esta métrica ya resultaba costosa en el grafo de aproximadamente 20.000 nodos del dataset 2016, su ejecución en una red cercana a los 300.000 nodos es posible sin recursos computacionales con los que no se cuentan a la hora del análisis. Por eso, se optó por estimar el betweenness centrality mediante una versión aproximada basada en muestreo. En este enfoque, en lugar de considerar todos los pares de nodos posibles, NetworkX selecciona aleatoriamente un subconjunto de nodos como fuentes para el cálculo de caminos mínimos, lo que reduce de manera significativa el tiempo de ejecución a costa de introducir una pequeña aproximación en los valores obtenidos.

En la Figura 2 se ilustran los nodos más relevantes según las métricas consideradas. Tal como ocurría en el dataset anterior, Requests vuelve a posicionarse entre los paquetes más influyentes del grafo, reflejando su uso sostenido dentro del ecosistema y su rol central en proyectos que requieren comunicación HTTP.

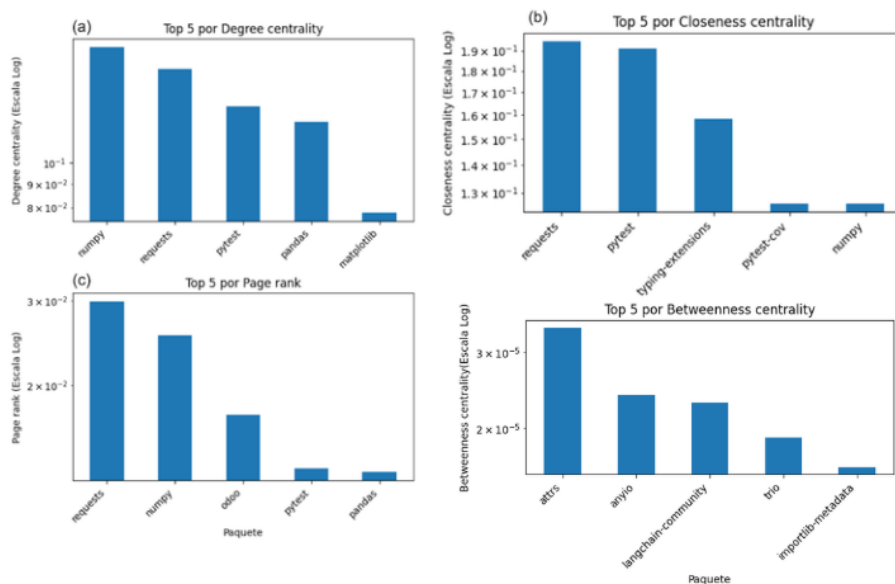


Figura 3: Análisis de centralidad para el dataset de 2025. Se reportan los top 5 nodos con valor más elevado en (a) Degree centrality, (b) Closeness centrality, (c) PageRank y (d) Betweenness centrality

Junto a Requests emerge con otra librería entre las más influyentes llamada `pytest`, que es una herramienta ampliamente utilizada para la automatización de pruebas que detecta archivos de test, ejecuta funciones de verificación y reporta resultados de manera sistemática (<https://www.python.org/downloads/>). Su ubicación en los primeros puestos se puede deber a la adopción masiva de prácticas de testing en el desarrollo de software en los últimos tiempos.

Otro nombre que adquiere relevancia es `pandas`, biblioteca fundamental para el análisis de datos, que proporciona estructuras como `Series` y `DataFrame` y que hoy supera los 100 millones de descargas mensuales [2]. Su presencia entre los nodos con mayor centralidad es coherente con el crecimiento exponencial de las tareas de procesamiento de datos en la comunidad Python.

El podio se lo lleva la librería `numpy`. NumPy es una biblioteca central del ecosistema científico de Python que provee estructuras de datos optimizadas para el cálculo numérico, en particular el arreglo multidimensional. Su implementación permite ejecutar operaciones vectorizadas y de álgebra lineal de manera altamente eficiente. Debido a estas características, NumPy constituye la base sobre la cual se desarrollan numerosas bibliotecas orientadas a análisis de datos, modelado matemático y aprendizaje automático.

En conjunto, estos resultados muestran que, si bien ciertos paquetes mantienen su predominio histórico, el grafo de 2025 revela un escenario donde nuevas áreas de interés —como el testing automatizado y el análisis de datos— adquieren un peso estructural considerable dentro de la red.

#### 3.2.4. Subgrafos

Para continuar con el análisis, se construyeron tres subgrafos sobre el dataset de 2025 correspondientes a las métricas de in-degree (subgrafo de nodos con in-degree mayor a 500), PageRank (top 300) y betweenness centrality (top 300). Luego se compararon sus librerías más centrales con las obtenidas previamente en el dataset de 2016. Esta comparación mostró qué dependencias se mantienen relevantes en el tiempo y cuáles reflejan cambios estructurales en el ecosistema. En particular, el contraste entre los subgrafos antiguos y los nuevos revela un desplazamiento de librerías asociadas al desarrollo web y compatibilidad (como `django` o `six`) y la aparición de librerías centrales en el Python moderno, especialmente en áreas de ciencia de datos, tipado estático y testing como `numpy`, `pytest`, `typing-extensions` y `pydantic`.

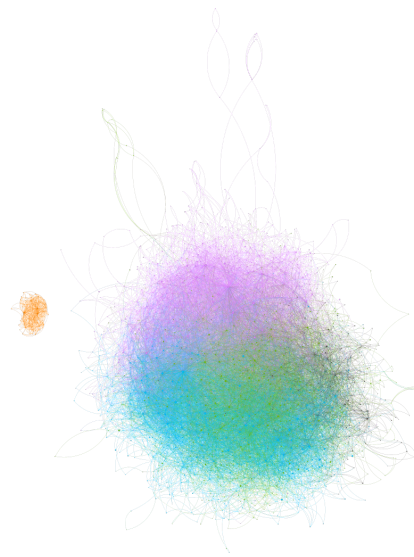
#### 3.2.5. Comunidades

El análisis hecho fué similar al hecho con la red anterior pero con resultados muy disintos. Se encontraron 3829 comunidades, la más grande con 116312 nodos. La modularidad cayó notablemente a 0.426. En la figura 4 podemos ver como quedaron algunos de los gráficos hechos sobre este dataset.

De estas imágenes podemos entender mejor la caída de la modularidad ya que se vé que las líneas divisorias de las comunidades son mucho más difusas. El nuevo dataset tiene muchas más aristas lo que complejiza la separación. Por ejemplo, en el top 10000 nodos



(a) Grado mayor a 100



(b) Top 10000 nodos por PageRank

Figura 4: Visualizaciones de la red separada por comunidades y filtrados por distintos criterios.

por pagerank, el dataset de 2016 tenía 18 mil nodos, mientras que éste tiene 29 mil.

### 3.2.6. Wordcloud

Al comparar los wordclouds de la red vieja con la actual se ven resultados interesantes. Primero, la comunidad de paquetes científicos (pandas, matplotlib, scipy, scikit-learn) pasó de ser la quinta comunidad más grande a ser la segunda. Pero además, la tercer comunidad más grande es de paquetes de data science (numpy, torch, pillow, networkx). Es interesante que en esta comunidad aparezca tqdm, una librería para mostrar barras de progreso, dando a entender que la mayoría de estos paquetes tienen tiempos de ejecución largos. Por último, mirando a nivel general, podemos ver que cuando antes las palabras más comunes eran api, django, client; ahora son data, api, y tool.

## 4. Conclusiones

El análisis comparativo de la Red de Dependencias de librerías de Python entre 2016 y 2025 evidencia un crecimiento significativo del ecosistema. La red pasó de 26,234 nodos y 72,252 aristas en 2016 a 295,898 nodos y 1,606,337 aristas en 2025, con un incremento en la densidad de conexiones y el grado promedio, indicando que los paquetes modernos dependen de un mayor número de librerías.

Si bien la estructura general se mantiene con componentes fuertemente conexas limitadas y una gran componente débilmente conexa que sostiene la red, se observa un cambio en los roles centrales. Librerías históricas de compatibilidad y desarrollo web, como Django y Six, ceden relevancia, mientras que Requests conserva su posición como nodo influyente.

En 2025 emergen nuevas áreas centrales: ciencia de datos y computación científica (NumPy, Pandas, SciPy), testing y calidad de código (Pytest), y tipado estático (Typing-Extensions, Pydantic). Además, se distinguen nodos con visibilidad global y nodos que actúan como puentes estructurales entre comunidades.

En síntesis, el ecosistema Python se ha vuelto más denso y diversificado, con un núcleo interconectado unidireccionalmente y una reestructuración de sus librerías centrales hacia ciencia de datos, computación científica y prácticas de testing automatizado.

## 5. Bibliografía

[1] Gullikson, Kevin. (2016, Febrero 18). Python Dependency Analysis. <https://kgullikson88.github.io/blanalysis.html>

Chugh, V.(2025, Febrero 9). Tutorial de pandas en Python: La guía definitiva para principiantes. Datacamp. <https://www.datacamp.com/es/tutorial/pandas>