pypi data

February 13, 2023

1 Análisis de importancia de paquetes en el ecosistema Python

1.1 I. Javier Mermet

2 Introducción al problema

El día 8 de julio de 2022 muchos maintainers de paquetes de PyPI recibieron un email similar al siguiente:

```
From: noreply@pypi.org
Subject: [PyPI] A project you maintain has been designated as critical

Congratulations! A project you ('crossnox') maintain has been designated as a critical
project on PyPI. You can see which project(s) has been designated
at http://pypi.org/manage/projects/. .

As part of this effort, in the coming months maintainers of projects designated as
critical, like yourself, will be required to enable two-factor authentication on
their account in order to add new releases or otherwise modify a critical project.

Since you already have two-factor authentication enabled on your account,
there's nothing you need to do at this time.

PS: To make it easier for maintainers like you to enable two-factor authentication,
we're also distributing security keys to eligible maintainers.
See http://pypi.org/security-key-giveaway/ for more details.
```

A simple vista parecía un intento de phishing. El lenguaje, felicitandonos, y links que no eran HTTPS.

Sin embargo, una breve investigación reveló que era real 1 2. PyPI estaba marcando el top 1% de paquetes como "Critical" y obligando a sus maintainers a activar 2FA para evitar ataques de cadena de suministro.

En una sección de preguntas frecuentes, se explicaba como se determina si un proyecto es critico:

> What determines if a project is a critical project?
>
> PyPI determines eligibility based on download counts derived from PyPI's public dataset of download statistics. Any project in the top 1% of downloads over the prior 6 months is designated as critical.

Lo cual abre el interrogante: es la cantidad de descargas la mejor metrica para definir si un proyecto es *critico* en el ecosistema?

El objetivo de este trabajo es investigar métricas alternativas, basadas en fuentes adicionales de datos y en el grafo de dependencias entre bibliotecas. Se aplican distintas heurísticas cuyo impacto se discute en las conclusiones finales.

3 Qué es un paquete?

Un paquete es un modulo que puede contener submodulos o, recursivamente, subpaquetes.

Un modulo es un objeto que sirve como una unidad organizacional de codigo python. Tiene un namespace conteniendo objetos arbitrarios de python. Los modulos se cargan en Python al importarlos.

En la documentación del lenguaje se provee más información sobre los módulos, y es particularmente relevante leer sobre el sistema de imports para entenderlos del todo.

4 Bibliotecas

Nos interesa pensar particularmente en los paquetes que son bibliotecas. En particular, vamos a hacer una distincion entre bibliotecas y aplicaciones, donde conceptualmente vamos a distinguir a las bibliotecas como paquetes desarrollados con la intención de ser usados por terceros de modo reutilizable. Para poder ser reutilizable, tiene que ser distribuido. Y eso es un problema.

4.1 Metadata

Muchos principantes que aprenden python aprenden a especificar las dependencias de su codigo en un pequeño archivito de texto plano llamado `requirements.txt`. Sin embargo, en muchos paquetes vemos un script llamado `setup.py`. Minimamente, se ven asi:

```
from setuptools import setup

setup(
    name="MyLibrary",
    version="1.0"
    ...
)
```

Volviendo al articulo de Donald Stufft, algo que queda evidente es la semantica de cada una de las dos opciones: 1. `requirements.txt` es deterministico en el sentido de que dado un cierto conjunto de paquetes disponibles, y la especificacion del nombre del paquete (y opcionalmente un especificador de version), siempre instalará las mismas dependencias. 2. Al ser un script, `setup.py` no garantiza intrinsicamente ser deterministico, permitiendo instalar dependencias condicionales a la plataforma donde se está instalando.

Es evidente que este último formato es mucho más apto para cosas redistribuibles. Lo cual hace *imposible* armar el grafo de dependencias a menos que se limiten *severamente* algunas variables que comunmente se usan para determinar las dependencias a instalar. Por ejemplo: - Version de python - Plataforma - OS

Y **eso** el formato wheel. Y por **eso** es que hay algo de información de dependencias disponible en pypi.

5 Información de PyPI

La primer forma de obtener información de pypi sería scrapear todo el sitio, abriendonos a la posibilidad de que nos bloqueen la IP, logrando, efectivamente, un ataque de supply chain (nuestro y de todas las personas en el bloque de la CG-NAT donde nuestros ISPs nos hayan puesto, en su avaricia por ahorrar direcciones IPv4). Esto no es deseable, generalmente. Adicionalmente, en la interfaz de PyPI no hay información sobre dependencias entre paquetes. Lamentablemente la API de PyPI no provee 3 4 información sobre paquetes críticos o con 2FA mandatorio.

El segundo modo es, por cada paquete, armar una maquina virtual de una determinada plataforma, crear un entorno virtual, instalar dicho paquete y revisar los paquetes instalados. Por ejemplo,

```
$ pip show numpy
Name: numpy
Version: 1.23.5
Summary: NumPy is the fundamental package for array computing with Python.
Home-page: https://www.numpy.org
Author: Travis E. Oliphant et al.
Author-email: None
License: BSD
Location: /home/nox/repos/venv/lib64/python3.8/site-packages
Requires:
Required-by: xgboost, ucx-py-cu11, treelite, treelite-runtime, torchvision, thinc, tensorflow,
```

Y luego reconstruir el grafo completo bajo esas constraints. Notar que en este esquema queremos que cada descarga se haga en un entorno sandboxeado para evitar ataques al instalar paquetes (lo cual es posible al usar `setup.py` porque es un script). 5 6 7 8 9

Afortunadamente hay un tercer modo más práctico y menos costoso. En la documentación de packaging hay una sección llamada Analyzing PyPI package downloads. Explican por que PyPI no muestra estadísticas de descarga. En su lugar, el proyecto linehaul hace streaming de los logs de descarga hacia un dataset publico en Google Bigquery a través de una cloud function. Este dataset es publico y se puede consultar. Una cuenta nueva de GCP tiene un crédito de 300 USD que puede usar por arriba del limite de 1TB gratuito de consultas por mes que nos da la plataforma.

Hay un poco más de detalles sobre este dataset en la documentación de warehouse.

> Warehouse is a web application that implements the canonical Python package index (repository); its production deployment is PyPI.

Aquí vemos algo que será muy relevante: además de una tabla con la información de descargas, hay una tabla con la metadata de todos los releases subidos a PyPI. Estos datos siguen la especificación de core metadata. En particular nos interesan los campos: - Name normado por el PEP 508 - Version normado por el PEP 440 - Classifier normados por el PEP 301, para filtrar por OS - Platform por el mismo motivo - Requires-Dist normado por el PEP 508 para extraer información de dependencias - Sidenote: gracias a twine este campo tiene más data recientemente - Requires-Python para filtrar versiones de distribución de python (util para deprecar un paquete para versiones viejas de Python)

Al buscar en BigQuery el dataset publico, encontramos una tercer tabla:

Buscando en discuss.python.org, encontramos este hilo del ya mencionado Donald Stufft, Core Developer de CPython. Y acá surge el primer punto: la cantidad de consultas a la API de `/simple/{project}` muchas veces es mas alta que la cantidad de descargas dado que `pip` (y otros clientes) mantienen una cache de descargas, por lo cual la cantidad de descargas sirve para comparar tendencias, pero no necesariamente es la mejor medida del uso. Por otro lado, hay clientes o bots que hacen consultas a la API pero no descargan nada, por lo que el valor puede estar inflado.

6 Trabajos relacionados

6.1 Analyzing PyPI Metadata - Martin Thoma

- Parte I
- Parte II

La primer entrega es un breve analisis descriptivo de la data de descargas. La segunda un pequeño armado del grafo y analisis de nombres.

6.2 PyPi interactive dependency graph

En este link se aprecia un bello grafo cuyo layout y color se armó con gephi y se visualiza con sigmajs.

Hay que notar que contiene pocos paquetes (solo 16536) y las dependencias se extraen desde el `setup.py` de cada uno.

6.3 pydepgraph – A dependencies analyzer for Python - Stefano Maggiolo

Biblioteca para graficar dependencias entre paquetes.

6.4 Python Dependency Analysis

Se extraen manualmente las dependencias, y se hace un analisis de la red resultante. Se ve centralidad, grado, conectividad, comunidades.

6.5 pipdeptree

Utilidad de linea de comandos para mostrar los paquetes instalados como un arbol.

6.6 deps.dev

Plataforma de recolección de datos de dependencias.

6.7 Visualizadores de paquetes top de pypi

- https://hugovk.github.io/top-pypi-packages/
- https://pypistats.org/top
- https://pepy.tech/

6.8 Thoth Project - RedHat

Cuya oferta principal es un resolvedor cloud de dependencias

7 Lecturas interesantes

- pip - Dependency Resolution
- pip - Caching
- blog de Donald Stufft
- PEP 425 para saber parsear versiones de python desde distribuciones compiladas (wheels)
- Packaging - Binary distribution format para entender el formato `wheel` de distribuciones binarias especificado en el PEP 427

8 Links miscealaneos

- Python infrastructure status
- pypinfo CLI para acceder a la data de BigQuery
  - Otros de esta lista
- criticality-score
  - Y su aplicación en pypi
- snakefood para armar un grafo de dependencias sin cargar los modulos (python2)

8.1 Packaging

Con esta biblioteca se puede parsear facilmente la lista de `requirements` de una biblioteca o `pasar su nombre a forma canónica`, sin implementar las gramáticas de sus respectivos PEPs.

9 Utilidades

[1]:
```
import re
from pprint import pprint

import pyarrow.parquet as pq
import rich.tree
```

```
def get_schema(f):
    l = f.flatten()
    if len(l) == 1:
        return str(l[0].type)

    return {str(li.name).split(".")[-1]: get_schema(li) for li in l}


def _get_rt(k, v):
    if not isinstance(v, dict):
        _vstr = str(v)
        _vstr = re.sub(r"list\<item: (\w+)\>", r"list<\1>", _vstr)
        # print(k,": " ,str(v), " --> ",_vstr)
        return rich.tree.Tree(f"{k}: {_vstr}")

    t = rich.tree.Tree(k)
    for _k, _v in v.items():
        child = _get_rt(_k, _v)
        t.children.append(child)
        child.parent = t

    return t


def get_rt(d, name):
    root = rich.tree.Tree(name, highlight=False)
    for k, v in d.items():
        child = _get_rt(k, v)
        # print(child, child.label)
        root.children.append(child)
        child.parent = root
    return root


def print_parquet_schema(fname, tname):
    pfile = pq.read_table(fname).schema

    d = {}

    for field in pfile:
        d[field.name] = get_schema(field)

    d = {k: d[k] for k in sorted(d.keys())}

    return get_rt(d, tname)
```

10 Analisis de datos

En esta sección se analizan los datasets descargados desde BigQuery, indicando las queries realizadas.

[2]:
```
import pandas as pd
```

10.1 Proyectos mas descargados

```
SELECT
  file.project as project,
  file.version as version,
  DATE(timestamp) as date,
  COUNT(*) as num_downloads
FROM `bigquery-public-data.pypi.file_downloads`
WHERE
  -- avoid bandersnatch and other mirrors
  details.installer.name = 'pip'
  -- use the oldest alive version
  AND details.python LIKE '3.8.%'
  -- pypy people are weird
  AND details.implementation.name = 'CPython'
  -- Only linux
  AND details.system.name = 'Linux'
  -- Only query the last 30 days of history
  AND DATE(timestamp)
    BETWEEN DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY)
    AND CURRENT_DATE()
GROUP BY 1, 2, 3
ORDER BY 1 ASC, 2 ASC, 3 ASC
```

[3]:
```
most_downloaded = pd.read_parquet("file_downloads__30d__20230204.parquet")
```

[4]:
```
most_downloaded.head()
```

[4]:

	project	version	date	num_downloads
0	0	0.0.0	2023-01-06	4
1	0	0.0.0	2023-01-09	4
2	0	0.0.0	2023-01-10	5
3	0	0.0.0	2023-01-11	3
4	0	0.0.0	2023-01-12	1

[5]:
```
most_downloaded = (
    most_downloaded.groupby("project").num_downloads.sum().
 ↪sort_values(ascending=False)
)
```

[6]:
```
top_downloaded = most_downloaded.head(int(len(most_downloaded) * 0.01))
```

[7]:
```
len(top_downloaded)
```

[7]:
```
1552
```

Pero hay mas paquetes marcados criticos. Al dia de escribir esto (2023-02-04) hay 4324 paquetes marcados como criticos.

[8]:
```
top_downloaded = most_downloaded.head(4324)
```

[9]:
```
top_downloaded.head()
```

[9]:
```
project
boto3               82825736
botocore            78837661
google-api-core     77962554
cryptography        73804490
requests            73227562
Name: num_downloads, dtype: int64
```

10.2 Proyectos mas consultados

```
SELECT
  project,
  DATE(timestamp) as date,
  COUNT(*) as num_requests
FROM `bigquery-public-data.pypi.simple_requests`
WHERE
  -- avoid bandersnatch and the likes
  details.installer.name = 'pip'
  -- use the oldest alive version
  AND details.python LIKE '3.8%'
  -- pypy people are weird
  AND details.implementation.name = 'CPython'
  -- Only linux
  AND details.system.name = 'Linux'
  -- Only query the last 30 days of history
  AND DATE(timestamp)
    BETWEEN DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY)
    AND CURRENT_DATE()
GROUP BY 1, 2
ORDER BY 1 ASC, 2 ASC
```

[10]:
```
most_requested = pd.read_parquet("simple_requests__30d__20230204.parquet")
```

[11]:
```
most_requested = (
    most_requested.groupby("project").num_requests.sum().
  ↪sort_values(ascending=False)
)
```

[12]:
```
top_requested = most_requested.head(4324)
```

[13]:
```
top_requested.head()
```

[13]:
```
project
pip             485823010
requests        200246682
oauthlib        169602357
setuptools      168930870
cryptography    156799618
Name: num_requests, dtype: int64
```

10.3 Son la misma métrica?

Para contestar a esta pregunta, vamos a ver utilizar el coeficiente Kendall Tau como medida de correlación entre los rankings de descargas y consultas.

[14]:
```
libmetrics = pd.merge(
    most_downloaded, most_requested, left_index=True, right_index=True,␣
 ↪how="outer"
)
```

[15]:
```
libmetrics["downloads_ranking"] = libmetrics.num_downloads.rank(
    ascending=False, na_option="keep",
)
```

[16]:
```
libmetrics["requests_ranking"] = libmetrics.num_requests.rank(
    ascending=False, na_option="keep",
)
```

[17]:
```
from scipy.stats import kendalltau
```

[18]:
```
mask = libmetrics.downloads_ranking.notna() & libmetrics.requests_ranking.
 ↪notna()

kendalltau(
    libmetrics[mask].downloads_ranking,
    libmetrics[mask].requests_ranking,
    nan_policy="omit",
)
```

[18]:
```
SignificanceResult(statistic=0.8821265353536296, pvalue=0.0)
```

Y si bien tienen una alta correlación, no son *exactamente* iguales. Veamos.

10.4 Diferencias del top 100

[19]:
```
import numpy as np
from IPython.display import Markdown, display
```

[20]:
```
def format_delta(x):
    if x > 0:
        return f" {int(x):d}"
    elif x < 0:
        return f" {int(abs(x)):d}"
    else:
        return f" "


libmetrics["change"] = libmetrics.downloads_ranking - libmetrics.
 ↪requests_ranking


def style_negative(v, neg="", pos=""):
    if v > 0:
        return pos
    elif v < 0:
        return neg
    else:
        return ""


libmetrics.sort_values("downloads_ranking").head(100).style.format(
    {"change": format_delta}, precision=0
).applymap(
    style_negative,
    neg="background-color:#a60101",
    pos="background-color:#045206",
    subset=["change"],
)
```

[20]:
```
<pandas.io.formats.style.Styler at 0x7f258879ce20>
```

[21]:
```
display(
    Markdown(
        f"""El {(libmetrics.sort_values("downloads_ranking").head(100).
 ↪requests_ranking <= 100).mean():.2%} de los paquetes mas descargados dentro␣
 ↪del top 100 no son los mas consultados.

La caída promedio es de {libmetrics.sort_values("downloads_ranking").head(100).
 ↪change.mean()} posiciones."""
    )
```

```
)
```

El 54.00% de los paquetes mas descargados dentro del top 100 no son los mas consultados.

La caída promedio es de -54.19 posiciones.

10.5 Paquetes mas descargados que pedidos

Es una circunstancia que debería darse poco. Veamos cuantas bibliotecas caen en esta categoria.

[22]:
```
(libmetrics.num_downloads > libmetrics.num_requests).sum()
```

[22]:
```
3957
```

[23]:
```
libmetrics[(libmetrics.num_downloads > libmetrics.num_requests)].sort_values(
    "downloads_ranking"
).head(25)
```

[23]:

project	num_downloads	num_requests	downloads_ranking	requests_ranking	change
google-api-core	77962554.0	46745668.0	3.0	91.0	-88.0
cfn-lint	20630824.0	1152568.0	48.0		
google-cloud-bigquery	20246170.0	11563134.0	52.0		
grpcio-status	15493572.0	13603532.0	73.0		
aiobotocore	13639815.0	5717551.0	89.0		
s3fs	12845298.0	7547873.0	95.0		
coverage	12720355.0	10796490.0	96.0		
nbconvert	9175822.0	5825404.0	135.0		
tensorflow-serving-api	8649142.0	908285.0	147.0		
google-cloud-pubsub	7353432.0	6937746.0	170.0		
grpcio-tools	7320259.0	4902745.0	171.0		
google-cloud-bigquery-storage	7151106.0	3643632.0	174.0		
sentry-sdk	6396017.0	5061180.0	185.0		
jupyter-server	6181530.0	4219075.0	190.0		
imageio	5211411.0	2871281.0	215.0		
tox	4916213.0	2517466.0	226.0		
delta-spark	3900300.0	3655276.0	287.0		
aws-sam-translator	3524890.0	1152589.0	314.0		
azureml-core	3144455.0	2272524.0	342.0		
databricks-sql-connector	3035002.0	2304612.0	349.0		
distributed	2185298.0	941346.0	441.0		
azureml-dataprep	2133840.0	1385593.0	450.0		
constructs	2062233.0	165985.0	455.0		
opentelemetry-sdk	1956764.0	1869175.0	473.0		
imbalanced-learn	1891805.0	1314674.0	489.0		

```
cfn-lint                                1043.0  -995.0
google-cloud-bigquery                    251.0  -199.0
grpcio-status                            227.0  -154.0
aiobotocore                              391.0  -302.0
s3fs                                     320.0  -225.0
coverage                                 261.0  -165.0
nbconvert                                383.0  -248.0
tensorflow-serving-api                  1161.0 -1014.0
google-cloud-pubsub                      338.0  -168.0
grpcio-tools                             448.0  -277.0
google-cloud-bigquery-storage            521.0  -347.0
sentry-sdk                               435.0  -250.0
jupyter-server                           482.0  -292.0
imageio                                  611.0  -396.0
tox                                      702.0  -476.0
delta-spark                              520.0  -233.0
aws-sam-translator                      1042.0  -728.0
azureml-core                             756.0  -414.0
databricks-sql-connector                 753.0  -404.0
distributed                             1133.0  -692.0
azureml-dataprep                         950.0  -500.0
constructs                              2591.0 -2136.0
opentelemetry-sdk                        814.0  -341.0
imbalanced-learn                         987.0  -498.0
```

Es interesante que muchas de estas librerias estan relacionadas a entornos cloud.

10.6 Metadata

```
select
  name,
  version,
  platform,
  requires_python,
  requires,
  provides,
  obsoletes,
  requires_dist,
  provides_dist,
  obsoletes_dist,
  requires_external,
  upload_time,
  filename,
  python_version
from `bigquery-public-data.pypi.distribution_metadata`
;
```

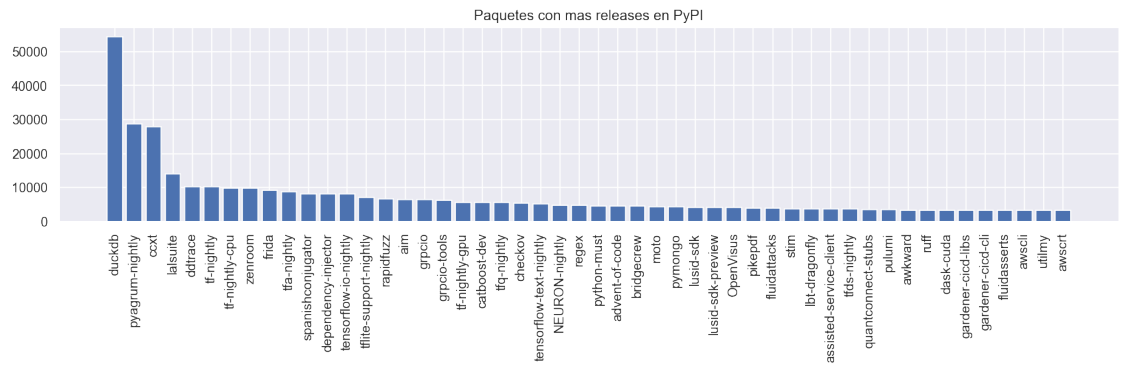
[24]:
```
metadata = pd.read_parquet("metadata__20230204.parquet")
```

[25]:
```
metadata = metadata.sort_values("upload_time", ascending=False)
```

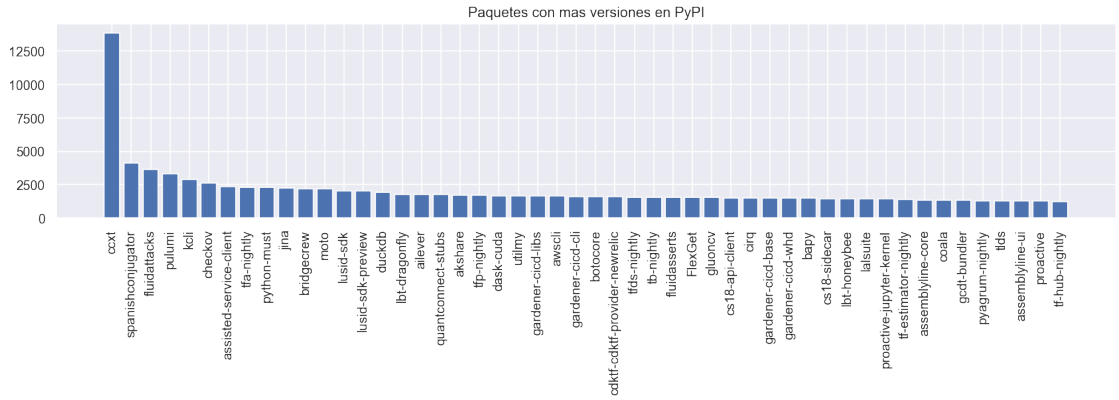
[26]:
```
rich.print(
    print_parquet_schema(
        "metadata__20230204.parquet", "bigquery-public-data.pypi.
 ↪distribution_metadata"
    )
)
```

```
bigquery-public-data.pypi.distribution_metadata
  filename: string
  name: string
  obsoletes: list<string>
  obsoletes_dist: list<null>
  platform: list<string>
  provides: list<string>
  provides_dist: list<null>
  python_version: string
  requires: list<string>
  requires_dist: list<string>
  requires_external: list<string>
  requires_python: string
  upload_time: timestamp
  version: string
```

[27]:
```
import seaborn as sns
from matplotlib import pyplot as plt

sns.set()

plt.figure(figsize=(16, 3), dpi=120)
packages = metadata.name.value_counts().head(50)
plt.bar(packages.index, packages.values)
plt.title("Paquetes con mas releases en PyPI")
plt.xticks(rotation=90)
plt.show()
```



[28]:
```
plt.figure(figsize=(16, 3), dpi=120)
packages = (
    metadata.groupby("name").version.nunique().sort_values(ascending=False).
  ↪head(50)
)
plt.bar(packages.index, packages.values)
plt.title("Paquetes con mas versiones en PyPI")
plt.xticks(rotation=90)
plt.show()
```



10.6.1 Filtrar releases

Considerar que nuestros datos de descarga son de: - Linux - CPython3.8.x - pip

CPython 3.8 Siguiendo PEP 425 - Python Tag.

Filtrar por `requires_python`

[29]:
```
display(f"Faltantes: {metadata.requires_python.isna().mean():.2%}")
```

```
'Faltantes: 55.09%'
```

[30]:
```
import logging
from functools import lru_cache

from packaging.requirements import InvalidRequirement, Requirement
from packaging.version import Version
from pandarallel import pandarallel
from tqdm.notebook import tqdm

pandarallel.initialize(nb_workers=8, progress_bar=True)
```

```
tqdm.pandas()


logger = logging.getLogger("pypidata")
logger.setLevel(logging.INFO)


@lru_cache(maxsize=5000)
def parse_pyreq(s):
    if s[-1] == ",":
        s = s[:-1]
    if s.startswith("=2"):
        s = f"={s}"
    if s.startswith("2"):
        s = f"=={s}"
    if s.startswith("3"):
        s = f"=={s}"
    if " " in s:
        s = s.replace(" ", "")

    s = s.replace("~3", "~=3").replace("~2", "~=2")

    s = s.replace("\n", ",")

    while ",," in s:
        s = s.replace(",,", ",")

    s = f"python{s}"

    try:
        return "3.8" in Requirement(s).specifier
    except InvalidRequirement as e:
        logger.error(f"At {repr(s)} {s} {',,' in s}", exc_info=True)


metadata.loc[metadata.requires_python.notna(), "requires_python"] = metadata[
    metadata.requires_python.notna()
].requires_python.progress_apply(parse_pyreq)

metadata = metadata[metadata.requires_python.isna() | metadata.requires_python]
```

```
INFO: Pandarallel will run on 8 workers.
INFO: Pandarallel will use Memory file system to transfer data between the main
process and workers.

  0%|          | 0/3834147 [00:00<?, ?it/s]
```

[31]:
```
for i in ["py3", "py2.py3", "cp38", "3.8", "any"]:
    assert (metadata.python_version == i).any(), i
```

Filtrar por python_version

[32]:
```
display(f"Faltantes: {metadata.python_version.isna().mean():.2%}")
```

```
'Faltantes: 0.00%'
```

[33]:
```
metadata = metadata[
    ~metadata.name.str.lower().str.contains("topsis")
]  # Funny story here, clearly an assignment
```

Can i delete source ones?

[34]:
```
source_releases = (
    metadata[(metadata.python_version == "source")]
    .drop_duplicates(subset=["version", "name"])[["version", "name"]]
    .assign(has_source=True)
)
non_source_releases = (
    metadata[metadata.python_version != "source"]
    .drop_duplicates(subset=["version", "name"])[["version", "name"]]
    .assign(has_nonsource=True)
)

has_both = pd.merge(
    left=source_releases,
    right=non_source_releases,
    how="outer",
    on=["version", "name"],
    suffixes=("_source", "_non_source"),
)

has_both.fillna(False, inplace=True)

pd.pivot_table(
    has_both,
    columns="has_source",
    index="has_nonsource",
    values="name",
    aggfunc="count",
    fill_value=0,
)
```

[34]:

has_source	False	True
has_nonsource		
False	0	1397461
True	491544	2450313

Yep.

[35]:
```
# Sacar outliers
top_python_versions = metadata.python_version.value_counts()
top_python_versions = top_python_versions[
    (top_python_versions.cumsum() / top_python_versions.sum()) <= 0.995
].index.tolist()
```

[36]:
```
orig_size = len(metadata)

metadata = metadata[
    (metadata.python_version.isin(top_python_versions))
    & ~(metadata.python_version.str.contains("pypy"))
    & ~(metadata.python_version.str.contains("cpcp"))
    & (
        (metadata.python_version.str.contains("^3\.8"))
        | (metadata.python_version.str.contains("(cp38|py38|cp3\.8|py3\.8)"))
        | (metadata.python_version.str.contains("py3(?![123456790])"))
        | (metadata.python_version.isin(["any",]))
    )
]

final_size = len(metadata)

print(f"Mantiene: {final_size / orig_size:.2%} de las filas")
```

```
/tmp/ipykernel_604015/1254240050.py:9: UserWarning: This pattern is interpreted
as a regular expression, and has match groups. To actually get the groups, use
str.extract.
  | (metadata.python_version.str.contains("(cp38|py38|cp3\.8|py3\.8)"))
```

```
Mantiene: 35.43% de las filas
```

[37]:
```
for i in ["py3", "py2.py3", "cp38", "3.8", "any"]:
    assert (metadata.python_version == i).any(), i
```

Filtrar por `filename` segun PEP 427

[38]:
```
from packaging.utils import InvalidVersion, InvalidWheelFilename,␣
↪parse_wheel_filename


def coalesce_parse_whl(w):
    try:
        name, version, build_number, tags = parse_wheel_filename(w)
        return name, version, build_number, tags
    except (InvalidWheelFilename, InvalidVersion):
        return None
```

```
metadata.head(100).filename.progress_apply(coalesce_parse_whl).iloc[50]
```

```
  0%|          | 0/100 [00:00<?, ?it/s]
```

[38]:
```
('nbdev-stdlib',
 <Version('0.0.594')>,
 (),
 frozenset({<py3-none-any @ 139786307057856>}))
```

Linux

[39]:
```
import operator as ops
```

[40]:
```
has_platform = metadata.platform.apply(len) > 0

metadata.loc[has_platform, "platform"] = metadata[has_platform].platform.apply(
    ops.itemgetter(0)
)
metadata.loc[~has_platform, "platform"] = None
```

[41]:
```
metadata.platform = metadata.platform.str.lower()
```

[42]:
```
metadata.loc[
    metadata.platform.notna() & metadata.platform.str.contains("linux"),␣
 ↪"platform"
] = "linux"
metadata.loc[
    metadata.platform.notna() & metadata.platform.str.contains("posix"),␣
 ↪"platform"
] = "linux"
metadata.loc[
    metadata.platform.notna() & metadata.platform.str.contains("ubuntu"),␣
 ↪"platform"
] = "linux"

top_platforms = metadata.drop_duplicates(
    subset=["name", "platform"]
).platform.value_counts()
top_platforms = top_platforms[top_platforms > 1].index.tolist()
len(top_platforms)
```

[42]:
```
119
```

[43]:
```
metadata = metadata[
    (metadata.platform.isna())
    | (
        (metadata.platform.isin(top_platforms))
        & (~metadata[metadata.platform.notna()].platform.str.contains("^mac"))
```

```
        & (
            ~metadata[metadata.platform.notna()].platform.isin(
                [
                    # insertar comentario cinico/sarcastico sobre estas opciones
                    "windows",
                    "win32",
                    "darwin",
                    "windows10",
                    "android",
                    "win32_amd64",
                    "operating system :: microsoft :: windows :: windows 10",
                    "win_amd64",
                    "operating system :: macos :: macos x",
                    "operating system :: microsoft :: windows",
                    "nt",
                    "osx",
                    "window10",
                    "win-amd64",
                    "windows only",
                    "win64",
                    "windows,",
                    "win",
                    "window",
                    "darwin-21.4.0-x86_64-i386-64bit",
                    "wind",
                    "win_x86_64",
                    "cel",
                    "a",
                    "windows 7",
                    "msvcp_dll_names:msvcp140.dll,msvcp140_1.dll",
                    "cygwin",
                    "win10",
                    "os independent (written in an interpreted language)",
                    "windows 10 x64",
                    "windows xp/2000/nt",
                    "windows 10",
                    "os x",
                    "ms windows",
                ]
            )
        )
    )
]
```

```
/tmp/ipykernel_604015/2015806324.py:1: UserWarning: Boolean Series key will be
reindexed to match DataFrame index.
  metadata = metadata[
```

[44]:
```
metadata = metadata[~metadata.filename.str.endswith(".exe")]
```

[45]:
```
metadata = metadata[~metadata.filename.str.endswith(".rpm")]
```

[46]:
```
metadata = metadata[~metadata.filename.str.contains("macos")]
```

[47]:
```
metadata = metadata[
    ~(
        metadata.filename.str.contains("win32.")
        | metadata.filename.str.contains("win_amd64.")
        | metadata.filename.str.contains("win-amd64.")
    )
]
```

Data on requires_dist

[48]:
```
# metadata = metadata[metadata.requires_dist.apply(len) > 0]
```

[49]:
```
display(
    f"A esta altura nos quedan {len(metadata)} releases de {metadata.name.
 ↪nunique()} paquetes"
)
```

```
'A esta altura nos quedan 2742303 releases de 275971 paquetes'
```

10.6.2 Parse requires_dist

[50]:
```
metadata["requires_dist_l"] = metadata.requires_dist.apply(lambda x: x.size)
```

[51]:
```
metadata = metadata.sort_values(by=["name", "requires_dist_l"], ascending=False)
```

[52]:
```
import re


def fix_pyver(l):
    if not any("python_version" in y for y in l):
        return l

    return [
        re.sub(
            r"(?P<PRELUDE>.*)python_version == (?P<MAJOR>\d)(?P<MINOR>\.\d)?(?
 ↪P<PATCH>\.\d)?",
            "\g<PRELUDE>python_version == '\g<MAJOR>\g<MINOR>\g<PATCH>'",
            x,
        )
        for x in l
    ]
```

```
metadata.requires_dist = metadata.requires_dist.progress_apply(fix_pyver)
```

```
  0%|          | 0/2742303 [00:01<?, ?it/s]
```

[53]:
```
@lru_cache(maxsize=512_000)
def parse_req(x):
    if x[-1] == ")":
        try:
            return Requirement(x)
        except:
            x = x.replace(" (", "").replace(")", "").replace(";", "")
            return Requirement(x)

    return Requirement(x)


@lru_cache(maxsize=32_000)
def parse_list(l):
    try:
        return [parse_req(r) for r in l]
    except InvalidRequirement:
        print(l)
        raise
```

[54]:
```
metadata.requires_dist = metadata.requires_dist.progress_apply(tuple).
 ↪progress_apply(
    parse_list
)
```

```
  0%|          | 0/2742303 [00:00<?, ?it/s]
```

```
  0%|          | 0/2742303 [00:00<?, ?it/s]
```

[55]:
```
def valid_version(v):
    try:
        Version(v)
        return True
    except:
        return False


print(
    f"{metadata.version.progress_apply(valid_version).mean():.2%} con versiones␣
 ↪validas"
)

metadata.version = metadata.version.progress_apply(valid_version)
```

```
  0%|          | 0/2742303 [00:00<?, ?it/s]
100.00% con versiones validas
  0%|          | 0/2742303 [00:00<?, ?it/s]
```

[56]:
```
metadata.head()
```

[56]:

	name	version	platform	requires_python	requires	provides	obsoletes	requires_dist	provides_dist	obsoletes_dist	requires_external	upload_time	filename	python_version	requires_dist_l
5355149	zzzzzTest2000	True	None	True	[]	[]	[]	[]	[]	[]	[]	2022-08-23 19:43:40.862210+00:00	zzzzztest2000-1.24.58-py3-none-any.whl	py3	0
5528628	zzzzzTest2	True	None	True	[]	[]	[]	[]	[]	[]	[]	2022-08-23 19:10:46.852963+00:00	zzzzztest2-1.24.58-py3-none-any.whl	py3	0
3324918	zzzzls-Spider	True	any	None	[]	[]	[]	[tqdm, requests, HeroSpider]	[]	[]	[]	2020-11-19 06:44:51.376565+00:00	zzzzls_Spider-1.2.5-py3-none-any.whl	py3	3
1342228	zzzzls-Spider	True	any	None	[]	[]	[]	[tqdm, requests]	[]	[]	[]	2020-11-19 07:09:13.308594+00:00	zzzzls_Spider-1.2.6-py3-none-any.whl	py3	2
1802452	zzzz	True	None	None	[]	[]	[]	[]	[]	[]	[]	2020-04-01 10:13:32.478193+00:00	zzzz-0.0.3-py3-none-any.whl	py3	0

11 Construir el grafo

Habiendo limpiado la data lo más posible vamos a construir el grafo de dependencia entre paquetes. Para ello, se utiliza `networkx` inicialmente. Por el tamaño del grafo, y limitaciones de la biblioteca, luego se utiliza `cugraph`.

11.1 Grafo dirigido de biblioteca a biblioteca

En este punto, se genera un grafo dirigido no pesado entre bibliotecas. Hay una arista e_{ij} que une los vertices $v_i \to v_j$ si i lista a j entre sus dependencias.

No hay ninguna noción de peso asociada.

[57]:
```
from packaging.utils import canonicalize_name
```

[58]:
```
import networkx as nx
```

[59]:
```
top_1p_downloads = set(
    [
        canonicalize_name(w)
        for w in most_downloaded.head(int(len(most_downloaded) * 0.01)).index.
 ↪tolist()
    ]
)
top_1p_requested = set(
    [
        canonicalize_name(w)
        for w in most_requested.head(int(len(most_requested) * 0.01)).index.
 ↪tolist()
    ]
)
```

[60]:
```
G = nx.DiGraph()
```

[61]:
```
for x in metadata.name.unique():
    x = canonicalize_name(x)
    G.add_node(
        x,
        from_metadata=True,
        top_1p_downloads=x in top_1p_downloads,
        top_1p_requested=x in top_1p_requested,
    )
```

[62]:
```
for lib, reqs in tqdm(metadata[["name", "requires_dist"]].
 ↪to_records(index=False)):
    lib = canonicalize_name(lib)
    for req in reqs:
        req_name = canonicalize_name(req.name)
        if not G.has_node(req_name):
            G.add_node(req_name, from_metadata=False)
        G.add_edge(lib, req_name, version=req.specifier)
```

```
  0%|          | 0/2742303 [00:00<?, ?it/s]
```

[63]:
```
isolated_nodes = set(n for n, deg in G.out_degree if deg == 0) & set(
    n for n, deg in G.in_degree if deg == 0
)
print(len(isolated_nodes))
```

```
75723
```

[64]:
```
G.number_of_nodes()
```

[64]:
```
286483
```

[65]:
```
G.remove_nodes_from(isolated_nodes)
```

[66]:
```
G.number_of_nodes()
```

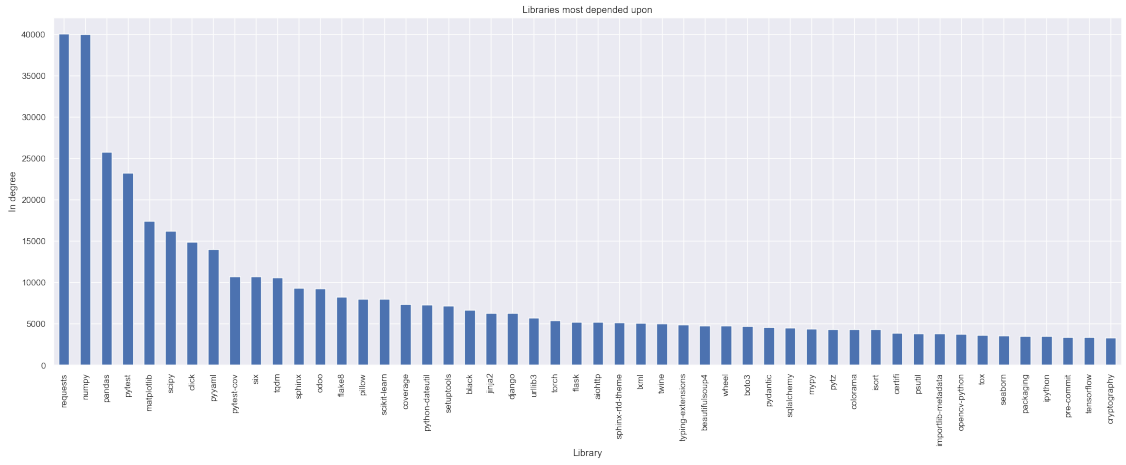
[66]:
```
210760
```

[67]:
```
G.number_of_edges()
```

[67]:
```
1125022
```

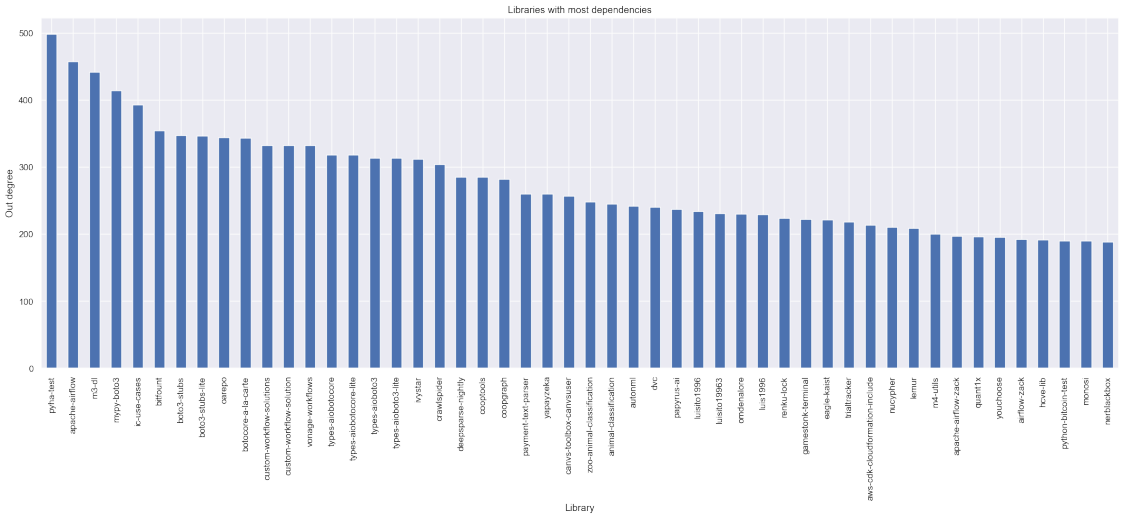
11.1.1 Analisis basico

[68]:
```
f, ax = plt.subplots(figsize=(24, 8), dpi=110)
nx.to_pandas_edgelist(G).target.value_counts().head(50).plot(kind="bar", ax=ax)
plt.ylabel("In degree")
plt.xlabel("Library")
plt.title("Libraries most depended upon")
plt.show()
```



[69]:
```
f, ax = plt.subplots(figsize=(24, 8), dpi=110)
nx.to_pandas_edgelist(G).source.value_counts().head(50).plot(kind="bar", ax=ax)
```

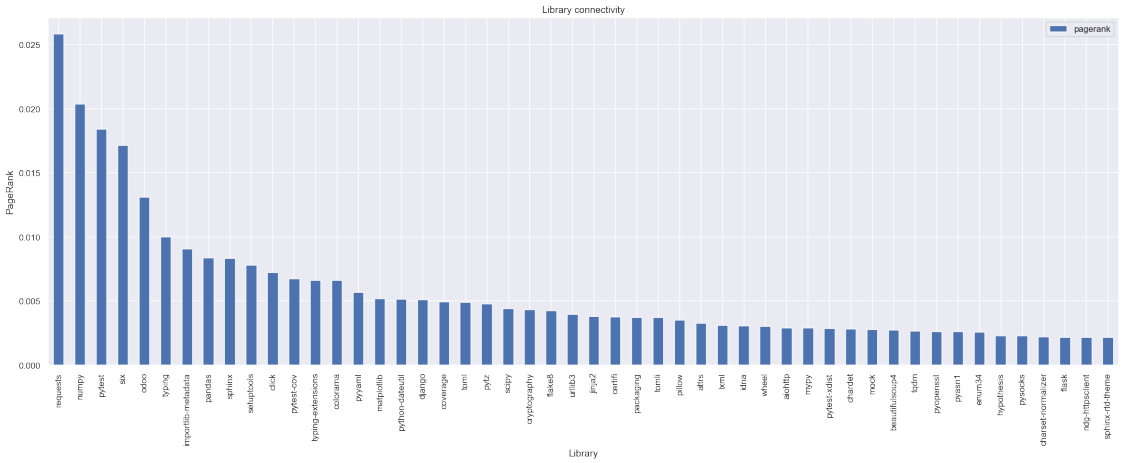
```
plt.ylabel("Out degree")
plt.xlabel("Library")
plt.title("Libraries with most dependencies")
plt.show()
```



En lo personal, puedo dar fe de `apache-airflow`.

[70]:
```
prank = nx.pagerank(G)
```

[71]:
```
prank_df = pd.DataFrame.from_dict(prank, columns=["pagerank"], orient="index",␣
↪).sort_values("pagerank", ascending=False).head(50)

f, ax = plt.subplots(figsize=(24, 8), dpi=110)
prank_df.plot(kind="bar", ax=ax)
plt.ylabel("PageRank")
plt.xlabel("Library")
plt.title("Library connectivity")
plt.show()
```



11.1.2 Centralidad como medida de importancia

En el escenario que nos concierne, estamos tratando de resolver el problema de marcar los paquetes criticos del ecosistema como tales para evitar problemas de supply chain. Esto puede tomar muchas formas, desde envenenamiento del paquete hasta su remoción absoluta. ¿Que pasaría si un día `pandas` desapareciera de PyPI?

Anteriormente se discutió que la cantidad de requests al índice sobre cada paquete es, intuitivamente, mejor métrica para denominar a un paquete crítico. Una segunda idea es que la centralidad es una métrica superadora. En particular, la `closeness centrality` nos indica que tan cercano es un paquete a todos los demás, por ende, el impacto que tiene en el ecosistema.

[72]:
```
import cugraph
import cupy
from scipy.stats import kendalltau
from sklearn.preprocessing import LabelEncoder

import cudf
```

[73]:
```
# Mas lento que nx
def cugraph_closeness_centrality(G):
    le = LabelEncoder()
    _ = le.fit(list(G.nodes))

    edges_df = nx.to_pandas_edgelist(G)
    edges_df["source_id"] = le.transform(edges_df.source)
    edges_df["target_id"] = le.transform(edges_df.target)
    edges_df.drop(columns=["version"]).to_parquet("edges_df.parquet",␣
 ↪index=False)

    # create a reversed graph view
    cuG = cugraph.from_pandas_edgelist(
```

```
        edges_df[["source_id", "target_id"]],
        source="target_id",
        destination="source_id",
        create_using=cugraph.DiGraph,
        renumber=False,
    )

    len_G = cuG.number_of_nodes()
    wf_improved = True
    nodes = cuG.nodes()

    closeness_arr = cupy.ndarray(shape=len_G, dtype=np.float64)

    for n in tqdm(nodes.values_host):
        sp = cugraph.sssp(cuG, source=n)
        mask = sp.predecessor != -1
        sp = sp[mask]
        totsp = sp.distance.sum()
        _closeness_centrality = 0.0
        if totsp > 0.0 and len_G > 1:
            _closeness_centrality = mask.sum() / totsp
            # normalize to number of nodes-1 in connected part
            if wf_improved:
                s = mask.sum() / (len_G - 1)
                _closeness_centrality *= s

        closeness_arr[n] = _closeness_centrality

    closeness_dict = dict(zip(le.inverse_transform(np.arange(len_G)),␣
↪closeness_arr))

    return closeness_dict
```

[74]:
```
# fast
# nx.centrality.eigenvector_centrality_numpy(G)
```

[75]:
```
%%time


def closeness_centrality2(G, wf_improved=True):
    # adapted from
    # https://networkx.org/documesntation/stable/_modules/networkx/algorithms/
↪centrality/closeness.html#closeness_centrality
    # to add a progressbar
    if G.is_directed():
        G = G.reverse()  # create a reversed graph view
```

```
    path_length = nx.single_source_shortest_path_length

    nodes = G.nodes

    closeness_dict = {}
    for n in tqdm(nodes):
        sp = path_length(G, n)
        totsp = sum(sp.values())
        len_G = len(G)
        _closeness_centrality = 0.0
        if totsp > 0.0 and len_G > 1:
            _closeness_centrality = (len(sp) - 1.0) / totsp
            # normalize to number of nodes-1 in connected part
            if wf_improved:
                s = (len(sp) - 1.0) / (len_G - 1)
                _closeness_centrality *= s
        closeness_dict[n] = _closeness_centrality

    return closeness_dict


closeness_dict2 = closeness_centrality2(G)
```

```
  0%|          | 0/210760 [00:00<?, ?it/s]
```

```
CPU times: user 14min 56s, sys: 1.98 s, total: 14min 58s
Wall time: 15min 1s
```

[76]:
```
centrality_df = (
    pd.Series(closeness_dict2.values(), index=closeness_dict2.keys())
    .sort_values(ascending=False)
    .to_frame(name="closeness_centrality")
)
centrality_df["closeness_centrality_rank"] = centrality_df.closeness_centrality.
 ↪rank(
    ascending=False
).astype(int)
```

[77]:
```
measures_df = libmetrics.merge(
    centrality_df, how="outer", left_index=True, right_index=True
)
measures_df["downloads_critical"] = measures_df.downloads_ranking <= 4324
measures_df["requests_critical"] = measures_df.requests_ranking <= 4324
```

Relacion con descargas

[78]:
```
mask = (
    measures_df.closeness_centrality_rank.notna()
    & measures_df.downloads_ranking.notna()
```

```
)

kendalltau(
    measures_df[mask].closeness_centrality_rank,
    measures_df[mask].downloads_ranking,
    nan_policy="omit",
)
```

[78]:
```
SignificanceResult(statistic=0.4369564256332471, pvalue=0.0)
```

Si bien la correlación es positiva, es bastante baja.

Relacion con requests

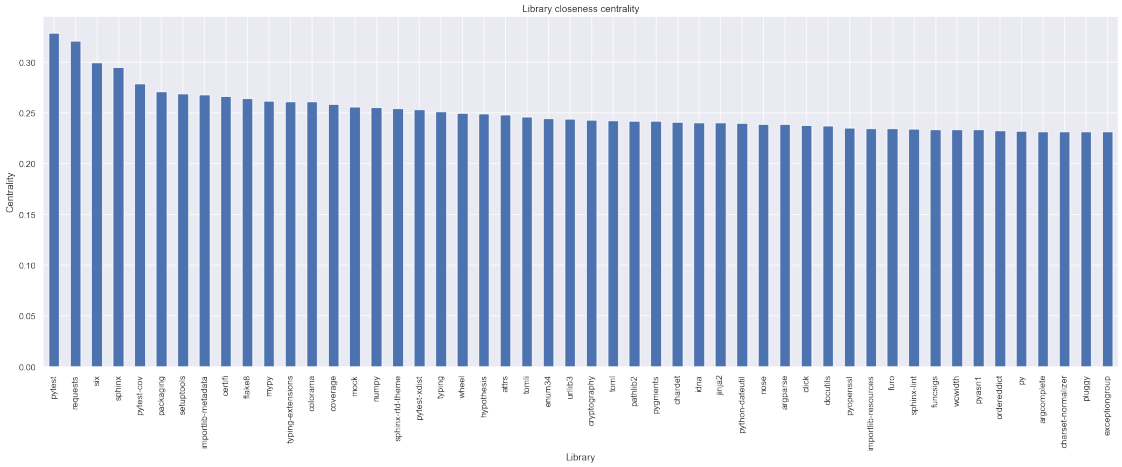
[79]:
```
mask = (
    measures_df.closeness_centrality_rank.notna() & measures_df.
 ↪requests_ranking.notna()
)

kendalltau(
    measures_df[mask].closeness_centrality_rank,
    measures_df[mask].requests_ranking,
    nan_policy="omit",
)
```

[79]:
```
SignificanceResult(statistic=0.4276573528213079, pvalue=0.0)
```

Igualmente que en el caso anterior, es una correlacion, si bien positiva, baja.

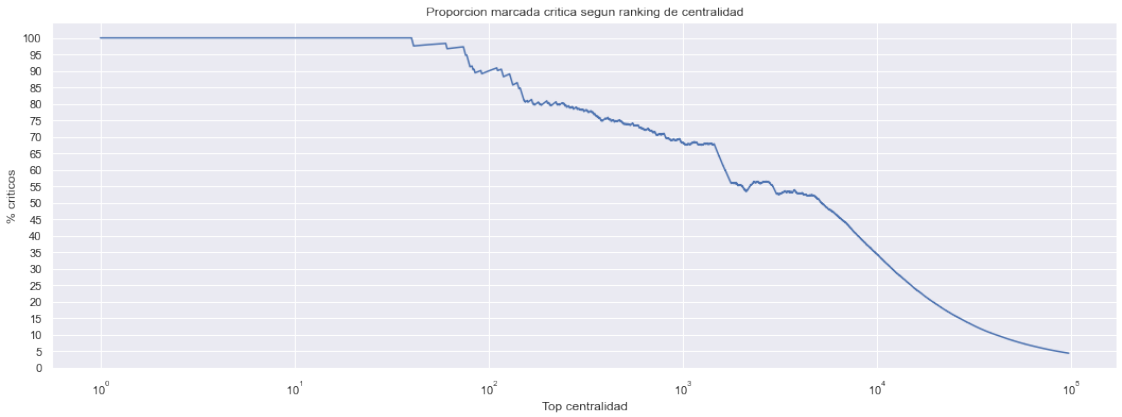
Visualizacion de mayor centralidad

[80]:
```
f, ax = plt.subplots(figsize=(24, 8), dpi=110)
measures_df.sort_values("closeness_centrality", ascending=False).
 ↪head(50)["closeness_centrality"].plot(kind="bar", ax=ax)
plt.ylabel("Centrality")
plt.xlabel("Library")
plt.title("Library closeness centrality")
plt.show()
```



Proporcion marcada como critica

[81]:
```
_nnm_df = measures_df.sort_values("closeness_centrality_rank", ascending=True).
↪dropna(
    subset=["closeness_centrality_rank", "downloads_ranking"]
)
x = np.arange(1, len(_nnm_df) + 1)
y = 100 * _nnm_df.downloads_critical.cumsum() / x

plt.figure(figsize=(18, 6))
plt.title("Proporcion marcada critica segun ranking de centralidad")
plt.ylabel("% criticos")
plt.xlabel("Top centralidad")
plt.plot(x, y)
plt.xscale("log")
plt.yticks(ticks=np.linspace(0, 100, num=21, dtype=int))
plt.show()
```



Es decir, muchas bibliotecas altamente centrales no serían marcadas como críticas si nos guiamos por la cantidad de descargas.

Esto parece evidencia suficiente para indicar que el criterio elegido podría ser complementado por esta métrica.

Veamos algunos ejemplos de bibliotecas en el top 100 de mayor centralidad que no hayan sido marcadas como criticas segun descargas.

[82]:
```
top100 = measures_df.sort_values("closeness_centrality_rank", ascending=True).
 ↪head(100)
top100[~top100.downloads_critical].sort_values("num_downloads",␣
 ↪ascending=False).head(
    25
)
```

[82]:

	num_downloads	num_requests	downloads_ranking	requests_ranking	change	closeness_centrality	closeness_centrality_rank	downloads_critical
pytest-black	21812.0	28006.0	4614.0	5623.0	-1009.0	0.218183	79.0	False
sphinx-inline-tabs	7869.0	9713.0	7023.0	8479.0	-1456.0	0.216632	84.0	False
win-inet-pton	5153.0	23742.0	8190.0	6002.0	2188.0	0.225889	61.0	False
docutils-stubs	2297.0	2509.0	11176.5	13840.5	-2664.0	0.214073	93.0	False
sphinx-lint	688.0	1276.0	17237.5	17557.5	-320.0	0.233875	41.0	False
pytest-checkdocs	308.0	310.0	22675.5	27713.0	-5037.5	0.218252		
pytest-enabler	286.0	286.0	23212.0	28382.0	-5170.0	0.217931		
rst-linker	32.0	150.0	43492.0	33989.0	9503.0	0.218332		
jaraco-packaging	18.0	127.0	50489.5	35572.5	14917.0	0.218332		
jaraco-test	13.0	14.0	54547.5	64222.0	-9674.5	0.216094		
jaraco-tidelift	NaN	NaN	NaN	NaN	NaN	0.216791		

```
pytest-checkdocs                        78.0              False
pytest-enabler                          80.0              False
rst-linker                              75.0              False
jaraco-packaging                        75.0              False
jaraco-test                             86.0              False
jaraco-tidelift                         83.0              False
```

```
                    requests_critical
pytest-black                        False
sphinx-inline-tabs                  False
win-inet-pton                       False
docutils-stubs                      False
sphinx-lint                         False
pytest-checkdocs                    False
pytest-enabler                      False
rst-linker                          False
jaraco-packaging                    False
jaraco-test                         False
jaraco-tidelift                     False
```

11.1.3 Comunidades en el grafo

Si bien con la centralidad tenemos una mirada holistica del ecosistema completo de python, es razonable pensar que un lenguaje de proposito general tenga comunidades en base a su uso. Por ejemplo: - Para scripting web: `requests` - Para aplicaciones web - El ecosistema `fastapi` - El ecosistema `django` - El ecosistema `flask` - Para machine learning - El stack de analisis de datos basado en `pandas` - ML tradicional con `scikit-learn` - Las bibliotecas construidas alrededor de `torch` - Las bibliotecas construidas alrededor de `tensorflow`

Vamos a hacer un analisis de comunidades, con el objetivo de entender si las bibliotecas mas importantes (medida por centralidad) dentro de la comunidad estan marcadas como criticas.

[83]:
```
#%%time
# communities = nx.community.louvain_communities(G, resolution=1, seed=117,␣
 ↪threshold=1e-2)
```

La implementación de `networkx` del algoritmo de detección de comunidades de Louvain es *extremadamente* lenta. Luego de más de una hora la ejecución no había terminado.
Por ello se decidió acelarar por GPU el calculo de comunidades. Esta sección del notebook solo se puede correr con una instalación de CUDA/ROCm de RAPIDS. Su instalación usando `pip` dista de ser sencilla, requiriendo prestar atención a cupy y requiere una GPU.

Un tradeoff asociado es que la implementación actual (v22.12.00) de `cugraph` solo soporta grafos no dirigidos.

[84]:
```
edges_df = nx.to_pandas_edgelist(G)

cuG = cugraph.from_pandas_edgelist(
    edges_df[["source", "target"]],
```

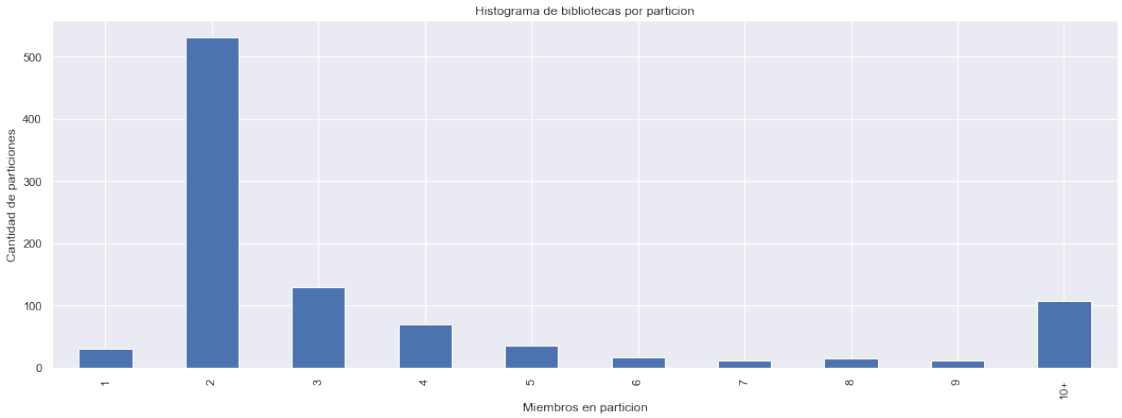
```
    source="source",
    destination="target",
    create_using=cugraph.Graph,
    renumber=True,
)
```

[85]:
```
%%time
parts, mod_score = cugraph.louvain(cuG, max_iter=500, resolution=0.85,)

print(f"Modularity was {mod_score}. Over {parts.partition.nunique()}␣
 ↪partitions")
```

```
Modularity was 0.5141443610191345. Over 957 partitions
CPU times: user 220 ms, sys: 198 ms, total: 418 ms
Wall time: 417 ms
```

[86]:
```
parts_df = measures_df.merge(
    parts.to_pandas().set_index("vertex"), left_index=True, right_index=True
)
```

[87]:
```
plt.figure(figsize=(18, 6))
cts = (
    parts_df.reset_index()
    .rename(columns={"index": "name"})
    .groupby("partition")
    .name.count()
)
cts_labels = cts.copy()
cts_labels[cts_labels >= 10] = 10

cts_labels.value_counts().sort_index().plot(kind="bar")
plt.ylabel("Cantidad de particiones")
plt.xlabel("Miembros en particion")
plt.title("Histograma de bibliotecas por particion")
plt.xticks(ticks=range(0, 10), labels=[*range(1, 10), f"10+"])
plt.show()
```



Voy a dejar de lado las particiones con solo 1 o 2 miembros, bajo la nocion de que no deben ser particularmente importantes.

[88]:
```
parts_df = parts_df.merge(
    cts.rename("partition_elements"), left_on="partition", right_index=True,␣
 ↪how="inner"
)
```

[89]:
```
parts_df["ranking_within_partition"] = (
    parts_df[parts_df.partition_elements > 2]
    .groupby("partition")
    .closeness_centrality.rank(method="dense", ascending=False)
)
```

[90]:
```
parts_df["top_1p_within_partition"] = (
    parts_df[parts_df.partition_elements > 2].ranking_within_partition
    <= parts_df[parts_df.partition_elements > 2].partition_elements * 0.01
)
```

[91]:
```
parts_df.head()
```

[91]:

	num_downloads	num_requests	downloads_ranking	requests_ranking	\
0	82.0	101.0	33405.0	37833.5	
0-0	NaN	7.0	NaN	76737.0	
0-0-1	6.0	6.0	66319.5	80018.0	
0-0-21	NaN	NaN	NaN	NaN	
0-8	NaN	NaN	NaN	NaN	

	change	closeness_centrality	closeness_centrality_rank	\
0	-4428.5	0.000005	49064.0	
0-0	NaN	0.000005	49064.0	
0-0-1	-13698.5	0.000005	49064.0	
0-0-21	NaN	0.000005	49064.0	

```
0-8          NaN              0.000005                    49064.0

        downloads_critical  requests_critical  partition  partition_elements  \
0                    False              False          3               36961
0-0                  False              False          3               36961
0-0-1                False              False          3               36961
0-0-21               False              False          3               36961
0-8                  False              False          3               36961

        ranking_within_partition top_1p_within_partition
0                          811.0                   False
0-0                        811.0                   False
0-0-1                      811.0                   False
0-0-21                     811.0                   False
0-8                        811.0                   False
```

[92]:
```
print(
    f"El {parts_df[(parts_df.partition_elements > 2) & (parts_df.
 ↪top_1p_within_partition)].downloads_critical.mean():.4%} de las bibliotecas␣
 ↪en el top 1% de sus respectivas comunidades estaría marcada como crítica en␣
 ↪base a las descargas"
)
```

```
El 58.8772% de las bibliotecas en el top 1% de sus respectivas comunidades
estaría marcada como crítica en base a las descargas
```

[93]:
```
partition_names = parts_df[parts_df.ranking_within_partition == 1].partition.
 ↪to_dict()
partition_names = {v: k for k, v in partition_names.items()}
```

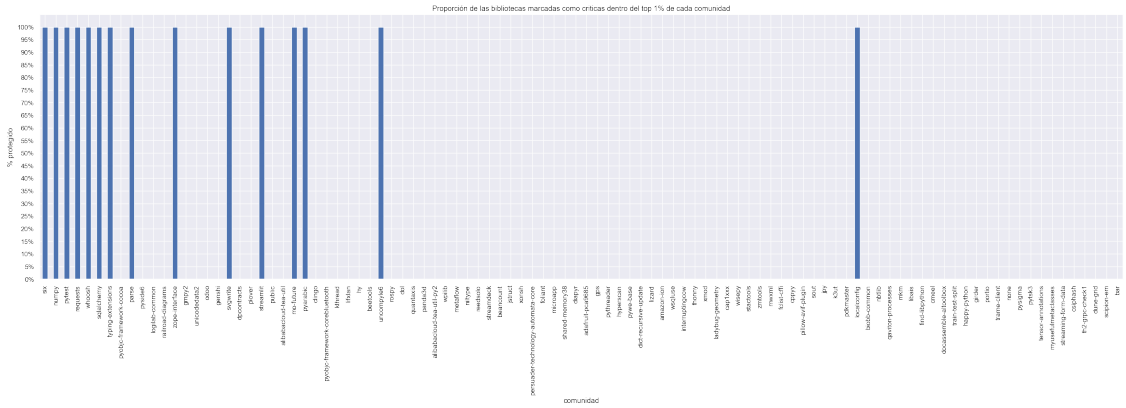
[94]:
```
parts_df["partition_name"] = parts_df.partition.map(partition_names)
```

[95]:
```
parts_df[parts_df.partition_name == "odoo"].sort_values(
    "closeness_centrality", ascending=False
).downloads_critical.mean()
```

[95]:
```
0.00020046106043900973
```

[96]:
```
cts2 = cts.copy()
cts2.index = cts2.index.map(partition_names)
order = cts2[cts2 > 2].index

plt.figure(figsize=(32, 8), dpi=110)
parts_df[
    (parts_df.partition_elements > 2) & (parts_df.ranking_within_partition == 1)
].groupby("partition_name").downloads_critical.mean()[order].head(100).
 ↪plot(kind="bar")
plt.yticks(
```

```
    ticks=np.linspace(0, 1, num=21, dtype=float),
    labels=[f"{x:.0%}" for x in np.linspace(0, 1, num=21, dtype=float)],
)
plt.ylabel("% protegido")
plt.xlabel("comunidad")
plt.title(
    "Proporción de las bibliotecas marcadas como criticas dentro del top 1% de␣
 ↪cada comunidad"
)
plt.show()
```



Algo que vemos ahora es que el modo propuesto de marcar bibliotecas como criticas en base a las descargas nada más expone mucho más a algunas comunidades que a otras. Un ejemplo muy claro es la comunidad del CRM y ERM Open Source Odoo.

11.1.4 Graficar comunidades

A modo explorativo, se grafican las bibliotecas dentro del ecosistema de las cuales dependen al menos 10 bibliotecas. Si graficara todo el grafo, en primer lugar no se vería nada por la densidad de puntos. En segundo lugar, Francia. En tercer lugar, no confío en el cooler de mi cpu.

El tamaño indica su centralidad, y el color la comunidad a la que pertenece. Pero habiendo tantas comunidades, es dificil armar un esquema de colores que las diferencie.

[97]:
```
degs = cuG.degrees()
degs = degs[degs.in_degree > 10]

small_cuG = cugraph.subgraph(cuG, degs.vertex)

pos = cugraph.force_atlas2(
    small_cuG, max_iter=1000, strong_gravity_mode=True, lin_log_mode=True
)
```

[98]:
```
centrality = cudf.DataFrame(
    {"centrality": centrality_df.closeness_centrality, "vertex": centrality_df.
 ↪index}
)
```

[99]:
```
pos_df = cudf.merge(parts, pos, left_on="vertex", right_on="vertex")

pos_df = cudf.merge(pos_df, centrality, on="vertex")

pos_df = cudf.merge(pos_df, cuG.degrees(), on="vertex")

pos_df = pos_df.to_pandas()
```

[100]:
```
pos_df["centrality_size"] = np.maximum(pos_df.centrality * 100, 1)
pos_df.centrality_size = pos_df.centrality_size.astype(int)
```

[101]:
```
pos_df = pos_df.sort_values(by="centrality", ascending=False)
pos_df.head()
```

[101]:

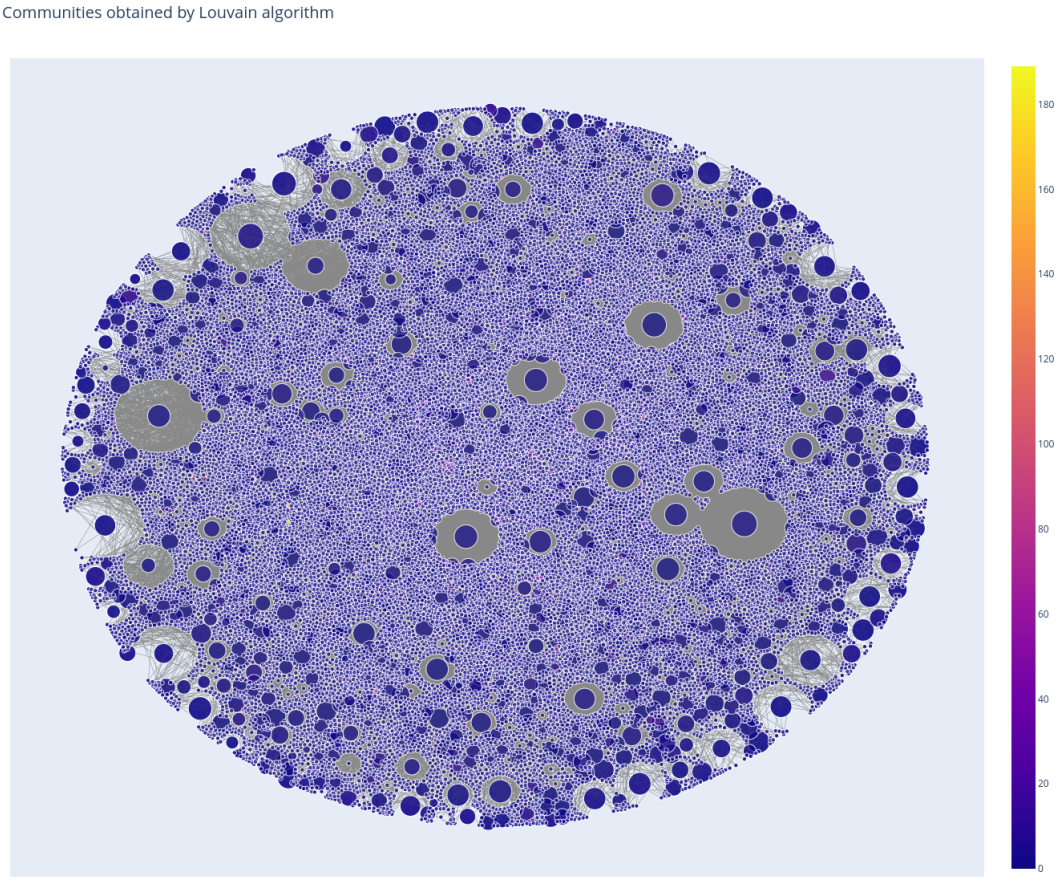
	partition	vertex	x	y	centrality	in_degree	\
8870	2	pytest	608.858826	-168.215759	0.328450	23263	
23175	3	requests	-598.177429	679.499451	0.320826	40021	
32356	0	six	388.365570	418.469055	0.299399	10720	
946	2	sphinx	-516.205383	834.266479	0.294859	9338	
27313	2	pytest-cov	-71.554665	-205.932404	0.278706	10733	

	out_degree	centrality_size
8870	23263	32
23175	40021	32
32356	10720	29
946	9338	29
27313	10733	27

[102]:
```
import plotly.express as px
import plotly.graph_objects as go
```

[103]:
```
edge_trace = px.scatter(pos_df, x="x", y="y", render_mode="webgl")
edge_trace.update_traces(line=dict(color="#888", width=0.5), mode="lines")

node_trace = px.scatter(
    pos_df,
    x="x",
    y="y",
    size="centrality_size",
    color="partition",
    color_discrete_sequence="g10",
    color_discrete_map="partition",
```

```
    hover_name="vertex",
    hover_data=["partition",],
    render_mode="webgl",
    size_max=pos_df.centrality_size.max(),
)


fig = go.Figure(
    data=node_trace.data + edge_trace.data + node_trace.data,
    layout=go.Layout(
        title="Communities obtained by Louvain algorithm",
        titlefont_size=20,
        showlegend=False,
        hovermode="closest",
        # margin=dict(b=20,l=5,r=5,t=40),
        xaxis=dict(showgrid=False, zeroline=False, showticklabels=False),
        yaxis=dict(showgrid=False, zeroline=False, showticklabels=False),
        autosize=False,
        height=1200,
        width=1200,
    ),
)
fig.show()
```



11.1.5 Puentes en el grafo

Sería muy raro que haya puentes en el grafo, dado que es mas similar a una red social.

[104]:
```
uG = G.to_undirected()
```

[105]:
```
nx.has_bridges(uG)
```

[105]: True

[106]:
```
print(f"Hay {sum(1 for _ in nx.bridges(uG))} puentes globales")
```

```
Hay 45562 puentes globales
```

Varios probablemente se deban a componentes conexas muy chicas.

[107]:
```
concoms2elems = 0

for concom in nx.connected_components(uG):
```

```
    if len(concom) == 2:
        concoms2elems += 1

print(f"De los cuales {concoms2elems} son de componentes conexas de 2␣
 ↪elementos")
```

```
De los cuales 531 son de componentes conexas de 2 elementos
```

11.1.6 Motifs

En base a la anterior sección anterior, esperaría que haya muchos motifs de cadenas de 3 y 4 elementos. Pero dificilmente grafos `k-completos`, para cualquier $k > 2$. Correr el siguiente codigo es inviable para todo el grafo (dado su tamaño), por lo que se hará solo para una comunidad. Será elegida por ser interesante y pequeña.

[108]:
```
parts_df.partition_name.value_counts().head(25)
```

[108]:
```
numpy                        51410
six                          43211
requests                     36961
pytest                       27248
odoo                          9977
whoosh                        9811
sqlalchemy                    7531
typing-extensions             6563
pyobjc-framework-cocoa        4697
dpcontracts                   1506
parse                         1361
zope-interface                1158
alibabacloud-tea-util          664
alibabacloud-tea-util-py2      622
logilab-common                 542
pyside6                        508
genshi                         481
railroad-diagrams              421
interruptingcow                384
streamlit                      337
gmpy2                          171
public                         157
kthread                        120
svgwrite                       112
plover                          99
Name: partition_name, dtype: int64
```

[109]:
```
streamlit_community = parts_df[
    (parts_df.partition_name == "streamlit")
].index.tolist()
```

```
subG = nx.subgraph(uG, streamlit_community)
print(f"{subG.number_of_nodes()=}")
print(f"{nx.is_connected(subG)=}")
```

```
subG.number_of_nodes()=337
nx.is_connected(subG)=True
```

Para encontrar los subgrafos se utiliza una versión paralelizable de ESU y chequeos por isomorfismo.

[110]:
```
import itertools
import multiprocessing as mp
import time
from collections import defaultdict

import networkx as nx
from progressbar import ProgressBar
from scipy.special import comb as combinatorial
from tqdm import tqdm as text_tqdm
from tqdm.notebook import tqdm

SENTINEL = "S"
PB_SENTINEL = None


def n_excl(G, w, v_subgraph):
    nv = nx.descendants_at_distance(G, w, 1)
    nvp = set()
    for z in v_subgraph:
        nvp |= nx.descendants_at_distance(G, z, 1)
    return nv - nvp


def extend_subgraph(G, v_subgraph, v_extension, v, k, k_subgraphs):
    if len(v_subgraph) == k:
        k_subgraphs.append(v_subgraph)
        return
    while len(v_extension) > 0:
        w = v_extension.pop()
        v_extension_ = v_extension | set(u for u in n_excl(G, w, v_subgraph) if␣
 ↪u > v)
        extend_subgraph(G, v_subgraph | {w}, v_extension_, v, k, k_subgraphs)


def get_subgraphs(G, k, nbunch=None):
    for v in tqdm(G.nodes):
        if nbunch is not None and v not in nbunch:
            continue
        v_extension = set(u for u in G.neighbors(v) if u > v)
```

```
        k_subgraphs = []
        extend_subgraph(G, {v}, v_extension, v, k, k_subgraphs)
        for _r in k_subgraphs:
            yield _r


def generate_motifs(n: int = 5):
    if n >= 6:
        warning.warning("You better make some coffee while this runs")

    nonisomorphs = defaultdict(list)

    G0 = nx.Graph()
    G0.add_edge(1, 2)

    nonisomorphs[G0.number_of_nodes()].append(G0)

    for i in range(3, n + 1):
        previous_nonisomorphs = nonisomorphs[i - 1]
        check_isomorphs = defaultdict(list)

        for graph in previous_nonisomorphs:
            possible_edges_list = list(
                itertools.chain.from_iterable(
                    [[*itertools.combinations(graph.nodes(), r)] for r in␣
 ↪range(1, i)]
                )
            )
            for new_edges in possible_edges_list:
                H = graph.copy()
                new_edges = list(itertools.product([i], new_edges))
                H.add_edges_from(new_edges)
                possible_isomorphs = check_isomorphs[H.number_of_edges()]
                is_isomorph = any(
                    nx.isomorphism.is_isomorphic(pi, H) for pi in␣
 ↪possible_isomorphs
                )
                if not is_isomorph:
                    check_isomorphs[H.number_of_edges()].append(H)
        all_non_isomorph = list(itertools.chain.from_iterable(check_isomorphs.
 ↪values()))
        nonisomorphs[i].extend(all_non_isomorph)

    return list(itertools.chain.from_iterable(nonisomorphs.values()))


def pbar_update(q):
```

```
    pbar = text_tqdm()
    for item in iter(q.get, PB_SENTINEL):
        pbar.update(item)


class PermProducer(mp.Process):
    def __init__(self, H, NM, out_q, pb_q, batch_size=256, anchor=0,␣
 ↪nproducers=1):
        super().__init__()
        self.H = H
        self.NM = NM
        self.out_q = out_q
        self.pb_q = pb_q
        self.anchor = anchor
        self.nproducers = nproducers
        self.batch_size = batch_size

    def run(self):
        nbunch = {x for idx, x in enumerate(self.H.nodes) if idx % self.
 ↪nproducers == self.anchor}

        batch = []
        seen = 0

        for r in range(2, self.NM + 1):
            for sg in get_subgraphs(self.H, r, nbunch=nbunch):
                    batch.append(sg)
                    if len(batch) == self.batch_size:
                        self.out_q.put(batch)
                        self.pb_q.put(len(batch))
                        batch = []

        self.out_q.put(batch)
        self.pb_q.put(len(batch))


class CombChecker(mp.Process):
    def __init__(self, H, in_q, out_q, nm):
        super().__init__()
        self.H = H
        self.in_q = in_q
        self.out_q = out_q
        self.motifs = generate_motifs(nm)

    def run(self):
        balanced_motifs = defaultdict(list)
        for motif in self.motifs:
```

```
            balanced_motifs[(motif.number_of_nodes(), motif.number_of_edges())].
 ↪append(
                motif
            )

        motif_count = {
            nx.weisfeiler_lehman_graph_hash(motif): 0 for motif in self.motifs
        }
        motif_lookup = {
            nx.weisfeiler_lehman_graph_hash(motif): motif for motif in self.
 ↪motifs
        }

        for batch in iter(self.in_q.get, SENTINEL):
            # print(f"{time.time()} Got batch")
            for sg_nodes in batch:
                sg = self.H.subgraph(sg_nodes)

                for possible_motif in balanced_motifs[
                    (sg.number_of_nodes(), sg.number_of_edges())
                ]:
                    if nx.is_isomorphic(possible_motif, sg):
                        r = nx.weisfeiler_lehman_graph_hash(possible_motif)
                        motif_count[r] += 1
        res = {
            motif_lookup[motif_hash]: count for motif_hash, count in␣
 ↪motif_count.items()
        }

        self.out_q.put(res)


def count_motifs(H, nm=4, n_checkers=16, qsize=1_000, batch_size=256,␣
 ↪n_producers=1):
    q_prod = mp.Queue(maxsize=n_checkers * qsize)
    q_bar = mp.Queue()

    bar_process = mp.Process(target=pbar_update, args=(q_bar,))
    bar_process.start()

    producers = [PermProducer(
        H, nm, out_q=q_prod, pb_q=q_bar, batch_size=batch_size,␣
 ↪nproducers=n_producers, anchor=i
    ) for i in range(n_producers)]

    q_check = mp.Queue()
```

```
    checkers = [CombChecker(H, q_prod, q_check, nm) for _ in range(n_checkers)]

    [p.start() for p in tqdm(producers, desc="Start producers")]
    [c.start() for c in tqdm(checkers, desc="Start checkers")]


    [p.join() for p in tqdm(producers, desc="Join producers")]
    [
        q_prod.put(SENTINEL)
        for _ in tqdm(checkers, desc="Putting sentinels for checkers")
    ]

    print("Fetch results")
    all_results = []

    for idx in tqdm(range(n_checkers), desc="Results"):
        r = q_check.get()
        all_results.append(r)

    [c.join() for c in tqdm(checkers, desc="Join checkers")]

    q_bar.put(PB_SENTINEL)
    print("Join progress bar")
    bar_process.join()

    res = {nx.weisfeiler_lehman_graph_hash(motif): 0 for motif in␣
↪generate_motifs(nm)}
    rev_m = {
        nx.weisfeiler_lehman_graph_hash(motif): motif for motif in␣
↪generate_motifs(nm)
    }

    for ree in all_results:
        for k, v in ree.items():
            hk = nx.weisfeiler_lehman_graph_hash(k)
            res[hk] += v

    res = {rev_m[k]: v for k, v in res.items()}
    return res
```

[112]:
```
motif_count = count_motifs(subG, nm=4, n_checkers=12, qsize=10, batch_size=512,␣
↪n_producers=2)
```

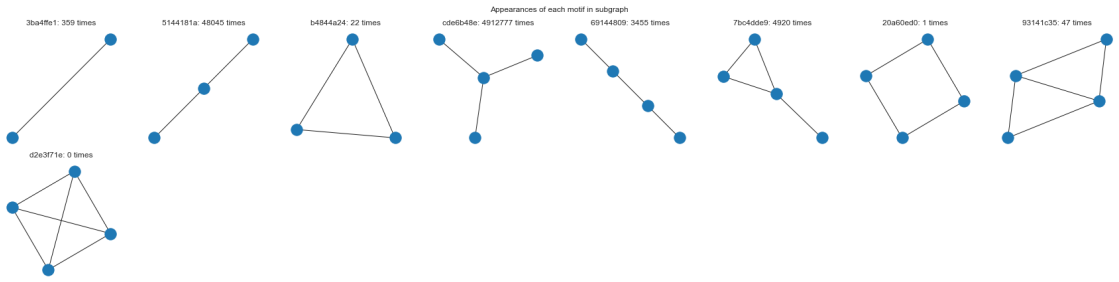
```
0it [00:00, ?it/s]

Start producers:   0%|          | 0/2 [00:00<?, ?it/s]

512it [00:00, 899.00it/s]
```

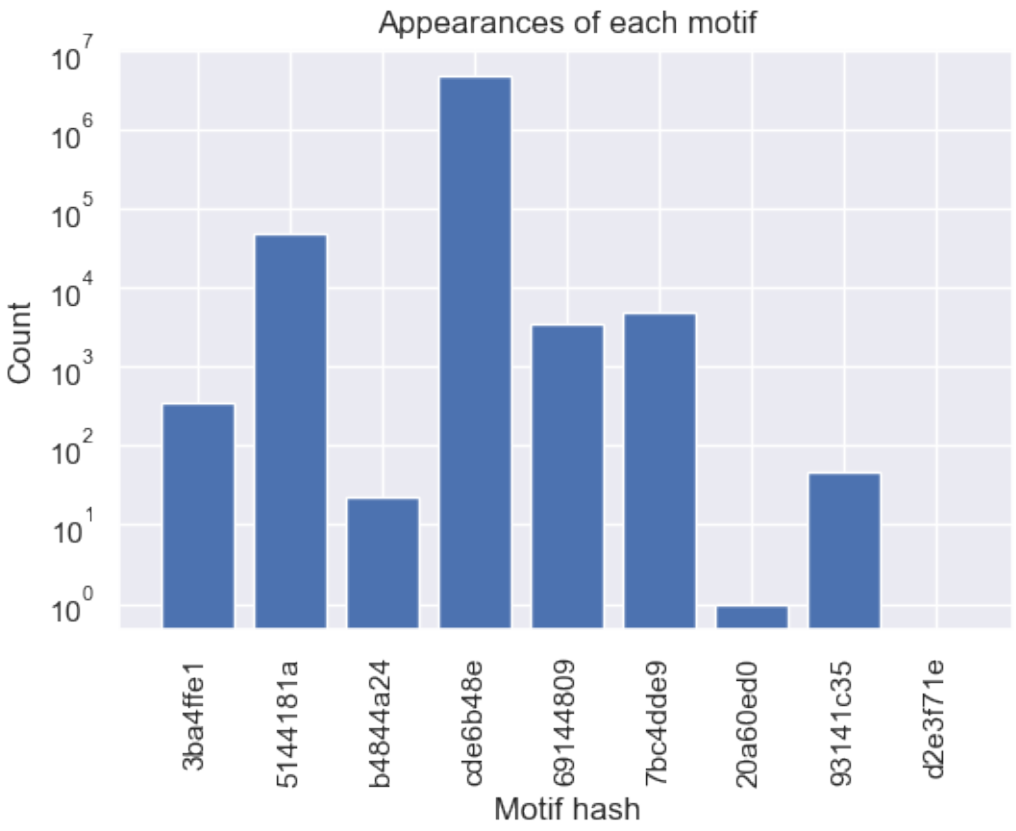
```
Start checkers:   0%|          | 0/12 [00:00<?, ?it/s]
16384it [00:04, 4244.46it/s]
Join producers:   0%|          | 0/2 [00:00<?, ?it/s]
4947145it [22:27, 3141.99it/s]
Putting sentinels for checkers:   0%|          | 0/12 [00:00<?, ?it/s]
Fetch results
Results:   0%|          | 0/12 [00:00<?, ?it/s]
Join checkers:   0%|          | 0/12 [00:00<?, ?it/s]
4969626it [22:32, 3673.21it/s]
Join progress bar
```

[119]:
```python
import math

from matplotlib import pyplot as plt


def plot_motif_count(motif_count, log=False):
    ncols = 8
    nrows = math.ceil(len(motif_count) / ncols)

    fig, axes = plt.subplots(
        nrows=nrows,
        ncols=ncols,
        figsize=(ncols * 3, nrows * 3),
        subplot_kw=dict(box_aspect=1),
        sharex=False,
        sharey=False,
    )

    for ax, (motif, seen) in zip(axes.flat, motif_count.items()):
        nx.draw(motif, ax=ax)
        ax.set_title(f"{nx.weisfeiler_lehman_graph_hash(motif)[:8]}: {seen}␣
 ↪times")

    for i in range(len(motif_count) - axes.size, 0):
        fig.delaxes(axes.flat[i])

    fig.suptitle("Appearances of each motif in subgraph")

    plt.tight_layout()
    plt.show()

    plt.figure(figsize=(6, 4), dpi=100)
```

```
    plt.bar(
        [nx.weisfeiler_lehman_graph_hash(x)[:8] for x in motif_count.keys()],
        motif_count.values(),
    )
    if log:
        plt.yscale("log")
    plt.title("Appearances of each motif")
    plt.xticks(rotation=90)
    plt.xlabel("Motif hash")
    plt.ylabel("Count")
    plt.show()
```

[120]:
```
plot_motif_count(motif_count, log=True)
```





Procedamos a calcular el significant profile de la red.

[114]:
```
degree_seq = [x for _, x in subG.degree()]

cms = defaultdict(list)

for _ in range(15):
    CM = nx.configuration_model(degree_seq, create_using=nx.Graph)
    cm = count_motifs(CM, nm=4, n_checkers=12, qsize=2, batch_size=128)
    for k, v in cm.items():
        cms[nx.weisfeiler_lehman_graph_hash(k)].append(v)
```

```
0it [00:00, ?it/s]

Start producers:   0%|          | 0/1 [00:00<?, ?it/s]

Start checkers:   0%|          | 0/12 [00:00<?, ?it/s]

37248it [00:03, 11036.57it/s]

Join producers:   0%|          | 0/1 [00:00<?, ?it/s]

599808it [00:31, 41065.54it/s]
```

```
Putting sentinels for checkers:   0%|          | 0/12 [00:00<?, ?it/s]
Fetch results
Results:   0%|          | 0/12 [00:00<?, ?it/s]
Join checkers:   0%|          | 0/12 [00:00<?, ?it/s]
603135it [00:32, 18505.93it/s]
Join progress bar
0it [00:00, ?it/s]
Start producers:   0%|          | 0/1 [00:00<?, ?it/s]
Start checkers:   0%|          | 0/12 [00:00<?, ?it/s]
31744it [00:03, 20772.92it/s]
Join producers:   0%|          | 0/1 [00:00<?, ?it/s]
942848it [00:55, 41447.82it/s]
Putting sentinels for checkers:   0%|          | 0/12 [00:00<?, ?it/s]
Fetch results
Results:   0%|          | 0/12 [00:00<?, ?it/s]
Join checkers:   0%|          | 0/12 [00:00<?, ?it/s]
946512it [00:56, 16636.98it/s]
Join progress bar
0it [00:00, ?it/s]
Start producers:   0%|          | 0/1 [00:00<?, ?it/s]
Start checkers:   0%|          | 0/12 [00:00<?, ?it/s]
24320it [00:04, 11375.22it/s]
Join producers:   0%|          | 0/1 [00:00<?, ?it/s]
577664it [00:30, 42457.54it/s]
Putting sentinels for checkers:   0%|          | 0/12 [00:00<?, ?it/s]
Fetch results
Results:   0%|          | 0/12 [00:00<?, ?it/s]
Join checkers:   0%|          | 0/12 [00:00<?, ?it/s]
579967it [00:31, 18153.60it/s]
Join progress bar
0it [00:00, ?it/s]
Start producers:   0%|          | 0/1 [00:00<?, ?it/s]
```

```
Start checkers:   0%|          | 0/12 [00:00<?, ?it/s]
47232it [00:03, 23358.30it/s]
Join producers:   0%|          | 0/1 [00:00<?, ?it/s]
622720it [00:33, 43617.56it/s]
Putting sentinels for checkers:   0%|          | 0/12 [00:00<?, ?it/s]
Fetch results
Results:   0%|          | 0/12 [00:00<?, ?it/s]
Join checkers:   0%|          | 0/12 [00:00<?, ?it/s]
626915it [00:34, 18198.48it/s]
Join progress bar
0it [00:00, ?it/s]
Start producers:   0%|          | 0/1 [00:00<?, ?it/s]
Start checkers:   0%|          | 0/12 [00:00<?, ?it/s]
37888it [00:03, 15467.67it/s]
Join producers:   0%|          | 0/1 [00:00<?, ?it/s]
751488it [00:41, 39359.16it/s]
Putting sentinels for checkers:   0%|          | 0/12 [00:00<?, ?it/s]
Fetch results
Results:   0%|          | 0/12 [00:00<?, ?it/s]
Join checkers:   0%|          | 0/12 [00:00<?, ?it/s]
755306it [00:42, 17882.66it/s]
Join progress bar
0it [00:00, ?it/s]
Start producers:   0%|          | 0/1 [00:00<?, ?it/s]
Start checkers:   0%|          | 0/12 [00:00<?, ?it/s]
37632it [00:03, 15620.42it/s]
Join producers:   0%|          | 0/1 [00:00<?, ?it/s]
766720it [00:43, 43052.39it/s]
Putting sentinels for checkers:   0%|          | 0/12 [00:00<?, ?it/s]
Fetch results
Results:   0%|          | 0/12 [00:00<?, ?it/s]
Join checkers:   0%|          | 0/12 [00:00<?, ?it/s]
```

```
767583it [00:44, 17372.65it/s]
Join progress bar
0it [00:00, ?it/s]
Start producers:   0%|          | 0/1 [00:00<?, ?it/s]
Start checkers:   0%|          | 0/12 [00:00<?, ?it/s]
35456it [00:03, 12098.39it/s]
Join producers:   0%|          | 0/1 [00:00<?, ?it/s]
556416it [00:29, 43826.53it/s]
Putting sentinels for checkers:   0%|          | 0/12 [00:00<?, ?it/s]
Fetch results
Results:   0%|          | 0/12 [00:00<?, ?it/s]
Join checkers:   0%|          | 0/12 [00:00<?, ?it/s]
556829it [00:30, 18433.86it/s]
Join progress bar
0it [00:00, ?it/s]
Start producers:   0%|          | 0/1 [00:00<?, ?it/s]
Start checkers:   0%|          | 0/12 [00:00<?, ?it/s]
36608it [00:03, 21121.72it/s]
Join producers:   0%|          | 0/1 [00:00<?, ?it/s]
672000it [00:37, 43666.75it/s]
Putting sentinels for checkers:   0%|          | 0/12 [00:00<?, ?it/s]
Fetch results
Results:   0%|          | 0/12 [00:00<?, ?it/s]
Join checkers:   0%|          | 0/12 [00:00<?, ?it/s]
675410it [00:38, 17670.43it/s]
Join progress bar
0it [00:00, ?it/s]
Start producers:   0%|          | 0/1 [00:00<?, ?it/s]
Start checkers:   0%|          | 0/12 [00:00<?, ?it/s]
42880it [00:03, 12665.73it/s]
Join producers:   0%|          | 0/1 [00:00<?, ?it/s]
491008it [00:25, 42483.14it/s]
```

```
Putting sentinels for checkers:   0%|          | 0/12 [00:00<?, ?it/s]
Fetch results
Results:   0%|          | 0/12 [00:00<?, ?it/s]
Join checkers:   0%|          | 0/12 [00:00<?, ?it/s]
493821it [00:26, 18870.19it/s]
Join progress bar
0it [00:00, ?it/s]
Start producers:   0%|          | 0/1 [00:00<?, ?it/s]
Start checkers:   0%|          | 0/12 [00:00<?, ?it/s]
31616it [00:04, 10078.26it/s]
Join producers:   0%|          | 0/1 [00:00<?, ?it/s]
825472it [00:47, 41342.27it/s]
Putting sentinels for checkers:   0%|          | 0/12 [00:00<?, ?it/s]
Fetch results
Results:   0%|          | 0/12 [00:00<?, ?it/s]
Join checkers:   0%|          | 0/12 [00:00<?, ?it/s]
825747it [00:48, 17009.09it/s]
Join progress bar
0it [00:00, ?it/s]
Start producers:   0%|          | 0/1 [00:00<?, ?it/s]
Start checkers:   0%|          | 0/12 [00:00<?, ?it/s]
32640it [00:03, 11460.18it/s]
Join producers:   0%|          | 0/1 [00:00<?, ?it/s]
811392it [00:46, 39463.74it/s]
Putting sentinels for checkers:   0%|          | 0/12 [00:00<?, ?it/s]
Fetch results
Results:   0%|          | 0/12 [00:00<?, ?it/s]
Join checkers:   0%|          | 0/12 [00:00<?, ?it/s]
811465it [00:47, 17088.49it/s]
Join progress bar
0it [00:00, ?it/s]
Start producers:   0%|          | 0/1 [00:00<?, ?it/s]
```

```
Start checkers:   0%|          | 0/12 [00:00<?, ?it/s]
23552it [00:03, 11098.75it/s]
Join producers:   0%|          | 0/1 [00:00<?, ?it/s]
544896it [00:28, 43127.47it/s]
Putting sentinels for checkers:   0%|          | 0/12 [00:00<?, ?it/s]
Fetch results
Results:   0%|          | 0/12 [00:00<?, ?it/s]
Join checkers:   0%|          | 0/12 [00:00<?, ?it/s]
545532it [00:29, 18333.27it/s]
Join progress bar
0it [00:00, ?it/s]
Start producers:   0%|          | 0/1 [00:00<?, ?it/s]
Start checkers:   0%|          | 0/12 [00:00<?, ?it/s]
38784it [00:04, 14957.20it/s]
Join producers:   0%|          | 0/1 [00:00<?, ?it/s]
635264it [00:34, 42323.31it/s]
Putting sentinels for checkers:   0%|          | 0/12 [00:00<?, ?it/s]
Fetch results
Results:   0%|          | 0/12 [00:00<?, ?it/s]
Join checkers:   0%|          | 0/12 [00:00<?, ?it/s]
638126it [00:35, 18041.83it/s]
Join progress bar
0it [00:00, ?it/s]
Start producers:   0%|          | 0/1 [00:00<?, ?it/s]
Start checkers:   0%|          | 0/12 [00:00<?, ?it/s]
46080it [00:03, 24830.34it/s]
Join producers:   0%|          | 0/1 [00:00<?, ?it/s]
460288it [00:22, 38465.67it/s]
Putting sentinels for checkers:   0%|          | 0/12 [00:00<?, ?it/s]
Fetch results
Results:   0%|          | 0/12 [00:00<?, ?it/s]
Join checkers:   0%|          | 0/12 [00:00<?, ?it/s]
```

```
461993it [00:23, 19398.86it/s]

Join progress bar

0it [00:00, ?it/s]

Start producers:   0%|          | 0/1 [00:00<?, ?it/s]

Start checkers:   0%|          | 0/12 [00:00<?, ?it/s]

37760it [00:03, 15472.16it/s]

Join producers:   0%|          | 0/1 [00:00<?, ?it/s]

752384it [00:42, 42260.57it/s]

Putting sentinels for checkers:   0%|          | 0/12 [00:00<?, ?it/s]

Fetch results

Results:   0%|          | 0/12 [00:00<?, ?it/s]

Join checkers:   0%|          | 0/12 [00:00<?, ?it/s]

754281it [00:43, 17343.33it/s]

Join progress bar
```

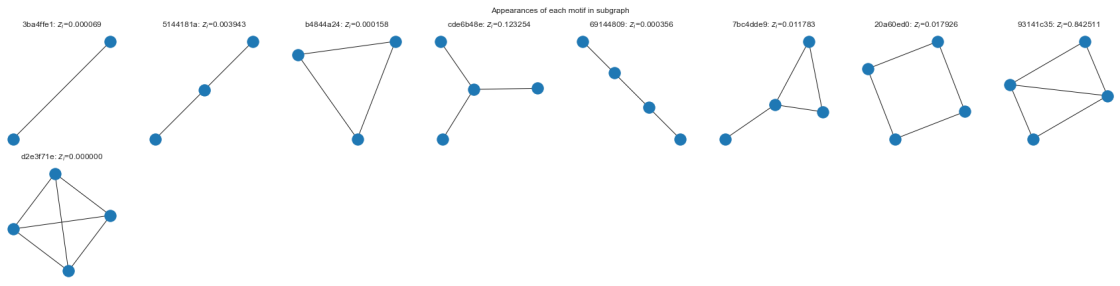
[127]:
```
_motif_count = {nx.weisfeiler_lehman_graph_hash(k):v for k, v in motif_count.
 ↪items()}
motifs = generate_motifs(4)
motifs_hashes = [nx.weisfeiler_lehman_graph_hash(x) for x in motifs]
```

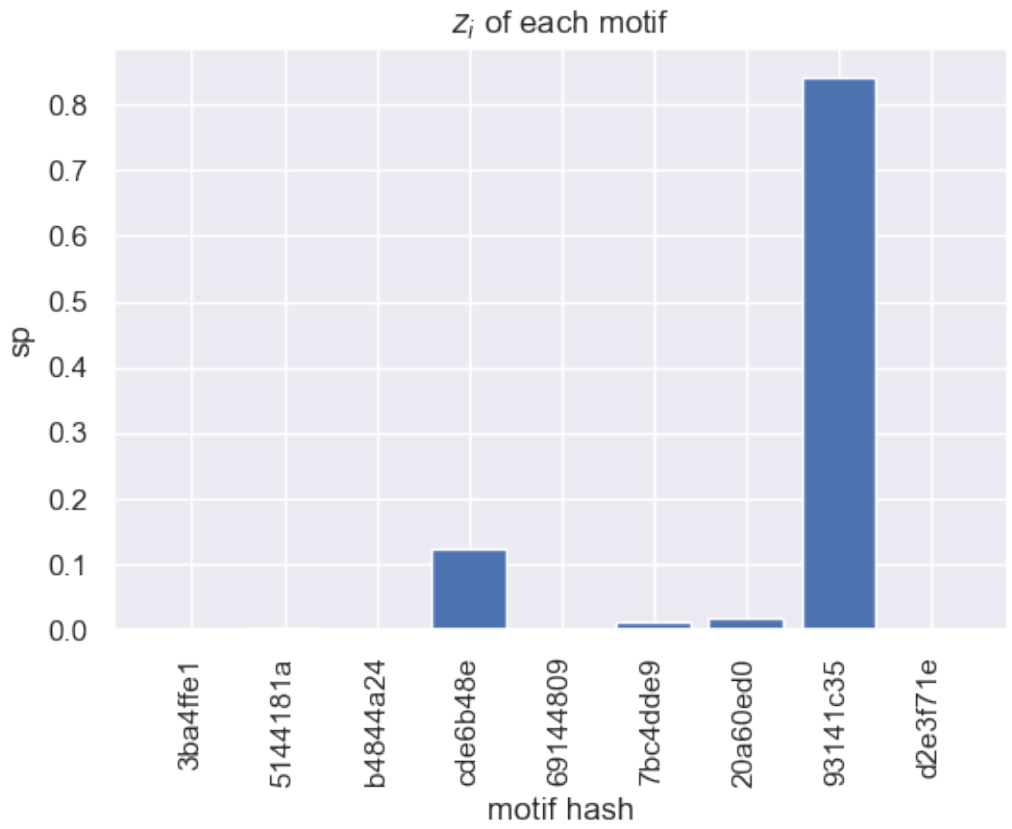
[129]:
```
eps = 1e-4

z = np.array(
    [
        np.divide(_motif_count[motif] - np.mean(cms.get(motif)), np.std(cms.
 ↪get(motif)) + eps)
        for motif in motifs_hashes
    ]
)
sp = z / z.sum()
sp
```

[129]:
```
array([6.91868264e-05, 3.94320027e-03, 1.57556144e-04, 1.23253791e-01,
       3.56409083e-04, 1.17834629e-02, 1.79257582e-02, 8.42510636e-01,
       0.00000000e+00])
```

[132]:
```
def plot_sp(motifs, sp):
    ncols = 8
    nrows = math.ceil(len(motifs) / ncols)

    fig, axes = plt.subplots(
```

```
        nrows=nrows,
        ncols=ncols,
        figsize=(ncols * 3, nrows * 3),
        subplot_kw=dict(box_aspect=1),
        sharex=False,
        sharey=False,
    )

    for ax, motif, zi in zip(axes.flat, motifs, sp):
        nx.draw(motif, ax=ax)
        ax.set_title(f"{nx.weisfeiler_lehman_graph_hash(motif)[:8]}: $z_i$={zi:.
↪6f}")

    for i in range(len(motif_count) - axes.size, 0):
        fig.delaxes(axes.flat[i])

    fig.suptitle("Appearances of each motif in subgraph")

    plt.tight_layout()
    plt.show()

    plt.figure(figsize=(6, 4), dpi=100)
    plt.bar([nx.weisfeiler_lehman_graph_hash(x)[:8] for x in motif_count.
↪keys()], sp)
    plt.title("$z_i$ of each motif")
    plt.xticks(rotation=90)
    plt.xlabel("motif hash")
    plt.ylabel("sp")
    plt.show()


plot_sp(motifs, sp)
```





No hay ninguna sorpresa en que el grafo $k-4$ no aparezca. Si quizás el grafo con hash `9314...` Pero sería el caso de una libreria de la cual dependen otras tres y una de ellas está altamente acoplada al resto. El grafo con hash `cde6...` es quizás el mas esperable que sea representativo: es una biblioteca de la cual dependen varias.

12 Conclusiones

Al inicio de este trabajo el planteo era si la cantidad de descargas era el mejor proxy para evaluar los paquetes criticos. La conclusión sería que este enfoque debería verse complementado por criterios adicionales que ayuden en el objetivo de evitar supply chain attacks. Principalmente, hay muchas comunidades de desarrollo que quedan particularmente expuestas a los mismos al no mandar 2FA sobre sus paquetes mas centrales.

Basarse en la representación como grafo del ecosistema nos permite encontrar estas comunidades y también los paquetes más relevantes disponibles en PyPI. En la siguiente sección plateamos mejoras que podrían dar mejores nociones de la importancia de cada dependencia.

13 Posibles mejoras

- Aplicar `criticality score` y comparar con las métricas obtenidas

- Trabajar con un periodo mas grande de tiempo
  - El costo se puede cubrir con los 300 USD de crédito de GCP, pero estimo que no tendría gran impacto
- Generar un nodo para cada versión de cada paquete y conectar cada nodo con todos aquellos compatibles acorde a la especificación de versiones
  - Este grafo sería inmanejable para muchos análisis
  - Las aristas entre versiones deberían estar diferenciadas de las aristas entre paquetes
- Introducir como noción de peso en una arista e_{ij} las descargas de i, bajo la noción de que es la contribución del paquete i al paquete j
  - Considerar pesos alternativos
- Agregar repositorios scrapeados desde gitlab/github, buscando `requirements.txt` o paquetes no publicados a PyPI