Proyecto final: la red de productos de PyShop

Vas a juntar todo el curso en un análisis completo de la red de productos que se compran juntos en PyShop, con recomendaciones que la tienda pueda aplicar. Se usan Python 3.10 o superior, NetworkX 3.6, pandas y Matplotlib. El notebook `proyecto-red-de-productos` tiene una solución de referencia: consúltalo al terminar.

Contexto

PyShop vende 41 productos: seis categorías más tres accesorios que se compran con varias de ellas. El equipo quiere saber qué poner junto, qué promocionar y qué producto queda fuera de toda relación. El catálogo no lo responde; la red de compras sí.

Datos

Copia este código en la primera celda. `generar_pedidos()` simula 3000 pedidos y `red_productos()` une dos productos si aparecieron juntos en al menos `minimo` pedidos (`weight` = coincidencias); cada nodo lleva su `categoria`.

```
import itertools
from collections import Counter

import networkx as nx
import numpy as np

CATALOGO = {
    "celulares": ["celular gama media",
                  "celular gama alta", "funda",
                  "mica de vidrio", "cargador rapido",
                  "soporte para auto", "power bank"],
    "computo": ["laptop", "mouse", "teclado", "monitor",
                "base para laptop", "hub usb",
                "mochila para laptop"],
    "audio": ["audifonos inalambricos",
              "audifonos de diadema", "barra de sonido",
              "microfono", "tocadiscos"],
    "cocina": ["cafetera", "molino de cafe",
               "freidora de aire", "licuadora",
               "juego de sartenes", "tabla para picar",
```

```
                "cuchillos"],
    "hogar": ["lampara de escritorio", "foco inteligente",
              "aspiradora robot", "ventilador", "cojines",
              "organizador"],
    "deportes": ["tapete de yoga", "mancuernas",
                 "liga de resistencia", "botella deportiva",
                 "tenis para correr", "reloj deportivo"],
}
PUENTES = {
    "cable usb-c": ["celulares", "computo", "audio"],
    "bocina bluetooth": ["audio", "hogar", "deportes"],
    "termo": ["cocina", "deportes"],
}
PESO_CAT = [0.22, 0.20, 0.14, 0.16, 0.14, 0.14]
AFINES = {"celulares": ["audio"],
          "computo": ["hogar", "audio"],
          "audio": ["celulares", "deportes"],
          "cocina": ["hogar"],
          "hogar": ["cocina", "computo"],
          "deportes": ["audio", "celulares"]}


def categorias():
    cat = {p: c for c, ps in CATALOGO.items() for p in ps}
    cat.update({p: "accesorios" for p in PUENTES})
    return cat


def generar_pedidos(n=3000, semilla=42):
    rng = np.random.default_rng(semilla)
    nombres = list(CATALOGO)
    pedidos = []
    for _ in range(n):
        c = nombres[rng.choice(len(nombres), p=PESO_CAT)]
        prods = CATALOGO[c]
        k = int(rng.integers(2, 4))
        pesos = np.linspace(2.0, 0.3, len(prods))
        elegidos = {str(x) for x in rng.choice(
            prods, size=min(k, len(prods)), replace=False,
            p=pesos / pesos.sum())}
        if rng.random() < 0.30:
            otra = CATALOGO[AFINES[c][
```

```
                    rng.integers(len(AFINES[c]))]]
                pa = np.linspace(2.0, 0.3, len(otra))
                pa = pa / pa.sum()
                elegidos.add(str(rng.choice(otra, p=pa)))
            for puente, cats in PUENTES.items():
                if c in cats and rng.random() < 0.18:
                    elegidos.add(puente)
            if rng.random() < 0.06:
                i = rng.integers(len(nombres))
                otra = CATALOGO[nombres[i]]
                elegidos.add(str(otra[rng.integers(len(otra))]))
            pedidos.append(sorted(elegidos))
        return pedidos


    def red_productos(pedidos=None, minimo=25):
        if pedidos is None:
            pedidos = generar_pedidos()
        pares = Counter()
        for p in pedidos:
            pares.update(itertools.combinations(p, 2))
        G = nx.Graph()
        for producto, c in categorias().items():
            G.add_node(producto, categoria=c)
        for (a, b), w in pares.items():
            if w >= minimo:
                G.add_edge(a, b, weight=w)
        return G


    pedidos = generar_pedidos()
    G = red_productos(pedidos)
    print(G.number_of_nodes(), G.number_of_edges())
    # 41 136
```

Los datos son sintéticos: describe lo que muestran sin inventarles causas.

Requisitos mínimos

1. **Red.** Construye la red con mínimo 25 y reporta nodos, aristas y la arista más pesada. Comprueba que `G["laptop"]["mouse"]` da `{'weight': 134}`.

2. **Umbral.** Reconstruye la red con mínimos de 1, 10, 25, 40 y 60. Tabla con aristas, densidad y componentes; justifica el umbral que eliges.
3. **Estructura.** Densidad, componentes y nodos aislados; diámetro, camino medio y clustering sobre la componente mayor; puentes y puntos de articulación. Quita los tres productos puente (cable usb-c, bocina bluetooth, termo) y reporta cómo se parte la red.
4. **Centralidad.** Grado, intermediación, cercanía, vector propio y PageRank en una sola tabla; los 5 primeros de cada medida. Explica por qué el producto con más vecinos no es el mejor puente y cómo trataste el peso.
5. **Comunidades.** Louvain con `weight="weight"` y `seed=42`: número de comunidades, modularidad y composición por categoría. Compara con `greedy_modularity_communities` y con la modularidad de las categorías del catálogo.
6. **Visualización.** Dos figuras con `spring_layout(G, seed=42)`: la red con color por comunidad, tamaño por centralidad y grosor por peso, y otra que responda una pregunta distinta (por ejemplo, la matriz de adyacencia ordenada).
7. **Recomendaciones.** Tres recomendaciones de negocio, cada una con su cifra: qué productos poner juntos, qué producto promocionar y qué hacer con el producto aislado.

Pistas de implementación

```
comp = max(nx.connected_components(G), key=len)
H = G.subgraph(comp)
print(nx.diameter(H))
lv = nx.community.louvain_communities(
    G, weight="weight", seed=42)
q = nx.community.modularity(G, lv, weight="weight")
```

Para comprobar: 6 comunidades, modularidad 0.609, diámetro 6.

Extras opcionales

- **Datos reales.** Repite centralidad y comunidades con `nx.les_miserables_graph()` o `nx.karate_club_graph()`; en el karate, cuenta cuántos miembros coinciden con el atributo `club`.
- **Estabilidad.** Repite Louvain con otras semillas y umbrales; reporta cuándo cambia.
- **Intermediación ponderada.** Crea el atributo `distancia = 1 / weight` y compara el ranking con el no ponderado.

Entregables

1. `proyecto.ipynb`, ejecutado de principio a fin sin errores, con las figuras a la vista.
2. `INFORME.md` (una página): umbral y justificación, tablas de estructura y centralidad, comunidades, las dos figuras, las tres recomendaciones y dos limitaciones.

Rúbrica (100 puntos, más 10 de extras)

- **Red y umbral (15 puntos):** red correcta, tabla de umbrales y justificación con cifras.
- **Estructura (15 puntos):** medidas sobre la componente correcta, puentes y efecto de quitarlos.
- **Centralidad (20 puntos):** cinco medidas, tratamiento del peso declarado y diferencia entre grado e intermediación explicada.
- **Comunidades (20 puntos):** Louvain con semilla, modularidad, composición por categoría y comparación con greedy y con el catálogo.
- **Visualización (15 puntos):** figuras legibles, reproducibles y con codificación justificada.
- **Recomendaciones e informe (15 puntos):** tres recomendaciones con cifras y limitaciones honestas.
- **Extras (hasta 10 puntos):** datos reales, estabilidad o intermediación ponderada bien interpretados.