Solucionarios

Semana 4

14.07.2025 - 20.07.2025

Caso General

Análisis de Interacción en Redes Sociales

Las empresas y creadores de contenido necesitan analizar qué publicaciones generan más interés para entender a su audiencia y mejorar su estrategia de comunicación. Medir "likes", comentarios y la actividad de los usuarios es clave para el crecimiento.

- Yamil Ayma

Nivel Fácil: Filtrar Posts por Autor

```python
def filtrar_posts_por_autor(lista_posts, nombre_autor):
    """
    Crea una nueva lista que contiene solo los posts de un autor específico.
    """
    # 1. Creamos una lista vacía para guardar los resultados.
    posts_filtrados = []

    # 2. Recorremos cada post en la lista original.
    for post in lista_posts:
        # 3. Verificamos si el valor de la clave 'autor' en el post actual
        #    es igual al nombre del autor que estamos buscando.
        if post['autor'] == nombre_autor:
            # 4. Si coincide, agregamos el post completo a nuestra lista de resultados.
            posts_filtrados.append(post)

    # 5. Devolvemos la nueva lista que solo contiene los posts del autor.
    return posts_filtrados

# --- Pruebas ---
def probar_filtrar_posts_por_autor():
    posts = [
        {'autor': 'ana_dev', 'contenido': '¡Aprendiendo Python!', 'likes': 50},
        {'autor': 'beto_data', 'contenido': 'Mi primer análisis de datos.', 'likes': 120},
        {'autor': 'ana_dev', 'contenido': 'Funciones Lambda son geniales.', 'likes': 95}
    ]

    posts_de_ana = filtrar_posts_por_autor(posts, 'ana_dev')
    esperado1 = [
        {'autor': 'ana_dev', 'contenido': '¡Aprendiendo Python!', 'likes': 50},
        {'autor': 'ana_dev', 'contenido': 'Funciones Lambda son geniales.', 'likes': 95}
    ]
    print(f"Prueba 1: {posts_de_ana == esperado1}")

    posts_de_carlos = filtrar_posts_por_autor(posts, 'carlos_design')
    print(f"Prueba 2: {posts_de_carlos == []}")

    posts_vacios = filtrar_posts_por_autor([], 'ana_dev')
    print(f"Prueba 3: {posts_vacios == []}")

probar_filtrar_posts_por_autor()
```

★ Click para acceder al código ★

Nivel Medio: Encontrar el Post con Más "Likes"

```python
def rankear_usuarios_por_engagement(lista_posts):
    """
    Calcula un puntaje de engagement para cada autor y los ordena de mayor a menor.
    """
    # 1. Creamos un diccionario para almacenar los datos de cada autor.
    #    Ejemplo: {'ana_dev': {'likes': 250, 'comentarios': 5}, ...}
    engagement_por_autor = {}

    # 2. Recorremos todos los posts para agrupar los datos por autor.
    for post in lista_posts:
        autor = post['autor']

        # Si el autor no está en nuestro diccionario, lo inicializamos.
        if autor not in engagement_por_autor:
            engagement_por_autor[autor] = {'likes': 0, 'comentarios': 0}

        # Sumamos los likes del post actual a su total.
        engagement_por_autor[autor]['likes'] += post['likes']

        # Sumamos la cantidad de comentarios (si existen).
        # .get('comentarios', []) es una forma segura de manejar posts sin comentarios.
        engagement_por_autor[autor]['comentarios'] += len(post.get('comentarios', []))

    # 3. Ahora, calculamos el puntaje final para cada autor.
    lista_ranking = []
    for autor, datos in engagement_por_autor.items():
        puntaje = datos['likes'] + (datos['comentarios'] * 2)
        lista_ranking.append((autor, puntaje))

    # 4. Ordenamos la lista de tuplas.
    #    key=lambda item: item[1] le dice a sorted que use el segundo elemento de la tupla (el puntaje)
para ordenar.
    #    reverse=True hace que el orden sea de mayor a menor.
    ranking_ordenado = sorted(lista_ranking, key=lambda item: item[1], reverse=True)

    # 5. Devolvemos el ranking final.
    return ranking_ordenado

# --- Pruebas ---
def probar_rankear_usuarios_por_engagement():
    posts = [
        {'autor': 'ana_dev', 'contenido': 'Post 1', 'likes': 100, 'comentarios': ['genial', 'útil']},
        {'autor': 'beto_data', 'contenido': 'Post 2', 'likes': 50, 'comentarios': ['interesante']},
        {'autor': 'ana_dev', 'contenido': 'Post 3', 'likes': 150, 'comentarios': ['gracias', 'me
sirvió', 'excelente']},
        {'autor': 'carla_ux', 'contenido': 'Post 4', 'likes': 250, 'comentarios': []}
    ]

    ranking = rankear_usuarios_por_engagement(posts)
    esperado = [
        ('ana_dev', 260),
        ('carla_ux', 250),
        ('beto_data', 52)
    ]
    print(f"Prueba 1: {ranking == esperado}")

    ranking_vacio = rankear_usuarios_por_engagement([])
    print(f"Prueba 2: {ranking_vacio == []}")

probar_rankear_usuarios_por_engagement()
```

★ Click para acceder al código ★

Nivel Avanzado: Rankear Usuarios por "Engagement"

```python
def rankear_usuarios_por_engagement(lista_posts):
    """
    Calcula un puntaje de engagement para cada autor y los ordena de mayor a menor.
    """
    # 1. Creamos un diccionario para almacenar los datos de cada autor.
    #    Ejemplo: {'ana_dev': {'likes': 250, 'comentarios': 5}, ...}
    engagement_por_autor = {}

    # 2. Recorremos todos los posts para agrupar los datos por autor.
    for post in lista_posts:
        autor = post['autor']

        # Si el autor no está en nuestro diccionario, lo inicializamos.
        if autor not in engagement_por_autor:
            engagement_por_autor[autor] = {'likes': 0, 'comentarios': 0}

        # Sumamos los likes del post actual a su total.
        engagement_por_autor[autor]['likes'] += post['likes']

        # Sumamos la cantidad de comentarios (si existen).
        # .get('comentarios', []) es una forma segura de manejar posts sin comentarios.
        engagement_por_autor[autor]['comentarios'] += len(post.get('comentarios', []))

    # 3. Ahora, calculamos el puntaje final para cada autor.
    lista_ranking = []
    for autor, datos in engagement_por_autor.items():
        puntaje = datos['likes'] + (datos['comentarios'] * 2)
        lista_ranking.append((autor, puntaje))

    # 4. Ordenamos la lista de tuplas.
    #    key=lambda item: item[1] le dice a sorted que use el segundo elemento de la tupla (el puntaje)
para ordenar.
    #    reverse=True hace que el orden sea de mayor a menor.
    ranking_ordenado = sorted(lista_ranking, key=lambda item: item[1], reverse=True)

    # 5. Devolvemos el ranking final.
    return ranking_ordenado

# --- Pruebas ---
def probar_rankear_usuarios_por_engagement():
    posts = [
        {'autor': 'ana_dev', 'contenido': 'Post 1', 'likes': 100, 'comentarios': ['genial', 'útil']},
        {'autor': 'beto_data', 'contenido': 'Post 2', 'likes': 50, 'comentarios': ['interesante']},
        {'autor': 'ana_dev', 'contenido': 'Post 3', 'likes': 150, 'comentarios': ['gracias', 'me
sirvió', 'excelente']},
        {'autor': 'carla_ux', 'contenido': 'Post 4', 'likes': 250, 'comentarios': []}
    ]

    ranking = rankear_usuarios_por_engagement(posts)
    esperado = [
        ('ana_dev', 260),
        ('carla_ux', 250),
        ('beto_data', 52)
    ]
    print(f"Prueba 1: {ranking == esperado}")

    ranking_vacio = rankear_usuarios_por_engagement([])
    print(f"Prueba 2: {ranking_vacio == []}")

probar_rankear_usuarios_por_engagement()
```

★ Click para acceder al código ★

Éxitos en los ejercicios

La simplicidad no precede a la complejidad, pero la sigue.
- Alan Perlis. Epigramas de la Programación

Aprende mucho