

Aplicación Web para Gestión y Análisis de Libros

Bienvenida al proyecto 📚. Este sistema web, desarrollado con **Python** y **Django**, facilita el manejo de información sobre libros, autores y géneros. Además, permite analizar datos y crear gráficos para visualizar estadísticas de lectura.

Entorno y Tecnologías

```
Python >= 3.10
Django >= 4.2
PostgreSQL >= 14
Pandas >= 1.5
Matplotlib >= 3.6
Seaborn >= 0.12
Scikit-learn >= 1.2
Virtualenv >= 20
```

Puesta en Marcha

1. Crear un entorno virtual

```
python -m venv venv
```

2. Activar el entorno

En Windows:

```
venv\Scripts\activate
```

En Linux/macOS:

```
source venv/bin/activate
```

3. Instalar dependencias

```
pip install django pandas matplotlib seaborn scikit-learn psycopg2-binary
```

Configuración de Base Datos

Para conectar Django a PostgreSQL, modifica settings.py así:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'libreria_db',
        'USER': 'postgres',
        'PASSWORD': 'tu_password',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

📖 Funcionalidades Principales

Con esta app podrás:

- Registrar información de autores, géneros y libros.
- Consultar registros de libros existentes.
- Obtener sugerencias de lectura por género, basadas en calificaciones.
- Generar gráficos para analizar datos de la biblioteca.
- Gestión de Autores

Modelo Author

```
from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=255)

    def __str__(self):
        return self.name
```

Vista para registrar autores

```
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
import json
from .models import Author

@csrf_exempt
def add_author(request):
    if request.method == 'POST':
        data = json.loads(request.body)
```

```
author = Author.objects.create(name=data['name'])
return JsonResponse({'message': 'Autor creado', 'id': author.id})
```

Ejemplo en Postman:

POST <http://127.0.0.1:8000/api/autores/>

Params Authorization Headers (10) Body Scripts Settings Cookies

(none) form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```

1 {
2   "nombre": "Julio Cortázar",
3   "nacionalidad": "Argentina"
4 }
5

```

Body Cookies Headers (10) Test Results 201 Created • 153 ms • 401 B Save Response

[{} JSON] Preview Visualize

```

1 {
2   "id": 6,
3   "nombre": "Julio Cortázar",
4   "nacionalidad": "Argentina"
5 }

```

Gestión de Géneros

Modelo Generos

```
from django.db import models

class Genre(models.Model):
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name
```

Vista para crear un género

```
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
import json
from .models import Genre

@csrf_exempt
def add_genre(request):
    if request.method == 'POST':
```

```

data = json.loads(request.body)
genre = Genre.objects.create(name=data['name'])
return JsonResponse({'message': 'Género creado', 'id': genre.id})

```

Ejemplo en Postman:

POST <http://127.0.0.1:8000/api/generos/> Send

Params Authorization Headers (10) **Body** Scripts Settings Cookies

(none) form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```

1 {
2   "nombre": "Autoayuda"
3 }
4

```

Body Cookies Headers (10) Test Results 201 Created 187 ms 369 B Save Response

[{} JSON] Preview Visualize

```

1 {}
2   "id": 14,
3   "nombre": "Autoayuda"
4

```

Registro de Libros

Vista para agregar libros

```

from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
import json
from .models import Book

@csrf_exempt
def add_book(request):
    if request.method == 'POST':
        data = json.loads(request.body)
        book = Book.objects.create(
            title=data['title'],
            author=data['author'],
            genre=data['genre'],
            year=data['year'],
            rating=data['rating']
        )
    return JsonResponse({'message': 'Libro creado', 'id': book.id})

```

Ejemplo en Postman:

The screenshot shows the Postman interface with a successful API call. The URL is `http://127.0.0.1:8000/api/libros/`. The request method is `POST`. The body contains the following JSON:

```
1 {  
2     "titulo": "Pedro Páramo",  
3     "autor": 1,  
4     "genero": 1,  
5     "fecha_publicacion": "1955-03-19",  
6     "isbn": "9681605021",  
7     "url": "https://ejemplo.com/pedro-paramo"  
8 }
```

The response status is `201 Created`, with a response time of `265 ms` and a size of `489 B`.

Visualización de Libros

Vista para listar todos los libros

```
from django.http import JsonResponse  
from .models import Book  
  
def list_books(request):  
    books = Book.objects.all().values()  
    return JsonResponse(list(books), safe=False)
```

Ejemplo en Postman:

GET <http://127.0.0.1:8000/api/libros/> Send

Params Authorization Headers (8) **Body** Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body.

Body Cookies Headers (10) Test Results ⚡ 200 OK • 242 ms • 2.34 KB • [Save Response](#)

{ } JSON ▾ ▶ Preview ⏷ Visualize

```

1  [
2    {
3      "id": 1,
4      "titulo": "Don Quijote de la Mancha",
5      "autor": 1,
6      "genero": 1,
7      "fecha_publicacion": "1605-01-16",
8      "isbn": "9788491050162",
9      "url": "https://ejemplo.com/don-quijote"
10     },
11   {

```

Actualización de Calificaciones

Vista para actualizar la puntuación de un libro

```

from django.views.decorators.csrf import csrf_exempt
import json
from django.http import JsonResponse
from .models import Book

@csrf_exempt
def rate_book(request, book_id):
    if request.method == 'POST':
        data = json.loads(request.body)
        book = Book.objects.get(id=book_id)
        book.rating = data['rating']
        book.save()
        return JsonResponse({'message': 'Calificación actualizada', 'rating': book.rating})

```

Ejemplo en Postman:

The screenshot shows a Postman interface. At the top, it says "POST" and the URL "http://127.0.0.1:8000/api/calificaciones/". On the right, there's a "Send" button. Below the URL, tabs for "Params", "Authorization", "Headers (10)", "Body", "Scripts", and "Settings" are visible, with "Body" being the active tab. Under "Body", options for "none", "form-data", "x-www-form-urlencoded", "raw", "binary", "GraphQL", and "JSON" are shown, with "raw" selected. To the right of these options is a "Beautify" button. The "Body" field contains the following JSON:

```
1 {  
2   "libro": 1,  
3   "calificacion": 5  
4 }  
5
```

On the right side of the interface, there are several icons: a magnifying glass, a refresh, a gear, and a copy/paste. Below the body editor, tabs for "Body", "Cookies", "Headers (10)", and "Test Results" are present, with "Body" being the active tab. The "Test Results" tab shows a green "201 Created" status with a timestamp of "240 ms" and a size of "378 B". There are also icons for "Save Response" and three dots. The "Body" tab shows the JSON response received:

```
1 {  
2   "id": 1,  
3   "libro": 1,  
4   "calificacion": "5.0"  
5 }
```

🔗 Lectura de Datos con Pandas

Para análisis de datos, puedes usar este script para leer datos de la base:

```
import pandas as pd
import psycopg2

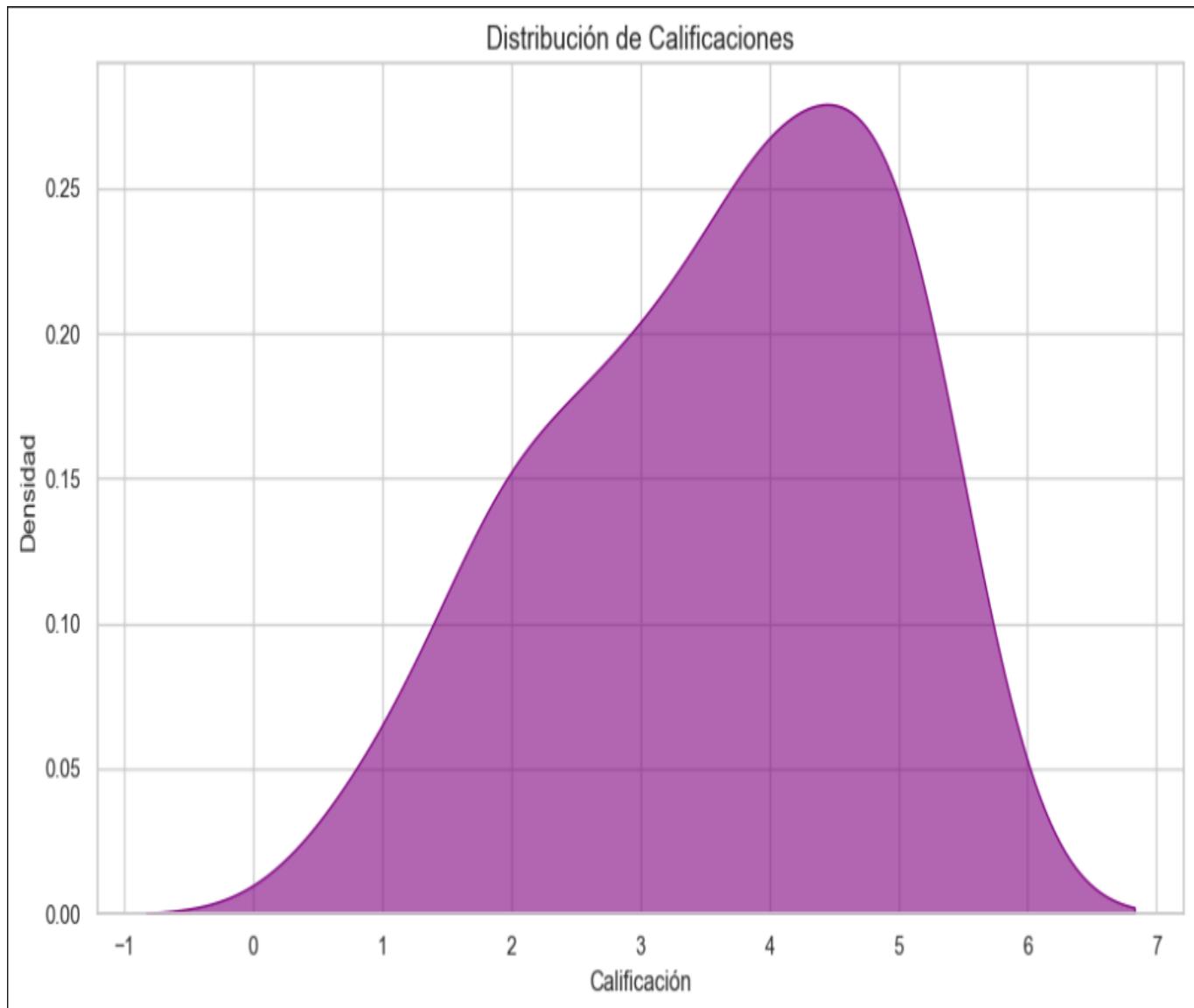
conn = psycopg2.connect(
    dbname='libreria_db',
    user='postgres',
    password='tu_password',
    host='localhost',
    port='5432'
)

query = "SELECT * FROM libros_book;"
df = pd.read_sql(query, conn)

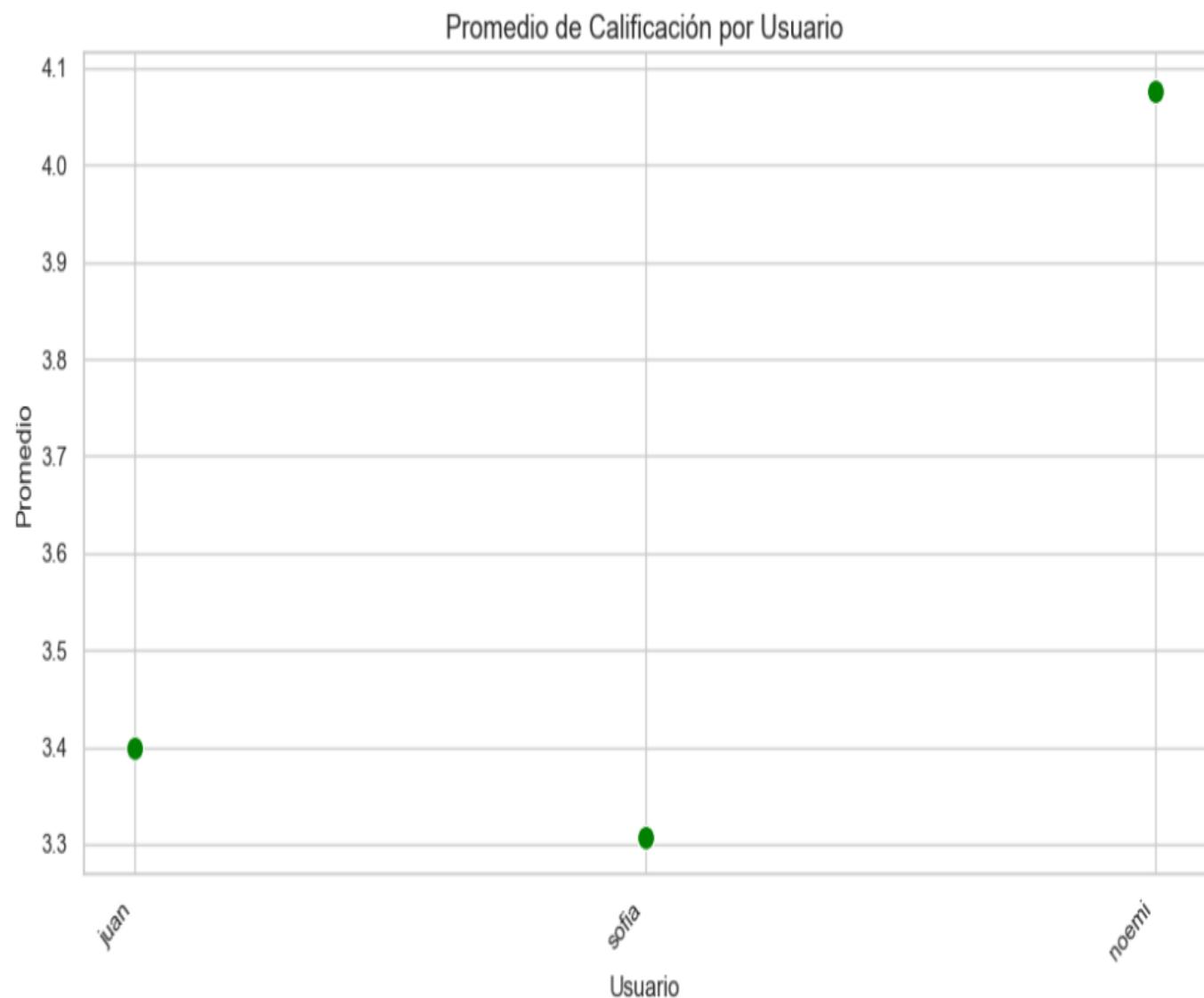
print(df.head())
```

📊 Análisis Visual

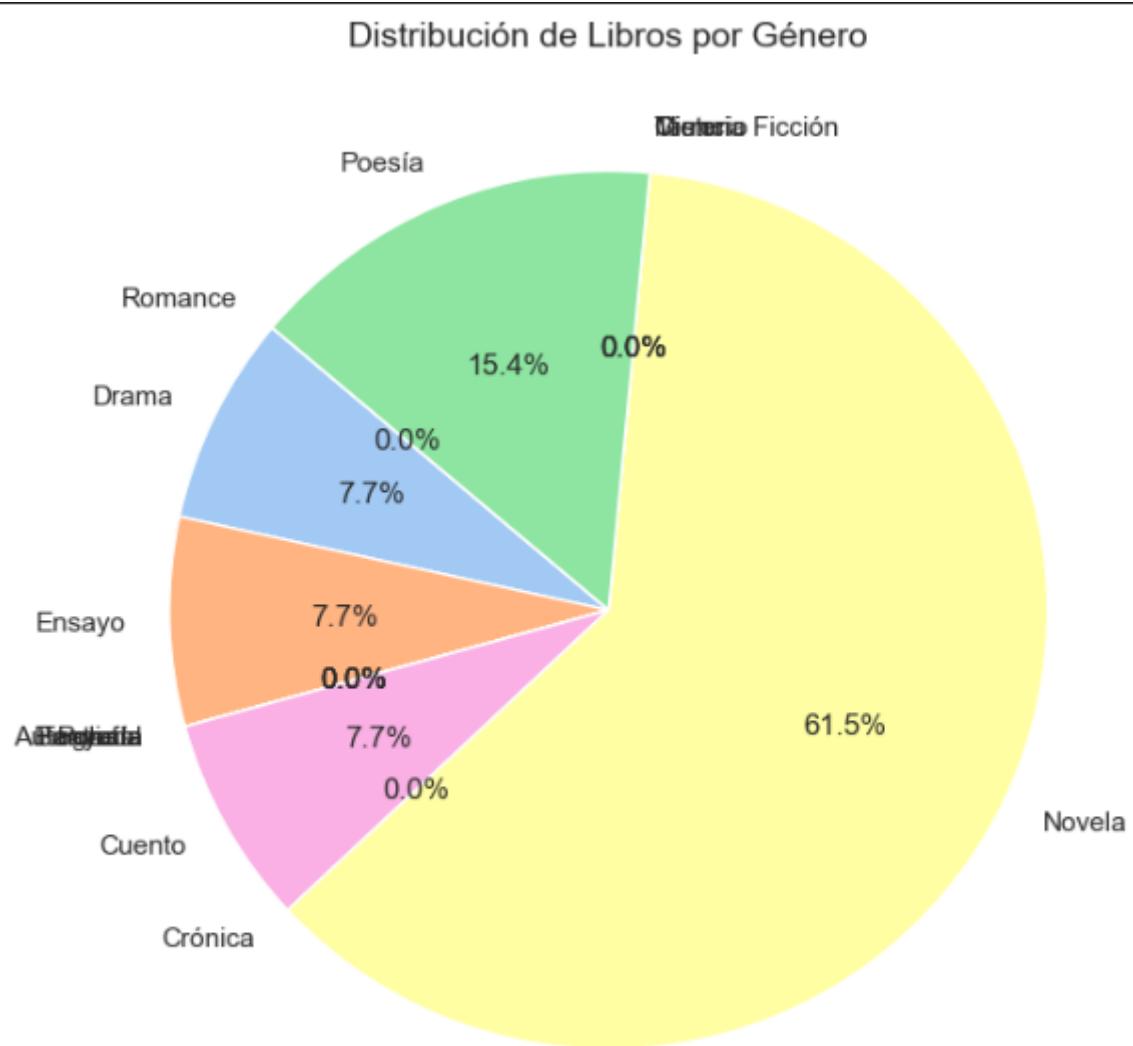
Distribución de las calificaciones:



Promedio de calificación por usuario:



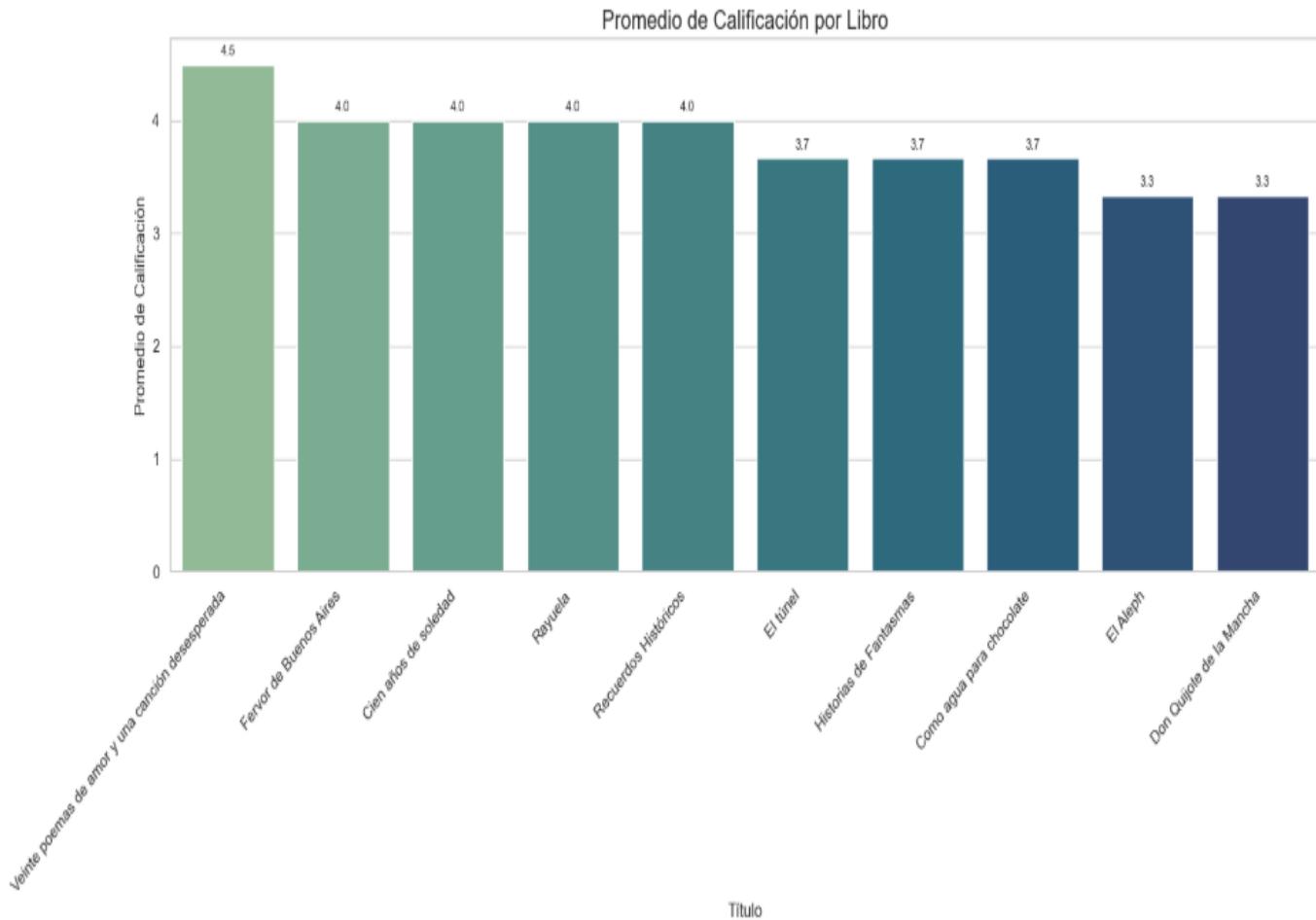
Cantidad de libros en cada género:



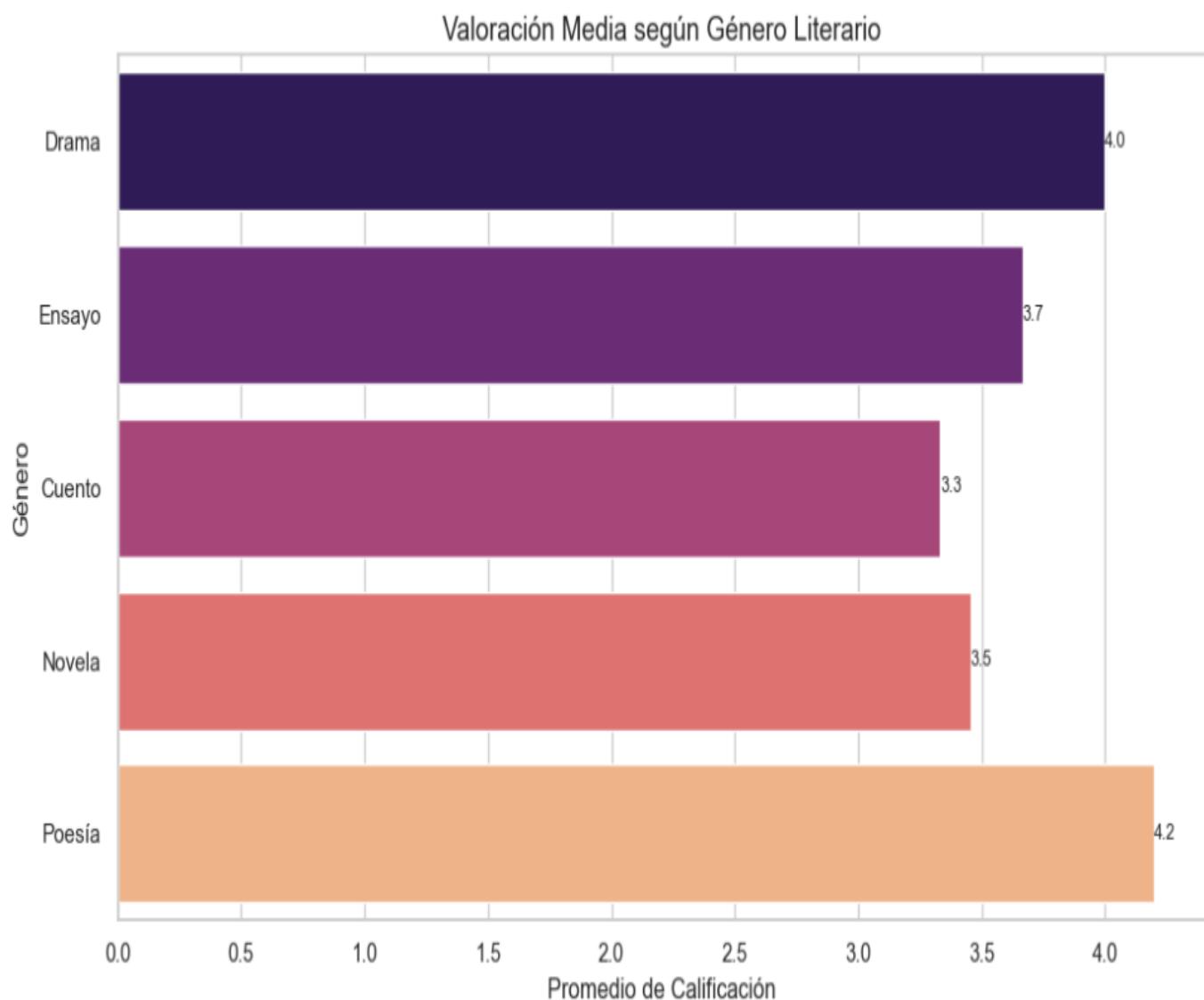
Autores más prolíficos

(imagen del gráfico autores más prolíficos)

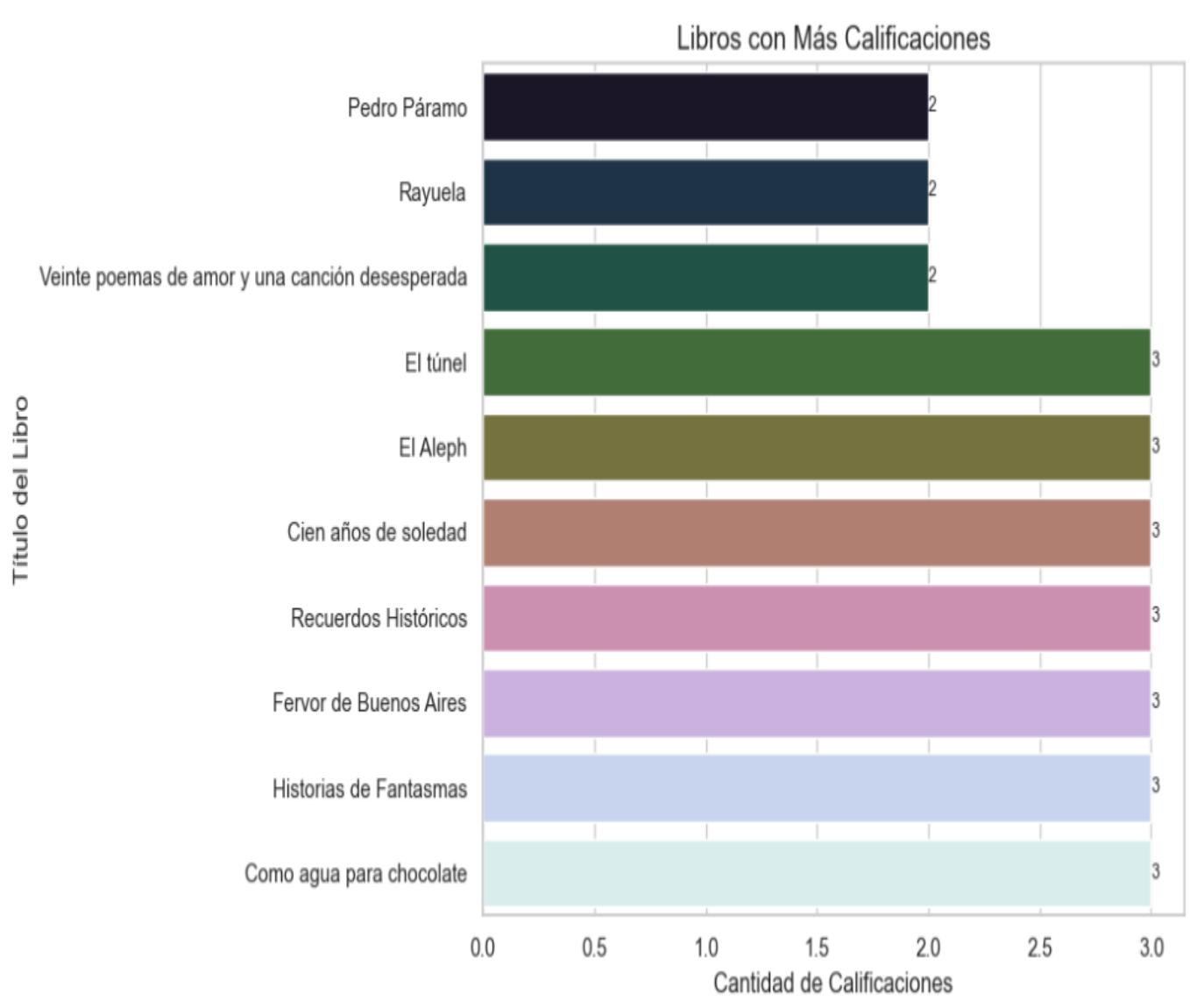
Libros con mejor promedio de calificación



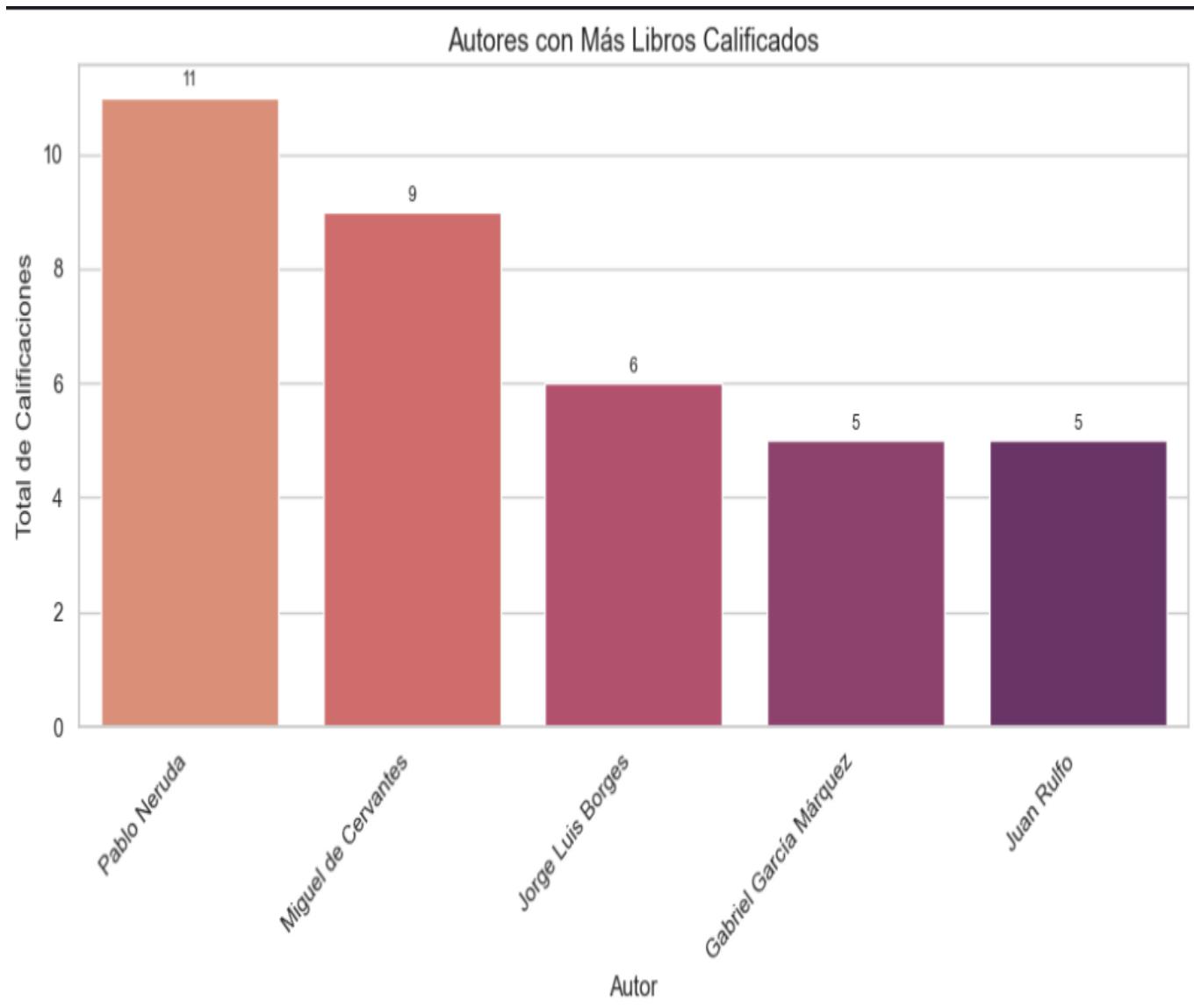
Media de valoración por género



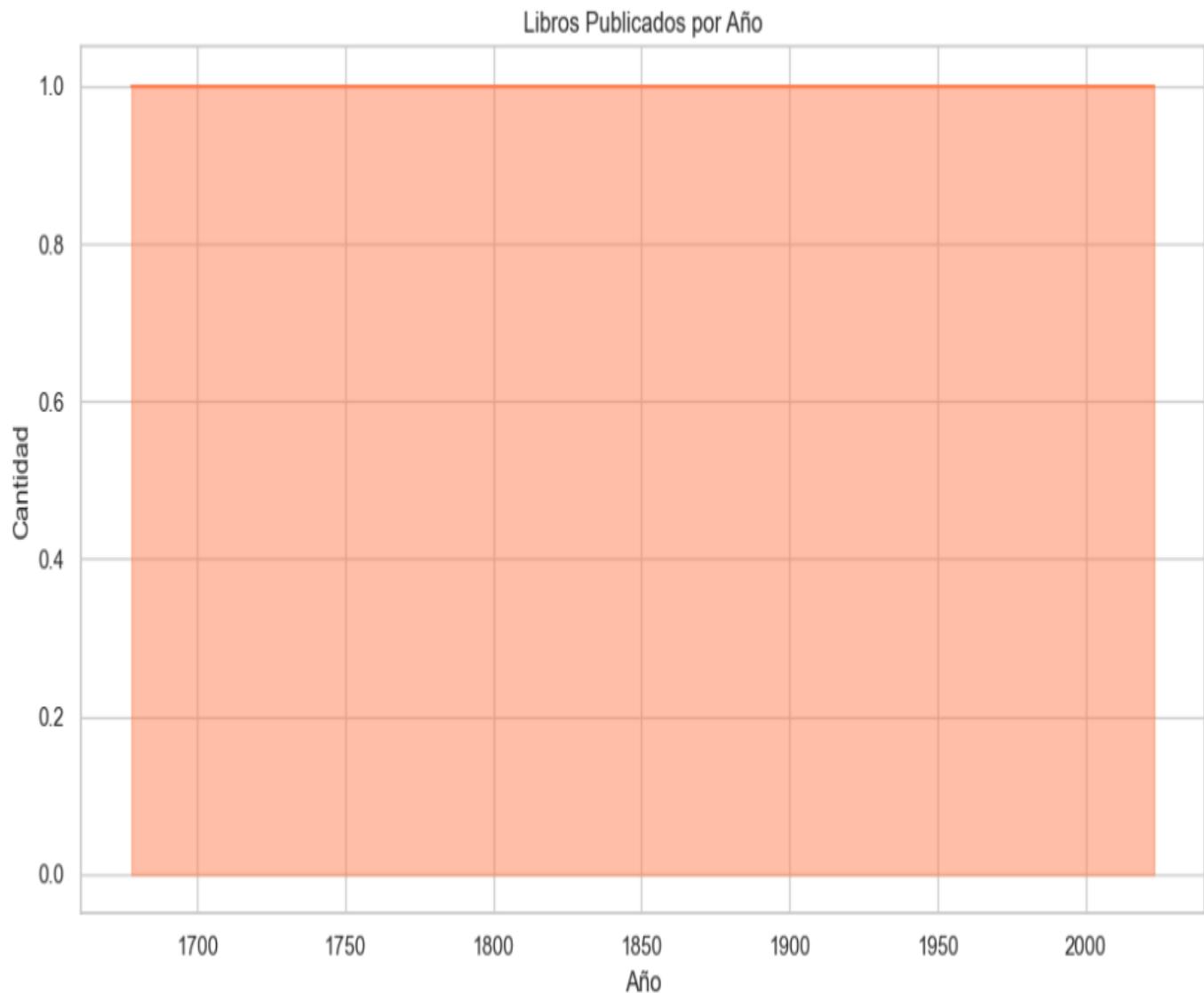
Libros más valorados



Autores con más calificaciones



Evolución de publicaciones a lo largo de los años



💡 Recomendaciones por Género

La app incluye un comando para recomendar lecturas. Filtra los libros por género y devuelve los mejor puntuados.

Ejemplo de uso:

```
python manage.py recomendar_por_genero aventura
```

Ejemplo de resultado:

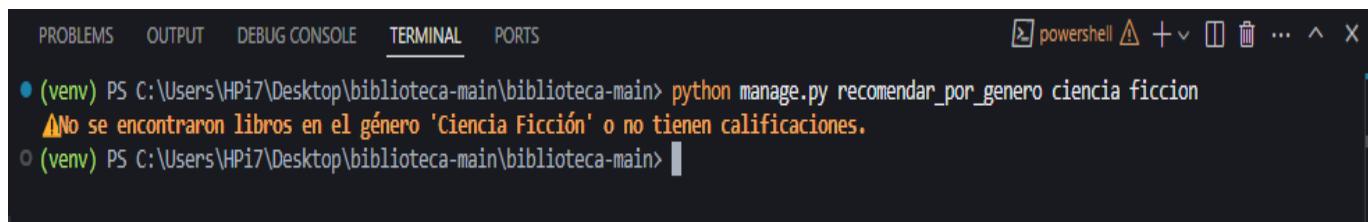


```
(venv) PS C:\Users\HPi7\Desktop\biblioteca-main\biblioteca-main> python manage.py recomendar_por_genero novela

Top libros en el género 'Novela':

- Cien años de soledad (Autor: Miguel de Cervantes) → Promedio: 4.0
- Rayuela (Autor: Pablo Neruda) → Promedio: 4.0
- Como agua para chocolate (Autor: Gabriel García Márquez) → Promedio: 3.7
- El túnel (Autor: Pablo Neruda) → Promedio: 3.7
- Don Quijote de la Mancha (Autor: Juan Rulfo) → Promedio: 3.3
- La casa de los espíritus (Autor: Jorge Luis Borges) → Promedio: 3.3
- Crónica de una muerte anunciada (Autor: Miguel de Cervantes) → Promedio: 3.0
- Pedro Páramo (Autor: Juan Rulfo) → Promedio: 2.5
```

Si el género no existe, se devuelve un aviso de error:



```
(venv) PS C:\Users\HPi7\Desktop\biblioteca-main\biblioteca-main> python manage.py recomendar_por_genero ciencia ficcion
⚠️ No se encontraron libros en el género 'Ciencia Ficción' o no tienen calificaciones.
```

Si el género existe, pero no hay libros calificados:

⚠️ No se encontraron libros en el género o no tienen calificaciones.

⚖️ Licencias

- Python → PSF License
- Django → BSD License
- PostgreSQL → PostgreSQL License
- Pandas → BSD License
- Matplotlib → PSF-based License
- Seaborn → BSD License
- Scikit-learn → BSD License

Este proyecto está cubierto por la licencia MIT. Consulta el archivo LICENSE para más detalles.