Coders of Delhi

Welcome - Your Data Science Internship begins

Congratulations! You have just been hired as a **Data Scientist Intern** at **CodeBook – The Social Media for Coders**. This Delhi-based company is offering you a **₹10 LPA job** if you successfully complete this **1-month internship**. But before you get there, you must prove your skills using **only Python**—no pandas, NumPy, or fancy libraries!

Your manager Puneet Kumar has assigned you your **first task**: analyzing a data dump of CodeBook users using pure Python. Your job is to **load and explore the data** to understand its structure.

Task 1: Load the User Data

Your manager has given you a dataset containing information about CodeBook users, their connections (friends), and the pages they have liked.

This is how the data will look (in JSON format):

```
{
    "users": [
        {"id": 1, "name": "Amit", "friends": [2, 3], "liked_pages": [101]},
        {"id": 2, "name": "Priya", "friends": [1, 4], "liked_pages": [102]},
        {"id": 3, "name": "Rahul", "friends": [1], "liked_pages": [101, 103]},
        {"id": 4, "name": "Sara", "friends": [2], "liked_pages": [104]}
    ],
    "pages": [
        {"id": 101, "name": "Python Developers"},
        {"id": 102, "name": "Data Science Enthusiasts"},
        {"id": 103, "name": "AI & ML Community"},
        {"id": 104, "name": "Web Dev Hub"}
```

```
    ]
}
```

We have to read this data and understand its structure. The data contains three main components: 1. **Users**: Each user has an ID, name, a list of friends (by their IDs), and a list of liked pages (by their IDs). 2. **Pages**: Each page has an ID and a name. 3. **Connections**: Users can have multiple friends and can like multiple pages.

Task 2: Read and Display the Data using Python

Your goal is to **load** this data and **print** it in a structured way. Use Python's built-in modules to accomplish this.

Steps:

1. Save the JSON data in a file (`codebook_data.json`).
2. Read the JSON file using Python.
3. Print user details and their connections.
4. Print available pages.

Code Implementation

Here's a simple way to load and display the data:

```
import json

# Load the JSON file
def load_data(filename):
    with open(filename, "r") as file:
        data = json.load(file)
    return data

# Display users and their connections
def display_users(data):
    print("Users and Their Connections:\n")
    for user in data["users"]:
```

```
        print(f"{user['name']} (ID: {user['id']}) - Friends: {user['friends']} -
Liked Pages: {user['liked_pages']}")
    print("\nPages:\n")
    for page in data["pages"]:
        print(f"{page['id']}: {page['name']}")

# Load and display the data
data = load_data("codebook_data.json")
display_users(data)
```

Expected Output:

```
Users and Their Connections:
Amit (ID: 1) - Friends: [2, 3] - Liked Pages: [101]
Priya (ID: 2) - Friends: [1, 4] - Liked Pages: [102]
Rahul (ID: 3) - Friends: [1] - Liked Pages: [101, 103]
Sara (ID: 4) - Friends: [2] - Liked Pages: [104]

Pages:
101: Python Developers
102: Data Science Enthusiasts
103: AI & ML Community
104: Web Dev Hub
```

Next Steps

Your manager is happy with your progress but says: **"The data looks messy. Can you clean and structure it better?"**

Cleaning and Structuring the Data

Introduction

Your manager is impressed with your progress but points out that the data is messy. Before we can analyze it effectively, we need to **clean and structure the data** properly.

Your task is to:

- Handle missing values
- Remove duplicate or inconsistent data
- Standardize the data format

Let's get started!

Task 1: Identify Issues in the Data

Your manager provides you with an example dataset where some records are incomplete or incorrect. Here's an example:

```
{
    "users": [
        {"id": 1, "name": "Amit", "friends": [2, 3], "liked_pages": [101]},
        {"id": 2, "name": "Priya", "friends": [1, 4], "liked_pages": [102]},
        {"id": 3, "name": "", "friends": [1], "liked_pages": [101, 103]},
        {"id": 4, "name": "Sara", "friends": [2, 2], "liked_pages": [104]},
        {"id": 5, "name": "Amit", "friends": [], "liked_pages": []}
    ],
    "pages": [
        {"id": 101, "name": "Python Developers"},
        {"id": 102, "name": "Data Science Enthusiasts"},
        {"id": 103, "name": "AI & ML Community"},
        {"id": 104, "name": "Web Dev Hub"},
        {"id": 104, "name": "Web Development"}
    ]
}
```

Problems:

1. User **ID 3** has an empty name.
2. User **ID 4** has a duplicate friend entry.
3. User **ID 5** has no connections or liked pages (inactive user).
4. The **pages list** contains duplicate page IDs.

Task 2: Clean the Data

We will:

1. Remove users with missing names.
2. Remove duplicate friend entries.
3. Remove inactive users (users with no friends and no liked pages).
4. Deduplicate pages based on IDs.

Code Implementation

```
import json

def clean_data(data):
    # Remove users with missing names
    data["users"] = [user for user in data["users"] if user["name"].strip()]

    # Remove duplicate friends
    for user in data["users"]:
        user["friends"] = list(set(user["friends"]))

    # Remove inactive users
    data["users"] = [user for user in data["users"] if user["friends"] or user["lik
ed_pages"]]

    # Remove duplicate pages
    unique_pages = {}
    for page in data["pages"]:
        unique_pages[page["id"]] = page
    data["pages"] = list(unique_pages.values())

    return data
```

```
# Load, clean, and display the cleaned data
data = json.load(open("codebook_data.json"))
data = clean_data(data)
json.dump(data, open("cleaned_codebook_data.json", "w"), indent=4)
print("Data cleaned successfully!")
```

Expected Output:

```
{
    "users": [
        {"id": 1, "name": "Amit", "friends": [2, 3], "liked_pages": [101]},
        {"id": 2, "name": "Priya", "friends": [1, 4], "liked_pages": [102]},
        {"id": 4, "name": "Sara", "friends": [2], "liked_pages": [104]}
    ],
    "pages": [
        {"id": 101, "name": "Python Developers"},
        {"id": 102, "name": "Data Science Enthusiasts"},
        {"id": 103, "name": "AI & ML Community"},
        {"id": 104, "name": "Web Development"}
    ]
}
```

The cleaned dataset will:

- Remove users with missing names
- Ensure friend lists contain unique entries
- Remove inactive users
- Deduplicate pages

Next Steps

Your manager is happy with the cleaned data and says: "**Great! Now that our data is structured, let's start analyzing it**. You are an intern, but he is so confident in your skills that he asks you - Can you build a 'People You May Know' feature?" Let's do that next!

Finding “People You May Know”

Now that our data is cleaned and structured, your manager assigns you a new task: **Build a ‘People You May Know’ feature!**

In social networks, this feature helps users connect with others by suggesting friends based on mutual connections. Your job is to **analyze mutual friends and recommend potential connections**.

Task 1: Understand the Logic

How ‘People You May Know’ Works:

- If **User A** and **User B** are not friends but have **mutual friends**, we suggest User B to User A and vice versa.
- More mutual friends = higher priority recommendation.

Example:

- **Amit (ID: 1)** is friends with **Priya (ID: 2)** and **Rahul (ID: 3)**.
- **Priya (ID: 2)** is friends with **Sara (ID: 4)**.
- Amit is not directly friends with Sara, but they share **Priya as a mutual friend**.
- Suggest **Sara to Amit** as “People You May Know”.

But there are cases where we will have more than one “People You May Know”. In those cases, greater the number of mutual friends, higher the probability that the user might know the person we are recommending.

Task 2: Implement the Algorithm

We’ll create a function that:

1. Finds all friends of a given user.
2. Identifies mutual friends between non-friends.
3. Ranks recommendations by the number of mutual friends.

Code Implementation

```
import json

def load_data(filename):
    with open(filename, "r") as file:
        return json.load(file)

def find_people_you_may_know(user_id, data):
    user_friends = {}
    for user in data["users"]:
        user_friends[user["id"]] = set(user["friends"])

    if user_id not in user_friends:
        return []

    direct_friends = user_friends[user_id]
    suggestions = {}

    for friend in direct_friends:
        # For all friends of friend
        for mutual in user_friends[friend]:
            # If mutual id is not the same user and not already a direct friend of
user
            if mutual != user_id and mutual not in direct_friends:
                # Count mutual friends
                suggestions[mutual] = suggestions.get(mutual, 0) + 1

    sorted_suggestions = sorted(suggestions.items(), key=lambda x: x[1], reverse=Tr
ue)
    return [user_id for user_id, _ in sorted_suggestions]

# Load data
data = load_data("cleaned_codebook_data.json")
user_id = 1  # Example: Finding suggestions for Amit
recommendations = find_people_you_may_know(user_id, data)
print(f"People You May Know for User {user_id}: {recommendations}")
```

Expected Output:

If Amit (ID: 1) and Sara (ID: 4) share Priya (ID: 2) as a mutual friend, the output might be:

```
People You May Know for User 1: [4]
```

This suggests that **Amit should connect with Sara!**

The Fontend developer of CodeBook can use this data via API and show it in the frontend when Amit logs in

---

Next Steps

Your manager is excited about your progress and now says: “Great job! Next, find ‘Pages You Might Like’ based on your connections and preferences.”

Let’s make sure we live up to his expectations.

Finding “Pages You Might Like”

We’ve officially reached the final milestone of our first data science project at **CodeBook – The Social Media for Coders**. After cleaning messy data and building features like **People You May Know**, it’s time to launch our last feature: **Pages You Might Like**.

Why This Matters

In real-world social networks, content discovery keeps users engaged. This feature simulates that experience using nothing but **pure Python**, showing how even simple logic can power impactful insights. So now the situation is that your manager is impressed with your ‘People You May Know’ feature and now assigns you a new challenge: **Recommend pages that users might like!**

On social media platforms, users interact with pages by liking, following, or engaging with posts. The goal is to analyze these interactions and suggest relevant pages based on user behavior.

Task 1: Understanding the Recommendation Logic

How 'Pages You Might Like' Works:

- Users engage with pages (like, comment, share, etc.).
- If two users have interacted with similar pages, they are likely to have common interests.
- For the sake of this implementation, we consider liking a page as an interaction
- Pages followed by similar users should be recommended.

Example:

- **Amit (ID: 1)** likes **Python Hub (Page ID: 101)** and **AI World (Page ID: 102)**.
- **Priya (ID: 2)** likes **AI World (Page ID: 102)** and **Data Science Daily (Page ID: 103)**.
- Since Amit and Priya both like **AI World (102)**, we suggest **Data Science Daily (103)** to Amit and **Python Hub (101)** to Priya.

What we are using here is called **collaborative filtering**:

> *"If two people like the same thing, maybe they'll like other things each one likes too."*

This is a basic form of a real-world recommendation engine, and our task is to implement it in pure Python. Let's Go!

Task 2: Implement the Algorithm

We'll create a function that:

1. Maps users to pages they have interacted with.
2. Identifies pages liked by users with similar interests.
3. Ranks recommendations based on common interactions.

Code Implementation

```
import json

# Function to load JSON data from a file
def load_data(filename):
    with open(filename, "r") as file:
        return json.load(file)

# Function to find pages a user might like based on common interests
def find_pages_you_might_like(user_id, data):
    # Dictionary to store user interactions with pages
    user_pages = {}
    for user in data["users"]:
        user_pages[user["id"]] = set(user["liked_pages"])

    # If the user is not found, return an empty list
    if user_id not in user_pages:
        return []

    user_liked_pages = user_pages[user_id]
    page_suggestions = {}

    for other_user, pages in user_pages.items():
        if other_user != user_id:
            shared_pages = user_liked_pages.intersection(pages)
            for page in pages:
                if page not in user_liked_pages:
                    page_suggestions[page] = page_suggestions.get(page, 0) + len(sh
ared_pages)

    # Sort recommended pages based on the number of shared interactions
    sorted_pages = sorted(page_suggestions.items(), key=lambda x: x[1], reverse=Tru
e)
    return [page_id for page_id, _ in sorted_pages]

# Load data
data = load_data("cleaned_codebook_data.json")
user_id = 1  # Example: Finding recommendations for Amit
```

```
page_recommendations = find_pages_you_might_like(user_id, data)
print(f"Pages You Might Like for User {user_id}: {page_recommendations}")
```

Expected Output:

If Amit (ID: 1) and Priya (ID: 2) both like AI World (102), and Priya also likes Data Science Daily (103), the output might be:

```
Pages You Might Like for User 1: [103]
```

This suggests that **Amit might be interested in 'Data Science Daily'!**

Now this was a very simple case, there is a chance that two users like 10 pages each, in that case all the pages liked by user 1 will be recommended to user 2 and vice versa. The score of this recommendation is the number of common pages they like

---

Conclusion

In simple terms, if two users like some of the same pages, it's likely they share similar interests. So, pages liked by one can be recommended to the other.

To make these recommendations smarter, we assign a similarity score — the number of pages they both like. The higher the score, the stronger the connection.

In larger networks where users follow many pages, this score helps us prioritize and personalize suggestions, just like how platforms like Facebook and LinkedIn recommend content or connections.

And that's a wrap!

From loading and cleaning data to recommending people and pages, you've successfully completed your first end-to-end data project using raw Python.

Your manager is impressed. Your ₹10 LPA dream offer letter is waiting for you!