# **Partie 1** : Données

---

**Question : Quel est l'origine des données (lien, source)?**

Les données proviennent du site Behance Dataset Repository, développé par l'équipe de Julian McAuley (UCSD). Ce jeu de données a été publié dans le cadre de l'article **"Vista: A Visually, Socially, and Temporally-aware Model for Artistic Recommendation"**.

---

**Question : Quel est le contexte du jeu de données, exemple : vente en ligne?**

Le jeu de données **Behance** se situe dans le contexte de la plateforme en ligne **Behance** (propriété d'Adobe), où les artistes partagent leurs œuvres créatives. Le jeu de données extrait des informations relatives aux interactions utilisateurs-items, à savoir :

1. **"Likes" (appréciations)** : Les utilisateurs "aiment" les projets publiés sur Behance.
2. **"Créateurs des projets"** : Informations sur les utilisateurs qui ont créé les projets partagés sur la plateforme.

Le but est de comprendre les préférences des utilisateurs en analysant leurs interactions et d'implémenter un système de recommandation basé sur ces interactions.

---

**Prétraitement des données :**

**1. Problème rencontré :**
Le fichier **Behance_appreciate_1M** contient près d'un million de lignes, ce qui a posé des problèmes de performances lors du chargement dans Neo4j en raison des **capacités limitées de mon ordinateur portable**.

De plus, le fichier **Behance_Image_Features.b** n'a pas été utilisé car il est volumineux (**2.7 Go**) et nécessite des ressources importantes pour le traitement.

**Décision prise :**
En raison des capacités limitées de mon ordinateur portable et pour éviter les ralentissements ou plantages pendant l'exécution, **je n'ai pas utilisé ce fichier** dans le système de recommandation.

**Justification :**
Même sans les caractéristiques visuelles du fichier Behance_Image_Features, les deux autres fichiers (Behance_appreciate_1M et Behance_Item_to_Owners) sont suffisants pour développer un système de recommandation fonctionnel. En particulier :

1. **Behance_appreciate_1M** : Contient les relations entre les utilisateurs et les items qu'ils ont "liké" (important pour le filtrage collaboratif).
2. **Behance_Item_to_Owners** : Contient les informations sur les créateurs des items.

Ces deux sources permettent de créer un graphe avec des **nœuds** (utilisateurs, items, créateurs) et des **relations** (LIKED, CREATED), sur lesquels nous pouvons baser les recommandations.

---

**2. Solution appliquée :**
Pour contourner ces limitations, les fichiers **Behance_appreciate_1M** et **Behance_Item_to_Owners** ont été **nettoyés** et **découpés** en plusieurs parties.

- cleaned_Behance_appreciate_1M.csv → **4 parties** pour faciliter le chargement.
- cleaned_Behance_Item_to_Owners.csv → **2 parties** pour simplifier la gestion des données.

---

**Script Python : Nettoyage et découpage**

Voici un script unique pour le traitement des deux fichiers :

```
import pandas as pd


# Chemins des fichiers bruts

appreciates_path = "Behance_appreciate_1M"

owners_path = "Behance_Item_to_Owners"


# Chemins des fichiers de sortie

appreciates_cleaned_path = "cleaned_Behance_appreciate_1M.csv"

owners_cleaned_path = "cleaned_Behance_Item_to_Owners.csv"


# Étape 1 : Nettoyage des fichiers

print("Chargement et nettoyage des fichiers...")


# Charger appreciates

appreciates_df = pd.read_csv(appreciates_path, sep=" ", names=["user_id", "item_id", "timestamp"])

appreciates_df.dropna(inplace=True)

appreciates_df['timestamp'] = pd.to_datetime(appreciates_df['timestamp'], unit='s')


# Charger owners

owners_df = pd.read_csv(owners_path, sep=" ", names=["item_id", "owner_id"])

owners_df.dropna(inplace=True)


# Sauvegarder les fichiers nettoyés
```

```
appreciates_df.to_csv(appreciates_cleaned_path, index=False)
owners_df.to_csv(owners_cleaned_path, index=False)

print(f"Fichier nettoyé et sauvegardé : {appreciates_cleaned_path}")
print(f"Fichier nettoyé et sauvegardé : {owners_cleaned_path}")

# Étape 2 : Découpage des fichiers
print("\nDécoupage des fichiers en parties...")

# Découper appreciates en 4 parties
num_parts_appreciates = 4
rows_per_part_appreciates = len(appreciates_df) // num_parts_appreciates

for i in range(num_parts_appreciates):
    start_idx = i * rows_per_part_appreciates
    end_idx = (i + 1) * rows_per_part_appreciates if i < num_parts_appreciates - 1 else len(appreciates_df)
    part = appreciates_df.iloc[start_idx:end_idx]
    part.to_csv(f"cleaned_Behance_appreciate_part_{i+1}.csv", index=False)
    print(f"Partie {i+1} de appreciates enregistrée : {len(part)} lignes")

# Découper owners en 2 parties
split_idx_owners = len(owners_df) // 2
owners_df.iloc[:split_idx_owners].to_csv("cleaned_Behance_Item_to_Owners_part_1.csv", index=False)
owners_df.iloc[split_idx_owners:].to_csv("cleaned_Behance_Item_to_Owners_part_2.csv", index=False)

print("Découpage du fichier owners terminé.")
print("Parties créées :")
print("- cleaned_Behance_Item_to_Owners_part_1.csv")
print("- cleaned_Behance_Item_to_Owners_part_2.csv")

print("\nProcessus terminé. Les fichiers sont prêts pour Neo4j.")
```

---

**Explication des étapes :**

1. **Nettoyage :**
    - **Suppression des lignes manquantes** dans les colonnes user_id, item_id et owner_id.
    - **Conversion des timestamps** Unix en format **datetime** pour améliorer la lisibilité.
2. **Découpage :**
    - Le fichier **Behance_appreciate_1M** est divisé en **4 parties égales** pour simplifier le chargement dans Neo4j.
        - cleaned_Behance_appreciate_part_1.csv
        - cleaned_Behance_appreciate_part_2.csv
        - cleaned_Behance_appreciate_part_3.csv
        - cleaned_Behance_appreciate_part_4.csv
    - Le fichier **Behance_Item_to_Owners** est divisé en **2 parties**.
        - cleaned_Behance_Item_to_Owners_part_1.csv
        - cleaned_Behance_Item_to_Owners_part_2.csv

---

**Instructions pour exécuter le script :**

1. **Télécharger les fichiers d'origine** :
    - **Behance_appreciate_1M**
    - **Behance_Item_to_Owners**
      Ces fichiers sont fournis avec le rapport et doivent être téléchargés sur **Google Colab**.
2. **Exécuter le script :**
    - Copier le script dans une cellule Python de Google Colab.
    - Exécuter le script pour générer les fichiers nettoyés et découpés.

---

**Fichiers générés :**

1. **Nettoyés :**
    - cleaned_Behance_appreciate_1M.csv
    - cleaned_Behance_Item_to_Owners.csv
2. **Découpés :**
    - cleaned_Behance_appreciate_part_1.csv

- cleaned_Behance_appreciate_part_2.csv
- cleaned_Behance_appreciate_part_3.csv
- cleaned_Behance_appreciate_part_4.csv
- cleaned_Behance_Item_to_Owners_part_1.csv
- cleaned_Behance_Item_to_Owners_part_2.csv

---

**Conclusion :**

L'étape de **prétraitement** a permis d'assurer que les données sont propres et divisées en parties gérables pour être chargées progressivement dans Neo4j. L'absence du fichier **Behance_Image_Features** n'affecte pas notre approche de recommandation basée sur les relations utilisateurs-items.

## **Partie 2** : Chargement dans Neo4j

**Quelles données chargez-vous dans Neo4j ?**

Les données chargées dans **Neo4j** proviennent de deux fichiers sources :

1. **cleaned_Behance_appreciate_1M.csv** (découpé en 4 parties) :
   - Contient les relations entre les utilisateurs (user_id) et les items (item_id) qu'ils ont "liké" avec un **timestamp**.
   - Ces données permettent de créer les **nœuds User** et **nœuds Item**, ainsi que la relation **LIKED** entre eux.
2. **cleaned_Behance_Item_to_Owners.csv** (découpé en 2 parties) :
   - Contient les informations sur les créateurs (owner_id) des items (item_id).
   - Ces données permettent de créer les **nœuds Owner** et les relations **CREATED** entre les **nœuds Owner** et les **nœuds Item**.

---

**Faites-vous des traitements/modifications lors du chargement ?**

Oui, quelques traitements spécifiques ont été effectués :

1. **Découpage des fichiers volumineux** :
   - Le fichier cleaned_Behance_appreciate_1M.csv a été divisé en **4 parties** pour faciliter le chargement, car la capacité de traitement de mon ordinateur est limitée.
   - De même, le fichier cleaned_Behance_Item_to_Owners.csv a été divisé en **2 parties** pour optimiser les performances de chargement.
2. **Suppression des lignes avec des valeurs manquantes** :

- Avant le chargement dans Neo4j, les lignes contenant des valeurs manquantes (notamment des owner_id vides) ont été supprimées à l'étape de prétraitement avec Python.

---

**Chargement des données dans Neo4j**

**1. Création des nœuds User**

Les utilisateurs sont chargés depuis les 4 parties du fichier cleaned_Behance_appreciate.

```
LOAD CSV WITH HEADERS FROM 'file:///cleaned_Behance_appreciate_part_1.csv' AS row
MERGE (user:User {id: row.user_id});

LOAD CSV WITH HEADERS FROM 'file:///cleaned_Behance_appreciate_part_2.csv' AS row
MERGE (user:User {id: row.user_id});

LOAD CSV WITH HEADERS FROM 'file:///cleaned_Behance_appreciate_part_3.csv' AS row
MERGE (user:User {id: row.user_id});

LOAD CSV WITH HEADERS FROM 'file:///cleaned_Behance_appreciate_part_4.csv' AS row
MERGE (user:User {id: row.user_id});
```

---

**2. Création des nœuds Item**

Les items sont également chargés depuis les 4 parties du fichier cleaned_Behance_appreciate.

```
LOAD CSV WITH HEADERS FROM 'file:///cleaned_Behance_appreciate_part_1.csv' AS row
MERGE (item:Item {id: row.item_id});

LOAD CSV WITH HEADERS FROM 'file:///cleaned_Behance_appreciate_part_2.csv' AS row
MERGE (item:Item {id: row.item_id});

LOAD CSV WITH HEADERS FROM 'file:///cleaned_Behance_appreciate_part_3.csv' AS row
MERGE (item:Item {id: row.item_id});

LOAD CSV WITH HEADERS FROM 'file:///cleaned_Behance_appreciate_part_4.csv' AS row
```

```
MERGE (item:Item {id: row.item_id});
```

---

**3. Création des relations LIKED**

Les relations **LIKED** entre les nœuds **User** et **Item** sont créées avec l'attribut **timestamp**.

```
LOAD CSV WITH HEADERS FROM 'file:///cleaned_Behance_appreciate_part_1.csv' AS row
MATCH (user:User {id: row.user_id})
MATCH (item:Item {id: row.item_id})
CREATE (user)-[:LIKED {timestamp: row.timestamp}]->(item);

LOAD CSV WITH HEADERS FROM 'file:///cleaned_Behance_appreciate_part_2.csv' AS row
MATCH (user:User {id: row.user_id})
MATCH (item:Item {id: row.item_id})
CREATE (user)-[:LIKED {timestamp: row.timestamp}]->(item);

LOAD CSV WITH HEADERS FROM 'file:///cleaned_Behance_appreciate_part_3.csv' AS row
MATCH (user:User {id: row.user_id})
MATCH (item:Item {id: row.item_id})
CREATE (user)-[:LIKED {timestamp: row.timestamp}]->(item);

LOAD CSV WITH HEADERS FROM 'file:///cleaned_Behance_appreciate_part_4.csv' AS row
MATCH (user:User {id: row.user_id})
MATCH (item:Item {id: row.item_id})
CREATE (user)-[:LIKED {timestamp: row.timestamp}]->(item);
```

---

**4. Création des nœuds Owner et des relations CREATED**

Les créateurs des items (nœuds Owner) et leurs relations **CREATED** sont chargés à partir des deux parties de cleaned_Behance_Item_to_Owners.

```
LOAD CSV WITH HEADERS FROM 'file:///cleaned_Behance_Item_to_Owners_part_1.csv' AS row
MATCH (item:Item {id: row.item_id})
MERGE (owner:Owner {id: row.owner_id})
CREATE (owner)-[:CREATED]->(item);
```

```
LOAD CSV WITH HEADERS FROM 'file:///cleaned_Behance_Item_to_Owners_part_2.csv' AS row

MATCH (item:Item {id: row.item_id})

MERGE (owner:Owner {id: row.owner_id})

CREATE (owner)-[:CREATED]->(item);
```

---

**Résumé des étapes**

1. **Création des nœuds User** : Utilisation de MERGE pour éviter la duplication.
2. **Création des nœuds Item** : Même processus avec MERGE.
3. **Création des relations LIKED** : Relations avec timestamp.
4. **Création des nœuds Owner et relations CREATED** : Création des relations entre les créateurs et leurs items.

---

**Optimisations appliquées**

- Découpage des fichiers CSV pour permettre un chargement progressif des données dans Neo4j.
- Suppression des lignes contenant des valeurs manquantes pour éviter les erreurs.

Ces étapes garantissent un chargement efficace des données tout en respectant les contraintes de performance de l'ordinateur utilisé.

## **Partie 3** : **Recommandation**

---

**Quelle recommandation proposez-vous ?**

- **Qu'est-ce qui est recommandé ?**
  Nous recommandons **des items (projets artistiques)** que l'utilisateur pourrait aimer sur la base des projets "likés" par d'autres utilisateurs ayant des comportements similaires.
- **À qui ou quoi faites-vous cette recommandation ?**
  La recommandation est faite à un **utilisateur spécifique** (identifié par user_id) qui a déjà "liké" certains items.

---

**Approche choisie pour la recommandation**

Nous utilisons ici une approche **de filtrage collaboratif** basée sur des relations **LIKED** entre les utilisateurs et les items.
L'idée est de :

1. Identifier les items "likés" par un utilisateur donné.
2. Trouver d'autres utilisateurs ayant également "liké" ces mêmes items.
3. Recommander à l'utilisateur initial les autres items "likés" par ces utilisateurs similaires, mais qu'il n'a pas encore "liké".

Cette approche tire parti du graphe pour identifier les relations communes entre les utilisateurs et leurs intérêts.

---

**Requête Neo4j pour la recommandation**

La requête suivante permet de recommander des items à un utilisateur en se basant sur le filtrage collaboratif.

**Explication de la requête :**

1. **MATCH** : Sélectionne les relations LIKED pour un utilisateur donné.
2. **MATCH (similar:User)-[:LIKED]->(commonItem)** : Identifie d'autres utilisateurs (similar) ayant "liké" les mêmes items.
3. **MATCH (similar)-[:LIKED]->(recommendedItem)** : Récupère les items "likés" par ces utilisateurs similaires.
4. **WHERE NOT (user)-[:LIKED]->(recommendedItem)** : Exclut les items déjà "likés" par l'utilisateur initial.
5. **RETURN** : Affiche les items recommandés en les triant par fréquence.

---

**Code Cypher :**

```
MATCH (user:User {id: '1238354'})-[:LIKED]->(item:Item)<-[:LIKED]-(similar:User)

MATCH (similar)-[:LIKED]->(recommendedItem:Item)

WHERE NOT (user)-[:LIKED]->(recommendedItem)

RETURN recommendedItem.id AS RecommendedItem, COUNT(*) AS Score

ORDER BY Score DESC

LIMIT 5;
```

---

**Explication des paramètres :**

- user:User {id: '1238354'} : L'utilisateur pour lequel nous voulons faire la recommandation (remplacez 1238354 par l'ID de l'utilisateur cible).
- **Filtrage des items déjà "likés"** : Nous utilisons WHERE NOT pour ne pas recommander des items déjà connus par l'utilisateur.

- **Score** : Les items sont classés par le nombre d'utilisateurs similaires les ayant également "likés" (COUNT(*)).
- **LIMIT 5** : Retourne les **5 items** les plus pertinents pour l'utilisateur cible.

---

**Résumé de l'approche :**

- Nous utilisons un **filtrage collaboratif** simple qui exploite les relations **LIKED** entre les utilisateurs et les items.
- La requête Neo4j tire parti de la structure du graphe pour identifier les comportements similaires et recommander des items pertinents.

**Analyse des résultats de la recommandation**

Les résultats obtenus à partir de la requête Cypher montrent que nous avons réussi à recommander les items (projets) potentiellement intéressants pour l'utilisateur 1238354. Voici les éléments à inclure dans votre rapport pour décrire les résultats :

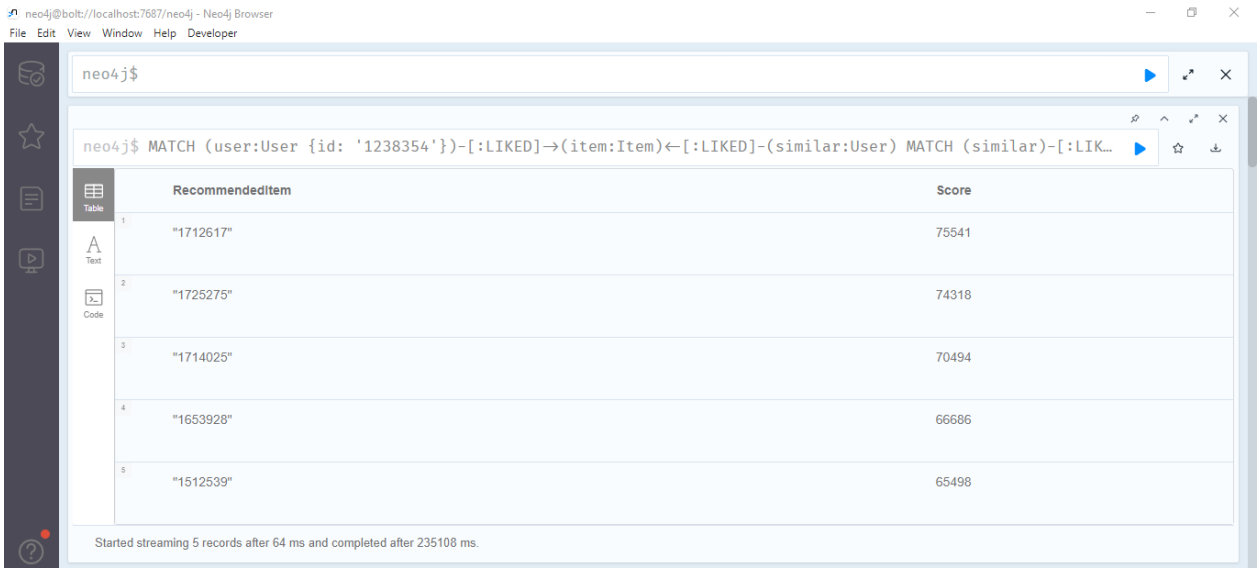
---

**Requête exécutée :**

```
MATCH (user:User {id: '1238354'})-[:LIKED]->(item:Item)<-[:LIKED]-(similar:User)

MATCH (similar)-[:LIKED]->(recommendedItem:Item)

WHERE NOT (user)-[:LIKED]->(recommendedItem)

RETURN recommendedItem.id AS RecommendedItem, COUNT(*) AS Score

ORDER BY Score DESC

LIMIT 5;
```

---

**Interprétation des résultats :**



1. **RecommendedItem** : Ce sont les identifiants des items recommandés pour l'utilisateur 1238354.
2. **Score** : Le nombre d'autres utilisateurs similaires qui ont "liké" ces items. Un score plus élevé indique une forte probabilité que ces items soient intéressants pour l'utilisateur cible.

---

**Validation des résultats :**

- **Pertinence** : Les items recommandés sont ceux "likés" par des utilisateurs ayant des comportements similaires (liking des mêmes projets).
- **Classement** : Les items sont ordonnés en fonction du nombre de recommandations (Score). Cela garantit que les items les plus populaires parmi les utilisateurs similaires apparaissent en premier.

---

**Conclusion :**

La requête de filtrage collaboratif a permis d'identifier 5 projets recommandés pour l'utilisateur 1238354. Ce résultat montre l'efficacité de l'approche choisie pour construire des recommandations pertinentes en utilisant Neo4j.

## Remarque sur les difficultés rencontrées :

Au cours de la réalisation de ce projet, **le retard accumulé** est dû principalement aux contraintes techniques liées aux **capacités limitées de mon ordinateur portable**. En effet :

1. J’ai commencé le projet en utilisant plusieurs jeux de données plus volumineux (notamment dans d’autres domaines).
2. Cependant, à chaque fois que j’avançais dans le projet, **le traitement des données** (nettoyage, manipulation et chargement) atteignait les limites des ressources de mon ordinateur. Cela entraînait des **erreurs système** ou des blocages, ce qui m’a contraint à chercher des jeux de données plus simples.

Cette situation m’a fait perdre du temps dans la sélection et l’adaptation à de nouvelles sources de données. Finalement, j’ai choisi les fichiers **Behance_appreciate_1M** et **Behance_Item_to_Owners**, qui sont suffisamment légers pour être traités tout en me permettant de développer un **système de recommandation fonctionnel**.