Manipulation de données

Pipeline d’agrégation avec MongoDB

Xavier Gendre

Motivation : une table de contingence

L'objectif est de compter le nombre de cartes de RingsDB par *Sphere* et par *Type*.

```
1  import requests
2
3  r = requests.get("https://ringsdb.com/api/public/cards")
4
5  if r.status_code != 200:
6      print(f"Erreur {r.status_code}")
7
8  rings = db["rings"]
9  result = rings.insert_many(r.json())
```

Le tableau donnant de tels effectifs croisés s'appelle une *table de contingence*. Il s'agit d'un outil très utilisé en statistique.

Méthode brutale

```
spheres = rings.distinct(key="sphere_name")
types = rings.distinct(key="type_name")
table = pd.DataFrame(0, index=spheres, columns=types)
for s in spheres:
    for t in types:
        table.loc[s, t] = rings.count_documents(
            {"$and": [{"sphere_name": s}, {"type_name": t}]}
        )
print(table)
```

```
             Ally  Attachment  Campaign  Contract  Event  Hero  \
Baggins         0           0         0         0      2     3
Fellowship      0           3         0         0      3     8
Leadership     88          62         0         0     78    61
Lore           90          63         0         0     67    64
Neutral        30          40       191        14     28     2
Spirit         91          59         0         0     67    64
Tactics        80          66         0         0     73    61

             Player Objective  Player Side Quest
Baggins                     0                  0
Fellowship                  0                  0
Leadership                  0                  2
Lore                        0                  2
```

Méthode brutale

```
spheres = rings.distinct(key="sphere_name")
types = rings.distinct(key="type_name")
table = pd.DataFrame(0, index=spheres, columns=types)
for s in spheres:
    for t in types:
        table.loc[s, t] = rings.count_documents(
            {"$and": [{"sphere_name": s}, {"type_name": t}]}
        )
print(table)
```

- Ce code n’a aucune élégance 🤮
- Il y a beaucoup d’aller-retours entre le client et le serveur.

Avec Pandas

```
1 df = pd.DataFrame(
2     rings.find( # Tout est retourné sans filter
3         projection={"_id": False, "sphere_name": True, "type_name":
4     )
5 )
6 # La fonction crosstab calcule les effectifs croisés
7 print(pd.crosstab(df.sphere_name, df.type_name))
```

```
type_name     Ally  Attachment  Campaign  Contract  Event  Hero  \
sphere_name
Baggins          0           0         0         0      2     3
Fellowship       0           3         0         0      3     8
Leadership      88          62         0         0     78    61
Lore            90          63         0         0     67    64
Neutral         30          40       191        14     28     2
Spirit          91          59         0         0     67    64
Tactics         80          66         0         0     73    61

type_name     Player Objective  Player Side Quest
sphere_name
Baggins                      0                  0
Fellowship                   0                  0
Leadership                   0                  2
Lore                         0                  2
```

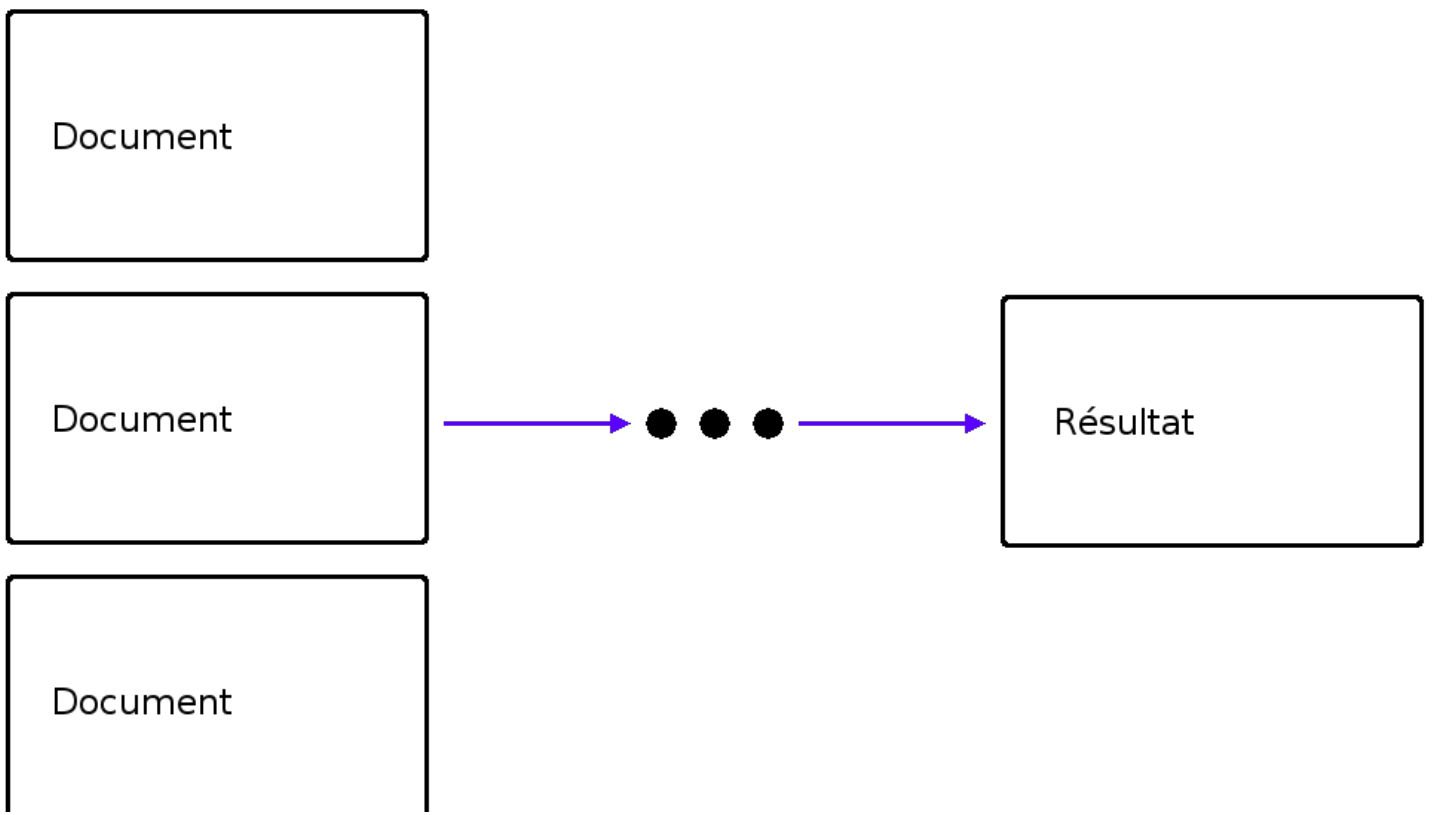
Avec Pandas

```
df = pd.DataFrame(
    rings.find( # Tout est retourné sans filter
        projection={"_id": False, "sphere_name": True, "type_name":
    )
)
# La fonction crosstab calcule les effectifs croisés
print(pd.crosstab(df.sphere_name, df.type_name))
```

- C’est plus propre ! 🤠
- Il n’y a plus qu’une seule requête mais elle retourne un objet dont on ne contrôle pas la taille.
- Tous les calculs se font côté client.

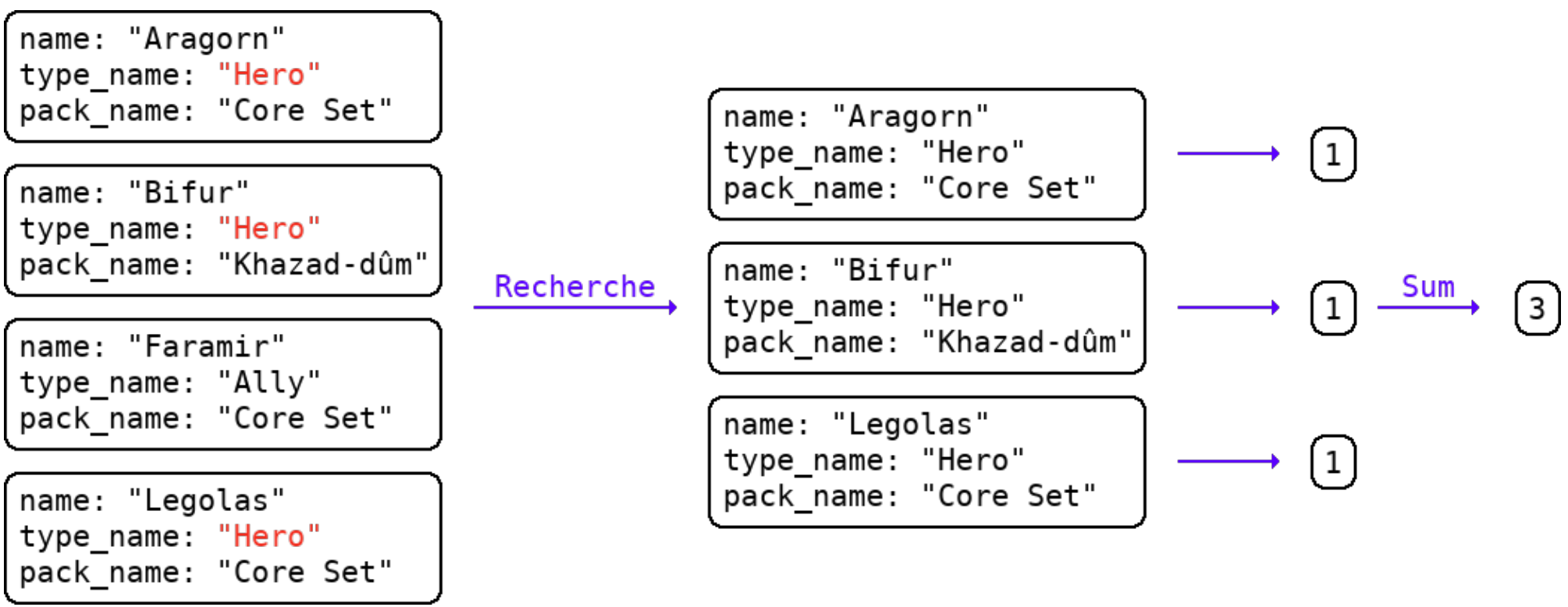
Agrégateurs

Un **agrégateur** regroupe les valeurs contenues dans plusieurs documents sélectionnés et retourne une structure contenant des objets “simples” et “plus informatifs”.



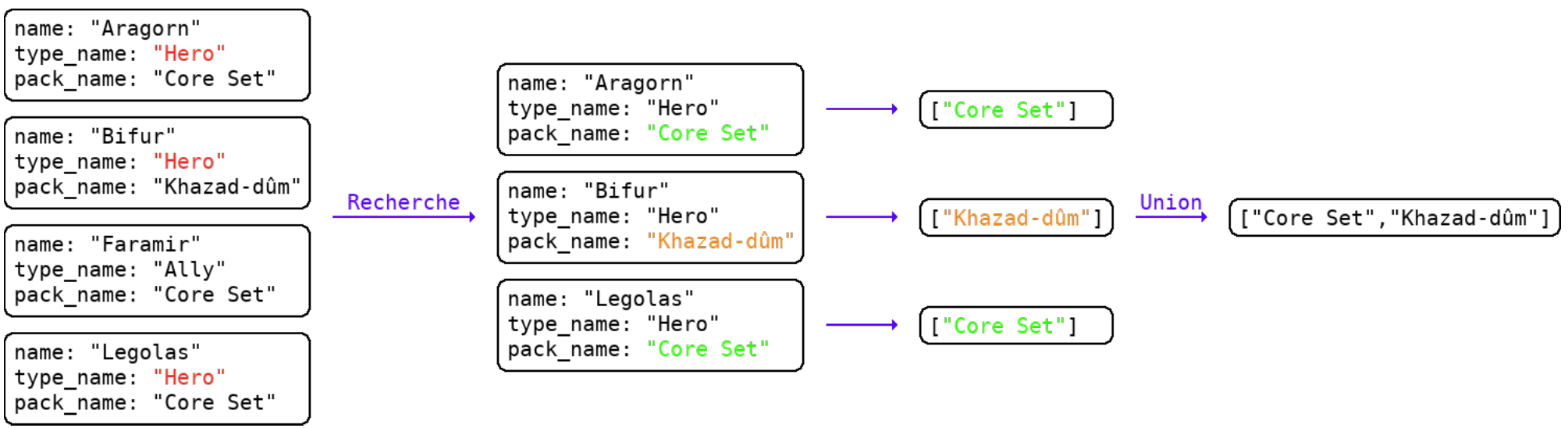
Les méthodes `count_documents` et `distinct` sont des agrégateurs.

```
1  rings.count_documents({"type_name": "Hero"})
```



Les méthodes `count_documents` et `distinct` sont des agrégateurs.

```
1  rings.distinct(key="pack_name", filter={"type_name": "Hero"})
```



Pipeline d'agrégation

Le **pipeline d'agrégation** de MongoDB est une séquence de **stages** à traverser pour transformer des documents en un résultat agrégé.

Les stages filtrent, transforment, groupent, trient, … les documents dans un ordre établi. Par exemple :

- `$match` recherche comme avec `find`,
- `$group` regroupe et accumule,
- `$sort` trie.

Pour les autres stages (`$project`, `$limit`, …) :

https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/

La méthode `aggregate` permet de construire un agrégateur basé sur le modèle du pipeline d'agrégation. Cette méthode prend la liste des différents stages à réaliser en argument.

- Exemple comme `count_documents` :

```
print(pd.DataFrame(
    rings.aggregate([
        {"$match": {"type_name": "Hero"}},
        {"$group": {"_id": None, "count": {"$sum": 1}}},
    ])
))
```

```
    _id  count
0  None    263
```

```
rings.aggregate([
    {"$match": {"type_name": "Hero"}},
    {"$group": {"_id": None, "count": {"$sum": 1}}},
])
```

- `$match` est similaire à l'argument `filter` de `find`,
- le champ `_id` de `$group` reçoit la clé utilisée pour les groupes ou `None` pour considérer tous les documents,
- le champ `count` de `$group` est le nom de l'accumulateur défini en suivant,
- la valeur des accumulateurs est maintenue groupe par groupe,
- la définition des accumulateurs est évaluée pour chaque document.

- Exemple comme `dictinct` :

```
print(pd.DataFrame(
    rings.aggregate([
        {"$group": {"_id": "$sphere_name"}},
    ])
))
```

```
          _id
0     Tactics
1     Neutral
2  Leadership
3        Lore
4      Spirit
5     Baggins
6  Fellowship
```

- Exemple comme `dictinct` :

```
print(pd.DataFrame(
    rings.aggregate([
        {"$group": {"_id": "$sphere_name"}},
    ])
))
```

ou bien

```
print(pd.DataFrame(
    rings.aggregate([
        {
            "$group": {
                "_id": None,
                "sphere_name": {"$addToSet": "$sphere_name"}
            }
        },
    ])
))
```

```
    _id                                                sphere_name
0  None  [Tactics, Neutral, Leadership, Lore, Spirit, B...
```

Tri

Le stage `$sort` permet de trier les résultats :

```
print(pd.DataFrame(
    rings.aggregate([
        {"$group": {"_id": "$sphere_name", "count": {"$sum": 1}}},
        {"$sort": {"count": pymongo.DESCENDING}},
    ])
))
```

	_id	count
0	Neutral	309
1	Leadership	291
2	Lore	286
3	Spirit	283
4	Tactics	282
5	Fellowship	14
6	Baggins	5

Table de contingence

Les effectifs croisés peuvent donc être totalement calculés côté MongoDB en adaptant `_id` :

```
table_df = pd.DataFrame(
    rings.aggregate([
        {
            "$group": {
                "_id": {
                    "sphere_name": "$sphere_name",
                    "type_name": "$type_name",
                },
                "count": {"$sum": 1}
            }
        }
    ])
)
```

```
1 # Résultat brut
2 print(table_df)
```

```
                                                  _id  count
0   {'sphere_name': 'Lore', 'type_name': 'Player S...      2
1     {'sphere_name': 'Spirit', 'type_name': 'Event'}     67
2      {'sphere_name': 'Spirit', 'type_name': 'Ally'}     91
3   {'sphere_name': 'Spirit', 'type_name': 'Attach...     59
4     {'sphere_name': 'Tactics', 'type_name': 'Hero'}     61
5   {'sphere_name': 'Leadership', 'type_name': 'He...     61
6   {'sphere_name': 'Neutral', 'type_name': 'Campa...    191
7   {'sphere_name': 'Fellowship', 'type_name': 'At...      3
8     {'sphere_name': 'Baggins', 'type_name': 'Hero'}      3
9   {'sphere_name': 'Fellowship', 'type_name': 'Ev...      3
10   {'sphere_name': 'Baggins', 'type_name': 'Event'}      2
11  {'sphere_name': 'Neutral', 'type_name': 'Attac...     40
12  {'sphere_name': 'Leadership', 'type_name': 'At...     62
13  {'sphere_name': 'Lore', 'type_name': 'Attachme...     63
14    {'sphere_name': 'Neutral', 'type_name': 'Hero'}      2
```

```
print( # Un peu de mise en forme
    table_df
    .assign(
        sphere_name=table_df["_id"].apply(lambda d: d["sphere_name"
        type_name=table_df["_id"].apply(lambda d: d["type_name"]),
    )
    .pivot(index="sphere_name", columns="type_name", values="count"
    .fillna(0).astype(int)
)
```

type_name	Ally	Attachment	Campaign	Contract	Event	Hero
sphere_name						
Baggins	0	0	0	0	2	3
Fellowship	0	3	0	0	3	8
Leadership	88	62	0	0	78	61
Lore	90	63	0	0	67	64
Neutral	30	40	191	14	28	2
Spirit	91	59	0	0	67	64
Tactics	80	66	0	0	73	61

type_name	Player Objective	Player Side Quest
sphere_name		
Baggins	0	0
Fellowship	0	0
Leadership	0	2
Lore	0	2

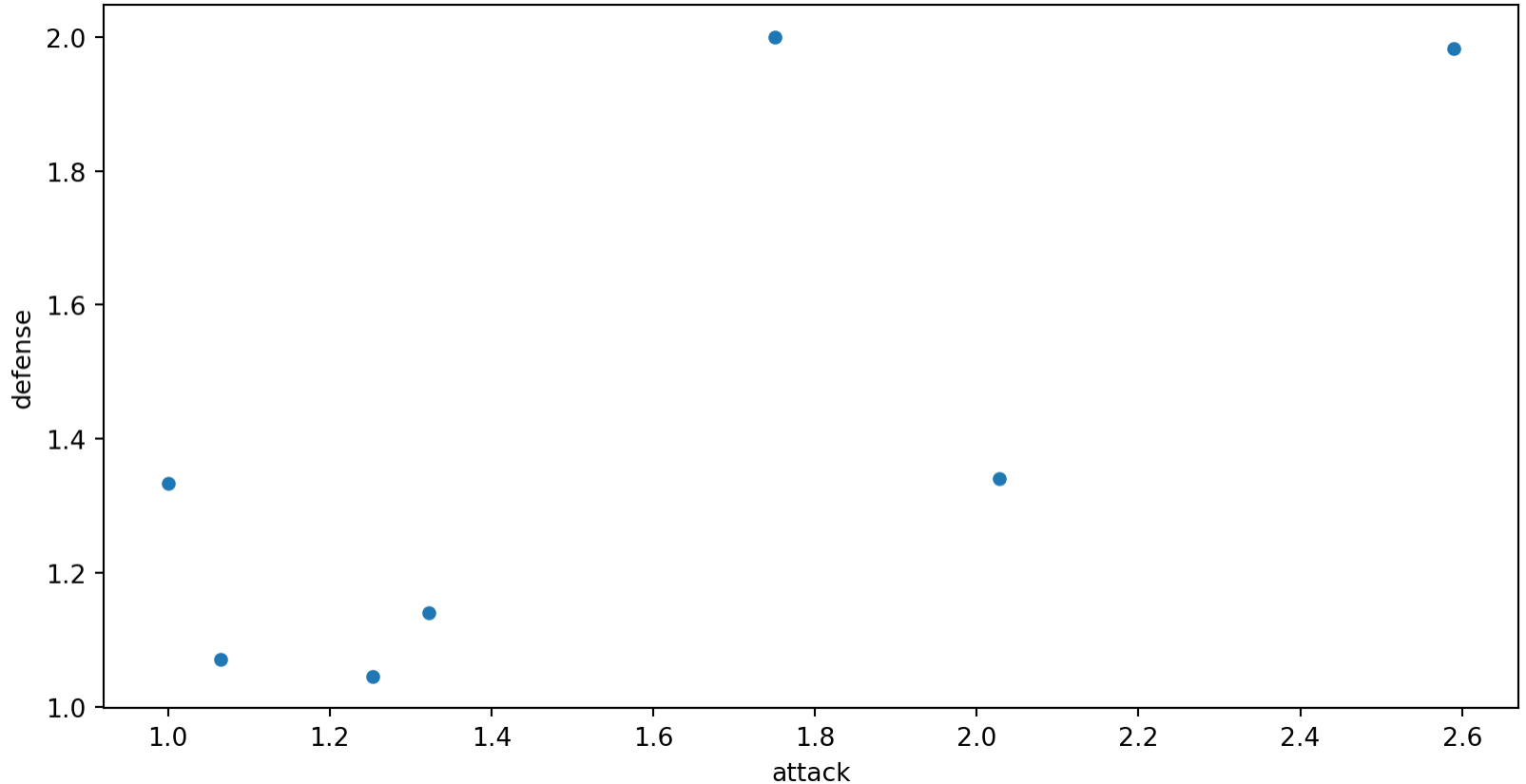
Opérateurs d'agrégation

MongoDB offre des opérateurs pour des pipelines d'agrégation plus avancés.

```
print(pd.DataFrame(
    rings.aggregate([
        {
            "$group": {
                "_id": "$sphere_name",
                "attack": {"$avg": "$attack"}, # Moyenne
                "defense": {"$avg": "$defense"}, # Moyenne
            }
        }
    ])
))
```

	_id	attack	defense
0	Tactics	2.028369	1.340426
1	Neutral	2.590164	1.983051
2	Leadership	1.322148	1.140940
3	Lore	1.253247	1.045455
4	Spirit	1.064516	1.070968
5	Baggins	1.000000	1.333333

```
pd.DataFrame(
    rings.aggregate([
        {
            "$group": {
                "_id": "$sphere_name",
                "attack": {"$avg": "$attack"}, # Moyenne
                "defense": {"$avg": "$defense"}, # Moyenne
            }
        }
    ])
).plot.scatter(x="attack", y="defense")
```


2.0
1.8
1.6
1.4
1.2
1.0
defense
1.0
1.2
1.4
1.6
1.8
2.0
2.2
2.4
2.6
attack

Beaucoup d'opérateurs d'agrégation à découvrir :

```
print(pd.DataFrame(
    rings.aggregate([
        {"$match": {"threat": {"$exists": True}}},
        {
            "$project": {
                "niveau": {
                    "$cond": [
                        {"$lt": ["$threat", 10]},
                        "Faiblard",
                        "Balèze"
                    ]
                }
            }
        },
        {"$group": {"_id": "$niveau", "count": {"$sum": 1}}}
    ])
))
```

```
        _id  count
0  Faiblard    193
1    Balèze     79
```

https://www.mongodb.com/docs/manual/reference/operator/aggregation/

Limites du pipeline d'agrégation

- Les documents produits par `aggregate` sont limités 16Mb qui est la limite de taille d'un document BSON. Cette limite ne s'applique pas aux documents intermédiaires entre les stages, seulement à ceux produits par le pipeline.
- Le nombre de stages est limité à 1000.
- Un stage est limité à 100Mb de mémoire, ce qui peut poser problème pour de grands jeux de données. Certains stages (`$group`, `$sort`, ...) offrent un mécanisme de *swap* pour aller au-delà avec l'argument `allowDiskUseByDefault`.

À vous de jouer !