epsi

l'école d'ingénierie
informatique

Amazing

MSPR - Bloc 2

RÉALISÉ PAR :

AHMED BOUSLAMA
ALEXANDRE CANTON CONDES
BENJAMIN HOARAU
SAMIR HOUBAD
ERIC MAJRI

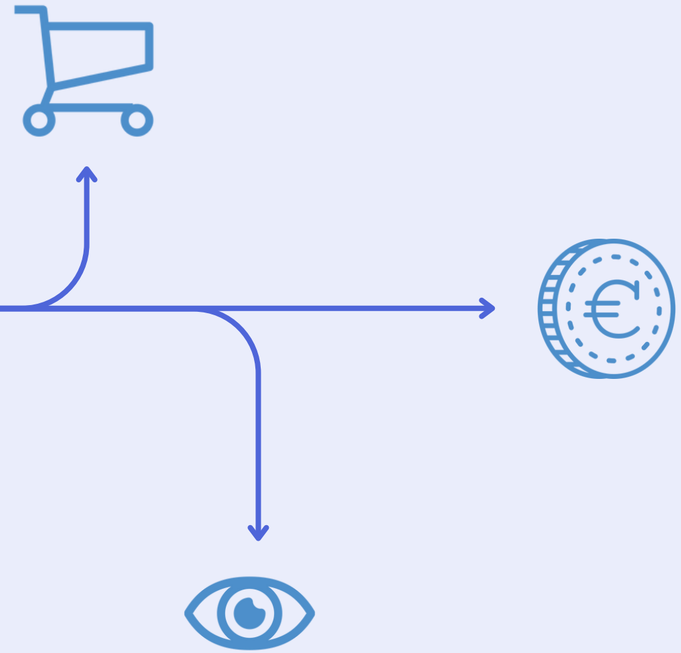
ANNÉE UNIVERSITAIRE 2023-2024

Sommaire

Contexte du projet

Amazing

SEGMENTER
Des profils types actionnables



Méthodologie utilisée

NETTOYER
les données

1

CONSTRUIRE
des indicateurs

2

RÉDUIRE
les dimensions

3

DÉTERMINER
le nombre de cluster optimal

4

ENTRAINER
le modèle

5

6

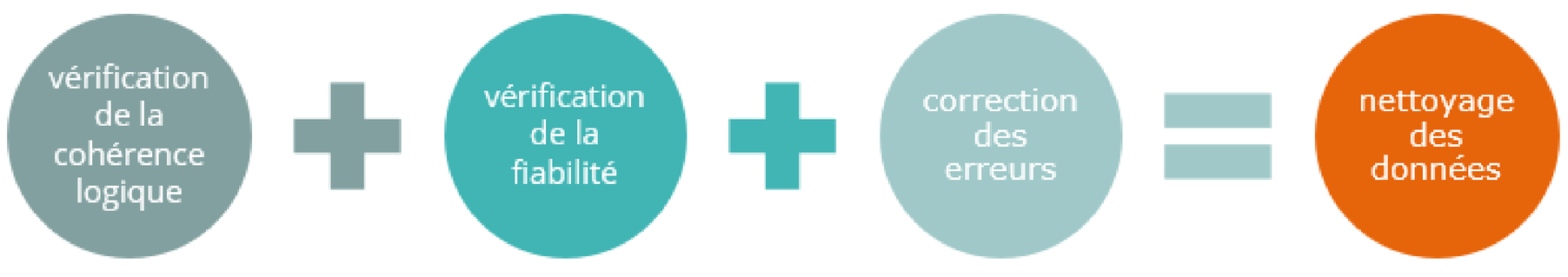
INTERPRÉTER
&
VISUALISER

DÉPLOYER
le modèle

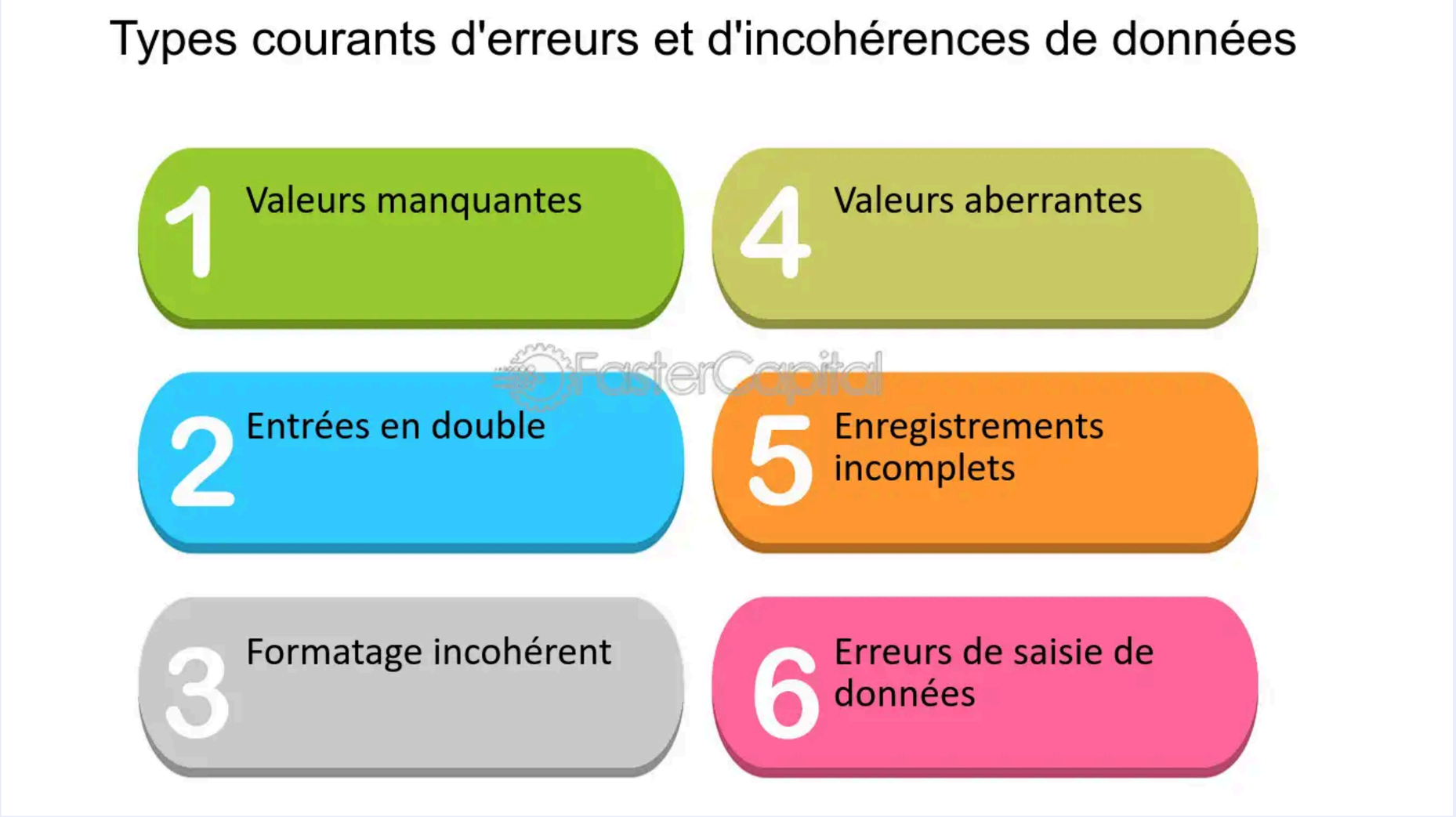
7

Préparation et Analyse des Données

Nettoyage du jeu de données

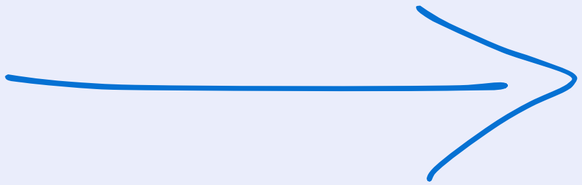


Nettoyage du jeu de données



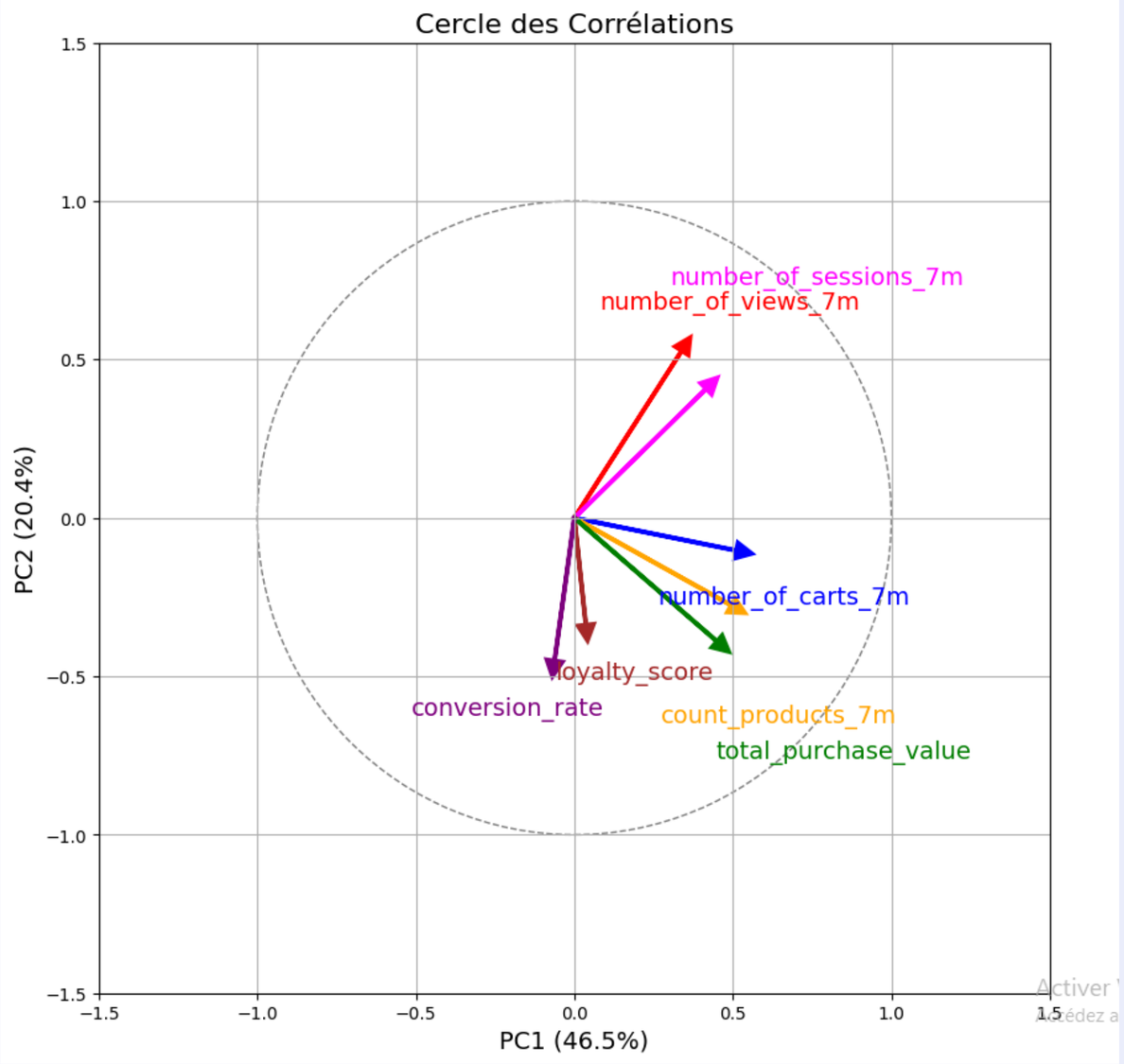
CONSTRUIRE

Column
event_time
event_type
product_id
category_id
category_code
brand
price
user_id
user_session



```
user_id                          int32
number_of_views_2m               int64
number_of_carts_2m               int64
number_of_sessions_2m            int64
count_products_2m                int64
avg_price_2m                   float64
number_of_views_5m               int64
number_of_carts_5m               int64
number_of_sessions_5m            int64
count_products_5m                int64
avg_price_5m                   float64
number_of_views_7m               int64
number_of_carts_7m               int64
number_of_sessions_7m            int64
count_products_7m                int64
avg_price_7m                   float64
last_purchase            datetime64[ns]
days_since_last_purchase         int32
total_purchase_value           float64
cart_abandonments                int64
preferred_brand                 object
preferred_category              object
most_active_time                object
most_active_day                 object
```

Réduction des dimensions - ACP



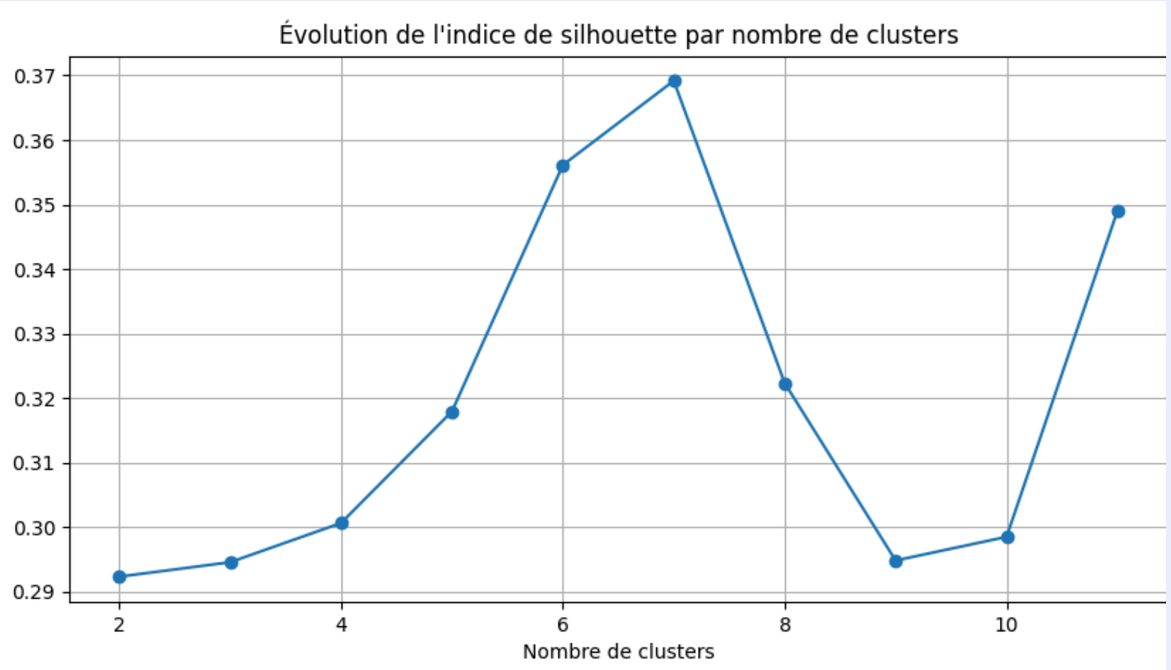
- **Variance expliquée par PC1: 46.51%**

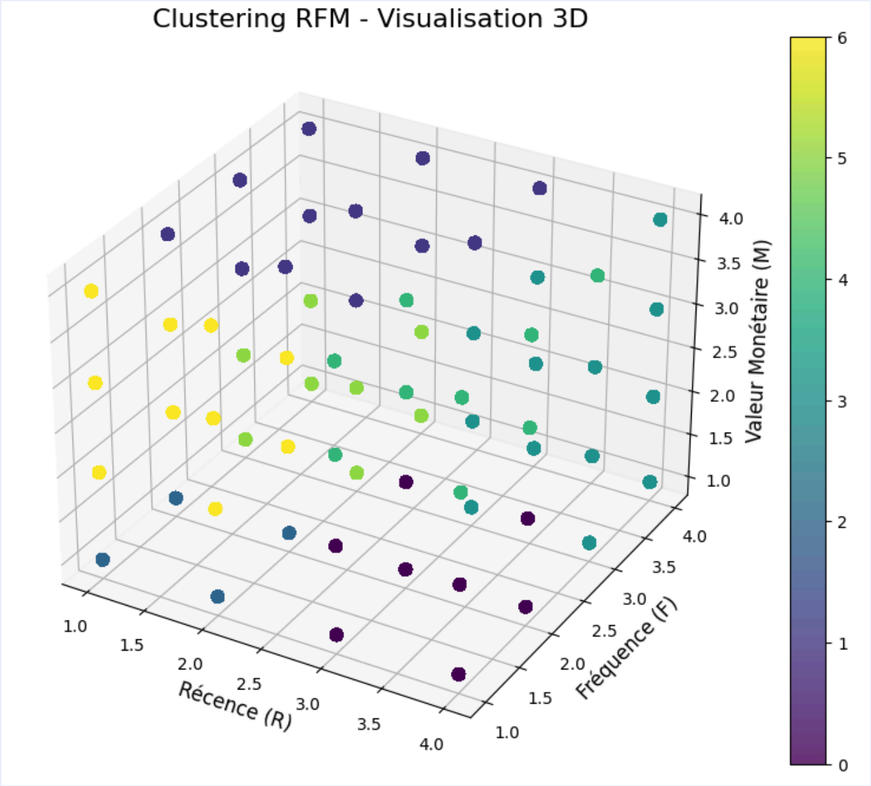
- **Variance expliquée par PC2: 20.44%**

Conception et optimisation du modèle

Visualisation du Clustering RFM et Évolution de l'Indice de Silhouette

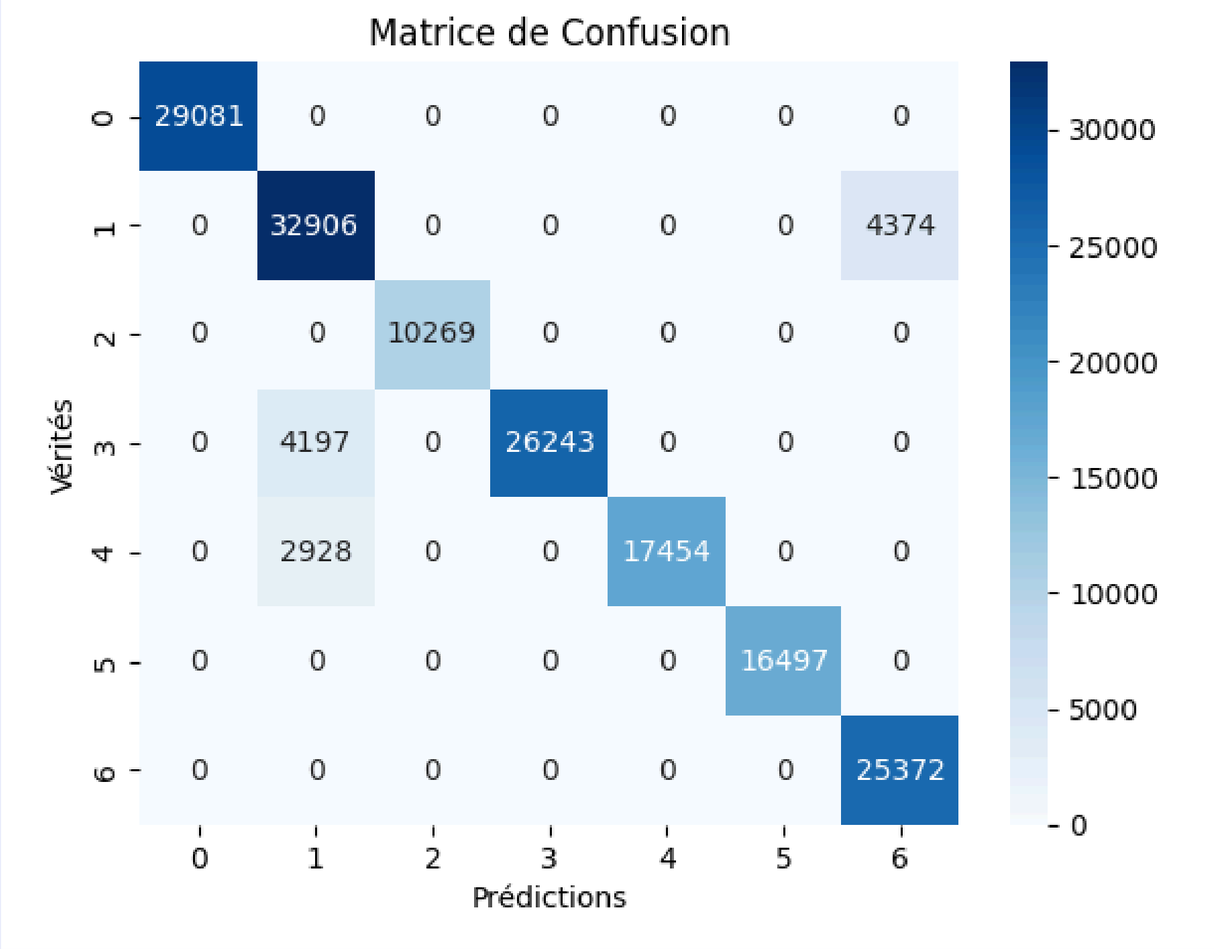
Évolution de l'indice de silhouette :
Meilleure qualité de clustering observée avec 7 clusters (indice de silhouette maximal).
Le graphique montre l'optimisation du nombre de clusters pour une segmentation efficace.





Visualisation 3D des clusters RFM :
Récence (R), Fréquence (F) et Valeur Monétaire (M) des clients visualisées
Clusters bien définis avec une répartition claire sur les trois dimensions.

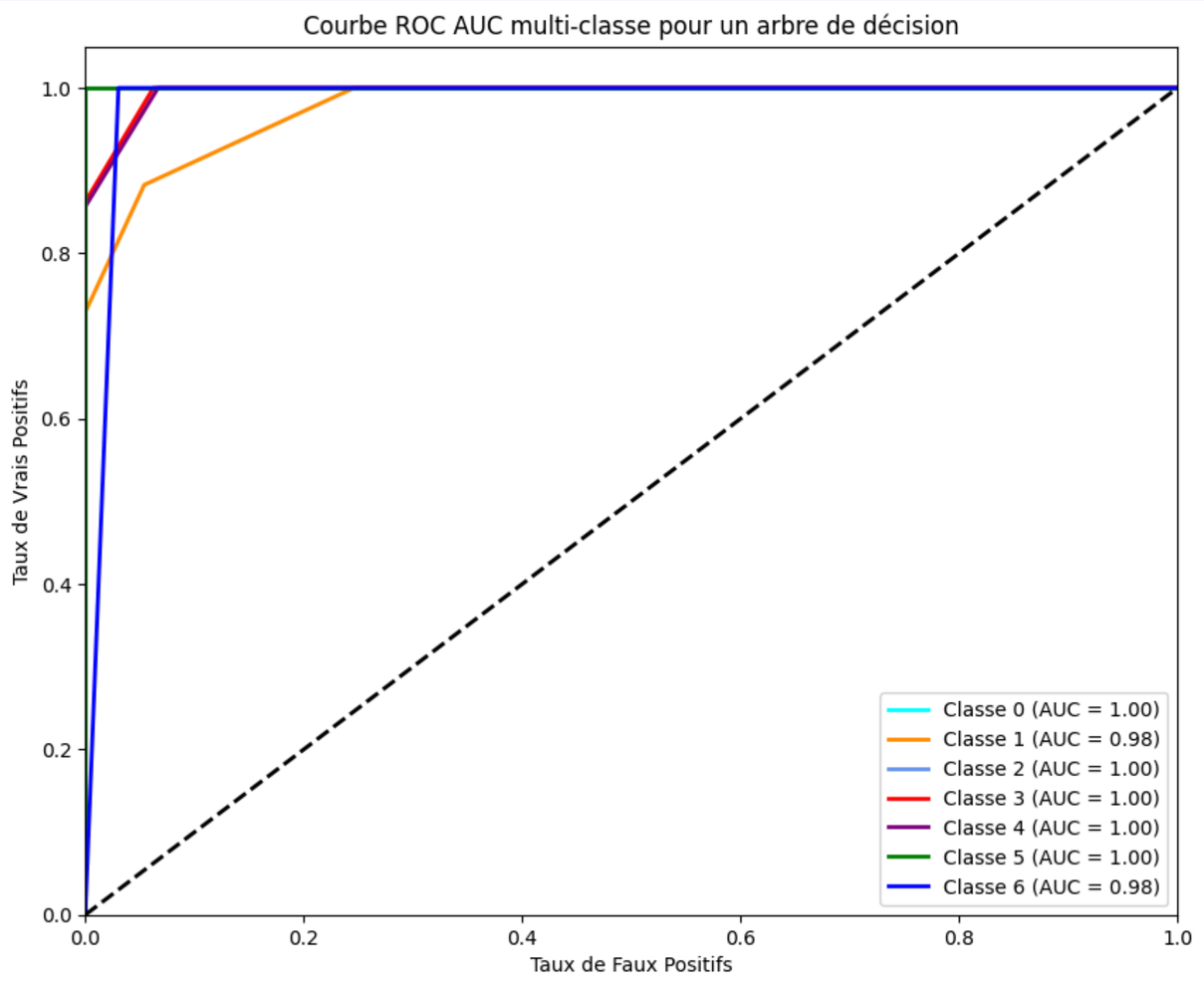
Matrice de Confusion - Analyse des Prédictions



- **Classe 0** : Excellent modèle avec 29081 prédictions correctes.
- **Classe 1** : 32906 corrects, mais 4374 erreurs classées dans la classe 6.
- **Classe 3** : 26243 corrects, mais 4197 erreurs vers la classe 1.
- **Classe 4** : 17454 corrects, mais 2928 erreurs vers la classe 1.
- **Autres classes** : Prédictions majoritairement correctes, erreurs minimes.

	Precision	Recall	F1-score	Support
0	1.00	1.00	1.00	29081
1	0.82	0.88	0.85	37280
2	1.00	1.00	1.00	10269
3	1.00	0.86	0.93	30440
4	1.00	0.86	0.92	20382
5	1.00	1.00	1.00	16497
6	0.85	1.00	0.92	25372
Accuracy			**0.93**	169321
Macro avg	0.95	0.94	0.95	169321
Weighted avg	0.94	0.93	0.93	169321

Courbe ROC AUC - Arbre de Décision Multiclasse

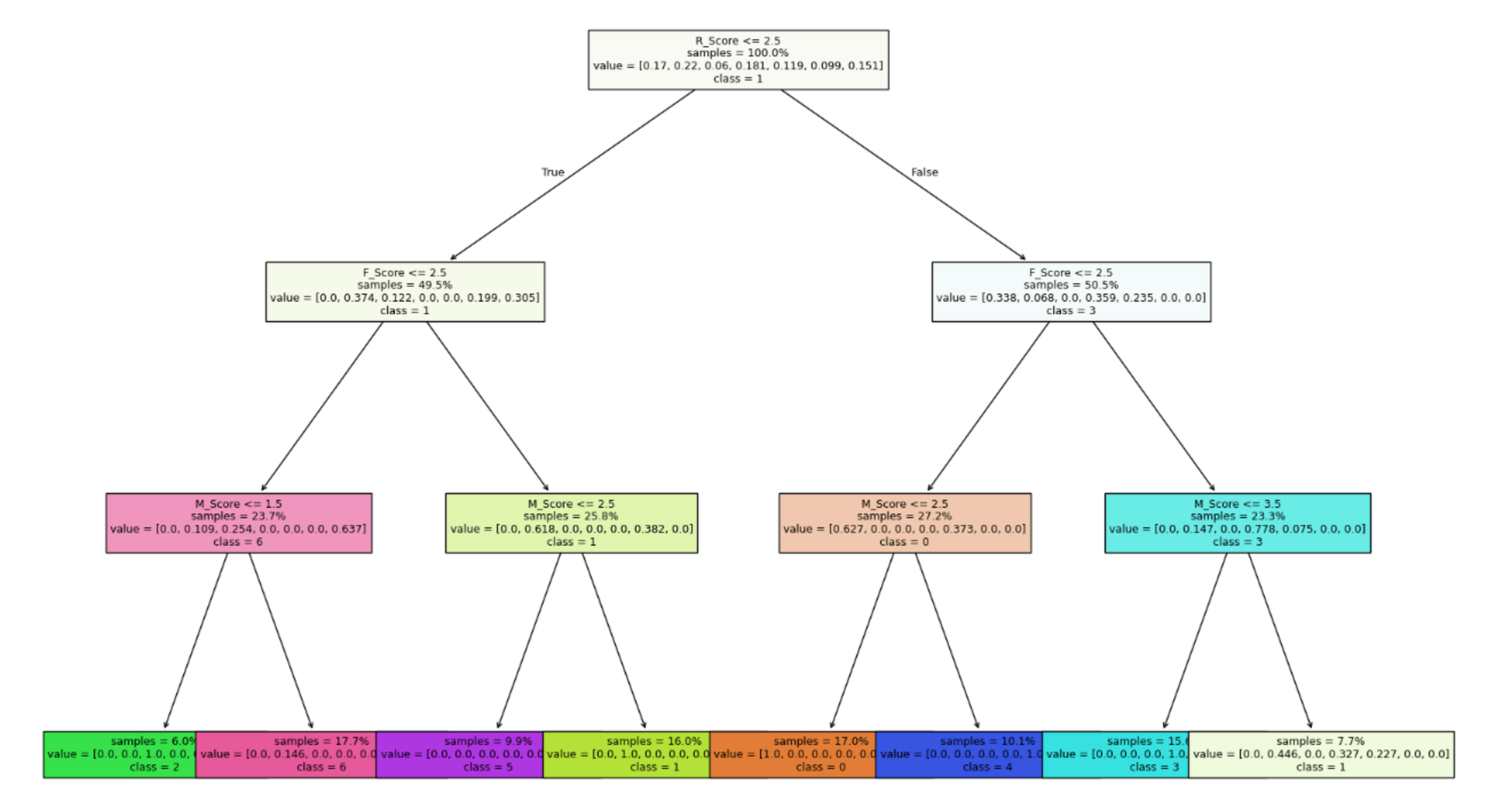


Outil clé pour évaluer la capacité du modèle à discriminer entre les classes, ici montrant un modèle performant

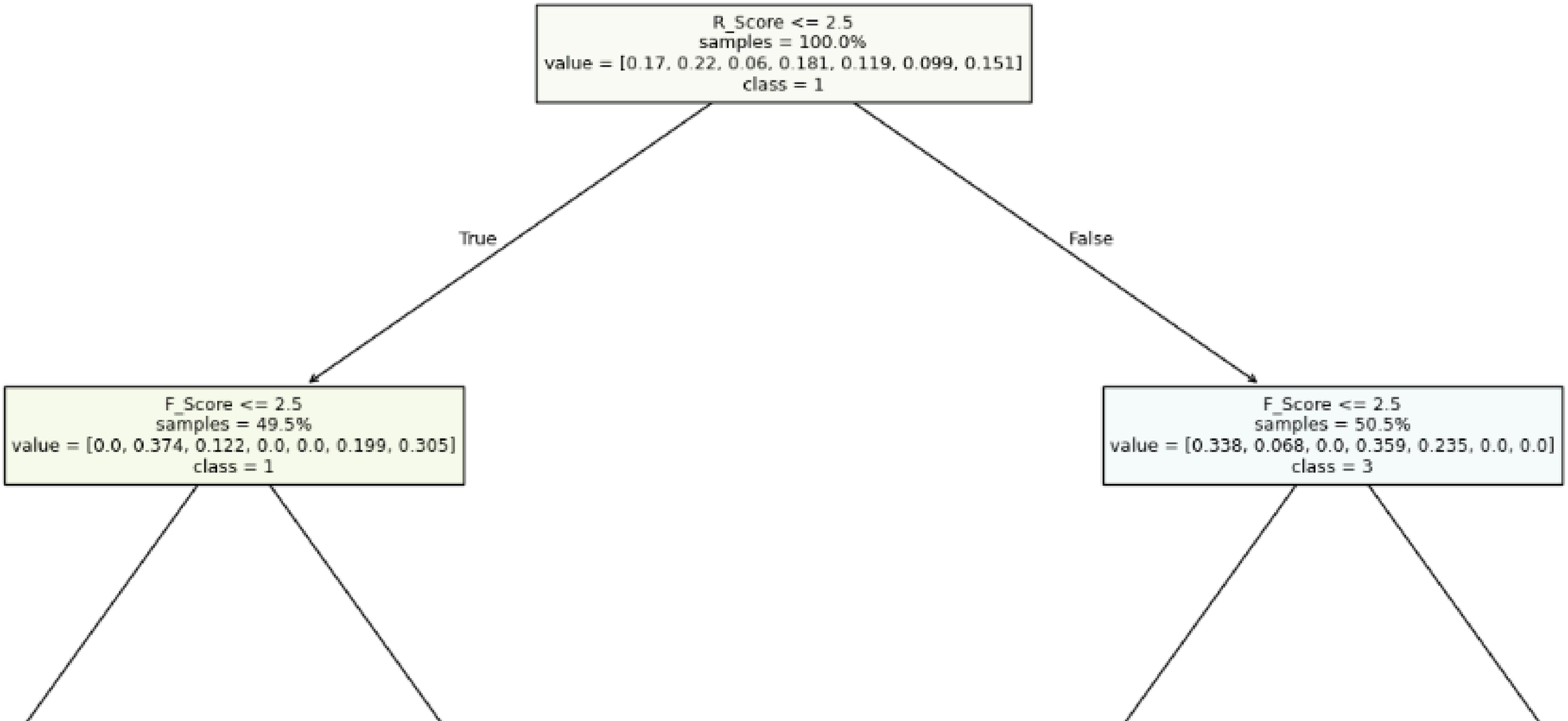
- **Performances excellentes** avec des AUC proches de 1.00 pour toutes les classes.
- **Classes 0, 2, 3, 4, 5** : AUC parfait de 1.00, indiquant **une séparation idéale.**
- **Classes 1 et 6** : AUC de 0.98, **avec de légères erreurs de classification, mais toujours très performantes.**

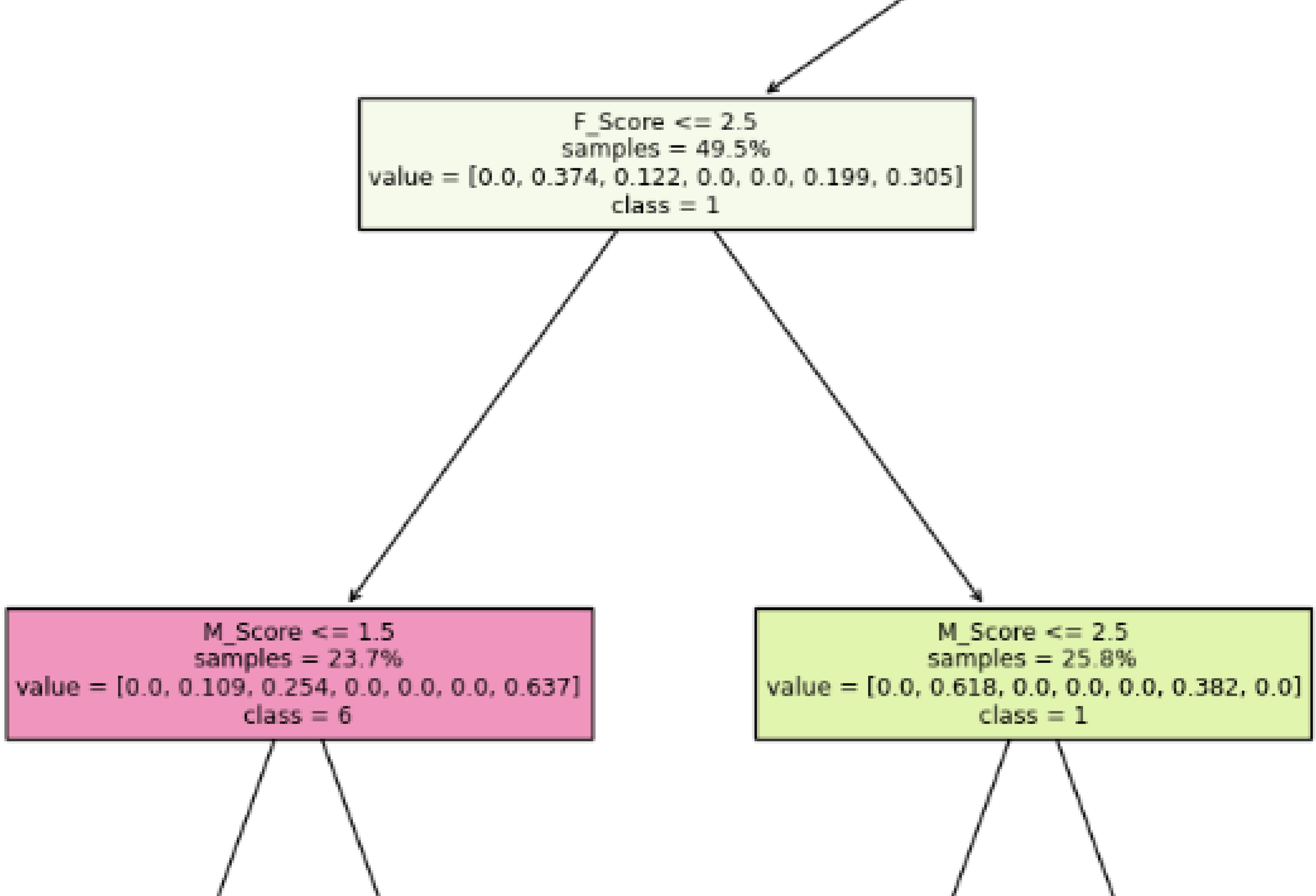
Analyse des catégories finales

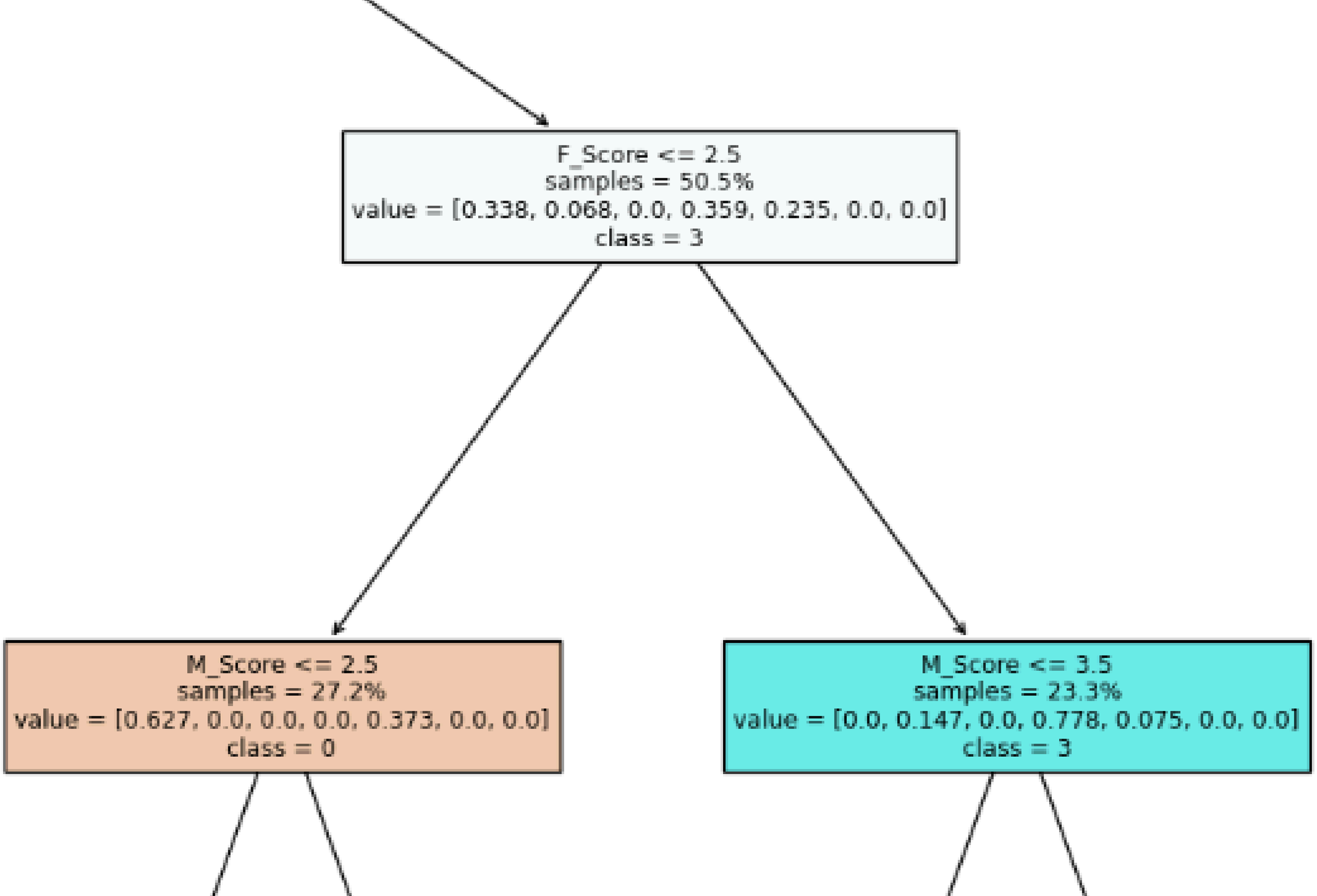
Arbre de décision



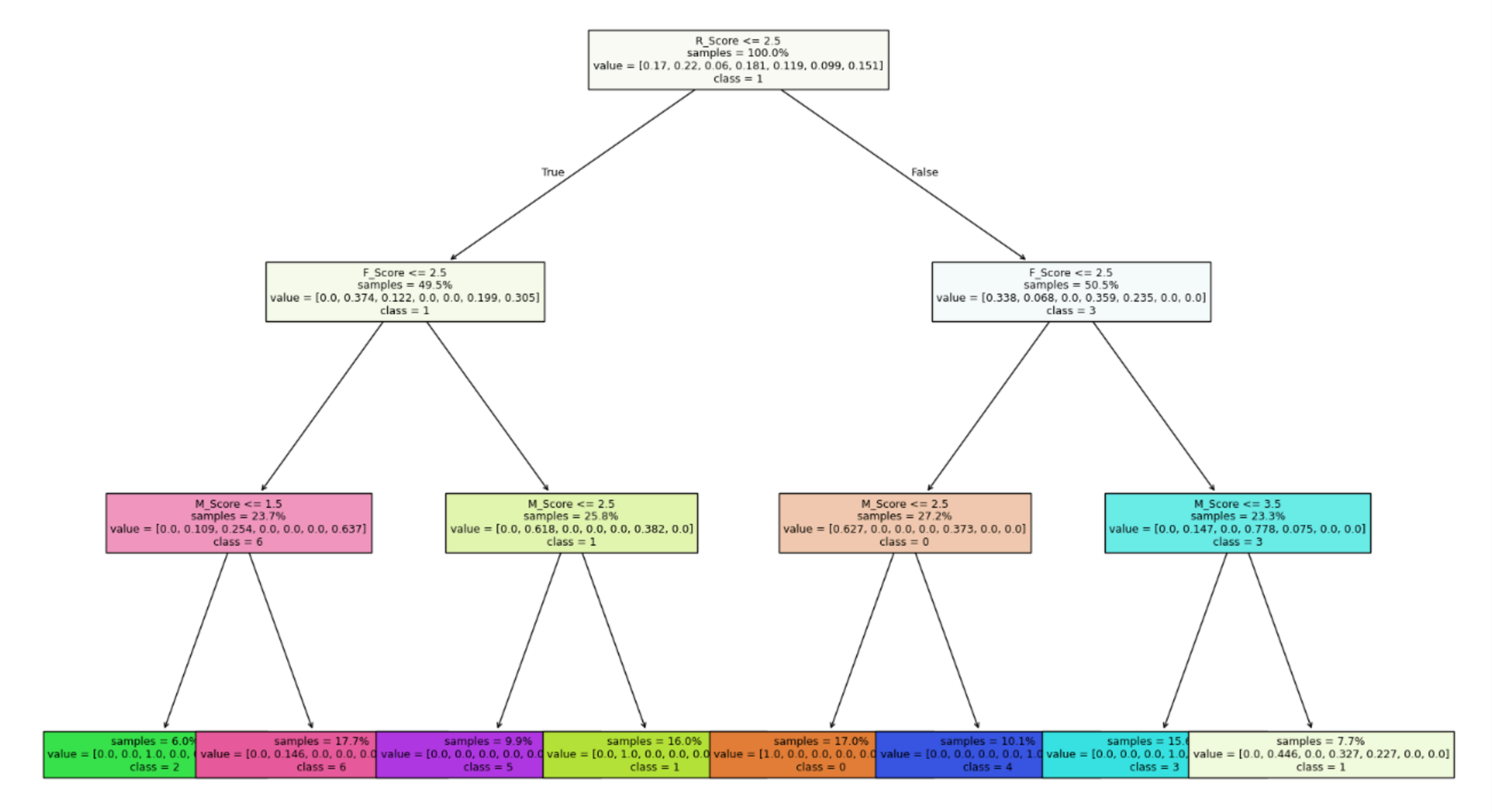
Arbre de décision




F_Score <= 2.5
samples = 49.5%
value = [0.0, 0.374, 0.122, 0.0, 0.0, 0.199, 0.305]
class = 1
M_Score <= 1.5
samples = 23.7%
value = [0.0, 0.109, 0.254, 0.0, 0.0, 0.0, 0.637]
class = 6
M_Score <= 2.5
samples = 25.8%
value = [0.0, 0.618, 0.0, 0.0, 0.0, 0.382, 0.0]
class = 1


F_Score <= 2.5
samples = 50.5%
value = [0.338, 0.068, 0.0, 0.359, 0.235, 0.0, 0.0]
class = 3
M_Score <= 2.5
samples = 27.2%
value = [0.627, 0.0, 0.0, 0.0, 0.373, 0.0, 0.0]
class = 0
M_Score <= 3.5
samples = 23.3%
value = [0.0, 0.147, 0.0, 0.778, 0.075, 0.0, 0.0]
class = 3

Arbre de décision



Clusters Finales

Lion

- Puissants,
- Dominants,
- Récents,
- Grandes Valeurs Marques

Ours

- Fidèles,
- Réguliers,
- Modérés,
- Stables

Écureuils

- Occasionnels,
- Petits dépensiers,
- Discrets,
- Potentiellement activables

Renards

- Discrets,
- Stratégiques,
- Grands dépensiers,
- À cibler

Castors

- Actifs
- Prévisibles ,
- Modérés,
- Fiables

Chouettes

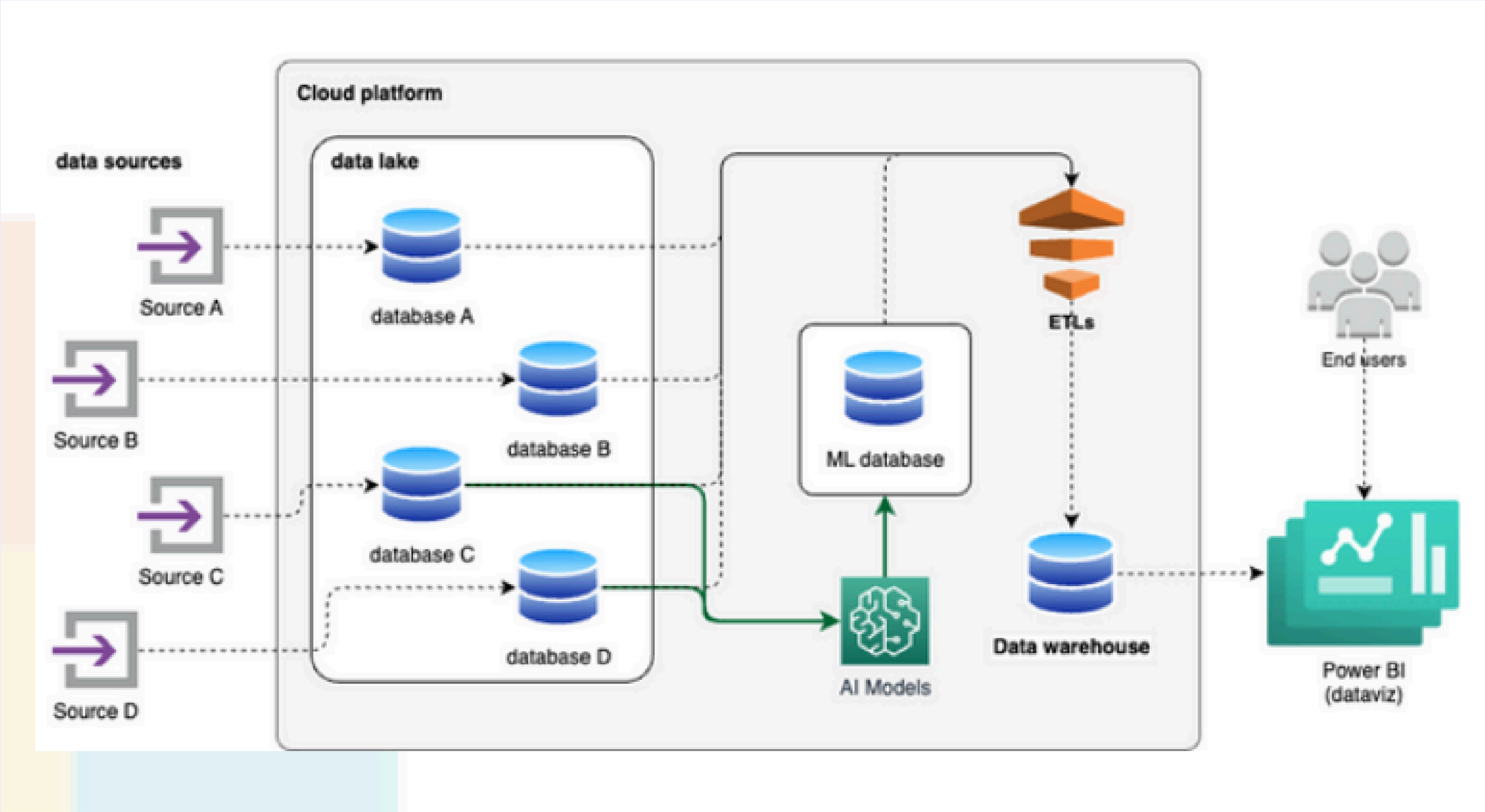
- Discrets,
- Silencieux,
- Modérés,
- Réactivables

Tortues

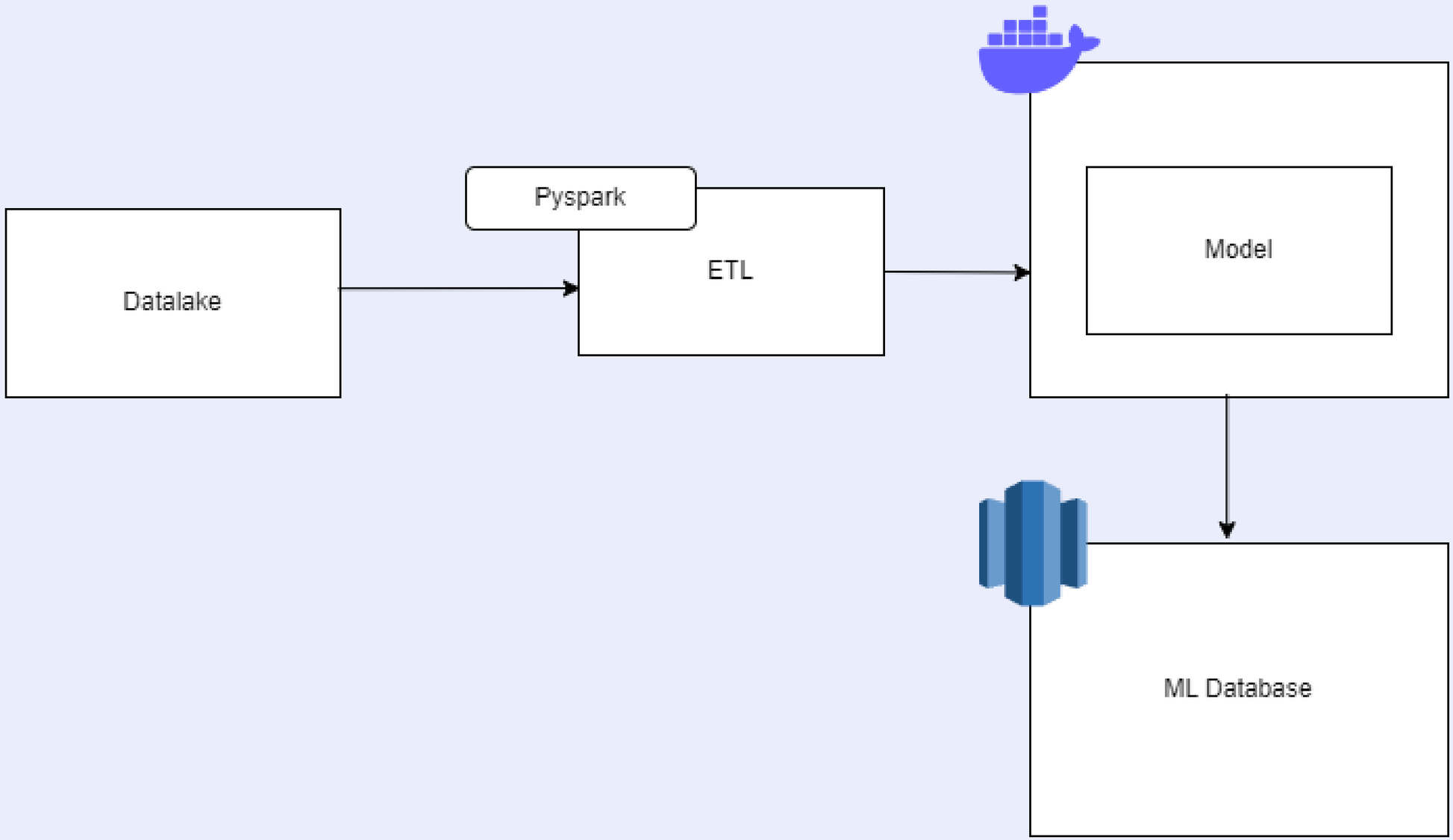
- Lents,
- Dépensiers,
- Inactifs,
- À fort potentiel de réengagement

Architecture et déploiement

Architecture Cloud d'Amazing



Architecture ML



Dockerizer du Modèle

Dockerfile

```dockerfile
# Utiliser une image de base officielle de Python 3.10
FROM python:3.10

# Définir un répertoire de travail dans le conteneur pour votre application
WORKDIR /app

# Copier le fichier .parquet dans le conteneur à l'emplacement approprié
COPY ./data/full_df_output.parquet /app/data/full_df_output.parquet

# Copier le script Python principal dans le conteneur
COPY ./5_RMF_full_data.py /app

# Copier le fichier requirements.txt pour gérer les dépendances Python
COPY ./requirements.txt .

# Installer les dépendances Python spécifiées dans requirements.txt sans utiliser le cache
RUN pip install --no-cache-dir -r requirements.txt

# Spécifier la commande à exécuter au démarrage du conteneur : exécuter le script Python
CMD ["python", "5_RMF_full_data.py"]
```

Docker compose

```yaml
version: '3.8'

services:
  ml_model:
    build: .
    ports:
      - "8888:8888"
    tty: true
```

Déploiement sur AWS ECS

```yaml
name: ML Docker Image CI

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:

  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v4
    - name: Build the Docker image
      run: docker build . --file Dockerfile --tag ml_model:$(date +%s)
```

```yaml
deploy:
  name: Deploy
  runs-on: ubuntu-latest
  environment: production

  steps:
  - name: Checkout
    uses: actions/checkout@v4

  - name: Configure AWS credentials
    uses: aws-actions/configure-aws-credentials@v1
    with:
      aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
      aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
      aws-region: ${{ env.AWS_REGION }}

  - name: Login to Amazon ECR
    id: login-ecr
    uses: aws-actions/amazon-ecr-login@v1

  - name: Build, tag, and push image to Amazon ECR
    id: build-image
    env:
      ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
      IMAGE_TAG: ${{ github.sha }}
    run: |
      # Build a docker container and
      # push it to ECR so that it can
      # be deployed to ECS.
      docker build -t $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG .
      docker push $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG
      echo "image=$ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG" >> $GITHUB_OUTPUT
```

Conclusion et Perspectives