



## UNIVERSITÉ GALATASARAY

Faculté d'Ingénierie et de Technologie

Département d'Informatique

## INF438 BASES DE DONNÉES AVANCÉES

---

### Projet Final

Analyse E-Sport (Dota 2) sur  
Architecture Lakehouse Azure

---

### GROUPE 3

**Membre 1 :** Sabri Taner Burak ALTINDAL – 22401030

**Membre 2 :** Emirhan Karatepe – 19401830

**Membre 3 :** Kaan Çolakoğlu – 21401946

**Membre 4 :** Ceren Akbaş – 22401028

**Année Académique 2025–2026**

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Contexte du Projet . . . . .	4
1.2	Jeu de Données Utilisé . . . . .	4
1.3	Objectifs du Projet . . . . .	4
1.4	Services Azure Utilisés . . . . .	5
<b>2</b>	<b>Conception Architecturale</b>	<b>5</b>
2.1	Medallion Architecture . . . . .	6
2.2	Structure Azure Data Lake Storage Gen2 . . . . .	7
2.3	Diagramme de Flux de Données . . . . .	7
<b>3</b>	<b>Implémentation</b>	<b>8</b>
3.1	Pipeline de Traitement Batch . . . . .	8
3.2	Configuration du Déclencheur . . . . .	9
3.3	Notebooks Databricks . . . . .	10
3.4	Simulation de Streaming . . . . .	10
3.5	Transformations ETL . . . . .	11
3.6	Gestion des Tables Delta Lake . . . . .	13
<b>4</b>	<b>Analyse des Données et Machine Learning</b>	<b>13</b>
4.1	Accès aux Tables Gold et Exécution SQL (Databricks) . . . . .	14
4.2	Requêtes SQL Analytiques (Résultats) . . . . .	15
4.3	Visualisation Power BI . . . . .	20
4.4	Modèle de Machine Learning . . . . .	24
4.5	Résultats du Modèle . . . . .	25
4.6	Analyse de l'Importance des Features . . . . .	25
4.7	Suivi MLflow . . . . .	26
<b>5</b>	<b>Difficultés Rencontrées et Solutions</b>	<b>26</b>
5.1	Limites des Ressources Azure . . . . .	27
5.2	Collision de Fichiers dans la Simulation de Streaming . . . . .	27
5.3	Volume de Données et Contraintes de Coûts . . . . .	27
5.4	Gestion des Clés Azure Storage . . . . .	27
5.5	Complexité de la Gestion des Accès (IAM) . . . . .	27
5.6	Problème d'accès au cluster Databricks . . . . .	28
<b>6</b>	<b>Conclusion</b>	<b>28</b>
6.1	Résumé du Projet . . . . .	29
6.2	Leçons Apprises . . . . .	29
6.3	Conclusion Générale . . . . .	29
	<b>Annexes</b>	<b>30</b>

# Table des figures

1	Vue Azure Portal de la structure des dossiers Bronze/Silver/Gold dans ADLS Gen2 . . . . .	7
2	Diagramme de flux de données . . . . .	8
3	Détails d'exécution de l'activité <i>Copy Data</i> confirmant l'ingestion réussie des 4 fichiers sources (212 MB) vers le conteneur Bronze. . . . .	8
4	Diagramme du pipeline pl_master_esports dans Azure Data Factory . . . . .	9
5	État “Succeeded” et “Triggered by tr_daily_esports” dans l'écran ADF Monitor . . . . .	10
6	Sortie d'exécution du code stream_simulator.py . . . . .	11
7	Sortie de la classification Tier dans le notebook silver_to_gold . . . . .	12
8	Liste des tables sous le schéma esports_gold dans le Databricks Catalog . . . . .	13
9	Chargement des tables Gold depuis ADLS Gen2 et création des vues temporaires dans Databricks . . . . .	15
10	Résultat SQL : Top 10 joueurs par KDA moyen (avec kills, deaths, assists) . . . . .	16
11	Résultat SQL : aperçu des statistiques quotidiennes (match_count et durée moyenne en minutes) . . . . .	17
12	Résultat SQL : Top 10 héros selon GPM et XPM moyens . . . . .	18
13	Résultat SQL : Top 10 héros orientés combat (kills/assists) et nombre de parties jouées . . . . .	19
14	Résultat SQL : Top 10 héros les plus populaires selon times_played (avec wins et win_rate) . . . . .	20
15	Power BI — Page Overview : filtre temporel, tendance journalière du nombre de matchs, et KPI globaux . . . . .	21
16	Power BI — Page Top 10 Players : comparaison KDA moyen vs win rate et table de détails . . . . .	22
17	Power BI — Page Hero Meta : relation presence rate vs win rate et Top 10 des héros les plus présents . . . . .	23
18	Power BI — Page Durée et Corrélations : tendance de la durée moyenne et corrélation durée vs kills . . . . .	24
19	Résultats du modèle et paramètres dans l'interface MLflow . . . . .	26

## Liste des tableaux

1	Caractéristiques du jeu de données . . . . .	4
2	Services Microsoft Azure utilisés . . . . .	5
3	Tables de la couche Silver . . . . .	6
4	Tables de la couche Gold . . . . .	7
5	Configuration du trigger . . . . .	9
6	Configuration du cluster Databricks . . . . .	10
7	Critères de classification Meta Tier . . . . .	12
8	Résultats de l'analyse de durée . . . . .	13
9	Caractéristiques (Features) utilisées . . . . .	24
10	Matrice de Confusion . . . . .	25
11	Classement de l'importance des features . . . . .	25
12	Principaux résultats obtenus . . . . .	29
13	Stack technologique du projet . . . . .	35

# 1. Introduction

## 1.1 Contexte du Projet

L'industrie de l'e-sport a connu une croissance exponentielle au cours de la dernière décennie, devenant aujourd'hui un écosystème de plusieurs milliards de dollars. Avec cette croissance, la quantité de données générées lors des tournois professionnels a également augmenté de manière significative. L'analyse des performances des joueurs, l'optimisation des stratégies d'équipe et la prédiction des résultats des matchs sont des domaines critiques où l'ingénierie des données et la science des données trouvent des applications pratiques.

Dans le cadre de ce projet, une solution complète d'ingénierie des données a été conçue et implémentée pour le traitement, l'analyse et le développement de modèles prédictifs sur les données des matchs professionnels de Dota 2. Le projet a été réalisé sur la plateforme Microsoft Azure, en adoptant les principes modernes d'architecture cloud.

## 1.2 Jeu de Données Utilisé

Le jeu de données utilisé dans ce projet est le “**Dota 2 Pro League Matches 2023**” publié sur la plateforme Kaggle. Ce jeu de données contient des enregistrements détaillés des matchs des tournois professionnels de Dota 2 tout au long de l'année 2023.

TABLEAU 1 – Caractéristiques du jeu de données

Nom du Fichier	Taille	Description
main_metadata.csv	9.09 MB	Métadonnées des matchs
players_reduced.csv	152.00 MB	Statistiques des joueurs
picks_bans.csv	33.01 MB	Données de sélection/bannissement
teams.csv	8.09 MB	Informations sur les équipes

Le format CSV et la structure relationnelle du jeu de données ont nécessité un processus complet de nettoyage et de fusion lors de la transition de la couche Bronze vers la couche Silver.

## 1.3 Objectifs du Projet

Les objectifs principaux de ce projet ont été définis comme suit :

1. **Mise en place de l'Architecture Lakehouse** : Implémentation de la Medallion Architecture (Bronze-Silver-Gold) sur Azure Data Lake Storage Gen2
2. **Pipeline de Traitement Batch** : Création de pipelines ETL automatisés utilisant Azure Data Factory et Databricks
3. **Simulation de Streaming** : Développement d'un simulateur imitant le flux de données en temps réel
4. **Transformation des Données** : Transformation des données brutes en tables analytiques à l'aide de PySpark

5. **Analyse Avancée** : Analyses de performance des joueurs et des héros avec SQL et PySpark
6. **Machine Learning** : Développement d'un modèle de prédiction des résultats de matchs et suivi avec MLflow

## 1.4 Services Azure Utilisés

TABLEAU 2 – Services Microsoft Azure utilisés

Service	Objectif d'Utilisation
Azure Data Lake Storage Gen2	Stockage central pour les couches Lakehouse
Azure Data Factory	Orchestration des pipelines et planification
Azure Databricks	Traitement des données et développement ML
Delta Lake	Format de données performant conforme ACID
MLflow	Suivi des expériences de machine learning
Power BI	Visualisation et création de tableaux de bord

Toutes les ressources ont été créées à l'aide de l'abonnement **Microsoft Azure for Students** et gérées en tenant compte de l'optimisation des coûts.

## 2. Conception Architecturale

### 2.1 Medallion Architecture

Dans ce projet, la **Medallion Architecture** (Architecture Médailon), largement adoptée dans les pratiques modernes d'ingénierie des données, a été appliquée. Cette architecture divise les données en trois couches selon leur niveau de qualité et de traitement : **Bronze**, **Silver** et **Gold**.

#### 2.1.1 Couche Bronze (Données Brutes)

La couche Bronze est la couche où les données provenant des systèmes sources sont stockées telles quelles, sans aucune transformation :

- **Objectif** : Conservation d'une copie exacte de la source de données
- **Format** : CSV et JSON (pour la simulation de streaming)
- **Contenu** : 4 fichiers CSV principaux + fichiers JSON de streaming
- **Utilisation** : Retour à la source en cas d'erreur, audit et traçabilité

#### 2.1.2 Couche Silver (Données Nettoyées)

La couche Silver est la couche intermédiaire où les données brutes sont nettoyées et standardisées :

- **Objectif** : Fournir des données propres et cohérentes
- **Format** : Delta Lake (basé sur Parquet)
- **Transformations** : Nettoyage des valeurs nulles, standardisation des types, élimination des doublons, détection d'outliers

TABLEAU 3 – Tables de la couche Silver

Table	Nombre d'Enregistrements
cleaned_matches	29 809
cleaned_players	8 456
cleaned_picks_bans	710 414

#### 2.1.3 Couche Gold (Données Analytiques)

La couche Gold contient les données enrichies avec la logique métier :

TABLEAU 4 – Tables de la couche Gold

Table	Description	Enregistrements
esports_gold.player_stats	Métriques de performance	813
esports_gold.hero_stats	Statistiques des héros	123
esports_gold.daily_stats	Statistiques quotidiennes	365
esports_gold.ml_features	Features pour le ML	8 312

## 2.2 Structure Azure Data Lake Storage Gen2

Toutes les couches Lakehouse sont positionnées sur Azure Data Lake Storage Gen2 :

- **Storage Account** : dota2lakehousenew
- **Container** : data
- **Chemins** :
  - Bronze : abfss://data@dota2lakehousenew.../bronze/
  - Silver : abfss://data@dota2lakehousenew.../silver/
  - Gold : abfss://data@dota2lakehousenew.../gold/

Name	Last modified	Access tier	Blob type	Size	Lease state
bronze	12/15/2023, 5:57:06 PM				...
gold	12/15/2023, 5:58:29 PM				...
landing	12/15/2023, 5:57:09 PM				...
silver	12/15/2023, 5:58:10 PM				...

FIGURE 1 – Vue Azure Portal de la structure des dossiers Bronze/Silver/Gold dans ADLS Gen2

## 2.3 Diagramme de Flux de Données

Le flux de données de bout en bout du système suit le parcours suivant : les données sources provenant de Kaggle et du simulateur de streaming sont d'abord stockées dans la couche Bronze. Ensuite, Azure Data Factory orchestre les notebooks Databricks qui transforment les données vers les couches Silver puis Gold. Enfin, les données Gold alimentent les analyses SQL, le modèle de machine learning (suivi par MLflow), et les visualisations Power BI.

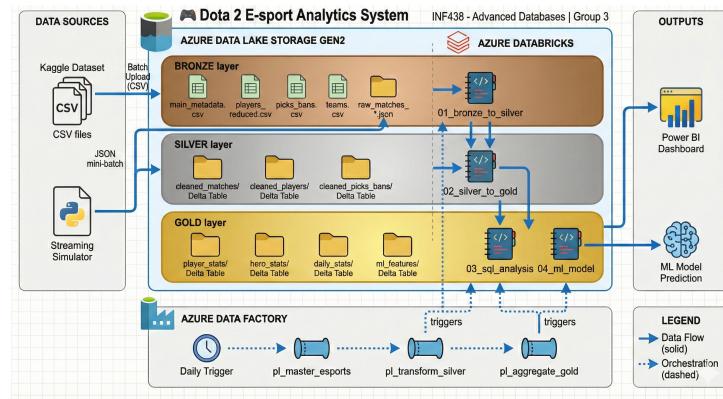


FIGURE 2 – Diagramme de flux de données

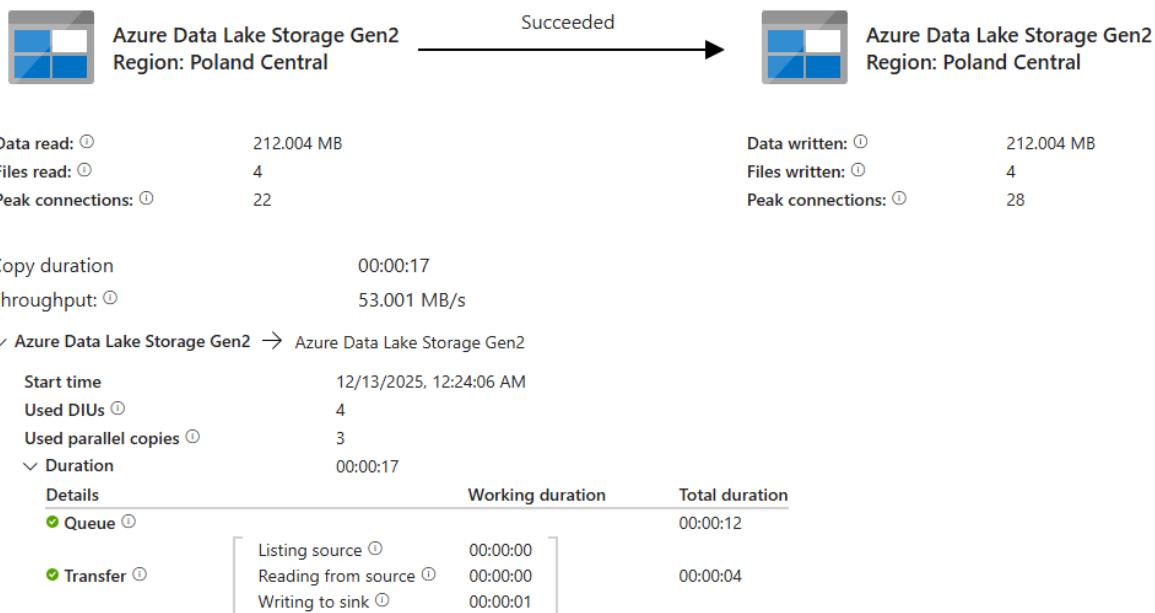
### 3. Implémentation

#### 3.1 Pipeline de Traitement Batch

Azure Data Factory (ADF) a été utilisé pour l'orchestration du pipeline de traitement batch. Le pipeline principal créé dans le projet porte le nom **pl\_master\_esports**.

En amont de ce traitement, un pipeline d'ingestion spécifique nommé **PL\_Ingest\_Kaggle\_To\_Bronze** a été mis en place. Ce pipeline utilise une activité *Copy Data* pour transférer automatiquement les fichiers bruts (CSV) et le jeu de données réduit vers le conteneur **bronze/** du Data Lake, assurant ainsi la disponibilité des données pour les notebooks Databricks.

Activity run id: 0d82b85a-cd5f-4180-8ed5-f1b8f717fae3

FIGURE 3 – Détails d'exécution de l'activité *Copy Data* confirmant l'ingestion réussie des 4 fichiers sources (212 MB) vers le conteneur Bronze.

La structure du pipeline est conçue comme suit :

- **Activité 1 :** nb\_bronze\_to\_silver (Databricks Notebook)
- **Activité 2 :** nb\_silver\_to\_gold (dépend de l'Activité 1)
- **Activité 3 :** nb\_ml\_model (dépend de l'Activité 2)

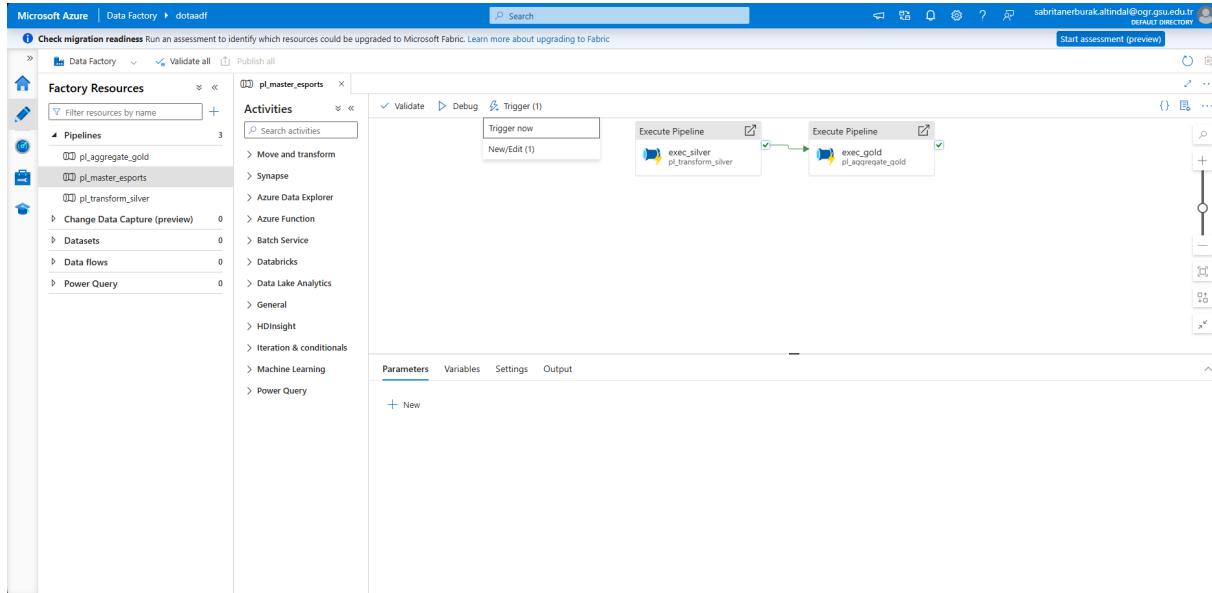


FIGURE 4 – Diagramme du pipeline pl\_master\_esports dans Azure Data Factory

### 3.2 Configuration du Déclencheur

Un déclencheur nommé **tr\_daily\_esports** a été créé pour l'exécution automatique du pipeline.

TABLEAU 5 – Configuration du trigger

Paramètre	Valeur
Nom du Trigger	tr_daily_esports
Type	Schedule Trigger
Fréquence	Quotidien (Daily)
Heure de Début	02 :00 UTC
État	Actif

The screenshot shows the Microsoft Azure Data Factory Monitor interface. The left sidebar has a 'Pipeline runs' section selected. The main area displays a table of pipeline runs with columns: Pipeline name, Run start, Run end, Duration, Triggered by, Status, Run, Parameters, Annotations, and Run ID. One row for 'pl\_transform\_silver' is highlighted with a red box, showing it was triggered at 12/21/2025, 1:43:23 AM, ended at 12/21/2025, 1:54:04 AM, took 10m 42s, and was triggered by 'tr\_daily\_esports'.

Pipeline name	Run start	Run end	Duration	Triggered by	Status	Run	Parameters	Annotations	Run ID
pl_aggregate_gold	12/21/2025, 2:04:22 AM	12/21/2025, 2:05:13 AM	51s	98ab8d28-5a69-42e...	Succeeded	Original			46f7882f-4a4f-4a30-959e-a75
pl_transform_silver	12/21/2025, 2:00:05 AM	12/21/2025, 2:04:20 AM	4m 15s	80f9d01b-b281-454...	Succeeded	Original			b118bef9-4582-4661-821b-19
pl_master_esports	12/21/2025, 2:00:01 AM	12/21/2025, 2:05:14 AM	5m 14s	tr_daily_esports	Succeeded	Original			c1794e0b-9e56-443c-9bd1-8c
pl_aggregate_gold	12/21/2025, 1:54:06 AM	12/21/2025, 1:55:01 AM	56s	dbcd679-b51c-4b1f...	Succeeded	Original			c539cb2c-3567-4b28-bd74-11
pl_transform_silver	12/21/2025, 1:43:23 AM	12/21/2025, 1:54:04 AM	10m 42s	e592d65c-6531-4c6...	Succeeded	Original			1912678d-6a93-4947-8267-f7
pl_master_esports	12/21/2025, 1:43:15 AM	12/21/2025, 1:55:03 AM	11m 48s	Manual trigger	Succeeded	Original			c95e9487-bd0b-48c4-bb1e-1f

FIGURE 5 – État “Succeeded” et “Triggered by tr\_daily\_esports” dans l’écran ADF Monitor

### 3.3 Notebooks Databricks

Les opérations de transformation des données ont été réalisées en utilisant PySpark sur Azure Databricks.

TABLEAU 6 – Configuration du cluster Databricks

Paramètre	Valeur
Mode du Cluster	Standard
Type de Nœud	Standard_DS3_v2
Auto Termination	120 minutes
Databricks Runtime	13.3 LTS (Spark 3.4.1)

Notebooks créés :

1. **setup\_connection.ipynb** : Test de connexion Azure ADLS Gen2
2. **01\_bronze\_to\_silver.ipynb** : Nettoyage des données brutes
3. **02\_silver\_to\_gold.ipynb** : Application de la logique métier
4. **04\_ml\_model.ipynb** : Entraînement du modèle ML

Le code source complet de ces notebooks est disponible en **Annexe 6.3**.

### 3.4 Simulation de Streaming

En l’absence d’une source de données en temps réel, un script Python simulant le comportement de streaming a été développé. Le script **stream\_simulator.py** fonctionne selon la logique suivante :

1. Les enregistrements sont lus à partir du fichier `main_metadata.csv`

2. Les enregistrements sont divisés en mini-batches de 5
3. Un délai artificiel de 3 secondes est appliqué pour chaque mini-batch
4. Le mini-batch est nommé avec un timestamp unique et écrit dans Bronze

```

data > bronze
Authentication method: Access key (Switch to Microsoft Entra user)
Search blobs by prefix (case-sensitive)

Showing all 7 items
Name
[.] main_metadata.csv
picks_bans.csv
players_reduced.csv
raw_matches_20251213_091757.json
raw_matches_20251213_091801.json
raw_matches_20251213_091804.json
teams.csv

25  def stream_data():
68      file_client.flush_data(len(json_data))
69      print("Done.")
70  except Exception as e:
71      print(f"Upload Error: {e}")
72
73  # 5. Wait (Simulate real-time gap)
74  time.sleep(SLEEP_TIME)
75
76  print("\n--- Simulation Complete ---")
77
78 if __name__ == "__main__":
79     stream_data()

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS python
PS C:\Users\emirhan\ders\04_01\ileri-veri-tabanlari\projet> python
--- Starting Streaming Simulation using main_metadata.csv ---
Loading dataset...
Loaded 29810 rows. Starting stream...
Uploading raw_matches_20251213_091757.json... Done.
Uploading raw_matches_20251213_091801.json... Done.
Uploading raw_matches_20251213_091804.json... Done.
Uploading raw_matches_20251213_091807.json... Done.

```

FIGURE 6 – Sortie d'exécution du code stream\_simulator.py

Lorsque le simulateur est exécuté, des fichiers sont créés dans la couche Bronze avec le format `raw_matches_YYYYMMDD_HHMMSS.json` (10 fichiers au total).

Le code source complet du simulateur est disponible en [Annexe 6.3](#).

## 3.5 Transformations ETL

### 3.5.1 Transformation Bronze → Silver

Les données CSV brutes de la couche Bronze ont subi un processus de transformation complet :

- **Chargement** : Lecture des fichiers CSV avec inférence de schéma
- **Nettoyage** : Suppression des colonnes inutiles, filtrage des valeurs nulles
- **Validation** : Filtrage des matchs avec durée invalide (<5 min ou >2 heures)
- **Enrichissement** : Création de nouvelles features (`duration_minutes`, `kda_calculated`)
- **Détection d'outliers** : Marquage avec la règle des 3 sigma

### 3.5.2 Transformation Silver → Gold

Les données nettoyées ont reçu une logique métier et des calculs avancés.

**Statistiques des Joueurs (player\_stats) :** Agrégation par compte joueur avec calcul des moyennes (KDA, GPM, XPM) et du taux de victoire. Résultat : statistiques calculées pour **813 joueurs uniques**.

**Analyse Méta des Héros (hero\_stats) :** Un algorithme de classification des héros en 5 tiers selon leur taux de présence (voir Tableau 7).

TABLEAU 7 – Critères de classification Meta Tier

Tier	Critère de Presence Rate
S-Tier	> 50%
A-Tier	> 30%
B-Tier	> 15%
C-Tier	> 5%
D-Tier	≤ 5%

```

# --- Hero Tiering Logic ---
df_hero_stats = df_hero_stats \
    .withColumn("presence_rate", round(col("total_presence") / (total_matches * 2) * 100, 2)) \
    .withColumn("meta_tier", \
        when(col("presence_rate") > 50, "S-Tier") \
        .when(col("presence_rate") > 30, "A-Tier") \
        .when(col("presence_rate") > 15, "B-Tier") \
        .when(col("presence_rate") > 5, "C-Tier") \
        .otherwise("D-Tier")
    )

print(f"Hero stats created: {df_hero_stats.count()} heroes")
df_hero_stats.orderBy(desc("presence_rate")).select(
    "hero_id", "times_played", "win_rate", "pick_count", "ban_count", "presence_rate", "meta_tier"
).show(10)
# --- Hero Statistics Analysis ---
[3/5] Creating Hero Statistics (Meta Analysis)...
Hero stats created: 123 heroes
+-----+-----+-----+-----+-----+
|hero_id|times_played|win_rate|pick_count|ban_count|presence_rate|meta_tier|
+-----+-----+-----+-----+-----+
| 86.0 | 330 | 51.21 | 9932 | 5983 | 26.69 | B-Tier |
| 137.0 | 81 | 51.85 | 3834 | 9844 | 21.57 | B-Tier |
| 93.0 | 89 | 53.93 | 2943 | 9794 | 21.36 | B-Tier |
| 106.0 | 187 | 58.27 | 5386 | 7416 | 21.34 | B-Tier |
| 128.0 | 122 | 44.26 | 4447 | 8167 | 21.16 | B-Tier |
| 71.0 | 39 | 51.28 | 4998 | 6394 | 19.09 | B-Tier |
| 38.0 | 96 | 55.21 | 3855 | 7274 | 18.67 | B-Tier |
| 53.0 | 98 | 57.78 | 3814 | 6895 | 17.9 | B-Tier |
| 10.0 | 330 | 51.21 | 4941 | 1770 | 17.8 | B-Tier |
| 114.0 | 64 | 57.81 | 3849 | 7215 | 17.22 | B-Tier |
+-----+-----+-----+-----+-----+
only showing top 10 rows

```

FIGURE 7 – Sortie de la classification Tier dans le notebook silver\_to\_gold

### Analyse de la Durée des Matchs :

TABLEAU 8 – Résultats de l'analyse de durée

Durée	Matchs	Avg Kills	Radiant Win
Very Short (<25min)	4 176	44.94	53.54%
Short (25-35min)	13 548	56.74	50.41%
Medium (35-45min)	8 162	65.33	49.36%
Long (45-55min)	2 839	68.19	49.52%
Very Long (>55min)	1 084	80.77	50.46%

### 3.6 Gestion des Tables Delta Lake

Les tables de la couche Gold sont gérées sous le schéma **esports\_gold** dans le Databricks Unity Catalog. Les commandes SQL de création sont disponibles en **Annexe 6.3**.

FIGURE 8 – Liste des tables sous le schéma esports\_gold dans le Databricks Catalog

## 4. Analyse des Données et Machine Learning

### 4.1 Accès aux Tables Gold et Exécution SQL (Databricks)

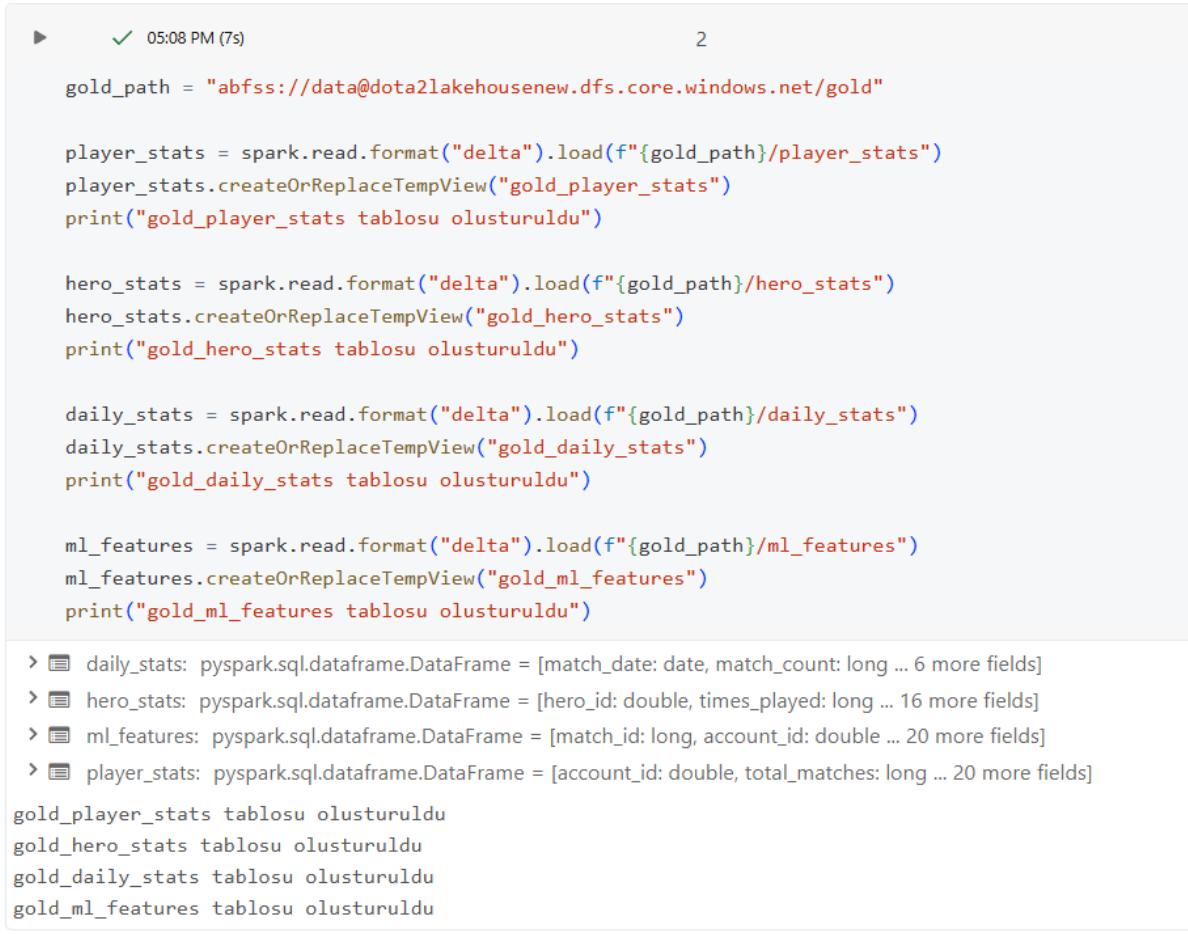
#### 4.1.1 Connexion à Azure Data Lake Storage (ADLS Gen2)

Les tables analytiques de la couche *Gold* sont stockées sur Azure Data Lake Storage Gen2 sous format Delta Lake. Dans notre cas, certaines limitations de l'abonnement *Azure for Students* (quotas / permissions RBAC) ont empêché un accès direct et stable pour tous les membres du groupe. Afin d'assurer la continuité du projet, l'accès au *Storage Account* a été effectué via une clé d'accès transmise par un membre du groupe disposant des autorisations nécessaires. Cette clé n'est jamais exposée dans le rapport ni en clair dans le code versionné ; elle est injectée via variables d'environnement Databricks.

**Contraintes CPU et utilisation d'un cluster partagé.** Pendant l'exécution des notebooks avec l'abonnement Azure for Students, nous avons rencontré des limitations de ressources (quota CPU/vCore) et des ralentissements pouvant interrompre certains traitements Spark. Pour éviter les blocages et terminer le travail à temps, nous avons donc utilisé le cluster Databricks déjà configuré par un membre du groupe (esports-cluster), qui disposait de ressources plus stables (16 GB de RAM et 4 cœurs). Grâce à cette solution, nous avons pu exécuter correctement les transformations ETL, accéder aux tables Delta de la couche Gold et lancer les requêtes SQL analytiques sans interruptions.

#### 4.1.2 Chargement des tables Gold et création de vues temporaires

Pour exécuter des requêtes SQL directement sur les fichiers Delta de la couche Gold, nous avons chargé chaque table depuis ADLS et créé des vues temporaires dans Databricks.



```

▶ ✓ 05:08 PM (7s) 2

gold_path = "abfss://data@dota2lakehousenew.dfs.core.windows.net/gold"

player_stats = spark.read.format("delta").load(f"{gold_path}/player_stats")
player_stats.createOrReplaceTempView("gold_player_stats")
print("gold_player_stats tablosu olusturuldu")

hero_stats = spark.read.format("delta").load(f"{gold_path}/hero_stats")
hero_stats.createOrReplaceTempView("gold_hero_stats")
print("gold_hero_stats tablosu olusturuldu")

daily_stats = spark.read.format("delta").load(f"{gold_path}/daily_stats")
daily_stats.createOrReplaceTempView("gold_daily_stats")
print("gold_daily_stats tablosu olusturuldu")

ml_features = spark.read.format("delta").load(f"{gold_path}/ml_features")
ml_features.createOrReplaceTempView("gold_ml_features")
print("gold_ml_features tablosu olusturuldu")

> daily_stats: pyspark.sql.dataframe.DataFrame = [match_date: date, match_count: long ... 6 more fields]
> hero_stats: pyspark.sql.dataframe.DataFrame = [hero_id: double, times_played: long ... 16 more fields]
> ml_features: pyspark.sql.dataframe.DataFrame = [match_id: long, account_id: double ... 20 more fields]
> player_stats: pyspark.sql.dataframe.DataFrame = [account_id: double, total_matches: long ... 20 more fields]

gold_player_stats tablosu olusturuldu
gold_hero_stats tablosu olusturuldu
gold_daily_stats tablosu olusturuldu
gold_ml_features tablosu olusturuldu

```

FIGURE 9 – Chargement des tables Gold depuis ADLS Gen2 et création des vues temporelles dans Databricks

## 4.2 Requêtes SQL Analytiques (Résultats)

Cette partie présente les principales requêtes SQL utilisées pour produire des analyses sur les joueurs, les héros et l'activité quotidienne. Pour garder le rapport lisible, nous présentons ici les sorties (captures d'écran), les requêtes complètes peuvent être ajoutées en annexe si nécessaire.

### 4.2.1 Top 10 joueurs par performance (KDA)

Objectif : identifier les 10 joueurs les plus performants selon leur KDA moyen (avec filtrage pour éviter les cas non représentatifs).

```
%sql
SELECT
    account_id,
    total_kills,
    total_deaths,
    total_assists,
    ROUND(avg_kda, 2) as kda_ratio
FROM
    gold_player_stats
WHERE
    total_matches > 5
ORDER BY
    avg_kda DESC
LIMIT 10;
```

▶ (1) Spark Jobs

▶ \_sqldf: pyspark.sql.dataframe.DataFrame = [account\_id: double, total\_kills: long ... 3 more fields]

	1.2 account_id	1 <sup>2</sup> <sub>3</sub> total_kills	1 <sup>2</sup> <sub>3</sub> total_deaths	1 <sup>2</sup> <sub>3</sub> total_assists	1.2 kda_ratio
1	202217968	49	16	49	12.27
2	116249155	68	44	193	11.97
3	173978074	70	25	99	11.79
4	320252024	68	12	62	11.76
5	132309493	40	12	48	11.75
6	392565237	53	22	100	11.67
7	221532774	60	24	127	11.18
8	118233883	52	24	78	10.95
9	361688848	177	58	180	10.77
10	220154695	54	22	57	10.63

FIGURE 10 – Résultat SQL : Top 10 joueurs par KDA moyen (avec kills, deaths, assists)

#### 4.2.2 Tendances quotidiennes : volume de matchs et durée moyenne

Objectif : observer l'évolution journalière du nombre de matchs ainsi que la durée moyenne (conversion en minutes).

```
%sql
SELECT
    match_date,
    match_count,
    ROUND(avg_duration / 60, 2) as avg_duration_minutes
FROM
    gold_daily_stats
ORDER BY
    match_date DESC
LIMIT 10;
```

▶ (1) Spark Jobs

➤ \_sqlldf: pyspark.sql.dataframe.DataFrame = [match\_date: date, match\_count: long ... 1 more field]

Table +

	match_date	match_count	avg_duration_minutes
1	2023-12-31	45	0.58
2	2023-12-30	49	0.57
3	2023-12-29	70	0.55
4	2023-12-28	74	0.54
5	2023-12-27	73	0.55
6	2023-12-26	71	0.58
7	2023-12-25	56	0.56
8	2023-12-24	67	0.54
9	2023-12-23	75	0.56
10	2023-12-22	60	0.58

FIGURE 11 – Résultat SQL : aperçu des statistiques quotidiennes (match\_count et durée moyenne en minutes)

#### 4.2.3 Top 10 héros : performance économique (GPM/XPM)

Objectif : identifier les héros ayant les meilleures performances économiques via avg\_gpm et avg\_xpm.

	1.2 hero_id	1.2 avg_gpm	1.2 avg_xpm
1	109	701	710
2	89	674	640
3	53	666	707
4	25	655	783
5	46	653	803
6	95	649	751
7	94	643	719
8	12	641	731
9	69	640	691
10	113	637	682

FIGURE 12 – Résultat SQL : Top 10 héros selon GPM et XPM moyens

#### 4.2.4 Top 10 héros : style combat (kills/assists)

Objectif : analyser les héros les plus agressifs via les métriques `avg_kills` et `avg_assists`, en tenant compte du volume (`times_played`).

```
%sql
SELECT
    hero_id,
    ROUND(avg_kills, 2) as avg_kills,
    ROUND(avg_assists, 2) as avg_assists,
    times_played
FROM
    gold_hero_stats
ORDER BY
    avg_kills DESC
LIMIT 10;
```

▶ (1) Spark Jobs

▶ \_sqldf: pyspark.sql.dataframe.DataFrame = [hero\_id: double, avg\_kills: double ... 2 more fields]

**Table** +

	1.2 hero_id	1.2 avg_kills	1.2 avg_assists	1.2 times_played
1	39	10.28	10.89	72
2	25	9.68	7.3	191
3	4	9.27	10.5	52
4	93	8.79	9.78	89
5	13	8.77	10.65	60
6	70	8.74	5.86	74
7	106	8.62	11.87	187
8	22	8.59	13.55	74
9	52	8.59	9.32	109
10	67	8.54	13.98	46

FIGURE 13 – Résultat SQL : Top 10 héros orientés combat (kills/assists) et nombre de parties jouées

#### 4.2.5 Top 10 héros les plus populaires

Objectif : identifier les héros les plus joués en se basant sur le nombre d'apparitions times played. Nous affichons aussi le nombre de victoires et le taux de victoire (en %).

```

SELECT
    hero_id,
    times_played as total_picks,
    wins as total_wins,
    ROUND(win_rate * 100, 2) as win_rate_percentage
FROM
    gold_hero_stats
ORDER BY
    times_played DESC
LIMIT 10;

```

▶ (1) Spark Jobs

➤ \_sqldf: pyspark.sql.dataframe.DataFrame = [hero\_id: double, total\_picks: long ... 2 more fields]

Table +

	1.2 hero_id	1.3 total_picks	1.3 total_wins	1.2 win_rate_percentage
1	86	330	169	5121
2	100	299	151	5050
3	19	255	117	4588
4	9	248	121	4879
5	128	225	101	4489
6	83	210	114	5429
7	5	202	109	5396
8	25	191	98	5131
9	106	187	94	5027
10	51	179	94	5251

FIGURE 14 – Résultat SQL : Top 10 héros les plus populaires selon times\_played (avec wins et win\_rate)

L'analyse révèle que dans les matchs très courts (<25 min), l'équipe Radiant a un léger avantage (53.54%). Ce taux s'équilibre autour de 50% pour les matchs plus longs.

## 4.3 Visualisation Power BI

### 4.3.1 Connexion et chargement des données dans Power BI

Nous avons réalisé la partie visualisation avec Power BI Desktop à partir des données Gold. Pour récupérer les données, nous avons utilisé *Get Data* puis la source *Azure Blob Storage*. Comme tout le monde ne pouvait pas se connecter facilement (droits/quotas Azure for Students), nous avons utilisé une clé d'accès partagée par un membre du groupe.

Après la connexion, nous avons trouvé les fichiers Parquet dans le conteneur *data* (couche Gold) et importé les tables *daily\_stats*, *player\_stats*, *hero\_stats* et *ml\_features*.

Ensuite, nous avons ajouté une table *Date* pour filtrer par temps, puis nous l'avons reliée à *daily\_stats* avec la colonne *match\_date*. Enfin, nous avons créé les pages et les graphiques pour analyser les tendances générales, les meilleurs joueurs et la météo des héros.

#### 4.3.2 Page 1 : Vue d'ensemble et tendances journalières

Cette page fournit une lecture rapide de l'activité 2023 : (i) un slicer de dates qui pilote l'ensemble des indicateurs, (ii) une tendance journalière du nombre de matchs, (iii) des cartes KPI : nombre total de matchs, taux moyen de victoire Radiant et durée moyenne de match.

**Observations.** On constate une activité relativement stable sur une grande partie de l'année, avec des fluctuations quotidiennes. Un pic marqué apparaît sur une période spécifique (fin d'année), ce qui peut correspondre à un événement (tournoi, patch majeur ou forte activité compétitive). Le taux moyen de victoire Radiant reste proche d'un équilibre, ce qui suggère l'absence d'avantage systématique durable sur l'année. La durée moyenne des matchs est globalement stable, avec des variations ponctuelles indiquant des changements de style de jeu ou de météo.

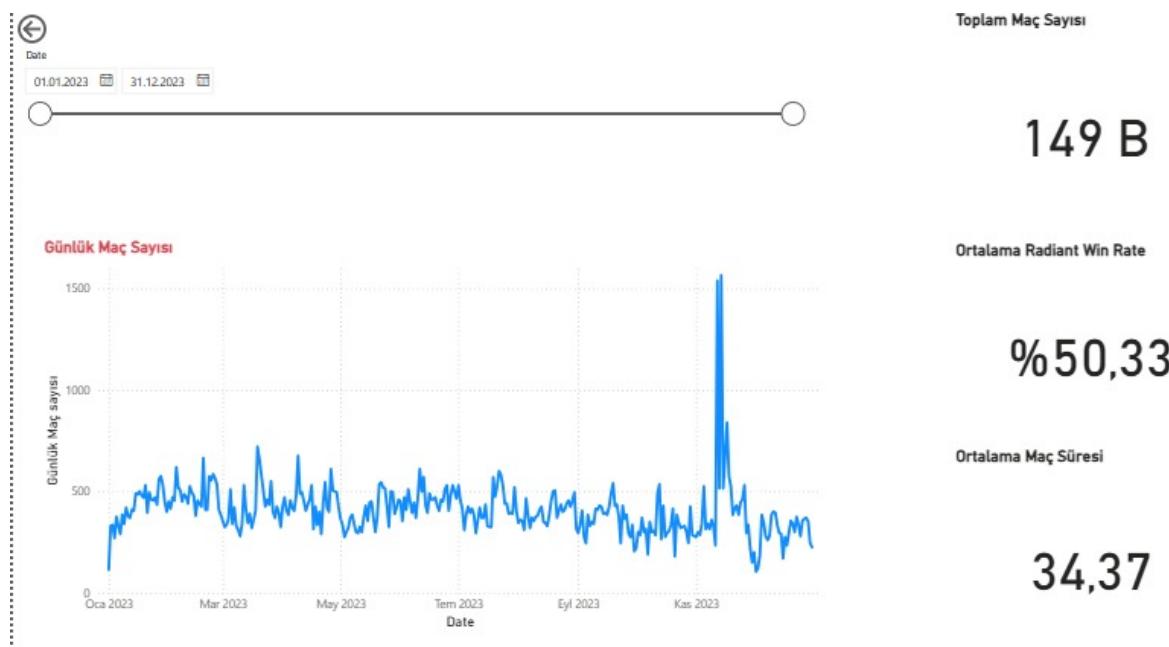


FIGURE 15 – Power BI — Page Overview : filtre temporel, tendance journalière du nombre de matchs, et KPI globaux

#### 4.3.3 Page 2 : Top 10 joueurs

Cette page compare les joueurs selon : (i) Top 10 joueurs par KDA moyen, (ii) Top 10 joueurs par taux de victoire (win rate), avec une table détaillée des statistiques (matches, wins, KDA, XPM, etc.).

**Observations.** Le classement par KDA met en évidence des profils orientés performance individuelle (survie + efficacité en combat). Le classement par win rate n'est pas forcément identique : un joueur peut avoir un KDA élevé mais un taux de victoire plus modéré, ce qui souligne l'importance du collectif et du contexte de match. La table permet de vérifier la cohérence des résultats en comparant simultanément volume de matchs et indicateurs, et d'éviter de sur-interpréter des joueurs avec trop peu de parties.

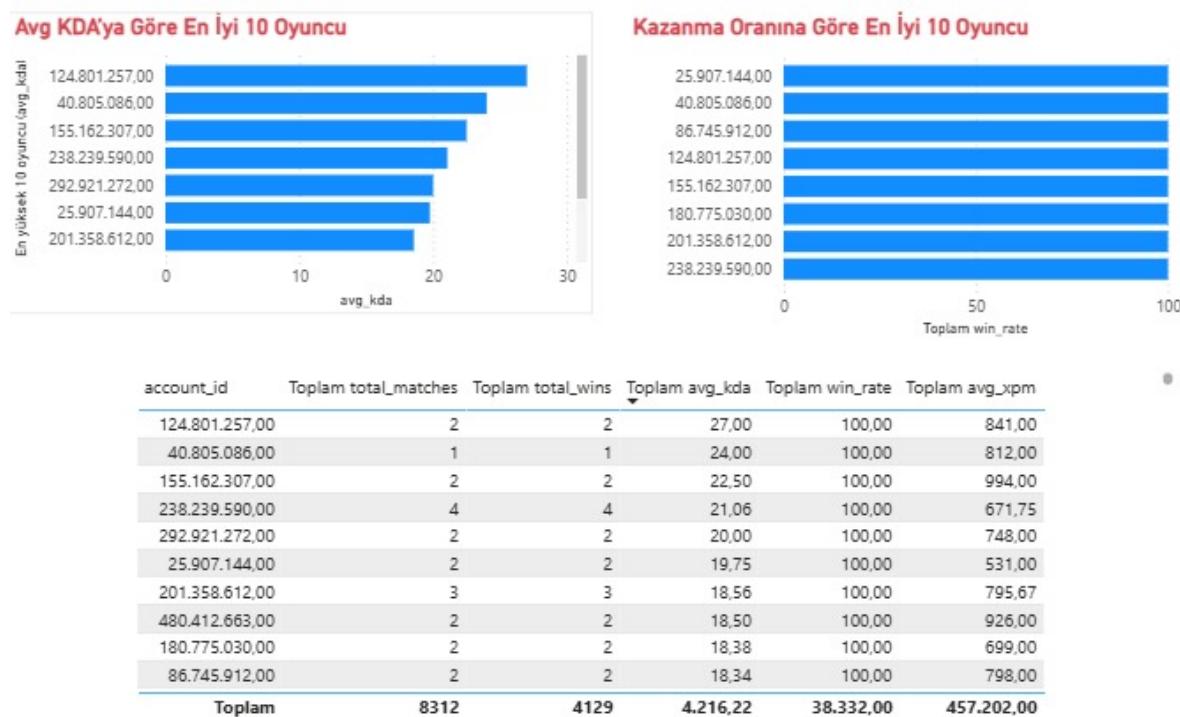


FIGURE 16 – Power BI — Page Top 10 Players : comparaison KDA moyen vs win rate et table de détails

#### 4.3.4 Page 3 : Analyse métta des héros

Cette page est dédiée à la métta : (i) un scatter reliant *presence\_rate* et *win\_rate* par héros, (ii) un Top 10 des héros les plus présents.

**Observations.** Le nuage de points met en évidence plusieurs zones intéressantes :

- des héros très présents mais avec un win rate moyen : souvent des choix "meta" ou flexibles, pas forcément dominants ;
- des héros peu présents mais avec un win rate élevé : candidats "underrated" ou picks situationnels très efficaces ;
- des héros présents et performants : ils constituent la métta dominante et méritent une attention particulière.

Le Top 10 des héros les plus présents synthétise la popularité, et facilite l'identification des héros joués le plus fréquemment sur la période filtrée.

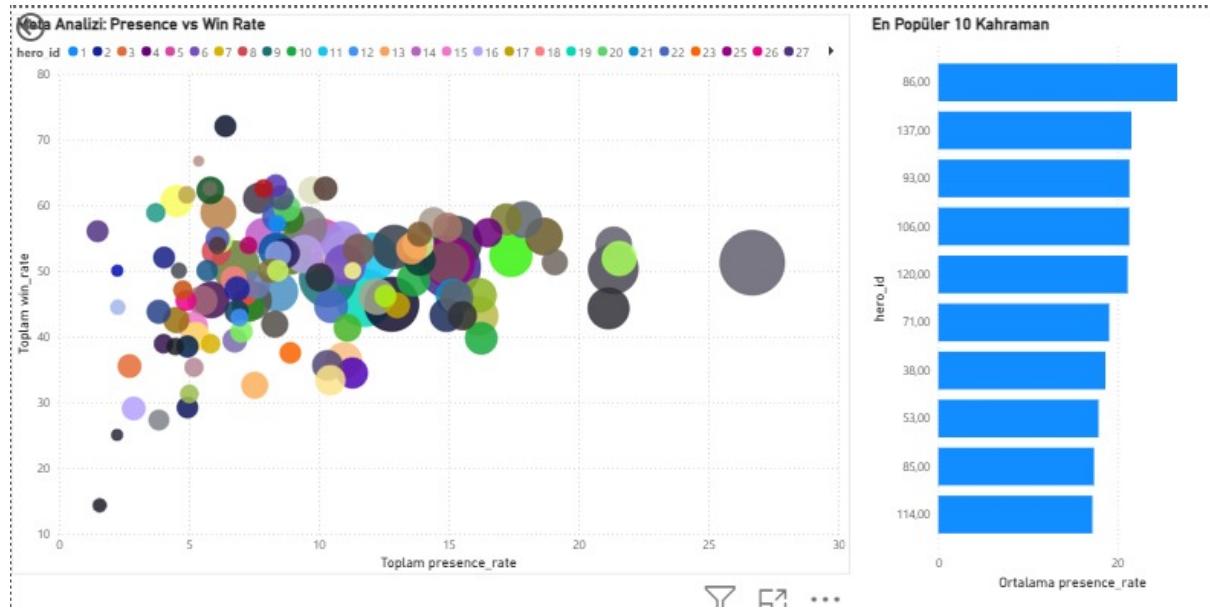


FIGURE 17 – Power BI — Page Hero Meta : relation presence rate vs win rate et Top 10 des héros les plus présents

#### 4.3.5 Page 4 : Durée des matchs et corrélations

Cette page analyse : (i) l'évolution temporelle de la durée moyenne des matchs, (ii) la relation entre durée et intensité (ex. kills moyens) via un scatter.

**Observations.** La durée moyenne varie dans une plage relativement limitée, mais certains creux/pics suggèrent des périodes où les matchs se terminent plus rapidement (stratégies agressives) ou au contraire s'allongent (jeux plus contrôlés). Le scatter durée vs kills montre une relation globalement faible à modérée : les matchs plus longs tendent à offrir davantage d'opportunités de combats, mais la dispersion indique que d'autres facteurs influencent fortement le nombre de kills (écart de niveau, drafts, style d'équipes). Ce graphique sert donc surtout à confirmer qu'il n'existe pas une relation parfaitement linéaire, mais plutôt une tendance générale.



FIGURE 18 – Power BI — Page Durée et Corrélations : tendance de la durée moyenne et corrélation durée vs kills

#### 4.4 Modèle de Machine Learning

Un modèle de classification binaire a été développé pour prédire le résultat d'un match. La variable cible est `win` (1 = Victoire, 0 = Défaite).

TABLEAU 9 – Caractéristiques (Features) utilisées

Feature	Description	Catégorie
kills	Nombre d'éliminations	Combat
deaths	Nombre de morts	Combat
assists	Nombre d'assistances	Combat
gold_per_min	Or par minute	Économie
xp_per_min	Expérience par minute	Économie
hero_damage	Dégâts aux héros	Dégâts
tower_damage	Dégâts aux tours	Dégâts
last_hits	Nombre de last hits	Progression
level	Niveau du joueur	Progression
duration_minutes	Durée du match	Info Match

L'algorithme **Random Forest Classifier** a été choisi avec les paramètres : numTrees=50, maxDepth=10, Train/Test Split=80%/20%.

Le code source complet du pipeline ML est disponible en **Annexe 6.3**.

## 4.5 Résultats du Modèle

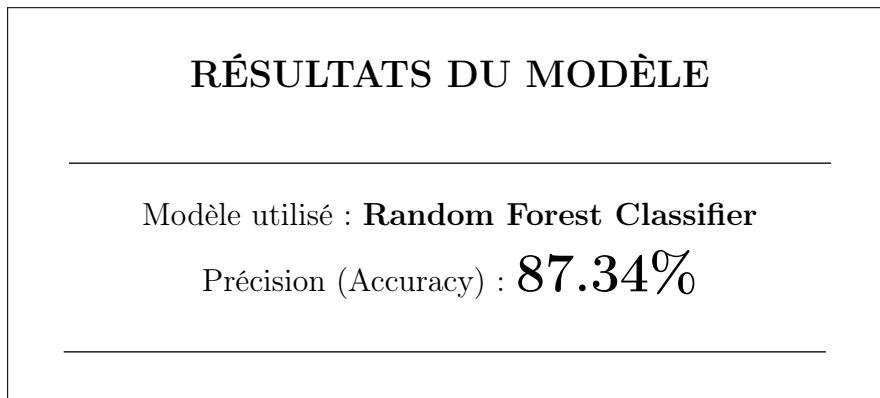


TABLEAU 10 – Matrice de Confusion

Réel / Prédit	Lose (0)	Win (1)
Lose (0)	727 (VN)	131 (FP)
Win (1)	81 (FN)	735 (VP)

Métriques calculées :

- Taux de Vrais Positifs (Recall) :  $\frac{735}{735+81} = 90.07\%$
- Taux de Vrais Négatifs :  $\frac{727}{727+131} = 84.73\%$
- Précision :  $\frac{735}{735+131} = 84.87\%$

## 4.6 Analyse de l'Importance des Features

TABLEAU 11 – Classement de l'importance des features

Rang	Feature	Importance
1	assists	0.311480
2	deaths	0.204537
3	tower_damage	0.147192
4	xp_per_min	0.073274
5	last_hits	0.058863
6	duration_minutes	0.049268
7	gold_per_min	0.048150
8	kills	0.040887
9	hero_damage	0.040081
10	level	0.026267

Ces résultats montrent que le **travail d'équipe** (assists avec 0.311) est le facteur le plus déterminant sur le résultat du match, suivi par la **survie** (deaths avec 0.205) et les **dégâts aux objectifs** (tower\_damage avec 0.147).

Contrairement aux attentes, les facteurs économiques (`gold_per_min` : 0.048, `xp_per_min` : 0.073) ont une importance relativement faible. Cela suggère que dans les matchs professionnels de Dota 2, la coordination d'équipe et le jeu orienté objectifs sont plus prédictifs de la victoire que l'accumulation individuelle de ressources.

## 4.7 Suivi MLflow

Toutes les expériences de modèle ont été automatiquement suivies sur MLflow avec `mlflow.autolog()`.

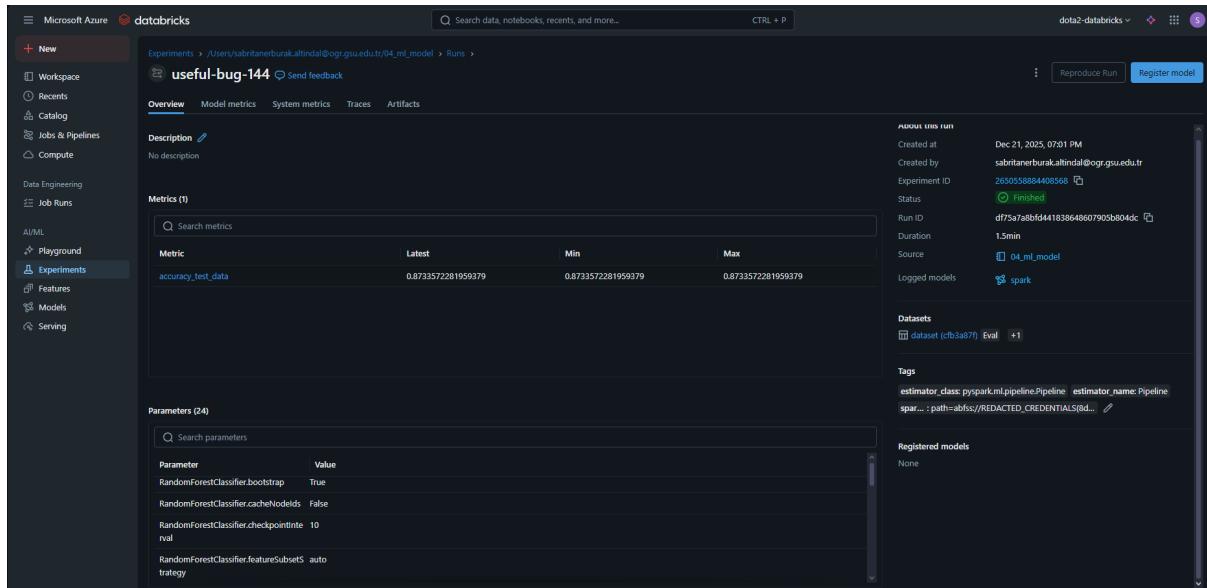


FIGURE 19 – Résultats du modèle et paramètres dans l’interface MLflow

## 5. Difficultés Rencontrées et Solutions

### 5.1 Limites des Ressources Azure

**Problème :** L'abonnement Azure for Students contient certaines limites de ressources strictes (notamment le quota vCore et les crédits limités).

**Solution :** La taille du cluster a été maintenue au minimum (Standard\_DS3\_v2), le temps d'auto-termination a été configuré à 120 minutes, et le cluster a été arrêté manuellement après la fin des opérations pour économiser les crédits.

### 5.2 Collision de Fichiers dans la Simulation de Streaming

**Problème :** Lors d'une exécution rapide du script de simulation, plusieurs mini-batches pouvaient être créés dans la même seconde, causant des collisions de noms de fichiers et des écrasements de données.

**Solution :** Un délai de `SLEEP_TIME = 3` secondes a été ajouté dans la boucle d'ingestion. De plus, les noms de fichiers utilisent désormais un horodatage précis au format `YYYYMMDD_HHMMSS`.

### 5.3 Volume de Données et Contraintes de Coûts

**Problème :** Le fichier source original `players.csv` dépassait les 5.3 GB. Le traitement de ce volume complet aurait rapidement épuisé les crédits Azure et les capacités de calcul du cluster étudiant disponible.

**Solution :** Un pré-traitement local a été effectué pour générer un fichier `players_reduced.csv` (152 MB) représentatif. Nous avons ensuite utilisé le format Delta Lake pour optimiser les performances de lecture/écriture sur ce jeu de données réduit.

### 5.4 Gestion des Clés Azure Storage

**Problème :** La clé du storage account devait être partagée entre les développeurs et les services sans être exposée en clair dans le code.

**Solution :** Utilisation de variables d'environnement (`AZURE_STORAGE_KEY`), stockage sécurisé avec Databricks secrets, et développement local avec fichier `.env`.

### 5.5 Complexité de la Gestion des Accès (IAM)

**Problème :** La collaboration en équipe a engendré des problèmes récurrents de permissions (IAM). L'attribution du rôle "Contributor" ne suffisait pas pour certaines actions (comme la lecture des données Blob ou l'accès au Cost Management), ce qui a causé des délais importants dans la gestion du temps et l'avancement du projet.

**Solution :** Une stratégie RBAC plus granulaire a été adoptée, en attribuant explicitement les rôles de *Storage Blob Data Owner* et *Cost Management Contributor* aux membres appropriés via le portail Azure.

## 5.6 Problème d'accès au cluster Databricks

**Problème :** En raison des limitations de l'abonnement *Azure for Students*, je ne pouvais pas créer ni sélectionner un cluster Databricks, ce qui empêchait l'exécution des notebooks et donc le lancement des analyses SQL sur les données Gold.

**Solution :** Pour assurer la progression du projet, un membre du groupe disposant des autorisations nécessaires a temporairement délégué l'accès à son cluster Databricks (*esports-cluster*). Cela m'a permis d'exécuter les requêtes SQL sur les tables Gold et de poursuivre la phase analytique. Les informations d'accès n'ont pas été exposées en clair : la clé et les paramètres de connexion ont été gérés de manière sécurisée via les variables d'environnement Databricks et *Databricks Secrets*.

## 6. Conclusion

### 6.1 Résumé du Projet

Ce projet a démontré comment les concepts modernes d'ingénierie des données (Lakehouse, Delta Lake, MLOps) peuvent être appliqués dans un scénario pratique d'e-sport.

TABLEAU 12 – Principaux résultats obtenus

Composant	Résultat
Medallion Architecture	Couches Bronze/Silver/Gold mises en place
Traitements des Données	29 809 matchs, 8 456 joueurs traités
Simulation Streaming	10 mini-batches écrits dans Bronze
Tables Gold	4 tables analytiques (813 joueurs, 123 héros)
Modèle ML	Random Forest avec 87.34% d'accuracy
Analyse Méta	Classification des héros à 5 niveaux

### 6.2 Leçons Apprises

- **Gestion des Ressources Cloud** : Optimisation des coûts et des ressources Azure
- **Avantages de Delta Lake** : Transactions ACID, évolution de schéma, performances
- **Orchestration** : Gestion des flux complexes avec Azure Data Factory
- **Pratiques MLOps** : Importance du suivi des expériences avec MLflow

### 6.3 Conclusion Générale

Les services intégrés offerts par l'écosystème Azure ont permis le développement rapide et efficace d'une solution de données de bout en bout. Le modèle ML avec un taux d'accuracy de **87.34%** a prouvé que les métriques de performance des joueurs peuvent prédire le résultat du match avec une haute précision. En particulier, les facteurs économiques (`gold_per_min`, `xp_per_min`) ont été identifiés comme les caractéristiques les plus déterminantes.

## ANNEXES

### Code Source des Notebooks Databricks

#### setup\_connection.ipynb

```

1 # Notebook: setup_connection
2 import os
3
4 storage_account = "data2lakehousenew"
5 container = "data"
6 storage_account_key = os.environ.get("AZURE_STORAGE_KEY")
7 print(f"Key loaded: {storage_account_key is not None}")
8
9 spark.conf.set(
10     f"fs.azure.account.key.{storage_account}.dfs.core.windows.net",
11     storage_account_key)
12
13 BRONZE = f"abfss://{{container}}@{{storage_account}}.dfs.core.windows.net/bronze"
14 SILVER = f"abfss://{{container}}@{{storage_account}}.dfs.core.windows.net/silver"
15 GOLD = f"abfss://{{container}}@{{storage_account}}.dfs.core.windows.net/gold"
16
17 print("== Files in Bronze Layer ==")
18 for f in dbutils.fs.ls(BRONZE):
19     print(f"  {f.name} ({f.size/1024/1024:.2f} MB)")

```

Code 1 – Test de connexion Azure ADLS Gen2

#### 01\_bronze\_to\_silver.ipynb (Extraits)

```

1 from pyspark.sql.functions import *
2
3 # LOAD RAW DATA FROM BRONZE
4 df_matches = spark.read.csv(f"{{BRONZE}}/main_metadata.csv",
5     header=True, inferSchema=True)
6 df_players = spark.read.csv(f"{{BRONZE}}/players_reduced.csv",
7     header=True, inferSchema=True)
8
9 # CLEAN MATCHES DATA
10 df_matches_clean = df_matches \
11     .drop("Unnamed: 0").dropDuplicates(["match_id"]) \
12     .filter(col("match_id").isNotNull()) \
13     .filter((col("duration") > 300) & (col("duration") < 7200))
14
15 # DERIVE NEW FEATURES
16 df_matches_clean = df_matches_clean \
17     .withColumn("duration_minutes", round(col("duration")/60, 2)) \
18     .withColumn("radiant_win", col("radiant_win").cast("boolean")) \
19     .withColumn("match_date", to_date(col("start_date_time")))
20
21 # OUTLIER DETECTION (3-sigma)
22 stats = df_players.select(
23     mean("kills").alias("m"), stddev("kills").alias("s")).collect()[0]
24 df_players = df_players.withColumn("is_outlier",
25     when(col("kills") > stats["m"] + 3*stats["s"], True).otherwise(False))
26
27 # WRITE TO SILVER AS DELTA
28 df_matches_clean.write.format("delta").mode("overwrite") \
29     .save(f"{{SILVER}}/cleaned_matches")

```

Code 2 – Chargement et nettoyage des données

#### 02\_silver\_to\_gold.ipynb (Extraits)

```
1 # PLAYER STATISTICS
2 df_player_stats = df_players \
3     .filter(col("account_id").isNotNull()) \
4     .groupBy("account_id").agg(
5         count("match_id").alias("total_matches"),
6         sum("win").alias("total_wins"),
7         round(avg("kills"), 2).alias("avg_kills"),
8         round(avg("kda_calculated"), 2).alias("avg_kda"),
9         round(avg("gold_per_min"), 2).alias("avg_gpm")) \
10    .withColumn("win_rate",
11        round(col("total_wins")/col("total_matches")*100, 2))
12
13 # HERO META TIER CLASSIFICATION
14 total = df_matches.count()
15 df_hero = df_hero \
16     .withColumn("presence_rate",
17         round(col("total_presence")/(total*2)*100, 2)) \
18     .withColumn("meta_tier",
19         when(col("presence_rate") > 50, "S-Tier")
20             .when(col("presence_rate") > 30, "A-Tier")
21             .when(col("presence_rate") > 15, "B-Tier")
22             .when(col("presence_rate") > 5, "C-Tier")
23             .otherwise("D-Tier"))
```

Code 3 – Création des tables Gold

## Code Source du Simulateur de Streaming

```

1 import time, pandas as pd, os
2 from datetime import datetime
3 from azure.storage.filedatalake import DataLakeServiceClient
4
5 STORAGE_ACCOUNT = "dota2lakehousenew"
6 STORAGE_KEY = os.getenv("AZURE_STORAGE_ACCOUNT_KEY")
7 CONTAINER = "data"
8 DIRECTORY = "bronze"
9 SOURCE_FILE = "main_metadata.csv"
10
11 def get_client():
12     url = f"https://'{STORAGE_ACCOUNT}'.dfs.core.windows.net"
13     return DataLakeServiceClient(account_url=url, credential=STORAGE_KEY)
14
15 def stream_data():
16     print("--- Starting Streaming Simulation ---")
17     df = pd.read_csv(SOURCE_FILE)
18     print(f"Loaded {len(df)} rows.")
19
20     client = get_client()
21     fs_client = client.get_file_system_client(CONTAINER)
22     dir_client = fs_client.get_directory_client(DIRECTORY)
23
24     BATCH_SIZE, SLEEP_TIME = 5, 3
25
26     for i in range(0, 50, BATCH_SIZE):
27         chunk = df.iloc[i:i+BATCH_SIZE]
28         json_data = chunk.to_json(orient='records')
29
30         ts = datetime.now().strftime("%Y%m%d_%H%M%S")
31         fname = f"raw_matches_{ts}.json"
32
33         fc = dir_client.get_file_client(fname)
34         fc.create_file()
35         fc.append_data(data=json_data, offset=0, length=len(json_data))
36         fc.flush_data(len(json_data))
37         print(f"Uploaded {fname}")
38
39         time.sleep(SLEEP_TIME)
40
41     print("--- Simulation Complete ---")
42
43 if __name__ == "__main__":
44     stream_data()

```

Code 4 – stream\_simulator.py

## Commandes SQL de Création des Tables

```
1  -- Creation du schema
2  CREATE DATABASE IF NOT EXISTS esports_gold;
3
4  -- Table player_stats
5  CREATE TABLE IF NOT EXISTS esports_gold.player_stats
6  USING DELTA LOCATION
7  'abfss://data@dota2lakehousenew.dfs.core.windows.net/gold/player_stats';
8
9  -- Table hero_stats
10 CREATE TABLE IF NOT EXISTS esports_gold.hero_stats
11 USING DELTA LOCATION
12 'abfss://data@dota2lakehousenew.dfs.core.windows.net/gold/hero_stats';
13
14 -- Table daily_stats
15 CREATE TABLE IF NOT EXISTS esports_gold.daily_stats
16 USING DELTA LOCATION
17 'abfss://data@dota2lakehousenew.dfs.core.windows.net/gold/daily_stats';
18
19 -- Table ml_features
20 CREATE TABLE IF NOT EXISTS esports_gold.ml_features
21 USING DELTA LOCATION
22 'abfss://data@dota2lakehousenew.dfs.core.windows.net/gold/ml_features';
23
24 -- Verification
25 SHOW TABLES IN esports_gold;
```

Code 5 – Création du schéma et des tables Gold

## Code Source du Modèle ML

```

1 import mlflow
2 from pyspark.ml.feature import VectorAssembler
3 from pyspark.ml.classification import RandomForestClassifier
4 from pyspark.ml.evaluation import MulticlassClassificationEvaluator
5 from pyspark.ml import Pipeline
6
7 # Chargement des données
8 path = f"abfss://{{container}}@{{storage_account}}.dfs.core.windows.net/gold/ml_features"
9 df = spark.read.format("delta").load(path)
10
11 # Sélection des features
12 cols = ["kills", "deaths", "assists", "gold_per_min", "xp_per_min",
13         "hero_damage", "tower_damage", "last_hits", "level",
14         "duration_minutes", "win"]
15 data = df.select(cols).dropna()
16
17 # Split Train/Test
18 train, test = data.randomSplit([0.8, 0.2], seed=42)
19
20 # Pipeline ML
21 features = [c for c in cols if c != "win"]
22 assembler = VectorAssembler(inputCols=features, outputCol="features")
23 rf = RandomForestClassifier(
24     labelCol="win", featuresCol="features", numTrees=50, maxDepth=10)
25 pipeline = Pipeline(stages=[assembler, rf])
26
27 # Entrainement avec MLflow
28 mlflow.autolog()
29 model = pipeline.fit(train)
30 predictions = model.transform(test)
31
32 # Evaluation
33 evaluator = MulticlassClassificationEvaluator(
34     labelCol="win", predictionCol="prediction", metricName="accuracy")
35 acc = evaluator.evaluate(predictions)
36 print(f"Accuracy: {acc*100:.2f}%")
37
38 # Matrice de confusion
39 predictions.groupBy("win", "prediction").count().show()
40
41 # Importance des features
42 import pandas as pd
43 imp = model.stages[-1].featureImportances.toArray()
44 df_imp = pd.DataFrame({"Feature": features, "Importance": imp})
45 print(df_imp.sort_values("Importance", ascending=False))

```

Code 6 – Pipeline Machine Learning complet

## Technologies Utilisées

TABLEAU 13 – Stack technologique du projet

Catégorie	Technologie	Version
Plateforme Cloud	Microsoft Azure	-
Storage Account	dota2lakehousenew	ADLS Gen2
Orchestration	Azure Data Factory	V2
Compute	Azure Databricks	13.3 LTS
Processing	Apache Spark	3.4.1
Langage	Python / PySpark	3.10
Format de Données	Delta Lake	2.4.0
ML Tracking	MLflow	Auto-enabled
Visualisation	Power BI	-