



GALATASARAY ÜNİVERSİTESİ

Mühendislik ve Teknoloji Fakültesi

Bilgisayar Mühendisliği Bölümü

INF438 İLERİ VERİTABANLARI

Final Proje

Azure Lakehouse Mimarisi Üzerinde
E-Sport Analizi (Dota 2)

GRUP 3

Üye 1 : Sabri Taner Burak ALTINDAL – 22401030

Üye 2 : Emirhan Karatepe – 19401830

Üye 3 : Kaan Çolakoğlu – 21401946

Üye 4 : Ceren Akbaş – 22401028

Akademik Yıl 2025–2026

İçindekiler

1 Giriş	5
1.1 Proje Bağlamı	5
1.2 Kullanılan Veri Seti	5
1.3 Proje Hedefleri	5
1.4 Kullanılan Azure Servisleri	6
2 Mimari Tasarım	6
2.1 Medallion Mimarisi	7
2.2 Azure Data Lake Storage Gen2 Yapısı	8
2.3 Veri Akış Diyagramı	8
3 Uygulama	8
3.1 Batch İşleme Pipeline’ı	9
3.2 Tetikleyici Yapılandırması	10
3.3 Databricks Notebook’ları	11
3.4 Streaming Simülasyonu	11
3.5 ETL Dönüşümleri	12
3.6 Delta Lake Tablo Yönetimi	14
4 Veri Analizi ve Makine Öğrenmesi	14
4.1 Gold Tablolarına Erişim ve SQL Çalıştırma (Databricks)	15
4.2 Analitik SQL Sorguları (Sonuçlar)	16
4.3 Power BI Görselleştirmesi	21
4.4 Makine Öğrenmesi Modeli	25
4.5 Model Sonuçları	26
4.6 Özellik Önem Analizi	26
4.7 MLflow Takibi	27
5 Karşılaşılan Zorluklar ve Çözümler	27
5.1 Azure Kaynak Sınırlamaları	28
5.2 Streaming Simülasyonunda Dosya Çakışmaları	28
5.3 Veri Hacmi ve Maliyet Kısıtlamaları	28
5.4 Azure Storage Anahtar Yönetimi	28
5.5 Erişim Yönetimi Karmaşıklığı (IAM)	28
5.6 Databricks Cluster Erişim Sorunları	29
6 Sonuç	29
6.1 Proje Özeti	30
6.2 Öğrenilen Dersler	30
6.3 Genel Sonuç	30
A EKLER	31
Ekler	31
A.1 Databricks Notebook Kaynak Kodları	31
A.2 Streaming Simülatörü Kaynak Kodu	33
A.3 Tablo Oluşturma SQL Komutları	34

A.4 ML Model Kaynak Kodu	35
A.5 Kullanılan Teknolojiler	36

Şekil Listesi

1	ADLS Gen2'deki Bronze/Silver/Gold klasör yapısının Azure Portal görünümü	8
2	Veri akış diyagramı	9
3	4 kaynak dosyanın (212 MB) Bronze konteynerine başarılı bir şekilde alın- diği onaylayan <i>Copy Data</i> aktivitesinin yürütme detayları.	9
4	Azure Data Factory'de pl_master_esports pipeline diyagramı	10
5	ADF Monitor ekranında "Succeeded" durumu ve "Triggered by tr_daily_esports" mesajı	11
6	stream_simulator.py yürütme çıktısı	12
7	silver_to_gold notebook'unda Tier sınıflandırma çıktısı	13
8	Databricks Catalog'da esports_gold şeması altındaki tabloların listesi . . .	14
9	ADLS Gen2'den Gold tablolarının yüklenmesi ve Databricks'te geçici gö- rünümlerin oluşturulması	16
10	SQL Sonucu: Ortalama KDA'ya göre ilk 10 oyuncu (kills, deaths, assists ile)	17
11	SQL Sonucu: Günlük istatistiklere genel bakış (match_count ve dakika cinsinden ortalama süre)	18
12	SQL Sonucu: Ortalama GPM ve XPM'ye göre ilk 10 kahraman	19
13	SQL Sonucu: Savaş odaklı ilk 10 kahraman (kills/assists) ve oynanan oyun sayısı	20
14	SQL Sonucu: times_played'e göre en popüler ilk 10 kahraman (wins ve win_rate ile)	21
15	Power BI — Genel Bakış Sayfası: zamansal filtre, maç sayısının günlük eğilimi ve global KPI'ler	22
16	Power BI — İlk 10 Oyuncu Sayfası: ortalama KDA vs kazanma oranı kar- şılaştırması ve detay tablosu	23
17	Power BI — Kahraman Meta Sayfası: presence rate vs win rate ilişkisi ve en çok bulunan kahramanların ilk 10'u	24
18	Power BI — Süre ve Korelasyonlar Sayfası: ortalama süre eğilimi ve süre vs öldürmeler korelasyonu	25
19	MLflow arayüzünde model sonuçları ve parametreler	27

Tablo Listesi

1	Veri Seti Özellikleri	5
2	Kullanılan Microsoft Azure Servisleri	6
3	Silver katmanı tabloları	7
4	Gold katmanı tabloları	7
5	Tetikleyici yapılandırması	10
6	Databricks küme yapılandırması	11
7	Meta Tier sınıflandırma kriterleri	13
8	Süre analizi sonuçları	14
9	Kullanılan Özellikler (Features)	25
10	Karışıklık Matrisi (Confusion Matrix)	26
11	Özellik Önem Sıralaması	26
12	Elde Edilen Temel Sonuçlar	30
13	Proje Teknoloji Yığını	36

1. Giriş

1.1 Proje Bağlamı

E-spor endüstrisi son on yılda üstel bir büyümeye yaşadı ve bugün milyarlarca dolarlık bir ekosistem haline geldi. Bu büyümeye ile birlikte, profesyonel turnuvalarda üretilen veri miktarı da önemli ölçüde arttı. Oyuncu performans analizi, takım stratejilerinin optimizasyonu ve maç sonuçlarının tahmini, veri mühendisliği ve veri biliminin pratik uygulamalar bulduğu kritik alanlardır.

Bu proje kapsamında, Dota 2 profesyonel maç verileri üzerinde işleme, analiz ve tahlime dayalı modellerin geliştirilmesi için eksiksiz bir veri mühendisliği çözümü tasarlanmıştır ve uygulanmıştır. Proje, Microsoft Azure platformunda gerçekleştirilmiş ve modern bulut mimarisi ilkeleri benimsenmiştir.

1.2 Kullanılan Veri Seti

Bu projede kullanılan veri seti, Kaggle platformunda yayınlanan **“Dota 2 Pro League Matches 2023”** tür. Bu veri seti, 2023 yılı boyunca gerçekleşen profesyonel Dota 2 turnuva maçlarının detaylı kayıtlarını içerir.

Tablo 1: Veri Seti Özellikleri

Dosya Adı	Boyut	Açıklama
main_metadata.csv	9.09 MB	Maç üst verileri (Metadata)
players_reduced.csv	152.00 MB	Oyuncu istatistikleri
picks_bans.csv	33.01 MB	Seçim/Yasaklama verileri
teams.csv	8.09 MB	Takım bilgileri

CSV formatı ve veri setinin ilişkisel yapısı, Bronze katmanından Silver katmanına geçiş sırasında kapsamlı bir temizleme ve birleştirme süreci gerektirmiştir.

1.3 Proje Hedefleri

Bu projenin ana hedefleri şu şekilde tanımlanmıştır:

1. **Lakehouse Mimarisinin Kurulması:** Azure Data Lake Storage Gen2 üzerinde Medallion Mimarisi'nin (Bronze-Silver-Gold) uygulanması
2. **Batch İşleme Pipeline'i:** Azure Data Factory ve Databricks kullanarak otomatik ETL pipeline'larının oluşturulması
3. **Streaming Simülasyonu:** Gerçek zamanlı veri akışını taklit eden bir simülatörün geliştirilmesi
4. **Veri Dönüşümü:** PySpark kullanarak ham verilerin analitik tablolara dönüştürülmesi
5. **Gelişmiş Analiz:** SQL ve PySpark ile oyuncu ve kahraman performans analizleri

- 6. Makine Öğrenmesi:** Maç sonucu tahmin modelinin geliştirilmesi ve MLflow ile izlenmesi

1.4 Kullanılan Azure Servisleri

Tablo 2: Kullanılan Microsoft Azure Servisleri

Service	Kullanım Amacı
Azure Data Lake Storage Gen2	Lakehouse katmanları için merkezi depolama
Azure Data Factory	Pipeline orkestrasyonu ve planlama
Azure Databricks	Veri işleme ve ML geliştirme
Delta Lake	ACID uyumlu performanslı veri formatı
MLflow	Makine öğrenmesi deneylerinin takibi
Power BI	Görselleştirme ve dashboard oluşturma

Tüm kaynaklar, Microsoft Azure for Students aboneliği kullanılarak oluşturulmuş ve maliyet optimizasyonu dikkate alınarak yönetilmiştir.

2. Mimari Tasarım

2.1 Medallion Mimarisi

Bu projede, modern veri mühendisliği uygulamalarında yaygın olarak benimsenen **Medallion Mimarisi** uygulanmıştır. Bu mimari, verileri kalite ve işleme seviyelerine göre üç katmana ayırrı: **Bronze**, **Silver** ve **Gold**.

2.1.1 Bronze Katmanı (Ham Veri)

Bronze katmanı, kaynak sistemlerden gelen verilerin herhangi bir dönüşüm olmadan olduğu gibi saklandığı katmandır:

- **Amaç:** Veri kaynağının birebir bir kopyasını korumak
- **Format:** CSV ve JSON (akış simülasyonu için)
- **İçerik:** 4 ana CSV dosyası + akış JSON dosyaları
- **Kullanım:** Hata durumunda kaynağına dönüş, denetim ve izlenebilirlik

2.1.2 Silver Katmanı (Temizlenmiş Veri)

Silver katmanı, ham verilerin temizlendiği ve standardize edildiği ara katmandır:

- **Amaç:** Temiz ve tutarlı veri sağlamak
- **Format:** Delta Lake (Parquet tabanlı)
- **Dönüşümler:** Null değer temizleme, tip standartizasyonu, tekrar eliminasyonu, aykırı değer tespiti

Tablo 3: Silver katmanı tabloları

Tablo	Kayıt Sayısı
cleaned_matches	29 809
cleaned_players	8 456
cleaned_picks_bans	710 414

2.1.3 Gold Katmanı (Analistik Veri)

Gold katmanı, iş mantığı ile zenginleştirilmiş verileri içerir:

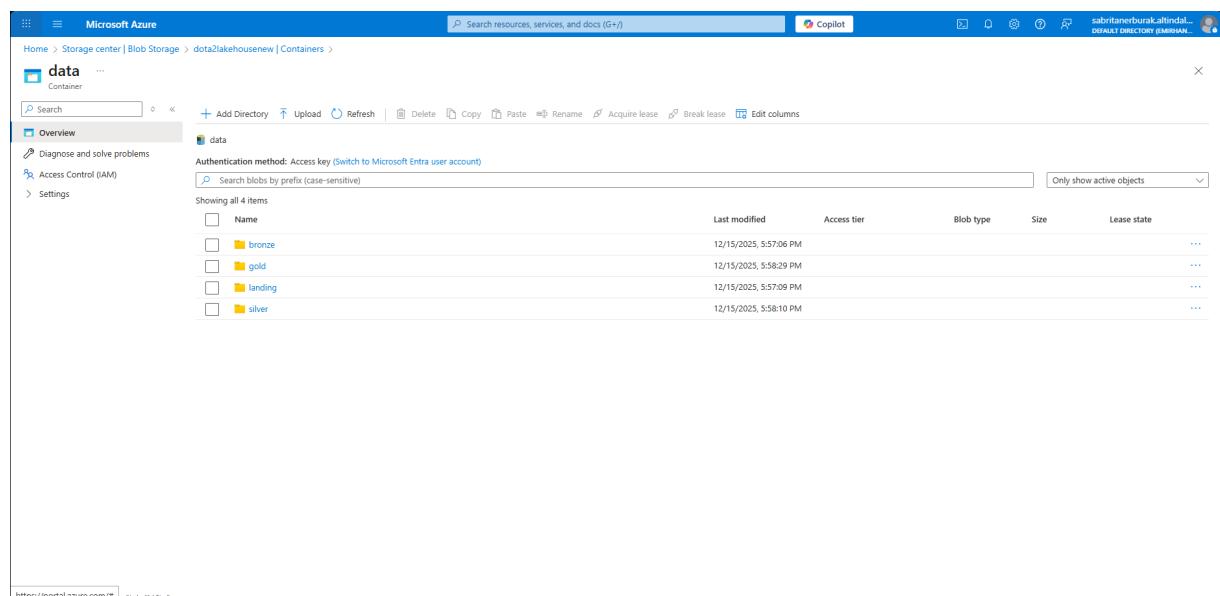
Tablo 4: Gold katmanı tabloları

Tablo	Açıklama	Kayıtlar
esports_gold.player_stats	Performans metrikleri	813
esports_gold.hero_stats	Kahraman istatistikleri	123
esports_gold.daily_stats	Günlük istatistikler	365
esports_gold.ml_features	ML özellikleri	8 312

2.2 Azure Data Lake Storage Gen2 Yapısı

Tüm Lakehouse katmanları Azure Data Lake Storage Gen2 üzerinde konumlandırılmıştır:

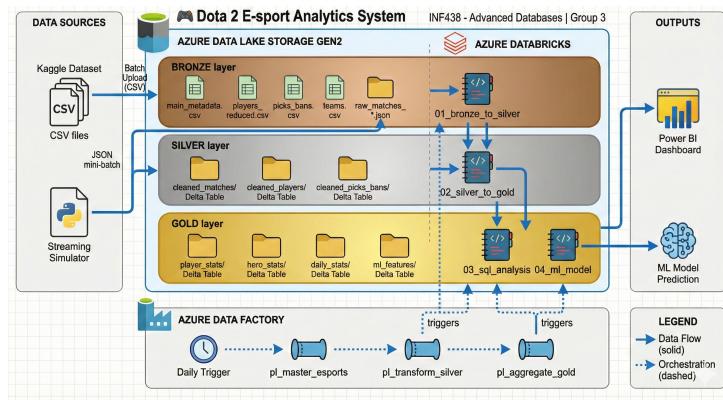
- **Storage Account :** dota2lakehousenew
- **Container :** data
- **Yollar:**
 - Bronze : abfss://data@dota2lakehousenew.../bronze/
 - Silver : abfss://data@dota2lakehousenew.../silver/
 - Gold : abfss://data@dota2lakehousenew.../gold/



Şekil 1: ADLS Gen2'deki Bronze/Silver/Gold klasör yapısının Azure Portal görünümü

2.3 Veri Akış Diyagramı

Sistemin uçtan uca veri akışı şu yolu izler: Kaggle ve akış simülatöründen gelen kaynak veriler önce Bronze katmanında depolanır. Ardından, Azure Data Factory, verileri Silver ve ardından Gold katmanlarına dönüştüren Databricks notebook'larını orkestre eder. Son olarak, Gold verileri SQL analizlerini, makine öğrenmesi modelini (MLflow tarafından izlenir) ve Power BI görselleştirmelerini besler.



Şekil 2: Veri akış diyagramı

3. Uygulama

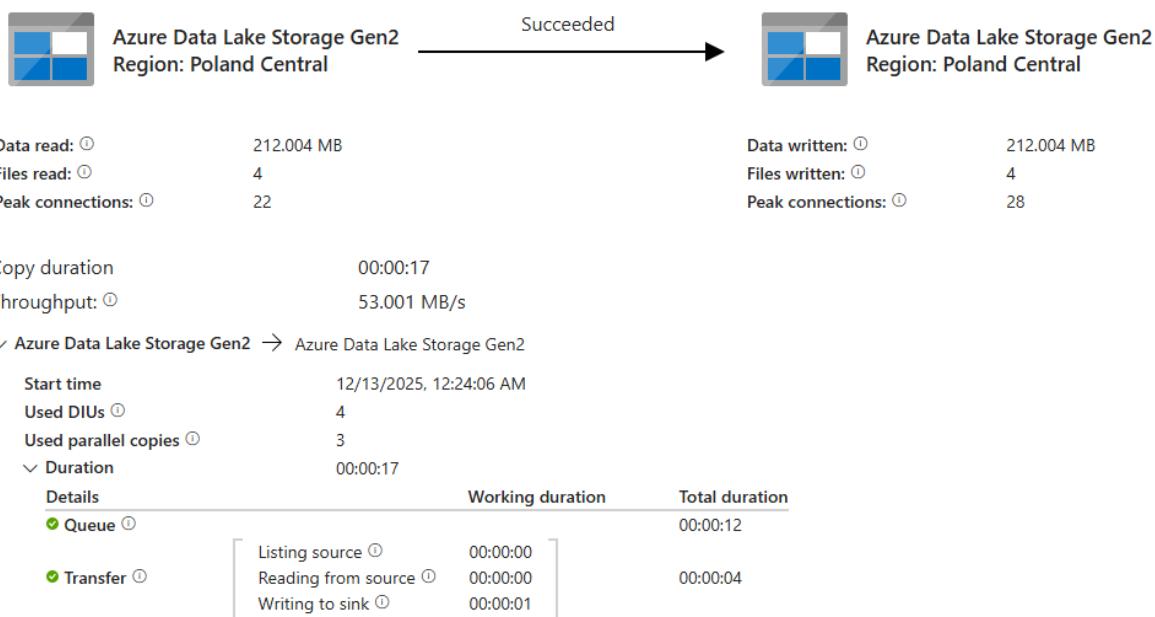
3.1 Batch İşleme Pipeline'sı

Batch işleme pipeline orkestrasyonu için Azure Data Factory (ADF) kullanılmıştır. Proje'de oluşturulan ana pipeline **pl_master_esports** adını taşımaktadır.

Bu işlemin yukarı akışında,

PL_Ingest_Kaggle_To_Bronze adlı özel bir veri alma pipeline'sı kurulmuştur. Bu pipeline, ham dosyaları (CSV) ve azaltılmış veri setini Data Lake'in **bronze/** konteynerine otomatik olarak aktarmak için bir *Copy Data* aktivitesi kullanarak Databricks notebook'ları için veri kullanılabilirliğini sağlar.

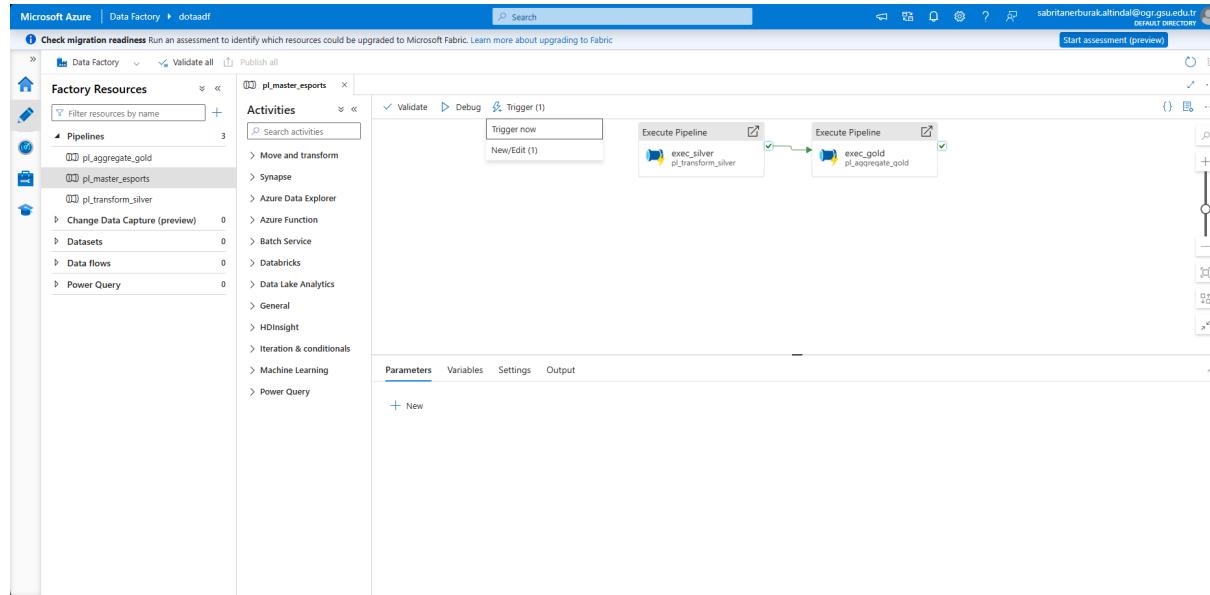
Activity run id: 0d82b85a-cd5f-4180-8ed5-f1b8f717fae3



Şekil 3: 4 kaynak dosyanın (212 MB) Bronze konteynerine başarılı bir şekilde alındığını onaylayan *Copy Data* aktivitesinin yürütme detayları.

Pipeline yapısı aşağıdaki gibi tasarlanmıştır:

- **Aktivite 1:** nb_bronze_to_silver (Databricks Notebook)
- **Aktivite 2:** nb_silver_to_gold (Aktivite 1'e bağlı)
- **Aktivite 3:** nb_ml_model (Aktivite 2'ye bağlı)



Şekil 4: Azure Data Factory'de pl_master_esports pipeline diyagramı

3.2 Tetikleyici Yapılandırması

Otomatik pipeline yürütmesi için tr_daily_esports adlı bir tetikleyici oluşturulmuştur.

Tablo 5: Tetikleyici yapılandırması

Parametre	Değer
Tetikleyici Adı	tr_daily_esports
Tip	Schedule Trigger
Frekans	Günlük (Daily)
Başlangıç Saati	02:00 UTC
Durum	Aktif

The screenshot shows the Microsoft Azure Data Factory Pipeline runs monitor. It displays a list of pipeline runs for the last 24 hours. One specific run, 'pl_transform_silver', is highlighted with a red box. This run was triggered by the message 'tr_daily_esports'. The run status is 'Succeeded'.

Pipeline name	Run start	Run end	Duration	Triggered by	Status	Run ID	
pl_aggregate_gold	12/21/2025, 2:04:22 AM	12/21/2025, 2:05:13 AM	51s	98abdd28-5a99-42e...	Succeeded	Original	4677882f-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	b18befa0-4582-4681-821b-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-9b61-bc
pl_aggregate_gold	12/21/2025, 1:54:06 AM	12/21/2025, 1:55:01 AM	56s	db00f79-b51c-4b1f...	Succeeded	Original	c539d82c-3567-4b28-bd74-11
pl_transform_silver	12/21/2025, 1:43:23 AM	12/21/2025, 1:54:04 AM	10m 42s	e592d65c-6331-4c0...	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	c95e5487-bd0b-48c4-bb1e-f9

Şekil 5: ADF Monitor ekranında “Succeeded” durumu ve “Triggered by tr_daily_esports” mesajı

3.3 Databricks Notebook’ları

Veri dönüştürme işlemleri Azure Databricks üzerinde PySpark kullanılarak gerçekleştirilmiştir.

Tablo 6: Databricks küme yapılandırması

Parametre	Değer
Küme Modu	Standard
Node Tipi	Standard_DS3_v2
Otomatik Sonlandırma	120 dakika
Databricks Runtime	13.3 LTS (Spark 3.4.1)

Oluşturulan Notebook’lar:

1. **setup_connection.ipynb** : Azure ADLS Gen2 bağlantı testi
2. **01_bronze_to_silver.ipynb** : Ham veri temizleme
3. **02_silver_to_gold.ipynb** : İş mantığı uygulaması
4. **04_ml_model.ipynb** : ML model eğitimi

Bu notebook’ların tam kaynak kodu **Ek A.1**’te mevcuttur.

3.4 Streaming Simülasyonu

Gerçek zamanlı bir veri kaynağı olmadığından, streaming davranışını simüle eden bir Python betiği geliştirilmiştir. **stream_simulator.py** betiği aşağıdaki mantığa göre çalışır:

1. Kayıtlar main_metadata.csv dosyasından okunur
2. Kayıtlar 5'li mini-batch'lere bölünür
3. Her mini-batch için 3 saniyelik yapay bir gecikme uygulanır
4. Mini-batch benzersiz bir zaman damgası ile adlandırılır ve Bronze'a yazılır

```

25     def stream_data():
26         file_client.flush_data(len(json_data))
27         print("Done.")
28     except Exception as e:
29         print(f"Upload Error: {e}")
30
31     # 5. Wait (Simulate real-time gap)
32     time.sleep(SLEEP_TIME)
33
34     print("\n--- Simulation Complete ---")
35
36 if __name__ == "__main__":
37     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.

```

Şekil 6: stream_simulator.py yürütme çıktısı

Simülatör çalıştırıldığında, Bronze katmanında raw_matches_YYYYMMDD_HHMMSS.json formatında dosyalar oluşturulur (toplamda 10 dosya).

Simülatörün tam kaynak kodu **Ek A.2**'te mevcuttur.

3.5 ETL Dönüşümleri

3.5.1 Bronze → Silver Dönüşümü

Bronze katmanındaki ham CSV verileri kapsamlı bir dönüşüm sürecinden geçmiştir:

- **Yükleme:** Şema çıkarımı ile CSV dosyalarının okunması
- **Temizleme:** Gereksiz sütunların kaldırılması, null değerlerin filtrelenmesi
- **Doğrulama:** Geçersiz süreye (<5 dk veya >2 saat) sahip maçların filtrelenmesi
- **Zenginleştirme:** Yeni özelliklerin oluşturulması (duration_minutes, kda_calculated)
- **Aykırı Değer Tespit:** 3-sigma kuralı kullanılarak işaretleme

3.5.2 Silver → Gold Dönüşümü

Temizlenmiş veriler iş mantığı ve gelişmiş hesaplamalar almıştır.

Oyuncu İstatistikleri (player_stats): Ortalamalar (KDA, GPM, XPM) ve kazanma oranı hesaplaması ile oyuncu hesabına göre toplama. Sonuç: **813 benzersiz oyuncu** için hesaplanan istatistikler.

Kahraman Meta Analizi (hero_stats): Kahramanları varlık oranlarına göre 5 katmana sınıflandıran bir algoritma (Tablo 7'ye bakınız).

Tablo 7: Meta Tier sınıflandırma kriterleri

Tier	Varlık Oranı Kriteri
S-Tier	> 50%
A-Tier	> 30%
B-Tier	> 15%
C-Tier	> 5%
D-Tier	≤ 5%

```

# MAGIC
# MAGIC %load ./silver_to_gold
# MAGIC
# MAGIC # Read data from yesterday
# MAGIC df_hero_stats = spark.read.parquet('dt_hero_stats')
# MAGIC
# MAGIC df_hero_stats = df_hero_stats \
# MAGIC     .withColumn("presence_rate", round(col("total_presence") / (total_matches * 2) * 100, 2)) \
# MAGIC     .withColumn("meta_tier",
# MAGIC     when(col("presence_rate") > 50, "S-tier")
# MAGIC     .when(col("presence_rate") > 30, "A-tier")
# MAGIC     .when(col("presence_rate") > 15, "B-tier")
# MAGIC     .when(col("presence_rate") > 5, "C-tier")
# MAGIC     .otherwise("D-tier")
# MAGIC )
# MAGIC
# MAGIC print(f"Hero stats created: {df_hero_stats.count()} heroes")
# MAGIC df_hero_stats.orderBy(desc("presence_rate")).select(
# MAGIC     "hero_id", "times_played", "win_rate", "pick_count", "ban_count", "presence_rate", "meta_tier"
# MAGIC ).show(10)
# MAGIC
# MAGIC # Create hero statistics
# MAGIC df_ban_stats = df_hero_stats.groupBy("hero_id").agg(
# MAGIC     sum("times_played").alias("times_played"),
# MAGIC     sum("win_rate").alias("win_rate"),
# MAGIC     sum("pick_count").alias("pick_count"),
# MAGIC     sum("ban_count").alias("ban_count"),
# MAGIC     sum("presence_rate").alias("presence_rate"),
# MAGIC     count("meta_tier").alias("meta_tier")
# MAGIC )
# MAGIC
# MAGIC df_ban_stats.createOrReplaceTempView("ban_heroes")
# MAGIC
# MAGIC df_hero_performance = df_ban_stats.select(
# MAGIC     "hero_id", "times_played", "win_rate", "pick_count", "ban_count", "presence_rate", "meta_tier"
# MAGIC )
# MAGIC
# MAGIC df_hero_stats = df_ban_stats.withColumn("meta_tier", when(df_ban_stats.meta_tier == 1, "S-tier")
# MAGIC .when(df_ban_stats.meta_tier == 2, "A-tier")
# MAGIC .when(df_ban_stats.meta_tier == 3, "B-tier")
# MAGIC .when(df_ban_stats.meta_tier == 4, "C-tier")
# MAGIC .otherwise("D-tier"))
# MAGIC
# MAGIC df_pick_stats = df_ban_stats.withColumn("meta_tier", when(df_ban_stats.meta_tier == 1, "S-tier")
# MAGIC .when(df_ban_stats.meta_tier == 2, "A-tier")
# MAGIC .when(df_ban_stats.meta_tier == 3, "B-tier")
# MAGIC .when(df_ban_stats.meta_tier == 4, "C-tier")
# MAGIC .otherwise("D-tier"))
# MAGIC
# MAGIC
# MAGIC [3/3] Creating Hero Statistics (Meta Analysis)...
# MAGIC Hero stats created: 123 heroes
# MAGIC +-----+-----+-----+-----+-----+
# MAGIC | hero_id|times_played|win_rate|pick_count|ban_count|presence_rate|meta_tier|
# MAGIC +-----+-----+-----+-----+-----+
# MAGIC | 86.0| 330| 51.21| 9932| 5983| 26.69| B-Tier|
# MAGIC | 137.0| 81| 51.85| 3814| 9844| 21.57| B-Tier|
# MAGIC | 93.0| 89| 53.93| 2943| 9794| 21.36| B-Tier|
# MAGIC | 106.0| 187| 58.27| 5306| 7416| 21.34| B-Tier|
# MAGIC | 120.0| 122| 58.26| 4447| 8167| 21.30| B-Tier|
# MAGIC | 1.0| 39| 53.21| 4959| 7274| 20.89| B-Tier|
# MAGIC | 38.0| 96| 55.21| 3855| 7274| 18.67| B-Tier|
# MAGIC | 53.0| 98| 57.78| 3814| 6855| 17.9| B-Tier|
# MAGIC | 85.0| 138| 52.31| 4941| 5435| 17.4| B-Tier|
# MAGIC | 114.0| 64| 57.81| 3049| 7215| 17.22| B-Tier|
# MAGIC +-----+-----+-----+-----+-----+
# MAGIC only showing top 10 rows
# MAGIC 
```

Şekil 7: silver_to_gold notebook'unda Tier sınıflandırma çıktısı

Maç Süresi Analizi:

Tablo 8: Süre analizi sonuçları

Süre	Maçlar	Ort Öldürme	Radiant Kazanma
Çok Kısa (<25dk)	4 176	44.94	53.54%
Kısa (25-35dk)	13 548	56.74	50.41%
Orta (35-45dk)	8 162	65.33	49.36%
Uzun (45-55dk)	2 839	68.19	49.52%
Çok Uzun (>55dk)	1 084	80.77	50.46%

3.6 Delta Lake Tablo Yönetimi

Gold katmanı tabloları Databricks Unity Catalog'da **esports_gold** şeması altında yönetilmektedir. SQL oluşturma komutları **Ek A.3**'de mevcuttur.

The screenshot shows the Databricks interface with the Catalog Explorer open. The left sidebar shows various workspace sections like Workspace, Recents, Catalog, etc. The Catalog section is selected. In the main area, the 'samples' schema is selected under the 'esports_gold' catalog. The 'Overview' tab is active, showing a list of tables: default, nyctaxi, trips, tpch, customer, lineitem, nation, orders, part, partsupp, region, and supplier. There are also sections for 'Legacy' and 'hive metastore' which are currently empty.

Şekil 8: Databricks Catalog'da esports_gold şeması altındaki tabloların listesi

4. Veri Analizi ve Makine Öğrenmesi

4.1 Gold Tablolarına Erişim ve SQL Çalıştırma (Databricks)

4.1.1 Azure Data Lake Storage'a Bağlantı (ADLS Gen2)

Gold katmanının analistik tabloları Azure Data Lake Storage Gen2 üzerinde Delta Lake formatında saklanmaktadır. Bizim durumumuzda, *Azure for Students* aboneliğinin belirli sınırlamaları (kotalar / RBAC izinleri) grup üyelerinin tümü için doğrudan ve istikrarlı erişimi engellemiştir. Projenin sürekliliğini sağlamak için, *Storage Accounta* erişim, gerekli izinlere sahip bir grup üyesi tarafından sağlanan bir erişim anahtarı aracılığıyla yapılmıştır. Bu anahtar raporda veya sürüm kontrol edilmiş kodda asla açık metin olarak gösterilmmez; Databricks ortam değişkenleri aracılığıyla enjekte edilir.

CPU kısıtlamaları ve paylaşılan küme kullanımı. *Azure for Students* aboneliği ile notebook çalışma sırasında, bazı Spark işlemlerini kesintiye uğratabilecek kaynak sınırlamalarıyla (CPU/vCore kotası) ve yavaşlamalarla karşılaştık. Kilitlenmeleri önlemek ve işi zamanında tamamlamak için, bir grup üyesi tarafından önceden yapılandırılmış (esports-cluster) ve daha istikrarlı kaynaklara sahip (16 GB RAM ve 4 çekirdek) Databricks kümесini kullandık. Bu çözüm sayesinde, ETL dönüşümlerini doğru şekilde yürütebildik.

İkinci, Gold katmanındaki Delta tablolarına erişebildik ve analistik SQL sorgularını kesintisiz çalıştırabildik.

4.1.2 Gold Tablolarının Yüklenmesi ve Geçici Görünümlerin Oluşturulması

Gold katmanındaki Delta dosyalarında doğrudan SQL sorguları çalıştmak için, her tabloyu ADLS'den yükledik ve Databricks'te geçici görünümler oluşturduk.

```

▶ ✓ 05:08 PM (7s) 2

gold_path = "abfss://data2lakehousenew.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

```

Şekil 9: ADLS Gen2'den Gold tablolarının yüklenmesi ve Databricks'te geçici görünüm-lerin oluşturulması

4.2 Analistik SQL Sorguları (Sonuçlar)

Bu bölüm, oyuncular, kahramanlar ve günlük aktivite üzerine analizler üretmek için kullanılan ana SQL sorgularını sunmaktadır. Raporun okunabilir kalması için burada çıktıları (ekran görüntüleri) sunuyoruz; tam sorgular gereklirse ekte eklenebilir.

4.2.1 Performansa Göre İlk 10 Oyuncu (KDA)

Amaç: Ortalama KDA'larına göre en başarılı 10 oyuncuyu belirlemek (temsili olmayan durumları önlemek için filtreleme ile).

```
%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]

Table +

	1.2 account_id	1 ² ₃ total_kills	1 ² ₃ total_deaths	1 ² ₃ 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

Şekil 10: SQL Sonucu: Ortalama KDA'ya göre ilk 10 oyuncu (kills, deaths, assists ile)

4.2.2 Günlük Eğilimler: Maç Hacmi ve Ortalama Süre

Amaç: Maç sayısının günlük evrimini ve ortalama süreyi (dakikaya dönüştürme) gözlemelemek.

```
%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

➤ _sqldf: 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

Şekil 11: SQL Sonucu: Günlük istatistiklere genel bakış (match_count ve dakika cinsinden ortalama süre)

4.2.3 İlk 10 Kahraman: Ekonomik Performans (GPM/XPM)

Amaç: avg_gpm ve avg_xpm üzerinden en iyi ekonomik performansa sahip kahramanları belirlemek.

	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

Şekil 12: SQL Sonucu: Ortalama GPM ve XPM'ye göre ilk 10 kahraman

4.2.4 İlk 10 Kahraman: Savaş Stili (Öldürme/Asist)

Amaç: Hacmi (times_played) dikkate alarak avg_kills ve avg_assists metrikleri üzerinden en agresif kahramanları analiz etmek.

```
%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 fi

Table +

	1.2 hero_id	1.2 avg_kills	1.2 avg_assists	1 ² 3 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

Şekil 13: SQL Sonucu: Savaş odaklı ilk 10 kahraman (kills/assists) ve oynanan oyun sayısı

4.2.5 En Popüler İlk 10 Kahraman

Amaç: Görülme sayısına (times played) göre en çok oynanan kahramanları belirlemek. Ayrıca galibiyet sayısını ve galibiyet oranını (%) da gösteriyoruz.

```

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 ² total_picks	1 ² 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

Şekil 14: SQL Sonucu: times_played'e göre en popüler ilk 10 kahraman (wins ve win_rate ile)

Analiz, çok kısa maçlarda (<25 dk) Radiant takımının hafif bir avantaja sahip olduğunu (53.54%) ortaya koymaktadır. Bu oran daha uzun maçlarda

4.3 Power BI Görselleştirmesi

4.3.1 Power BI'da Bağlantı ve Veri Yükleme

Görselleştirme kısmı, Gold verileri kullanılarak Power BI Desktop ile gerçekleştirilmiştir. Verileri almak için "Get Data" ve ardından Azure Blob Storage kaynağı kullanılmıştır. Herkes kolayca bağlantı kuramadığı için (Azure for Students izin/kota sınırlamaları), bir grup üyesi tarafından paylaşılan erişim anahtarı kullanılmıştır.

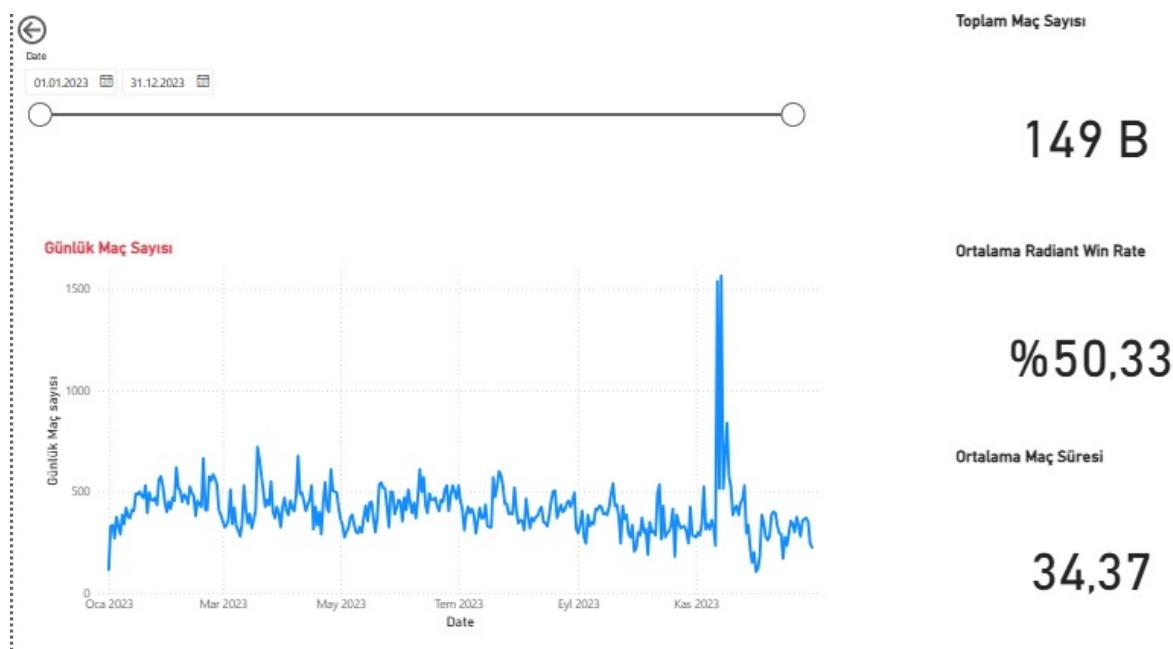
Bağlantı sağlandıktan sonra, data konteynerindeki (Gold katmanı) Parquet dosyaları bulunmuş ve daily_stats, player_stats, hero_stats ve ml_features tabloları içe aktar-

rılmıştır. Ardından, zamana göre filtreleme yapmak için bir Date tablosu eklenmiş ve match_date sütunu ile daily_stats tablosuna bağlanmıştır. Son olarak, genel eğilimleri, en iyi oyuncuları ve kahraman metasını analiz etmek için sayfalar ve grafikler oluşturulmuştur.

4.3.2 Sayfa 1: Genel Bakış ve Günlük Eğilimler

Bu sayfa 2023 aktivitesinin hızlı bir okumasını sağlar: (i) tüm göstergeleri kontrol eden bir tarih dilimleyici, (ii) maç sayısının günlük eğilimi, (iii) KPI kartları: toplam maç sayısı, ortalama Radiant kazanma oranı ve ortalama maç süresi.

Gözlemler. Yılın büyük bir bölümünde günlük dalgalanmalarla birlikte nispeten istikrarlı bir aktivite gözlemliyoruz. Belirli bir dönemde (yıl sonu) belirgin bir zirve görünür, bu bir olaya (turnuva, büyük yama veya yüksek rekabetçi aktivite) karşılık gelebilir. Ortalama Radiant kazanma oranı dengeye yakın kalır, bu da yıl boyunca kalıcı sistematik bir avantajın olmadığını gösterir. Ortalama maç süresi genel olarak istikrarlıdır, oyun stili veya meta değişikliklerini gösteren ara sıra varyasyonlarla.

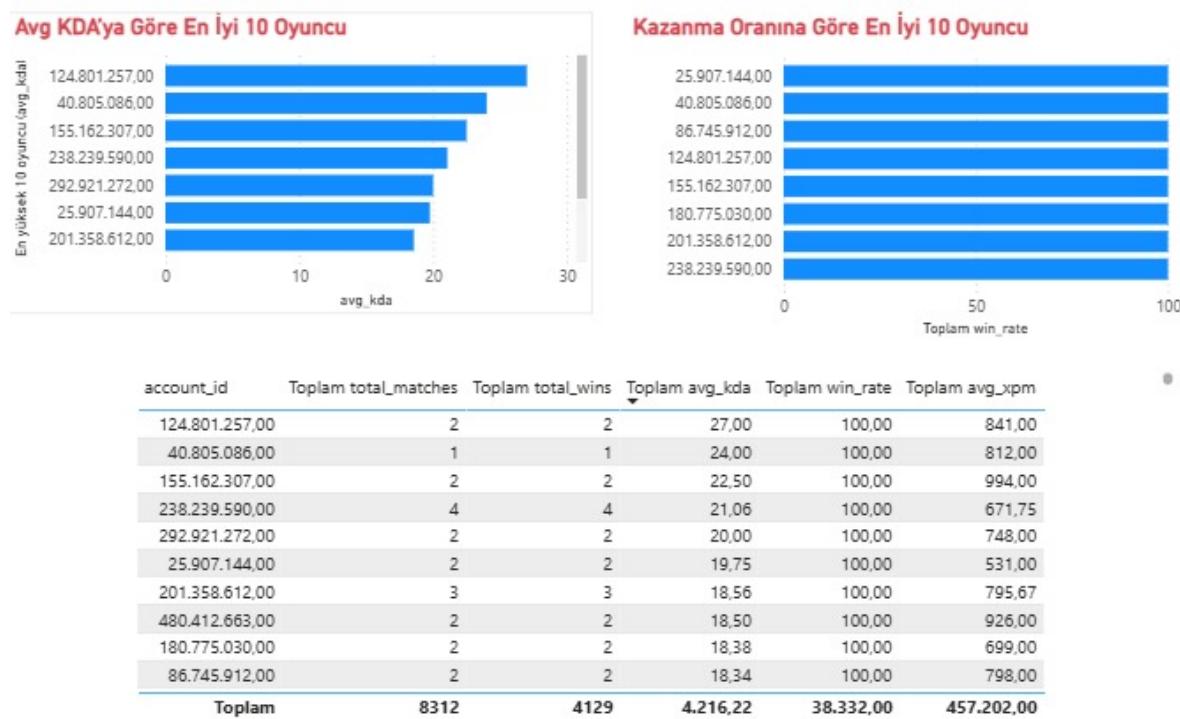


Şekil 15: Power BI — Genel Bakış Sayfası: zamansal filtre, maç sayısının günlük eğilimi ve global KPI’ler

4.3.3 Sayfa 2: İlk 10 Oyuncu

Bu sayfa oyuncuları şunlara göre karşılaştırır: (i) Ortalama KDA’ya göre ilk 10 oyuncu, (ii) Kazanma oranına göre ilk 10 oyuncu, istatistiklerin detaylı bir tablosu ile (maçlar, kazanmalar, KDA, XPM, vb.).

Gözlemler. KDA sıralaması bireysel performansa yönelik profilleri vurgular (hayatta kalma + savaş verimliliği). Kazanma oranı sıralaması mutlaka aynı değildir: bir oyuncu yüksek KDA’ya sahip olabilir ancak daha ihlaklı bir kazanma oranına sahip olabilir, bu da takım çalışmasının ve maç bağlamının önemini vurgular. Tablo, maç hacmini ve göstergeleri eşzamanlı olarak karşılaştırarak sonuçların tutarlığını doğrulamaya ve çok az maçı olan oyuncuları aşırı yorumlamaktan kaçınmaya olanak tanır.



Şekil 16: Power BI — İlk 10 Oyuncu Sayfası: ortalama KDA vs kazanma oranı karşılaştırması ve detay tablosu

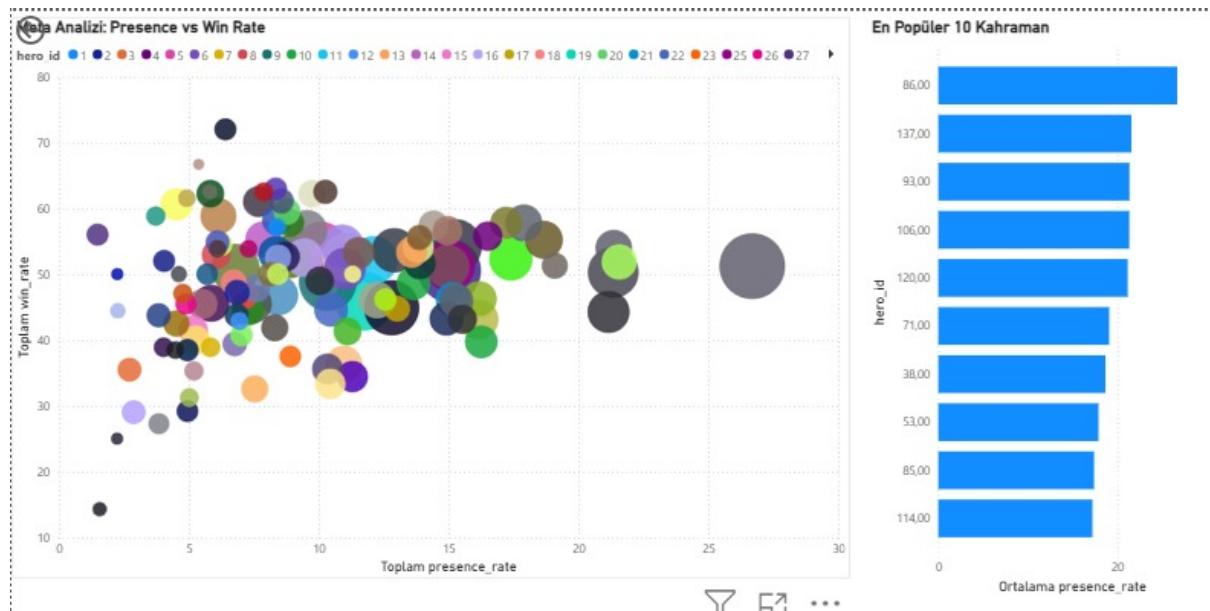
4.3.4 Sayfa 3: Kahraman Meta Analizi

Bu sayfa metaya ayrılmıştır: (i) kahraman başına *presence_rate* ve *win_rate*'i ilişkilendiren bir dağılım grafiği, (ii) en çok bulunan kahramanların ilk 10'u.

Gözlemler. Dağılım grafiği birkaç ilginç bölgeyi vurgular:

- çok mevcut ancak ortalama kazanma oranına sahip kahramanlar: genellikle "meta" veya esnek seçimler, mutlaka baskın değil;
- az mevcut ancak yüksek kazanma oranına sahip kahramanlar: "underrated" adaylar veya çok etkili durumsal seçimler;
- mevcut ve performanslı kahramanlar: baskın metayı oluştururlar ve özel dikkat hak ederler.

En çok bulunan kahramanların ilk 10'u popülariteyi sentezler ve filtrelenen dönemde en sık oynanan kahramanların tanımmasını kolaylaştırır.

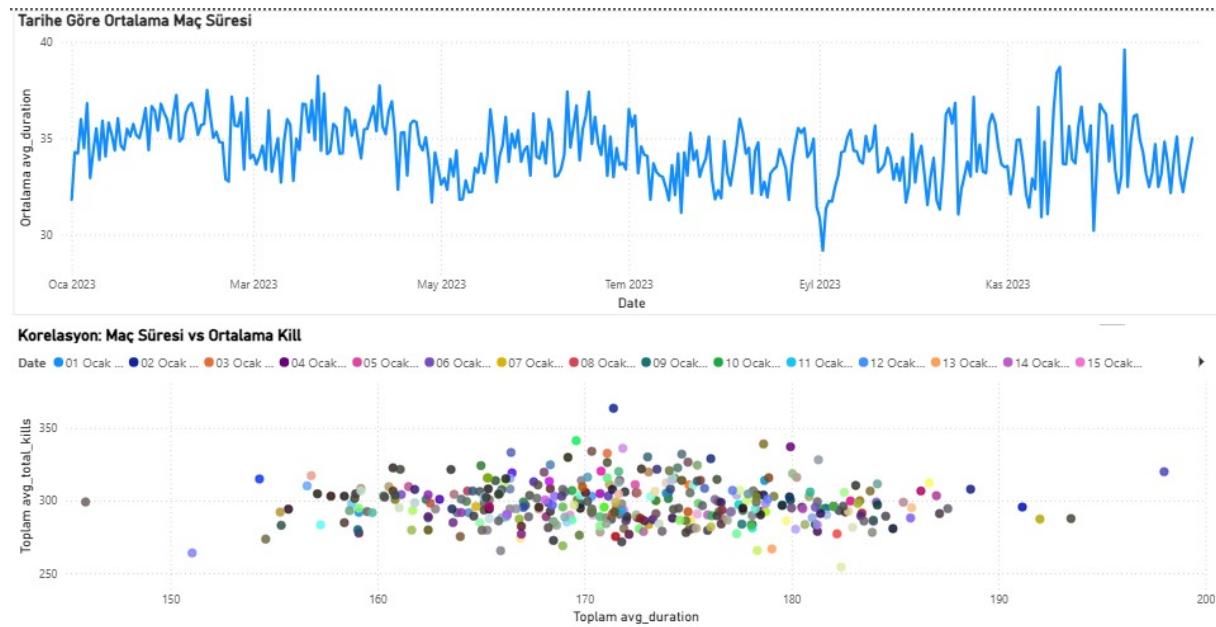


Şekil 17: Power BI — Kahraman Meta Sayfası: presence rate vs win rate ilişkisi ve en çok bulunan kahramanların ilk 10'u

4.3.5 Sayfa 4: Maç Süresi ve Korelasyonlar

Bu sayfa şunları analiz eder: (i) ortalama maç süresinin zamansal evrimi, (ii) bir dağılım grafiği aracılığıyla süre ve yoğunluk (örn. ortalama öldürmeler) arasındaki ilişki.

Gözlemler. Ortalama süre nispeten sınırlı bir aralıkta değişir, ancak bazı çukurlar/zirveler maçların daha hızlı bittiği (agresif stratejiler) veya tersine uzadığı (daha kontrollü oyuncular) dönemleri önerir. Süre vs öldürmeler dağılım grafiği genel olarak zayıf ila orta bir ilişki gösterir: daha uzun maçlar daha fazla savaş fırsatı sunma eğilimindedir, ancak dağılım diğer faktörlerin öldürme sayısını güclü bir şekilde etkilediğini gösterir (seviye farkı, draft'lar, takım stilleri). Bu grafik bu nedenle mükemmel doğrusal bir ilişkinin olmadığını, daha çok genel bir eğilim olduğunu doğrulamaya yarar.



Şekil 18: Power BI — Süre ve Korelasyonlar Sayfası: ortalama süre eğilimi ve süre vs öldürmeler korelasyonu

4.4 Makine Öğrenmesi Modeli

Maç sonucunu tahmin etmek için bir ikili sınıflandırma modeli geliştirilmiştir. Hedef değişken *win*'dır (1 = Kazanma, 0 = Kaybetme).

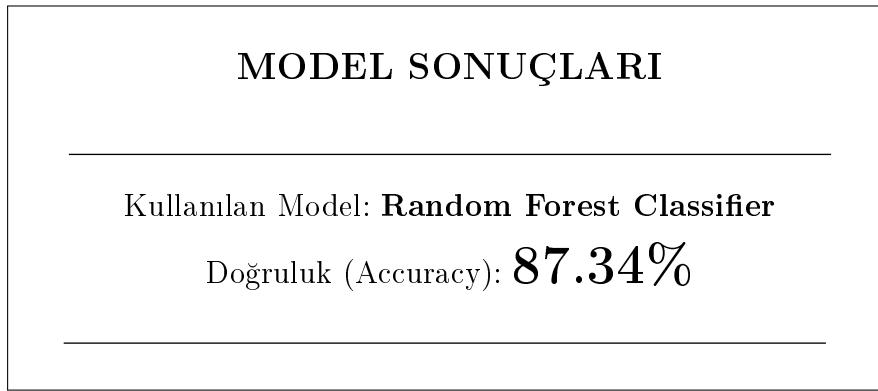
Tablo 9: Kullanılan Özellikler (Features)

Feature	Açıklama	Kategori
kills	Öldürme Sayısı	Savaş
deaths	Ölüm Sayısı	Savaş
assists	Asist Sayısı	Savaş
gold_per_min	Dakika başına altın	Ekonomi
xp_per_min	Dakika başına tecrübe (XP)	Ekonomi
hero_damage	Kahraman hasarı	Hasar
tower_damage	Kule hasarı	Hasar
last_hits	Son vuruş sayısı	İlerleme
level	Oyuncu seviyesi	İlerleme
duration_minutes	Maç süresi	Maç Bilgisi

Random Forest Classifier algoritması şu parametrelerle seçilmiştir: numTrees=50, maxDepth=10, Train/Test Split=80%/20%.

ML pipeline'sının tam kaynak kodu **Ek A.4**'de mevcuttur.

4.5 Model Sonuçları



Tablo 10: Karışıklık Matrisi (Confusion Matrix)

Gerçek / Tahmin	Kaybetme (0)	Kazanma (1)
Kaybetme (0)	727 (GN)	131 (YP)
Kazanma (1)	81 (YN)	735 (GP)

Hesaplanan Metrikler :

- Gerçek Pozitif Oranı (Recall) : $\frac{735}{735+81} = 90.07\%$
- Gerçek Negatif Oranı : $\frac{727}{727+131} = 84.73\%$
- Kesinlik (Precision) : $\frac{735}{735+131} = 84.87\%$

4.6 Özellik Önem Analizi

Tablo 11: Özellik Önem Sıralaması

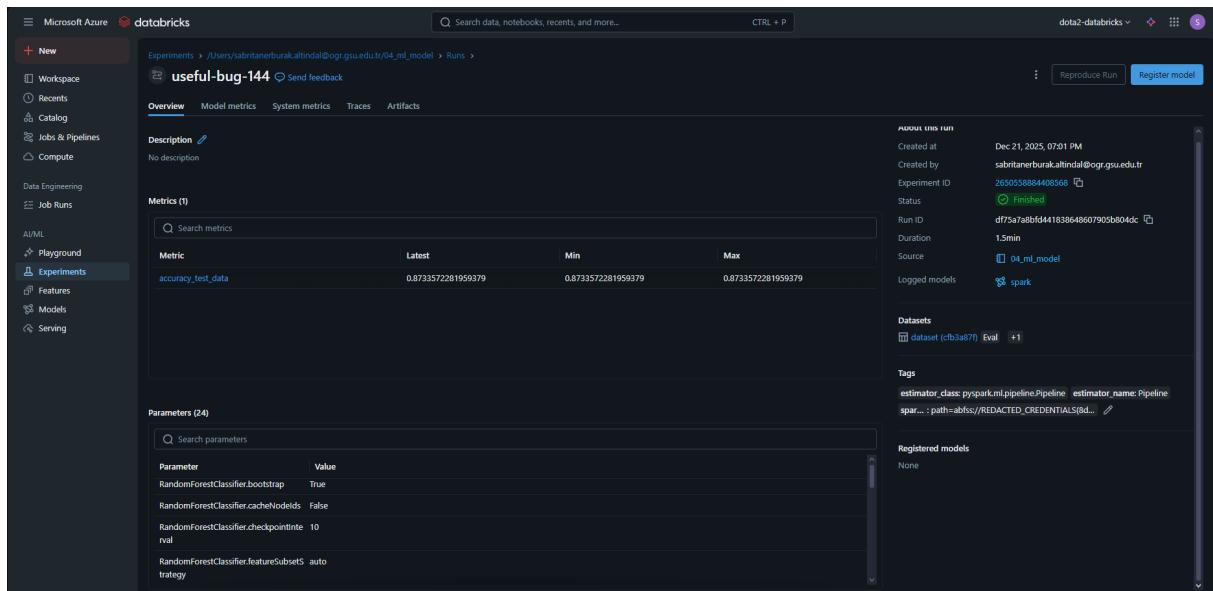
Sıra	Feature	Önem
1	assists	0.311480
2	deaths	0.204537
3	tower_damage	0.147192
4	xp_per_min	0.073274
5	last_hits	0.058863
6	duration_minutes	0.049268
7	gold_per_min	0.048150
8	kills	0.040887
9	hero_damage	0.040081
10	level	0.026267

Bu sonuçlar, **takım çalışmasının** (0.311 ile **assists**) maç sonucu üzerinde en belirleyici faktör olduğunu, bunu **hayatta kalmanın** (0.205 ile **deaths**) ve **hedef hasarının** (0.147 ile **tower_damage**) izlediğini göstermektedir.

Beklentilerin aksine, ekonomik faktörler (**gold_per_min**: 0.048, **xp_per_min**: 0.073) nispeten düşük öneme sahiptir. Bu, profesyonel Dota 2 maçlarında takım koordinasyonunun ve hedef odaklı oyunun, bireysel kaynak birikiminden daha fazla zafer öngördürücü olduğunu göstermektedir.

4.7 MLflow Takibi

Tüm model denemeleri `mlflow.autolog()` ile MLflow üzerinde otomatik olarak izlenmiştir.



Şekil 19: MLflow arayüzünde model sonuçları ve parametreler

5. Karşılaşılan Zorluklar ve Çözümler

5.1 Azure Kaynak Sınırlamaları

Sorun: Azure for Students aboneliği belirli katı kaynak sınırlamaları içermektedir (özellikle vCore kotası ve sınırlı krediler).

Çözüm: Küme boyutu minimumda tutuldu (Standard_DS3_v2), otomatik sonlandırma süresi 120 dakikaya ayarlandı ve küme kredileri biriktirmek için işlemler bittikten sonra manuel olarak durduruldu.

5.2 Streaming Simülasyonunda Dosya Çakışmaları

Sorun: Simülasyon betiğinin hızlı çalıştırılması sırasında, aynı saniye içinde birden fazla mini-batch oluşturulabiliyordu, bu da dosya adı çakışmalarına ve veri üzerine yazmaya neden oluyordu.

Çözüm: Veri alma döngüsüne SLEEP_TIME = 3 saniyelik bir gecikme eklendi. Ayrıca, dosya adları artık YYYYMMDD_HHMMSS formatında kesin bir zaman damgası kullanmaktadır.

5.3 Veri Hacmi ve Maliyet Kısıtlamaları

Sorun: Orijinal kaynak dosyası players.csv 5.3 GB'ı aşıyordu. Bu tam hacmin işlenmesi Azure kredilerini ve mevcut öğrenci kümesinin hesaplama kapasitesini hızla tüketecekti.

Çözüm: Temsili bir players_reduced.csv dosyası (152 MB) oluşturmak için yerel ön işleme yapıldı. Ardından bu azaltılmış veri setinde okuma/yazma performansını optimize etmek için Delta Lake formatı kullandık.

5.4 Azure Storage Anahtar Yönetimi

Sorun: Storage account anahtarının, kodda açık metin olarak gösterilmeden geliştiriciler ve hizmetler arasında paylaşılması gerekiyordu.

Çözüm: Ortam değişkenlerinin kullanımı (AZURE_STORAGE_KEY), Databricks secrets ile güvenli depolama ve .env dosyası ile yerel geliştirme.

5.5 Erişim Yönetimi Karmaşıklığı (IAM)

Sorun: Ekip işbirliği tekrarlayan izin sorunları (IAM) yarattı. "Contributor" rolünün atanması belirli işlemler için (Blob verilerini okuma veya Cost Management'a erişim gibi) yeterli değildi, bu da zaman yönetiminde ve proje ilerlemesinde önemli gecikmelere neden oldu.

Çözüm: Daha ayrıntılı bir RBAC stratejisi benimsendi, uygun üyelere Azure portalı üzerinden açıkça *Storage Blob Data Owner* ve *Cost Management Contributor* rolleri atandı.

5.6 Databricks Cluster Erişim Sorunları

Sorun: *Azure for Students* aboneliğinin sınırlamaları nedeniyle, bir Databricks kümesi oluşturamıyor veya seçemiyorduk, bu da notebook yürütmesini ve dolayısıyla Gold verileri üzerinde SQL analizlerinin başlatılmasını engelliyordu.

Çözüm: Projenin ilerlemesini sağlamak için, gerekli izinlere sahip bir grup üyesi geçici olarak Databricks kümesine (*esports-cluster*) erişim yetkisi verdi. Bu, Gold tablolarında SQL sorguları yürütmemize ve analistik aşamaya devam etmemize olanak sağladı. Erişim bilgileri açık metin olarak gösterilmedi: anahtar ve bağlantı parametreleri Databricks ortam değişkenleri ve *Databricks Secrets* aracılığıyla güvenli bir şekilde yönetildi.

6. Sonuç

6.1 Proje Özeti

Bu proje, modern veri mühendisliği kavramlarının (Lakehouse, Delta Lake, MLOps) pratik bir e-spor senaryosunda nasıl uygulanabileceğini göstermiştir.

Tablo 12: Elde Edilen Temel Sonuçlar

Bileşen	Sonuç
Medallion Mimarisi	Bronze/Silver/Gold katmanları kuruldu
Veri İşleme	29.809 maç, 8.456 oyuncu işlendi
Streaming Simülasyonu	Bronze katmanına 10 mini-batch yazıldı
Gold Tablolari	4 analitik tablo (813 oyuncu, 123 kahraman)
ML Modeli	%87.34 doğruluk oranlı Random Forest
Meta Analizi	5 seviyeli kahraman sınıflandırması

6.2 Öğrenilen Dersler

- **Bulut Kaynak Yönetimi:** Azure maliyet ve kaynak optimizasyonu
- **Delta Lake Avantajları:** ACID işlemleri, şema evrimi, performans
- **Orkestrasyon:** Azure Data Factory ile karmaşık akışları yönetme
- **MLOps Uygulamaları:** MLflow ile deney takibinin önemi

6.3 Genel Sonuç

Azure ekosistemi tarafından sunulan entegre hizmetler, uçtan uca veri çözümünün hızlı ve verimli geliştirilmesini sağladı. **%87.34** doğruluk oranına sahip ML modeli, oyuncu performans metriklerinin maç sonucunu yüksek kesinlikle tahmin edebileceğini kanıtladı. Özellikle, ekonomik faktörler (`gold_per_min`, `xp_per_min`) en belirleyici özellikler olarak tanımlandı.

A. EKLER

A.1 Databricks Notebook Kaynak Kodları

A.1.1 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: Azure ADLS Gen2 Bağlantı Testi

A.1.2 01_bronze_to_silver.ipynb (Extraits)

```

1 from pyspark.sql.functions import *
2 df_matches = spark.read.csv(f"{{BRONZE}}/main_metadata.csv",
3     header=True, inferSchema=True)
4 df_players = spark.read.csv(f"{{BRONZE}}/players_reduced.csv",
5     header=True, inferSchema=True)
6 df_matches_clean = df_matches \
7     .drop("Unnamed: 0").dropDuplicates(["match_id"]) \
8     .filter(col("match_id").isNotNull()) \
9     .filter((col("duration") > 300) & (col("duration") < 7200))
10 df_matches_clean = df_matches_clean \
11     .withColumn("duration_minutes", round(col("duration")/60, 2)) \
12     .withColumn("radianc_win", col("radianc_win").cast("boolean")) \
13     .withColumn("match_date", to_date(col("start_date_time")))
14 stats = df_players.select(
15     mean("kills").alias("m"), stddev("kills").alias("s")).collect()[0]
16 df_players = df_players.withColumn("is_outlier",
17     when(col("kills") > stats["m"] + 3*stats["s"], True).otherwise(False))
18 df_matches_clean.write.format("delta").mode("overwrite") \
19     .save(f"{{SILVER}}/cleaned_matches")

```

Code 2: Veri Yükleme ve Temizleme

A.1.3 02_silver_to_gold.ipynb (Extraits)

```

1 df_player_stats = df_players \
2     .filter(col("account_id").isNotNull()) \
3     .groupBy("account_id").agg(
4         count("match_id").alias("total_matches"),
5         sum("win").alias("total_wins"),
6         round(avg("kills"), 2).alias("avg_kills"),
7         round(avg("kda_calculated"), 2).alias("avg_kda"),
8         round(avg("gold_per_min"), 2).alias("avg_gpm")) \
9     .withColumn("win_rate",
10         round(col("total_wins")/col("total_matches")*100, 2))
11 total = df_matches.count()

```

```
12 | df_hero = df_hero \  
13 |     .withColumn("presence_rate",  
14 |         round(col("total_presence")/(total*2)*100, 2)) \  
15 |     .withColumn("meta_tier",  
16 |         when(col("presence_rate") > 50, "S-Tier")  
17 |             .when(col("presence_rate") > 30, "A-Tier")  
18 |                 .when(col("presence_rate") > 15, "B-Tier")  
19 |                     .when(col("presence_rate") > 5, "C-Tier")  
20 |                         .otherwise("D-Tier"))
```

Code 3: Gold Tablolarının Oluşturulması

A.2 Streaming Simülatörü Kaynak Kodu

```

1 import time, pandas as pd, os
2 from datetime import datetime
3 from azure.storage.filedatalake import DataLakeServiceClient
4
5 STORAGE_ACCOUNT = "data2lakehousenew"
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 - Streaming Simülatörü

A.3 Tablo Oluşturma SQL Komutları

```
1 CREATE DATABASE IF NOT EXISTS esports_gold;
2 CREATE TABLE IF NOT EXISTS esports_gold.player_stats
3 USING DELTA LOCATION
4   'abfss://data@dota2lakehousenew.dfs.core.windows.net/gold/player_stats';
5 CREATE TABLE IF NOT EXISTS esports_gold.hero_stats
6 USING DELTA LOCATION
7   'abfss://data@dota2lakehousenew.dfs.core.windows.net/gold/hero_stats';
8 CREATE TABLE IF NOT EXISTS esports_gold.daily_stats
9 USING DELTA LOCATION
10  'abfss://data@dota2lakehousenew.dfs.core.windows.net/gold/daily_stats';
11 CREATE TABLE IF NOT EXISTS esports_gold.ml_features
12 USING DELTA LOCATION
13   'abfss://data@dota2lakehousenew.dfs.core.windows.net/gold/ml_features';
14 SHOW TABLES IN esports_gold;
```

Code 5: Şema ve Gold Tablolarının Oluşturulması

A.4 ML Model Kaynak Kodu

```

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 path = f"abfss://{{container}}@{{storage_account}}.dfs.core.windows.net/gold/ml_features"
7 df = spark.read.format("delta").load(path)
8 cols = ["kills", "deaths", "assists", "gold_per_min", "xp_per_min",
9         "hero_damage", "tower_damage", "last_hits", "level",
10        "duration_minutes", "win"]
11 data = df.select(cols).dropna()
12 train, test = data.randomSplit([0.8, 0.2], seed=42)
13 features = [c for c in cols if c != "win"]
14 assembler = VectorAssembler(inputCols=features, outputCol="features")
15 rf = RandomForestClassifier(
16     labelCol="win", featuresCol="features", numTrees=50, maxDepth=10)
17 pipeline = Pipeline(stages=[assembler, rf])
18 mlflow.autolog()
19 model = pipeline.fit(train)
20 predictions = model.transform(test)
21 evaluator = MulticlassClassificationEvaluator(
22     labelCol="win", predictionCol="prediction", metricName="accuracy")
23 acc = evaluator.evaluate(predictions)
24 print(f"Accuracy: {acc*100:.2f}%")
25
26 predictions.groupBy("win", "prediction").count().show()
27 import pandas as pd
28 imp = model.stages[-1].featureImportances.toArray()
29 df_imp = pd.DataFrame({"Feature": features, "Importance": imp})
30 print(df_imp.sort_values("Importance", ascending=False))

```

Code 6: Tam Makine Öğrenmesi Pipeline’ı

A.5 Kullanılan Teknolojiler

Tablo 13: Proje Teknoloji Yığını

Kategori	Teknoloji	Sürüm
Bulut Platformu	Microsoft Azure	-
Storage Account	dota2lakehousenew	ADLS Gen2
Orkestrasyon	Azure Data Factory	V2
İşlem	Azure Databricks	13.3 LTS
İşleme	Apache Spark	3.4.1
Dil	Python / PySpark	3.10
Veri Formatı	Delta Lake	2.4.0
ML Takibi	MLflow	Otomatik
Görselleştirme	Power BI	-