

# RELATÓRIO

**Data Science & Business Analytics – 13.ª Edição**  
Data Lake of Unstructured Data



André Garcia (31811) | Dário Sequeira (31772) | Mariana António (31635) | Rodrigo Relvas (31783)

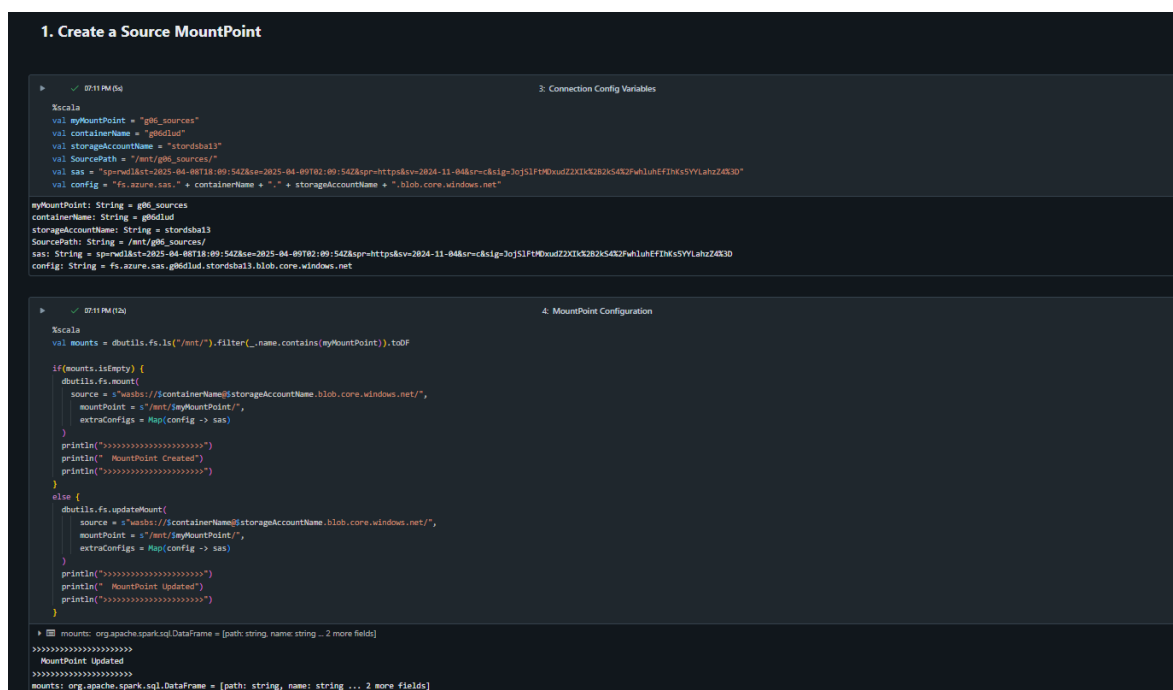
O presente relatório visa documentar as principais etapas do trabalho final de DLUD, desenvolvido em *Azure Databricks*. Este trabalho teve como principal objetivo a importação, transformação e exploração de dados para posterior análise e visualização através de um *dashboard* criado na própria plataforma.

Links:

- [Workspace do grupo G06](#)
- [Notebook](#)
- [Dashboard](#)

Descrevem-se, em seguida, os principais passos do trabalho, acompanhados dos respectivos *screenshots* do código:

1. Começamos por criar um MountPoint (g06\_sources) para estabelecer a ligação entre o Databricks e o *Storage Container* no *Azure Data Lake Storage* (ADLS). Definimos os parâmetros base e validamos que o MountPoint estava operacional;



The screenshot displays a Databricks notebook with two code cells. The first cell, titled '1. Create a Source MountPoint', contains Scala code that defines variables for the mount point name, container name, storage account name, source path, SAS token, and configuration. The second cell, titled '4. MountPoint Configuration', contains Scala code that checks if the mount point exists and either creates it or updates its configuration. The output of the second cell shows the mount point being updated.

```
1. Create a Source MountPoint

Scala
val myMountPoint = "g06_sources"
val containerName = "g06dlud"
val storageAccountName = "stordba13"
val SourcePath = "/mnt/g06_sources/"
val sas = "sp=we11st-2025-04-08T18:09:54Z&se=2025-04-09T02:09:54Z&spr=https&sv=2024-11-04&sr=ck&sig=3cj51F0Dnd22X1k32R2K54K2PwLuhEFFDKs5YPLahz24E3D"
val config = "fs.azure.sas." + containerName + "." + storageAccountName + ".blob.core.windows.net"

myMountPoint: String = g06_sources
containerName: String = g06dlud
storageAccountName: String = stordba13
SourcePath: String = /mnt/g06_sources/
sas: String = sp=we11st-2025-04-08T18:09:54Z&se=2025-04-09T02:09:54Z&spr=https&sv=2024-11-04&sr=ck&sig=3cj51F0Dnd22X1k32R2K54K2PwLuhEFFDKs5YPLahz24E3D
config: String = fs.azure.sas.g06dlud.stordba13.blob.core.windows.net

4. MountPoint Configuration

Scala
val mounts = dbutils.fs.ls("/mnt/").filter(_.name.contains(myMountPoint)).toDF

if(mounts.isEmpty) {
  dbutils.fs.mount(
    source = s"wasbs://$containerName@$storageAccountName.blob.core.windows.net/",
    mountPoint = s"/mnt/$myMountPoint/",
    extraConfigs = Map(config -> sas)
  )
  println("MountPoint Created")
} else {
  dbutils.fs.updateMount(
    source = s"wasbs://$containerName@$storageAccountName.blob.core.windows.net/",
    mountPoint = s"/mnt/$myMountPoint/",
    extraConfigs = Map(config -> sas)
  )
  println("MountPoint Updated")
}

mounts: org.apache.spark.sql.DataFrame = [path: string, name: string ... 2 more fields]
MountPoint Updated
mounts: org.apache.spark.sql.DataFrame = [path: string, name: string ... 2 more fields]
```

2. De seguida, lemos o ficheiro base do trabalho ('DLUD\_imdbTop250\_v2.csv'), previamente carregado no *Data Lake Storage Container* ('g06dlud');

## Data Lake of Unstructured Data

### 2. Read Data from Azure Blob Storage

```
// Leitura do ficheiro CSV do Azure Blob Storage através do mountpoint criado (variável SourcePath).

val imdbSrc = spark.read
  .option("header", "true")
  .option("inferSchema", "true")
  .csv(sourcePath + "/imdbtop250_v2.csv")
display(imdbSrc.limit(5))
```

Table

	Ranking	IMDByear	IMDBtitle	Title	Date	RunTime	Genre	1.2 Rating	1.2 Score	1.2 Votes	1.2 Gross	Director	Cast1	Cast2	Cast3
1	1	1996	/96%nd0707020/	Star Wars: Episode IV - A New Hope	1977	121	Action, Adventure, Fanta...	8.8	90	1297131	322.74	George Lucas	Mark Hamill	Harrison Ford	Carrie F
2	2	1996	/96%nd0111160/	The Shawshank Redemption	1994	142	Drama	9.3	80	2329413	28.34	Frank Darabont	Tim Robbins	Morgan Freeman	Bob Gu
3	3	1996	/96%nd0117051/	The Godfather Part II	1973	175	Drama	9.2	83	1800000	19.13	Francis Ford Coppola	Al Pacino	Al Pacino	Diane Ke
4	4	1996	/96%nd0114014/	The Usual Suspects	1995	106	Crime, Drama, Mystery	8.5	77	1045408	23.34	Bryan Singer	Kevin Spacey	Gabriel Byrne	Chazz P

5 rows | 1.63s runtime

Refreshed 47 minutes ago

3. Criámos as 9 tabelas do modelo de dados (4 *DimensionsTables*, 2 *FactTables* e 3 *BridgeTables*);

### 2.1. Get DimDirector

```
// Importa as funções da biblioteca Spark SQL, como split, trim, explode, etc., para a manipulação e transformação de dados.
import org.apache.spark.sql.functions._

val dfDirectorStg = imdbSrc.select("Director")

// Cria uma nova coluna "DirectorList" a partir da separação dos nomes dos diretores por vírgulas.

val dfDirectorStg1 = dfDirectorStg
  .withColumn("DirectorList", split(dfDirectorStg("Director"), ","))
  .select("DirectorList")

// Expande a lista de diretores em várias linhas, criando uma nova coluna "Director" através do Dataframe anterior.

val dfDirectorStg2 = dfDirectorStg1
  .withColumn("Director", explode(dfDirectorStg1("DirectorList")))
  .select("Director")

val dfDirectorStg3 = dfDirectorStg2
  .select(trim(dfDirectorStg2("Director")).alias("Director")) // Remove os espaços em branco e renomeia a coluna.
  .dropDuplicates() // Elimina os diretores duplicados.
  .withColumn("DirectorID", monotonically_increasing_id().plus(1)) // Adiciona um ID único a cada diretor.
  .select("DirectorID", "Director")

val DimDirector = dfDirectorStg3
display(DimDirector.limit(5))
```

Table

	DirectorID	Director
1	1	Tony Chong
2	2	James
3	3	David
4	4	King
5	5	Michael

5 rows | 0.49s runtime

Refreshed 1 hour ago

### 2.2. Get DimCast

```
// Seleciona cada uma das colunas do Cast a cada variável, com a remoção dos espaços em branco e linha ou Null.

val cast1 = imdbSrc.select("Cast1").withColumn("Cast", trim($"Cast1")).filter($"Cast" != null)
val cast2 = imdbSrc.select("Cast2").withColumn("Cast", trim($"Cast2")).filter($"Cast" != null)
val cast3 = imdbSrc.select("Cast3").withColumn("Cast", trim($"Cast3")).filter($"Cast" != null)
val cast4 = imdbSrc.select("Cast4").withColumn("Cast", trim($"Cast4")).filter($"Cast" != null)

// Faz o merge das variáveis do Cast em um único Dataframe.

val dfCastStg = cast1
  .union(cast2)
  .union(cast3)
  .union(cast4)

// Cria o Dataframe final, removendo os duplicados e adiciona um ID único a cada Cast.

val DimCast = dfCastStg
  .dropDuplicates()
  .withColumn("CastID", monotonically_increasing_id().plus(1))
  .select("CastID", "Cast")

display(DimCast.limit(5))
```

Table

	CastID	Cast
1	1	Diana
2	2	Patricia
3	3	James
4	4	Raphael
5	5	Paul

5 rows | 0.63s runtime

Refreshed 1 hour ago

## Data Lake of Unstructured Data

2.3. GetDimGenre

```
Scala
// Processo semelhante ao DimDirector.

val dfGenreStg = indSrc.select("Genre")

val dfGenreStg1 = dfGenreStg
  .withColumn("GenreList", split(dfGenreStg("Genre"), ","))
  .select("GenreList")

val dfGenreStg2 = dfGenreStg1
  .withColumn("Genre", explode(dfGenreStg1("GenreList")))
  .select("Genre")

val dfGenreStg3 = dfGenreStg2
  .select(trim(dfGenreStg2("Genre")).alias("Genre"))
  .dropDuplicates()
  .withColumn("GenreID", monotonically_increasing_id_pla(1))
  .select("GenreID", "Genre")

val DimGenre = dfGenreStg3
display(DimGenre.limit(5))
```

3 Spark Jobs

- dfGenreStg: org.apache.spark.sql.DataFrame = [Genre: string]
- dfGenreStg1: org.apache.spark.sql.DataFrame = [GenreList: array]
- dfGenreStg2: org.apache.spark.sql.DataFrame = [Genre: string]
- dfGenreStg3: org.apache.spark.sql.DataFrame = [GenreID: long, Genre: string]
- DimGenre: org.apache.spark.sql.DataFrame = [GenreID: long, Genre: string]

GenreID	Genre
1	Cine
2	Romance
3	Thriller
4	Adventure
5	Drama

5 rows | 0.47s runtime

Refreshed 1 hour ago

2.4. Get DimFilm

```
Scala
val DimFilm = indSrc
  .select("Title", "TMDBLink", "Year", "Runtime").dropDuplicates()
  .withColumn("FilmID", monotonically_increasing_id_pla(1))
  .select("FilmID", "Title", "TMDBLink", "Year", "Runtime")

display(DimFilm.limit(5))
```

3 Spark Jobs

- DimFilm: org.apache.spark.sql.DataFrame = [FilmID: long, Title: string, ... 3 more fields]

FilmID	Title	TMDBLink	Year	Runtime
1	The Double Life of Veronique	AtMuN0101763V/	1991	98
2	Dead Men	AtMuN0112817V/	1995	121
3	Un Chien Andalou	AtMuN0100030V/	1929	16
4	Chicken Run	AtMuN0120630V/	2000	84
5	A Separation	AtMuN1832382V/	2011	123

5 rows | 0.40s runtime

Refreshed 1 hour ago

2.5. Get BridgeDirector

```
Scala
// Criação do dataframe com as colunas "TMDBLink" e "Director" para apoiar a criação da bridge entre DimDirector e DimFilm.

val stgDirC = indSrc
  .select("Director", "TMDBLink")
  .withColumn("DirectorList", split(indSrc("Director"), ","))

val stgDirC1 = stgDirC
  .withColumn("Director", explode(stgDirC("DirectorList")))
  .select("TMDBLink", "Director")

val stgDirC1 = stgDirC1
  .select(stgDirC1("TMDBLink"), trim(stgDirC1("Director")).alias("Director"))
  .dropDuplicates()
  .orderBy("TMDBLink", "Director")

// Referência das tabelas de dimensões para diretores e filmes.

val dimDirector = DimDirector
val dimFilm = DimFilm

// Criação da bridge entre Directors e films através de inner joins das tabelas.

val BridgeDirector = stgSource
  .join(dimDirector, stgSource("Director") === dimDirector("Director"), "inner")
  .join(dimFilm, stgSource("TMDBLink") === dimFilm("TMDBLink"), "inner")
  .select("DirectorID", "FilmID")

display(BridgeDirector.limit(5))
```

4 Spark Jobs

- BridgeDirector: org.apache.spark.sql.DataFrame = [DirectorID: long, FilmID: long]
- dimDirector: org.apache.spark.sql.DataFrame = [DirectorID: long, Director: string]
- dimFilm: org.apache.spark.sql.DataFrame = [FilmID: long, Title: string, ... 3 more fields]
- stgSource: org.apache.spark.sql.DataFrame = [Director: string, TMDBLink: string, Director: string]
- stgDirC: org.apache.spark.sql.DataFrame = [Director: string, TMDBLink: string, ... 1 more field]
- stgDirC1: org.apache.spark.sql.DataFrame = [TMDBLink: string, Director: string]

DirectorID	FilmID
357	64
262	223
389	533
437	266
185	36

5 rows | 0.71s runtime

Refreshed 1 hour ago

## Data Lake of Unstructured Data

2.6. Get BridgeCast

```
Scala

// Criação do dataframe para apoiar a criação da bridge

val stgCast1SRC = IndbSrc.select("Cast1","DDBLink").withColumn("Cast", trim($"Cast1")).filter($"Cast".isNotNull)
val stgCast2SRC = IndbSrc.select("Cast1","DDBLink").withColumn("Cast", trim($"Cast1")).filter($"Cast".isNotNull)
val stgCast3SRC = IndbSrc.select("Cast1","DDBLink").withColumn("Cast", trim($"Cast1")).filter($"Cast".isNotNull)
val stgCast4SRC = IndbSrc.select("Cast4","DDBLink").withColumn("Cast", trim($"Cast4")).filter($"Cast".isNotNull)

val stgSRC1 = stgCast1SRC
               .union(stgCast2SRC)
               .union(stgCast3SRC)
               .union(stgCast4SRC)

val stgSource = stgSRC1
               .select(stgSRC1("DDBLink"),trim(stgSRC1("Cast")).alias("Cast"))
               .dropDuplicates()
               .orderBy("DDBLink","Cast")

val diaCast = DiaCast
val diaFile = DiaFile

// Criação da bridge entre o Cast e File através de inner join das tabelas.

val BridgeCast = stgSource
               .join(diaCast, stgSource("Cast") === diaCast("Cast"), "inner")
               .join(diaFile, stgSource("DDBLink") === diaFile("DDBLink"), "inner")
               .select("CastID","FileID")

display(BridgeCast.limit(5))
```

19: BridgeCast

Table

CastID	FileID
1	381
2	297
3	448
4	175
5	454

5 rows | 1.81s runtime

Refreshed 1 hour ago

2.7. Get BridgeGerre

```
Scala

val stgSRC = IndbSrc
               .select("Gerre","DDBLink")
               .withColumn("GerreList",split(indbSrc("Gerre"), ","))

val stgSRC1 = stgSRC
               .withColumn("Gerre",explode(stgSRC("GerreList")))
               .select("DDBLink","Gerre")

val stgSource = stgSRC1
               .select(stgSRC1("DDBLink"),trim(stgSRC1("Gerre")).alias("Gerre"))
               .dropDuplicates()
               .orderBy("DDBLink","Gerre")

val diaGerre = DiaGerre
val diaFile = DiaFile

val BridgeGerre = stgSource
               .join(diaGerre, stgSource("Gerre") === diaGerre("Gerre"), "inner")
               .join(diaFile, stgSource("DDBLink") === diaFile("DDBLink"), "inner")
               .select("GerreID","FileID")

display(BridgeGerre.limit(5))
```

21: BridgeGerre

Table

GerreID	FileID
1	426
2	68
3	10
4	130
5	672

5 rows | 0.55s runtime

Refreshed 1 hour ago

2.8. Get FactFilmsManagement

```
Scala

val stgSource = IndbSrc
               .select("DDBLink","Extractionyear","Rating","Score","Votes","Gross")

val diaFile = DiaFile

val FactFilmsManagement = stgSource
               .dropDuplicates()
               .join(diaFile, stgSource("DDBLink") === diaFile("DDBLink"), "inner")
               .select("Extractionyear","FileID","Rating","Score","Votes","Gross")

display(FactFilmsManagement.limit(5))
```

23: FactFilmsManagement

Table

Extractionyear	FileID	Rating	Score	Votes	Gross
2022	160	8.1	85	654107	24.61
2022	216	7.7	1000	456862	63.54
2022	126	8.7	87	1577278	342.55
2022	576	8.3	93	431483	40.06
2022	177	8.3	75	306687	5.32

5 rows | 0.45s runtime

Refreshed 1 hour ago

## Data Lake of Unstructured Data

```

2.9. Get FactFimsRanking

scala> val stgSource = imdbSrc
      |   .select("IMDBYear", "Ranking", "IMDBLink")
      |
      | val distFile = distFile
      |
      | val FactFimsRanking = stgSource
      |   .join(distFile, stgSource("IMDBLink") === distFile("IMDBLink"), "inner")
      |   .select("IMDBYear", "FilmID", "Ranking")
      |
      | display(FactFimsRanking.limit(5))

+ (3) Spark Jobs

+ [x] distFile: org.apache.spark.sql.DataFrame = [FilmID: long, Title: string ... 3 more fields]
+ [x] FactFimsRanking: org.apache.spark.sql.DataFrame = [IMDBYear: integer, FilmID: long ... 1 more field]
+ [x] stgSource: org.apache.spark.sql.DataFrame = [IMDBYear: integer, Ranking: integer ... 1 more field]

Table =
+
+-----+-----+-----+
+ | IMDBYear | FilmID | Ranking |
+ |-----+-----+-----+
+ | 1996     | 734    | 1       |
+ | 1996     | 495    | 2       |
+ | 1996     | 256    | 3       |
+ | 1996     | 222    | 4       |
+ | 1996     | 482    | 5       |
+ |-----+-----+-----+

5 rows | 0.50s runtime

```

4. Criámos um MountPoint de destino para o *Storage Container*, destinado a guardar os dados processados anteriormente;

```
3. Create a Destination MountPoint

scala
// Configuracão do mount point para gravar o Output da Informacão para o Azure Blob Storage.

val myMountPointM = "gbl_Destination"
val containerNameM = "gblodid"
val storageAccountNameM = "stordbhl1"
val sasM = "sprraueuocpda2025-04-07T15:40:51Zsk=2025-04-07T23:40:51Zskpr=https&sv=2024-11-04&sr=cs&sig=1D8yGp8pMh1EP9K12F5VulQ2H8W8GIncIrfW52m4U3"
val configM = "fs.azure.sas:" + containerNameM + "/" + storageAccountNameM + ".blob.core.windows.net"
val destinationPath = "/mnt/sayMountPointM/Processed/"

myMountPointM: String = gbl_Destination
containerNameM: String = gblodid
storageAccountNameM: String = stordbhl1
sasM: String = sprraueuocpda2025-04-07T15:40:51Zsk=2025-04-07T23:40:51Zskpr=https&sv=2024-11-04&sr=cs&sig=1D8yGp8pMh1EP9K12F5VulQ2H8W8GIncIrfW52m4U3
configM: String = fs.azure.sas:gblodid,stordbhl1.blob.core.windows.net
destinationPath: String = /mnt/gbl_Destination/Processed/

scala
// Criacao do mount point de "Read-Write".

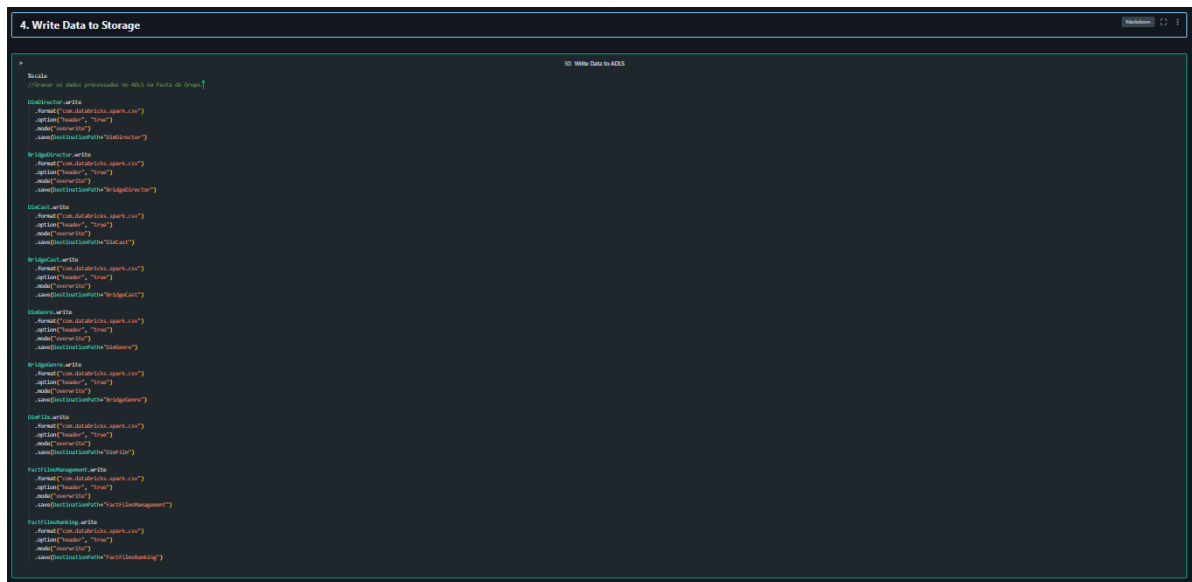
val mounts = dHutil.fs.ls("/mnt/").filter(_._name.contains(myMountPointM)).toDF

if(mounts.isEmpty) {
  dHutil.fs.mount(
    source = s"wasbs://${containerNameM}/${storageAccountNameM}.blob.core.windows.net/",
    mountPoint = s"/mnt/sayMountPointM/",
    extraConfigs = Map(configM -> sasM)
  )
  println("*****")
  println(" MountPoint Created")
  println("*****")
} else {
  dHutil.fs.updateMount(
    source = s"wasbs://${containerNameM}/${storageAccountNameM}.blob.core.windows.net/",
    mountPoint = s"/mnt/sayMountPointM/",
    extraConfigs = Map(configM -> sasM)
  )
  println("*****")
  println(" MountPoint Updated")
  println("*****")
}

mounts: org.apache.spark.sql.DataFrame = [path: string, name: string ... 2 more fields]
*****
MountPoint Updated
*****
mounts: org.apache.spark.sql.DataFrame = [path: string, name: string ... 2 more fields]
```

5. Gravámos os dados processados na pasta 'Processed' no ADLS através do MountPoint criado no passo anterior;

## Data Lake of Unstructured Data



```
4. Write Data to Storage

// Escrever os dados processados no ADS na Pasta de Green

// Escrever em Parquet
val dfParquet = df.write.parquet("parquet")

// Escrever em Avro
val dfAvro = df.write.avro("avro")

// Escrever em JSON
val dfJson = df.write.json("json")

// Escrever em CSV
val dfCsv = df.write.csv("csv")

// Escrever em Delta
val dfDelta = df.write.format("delta").saveAsTable("delta")

// Escrever em Parquet com compressão
val dfParquetCompressed = df.write.parquet("parquet_compressed", compression="snappy")

// Escrever em Avro com compressão
val dfAvroCompressed = df.write.avro("avro_compressed", compression="snappy")

// Escrever em JSON com compressão
val dfJsonCompressed = df.write.json("json_compressed", compression="snappy")

// Escrever em CSV com compressão
val dfCsvCompressed = df.write.csv("csv_compressed", compression="snappy")

// Escrever em Delta com compressão
val dfDeltaCompressed = df.write.format("delta").saveAsTable("delta_compressed")

// Escrever em Parquet com compressão e partição
val dfParquetCompressedPartitioned = df.write.parquet("parquet_compressed_partitioned", compression="snappy", partitionBy("date"))

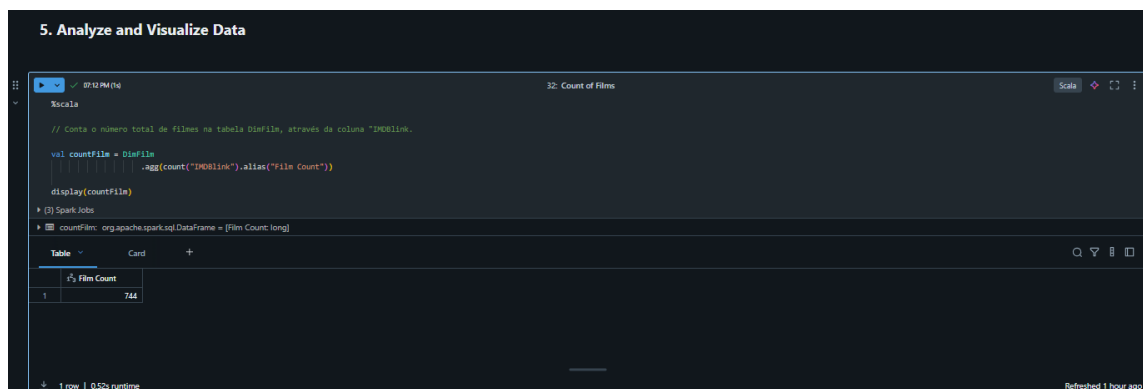
// Escrever em Avro com compressão e partição
val dfAvroCompressedPartitioned = df.write.avro("avro_compressed_partitioned", compression="snappy", partitionBy("date"))

// Escrever em JSON com compressão e partição
val dfJsonCompressedPartitioned = df.write.json("json_compressed_partitioned", compression="snappy", partitionBy("date"))

// Escrever em CSV com compressão e partição
val dfCsvCompressedPartitioned = df.write.csv("csv_compressed_partitioned", compression="snappy", partitionBy("date"))

// Escrever em Delta com compressão e partição
val dfDeltaCompressedPartitioned = df.write.format("delta").saveAsTable("delta_compressed_partitioned")
```

6. Realizamos diversas análises para explorar os dados, complementando com visualizações para facilitar a compreensão dos resultados e destacar *insights* relevantes;

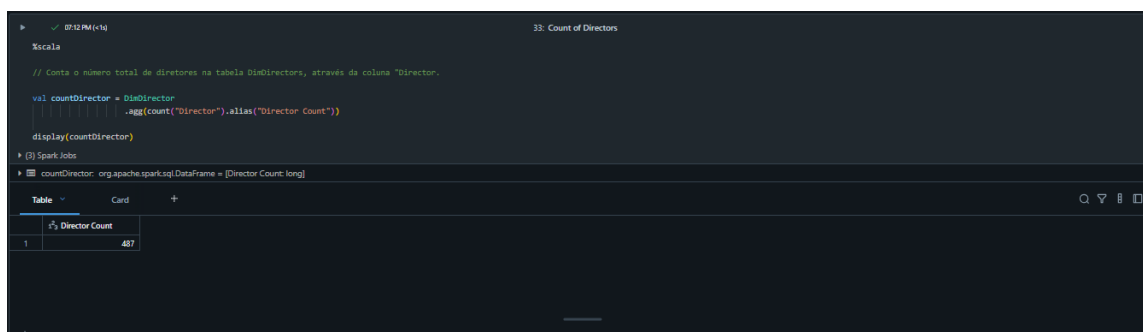


```
5. Analyze and Visualize Data

// Conta o número total de filmes na tabela DisFile, através da coluna "IMDblink".
val countFilm = DisFile
  .agg(count("IMDblink").alias("Film Count"))
  .display(countFilm)

// (3) Spark Jobs
countFilm: org.apache.spark.sql.DataFrame = [Film Count: long]

Table | Card | +
-----|-----|+
1 | 744 |
1 row | 0.52s runtime
```



```
5. Analyze and Visualize Data

// Conta o número total de diretores na tabela DisDirectors, através da coluna "Director".
val countDirector = DisDirectors
  .agg(count("Director").alias("Director Count"))
  .display(countDirector)

// (3) Spark Jobs
countDirector: org.apache.spark.sql.DataFrame = [Director Count: long]

Table | Card | +
-----|-----|+
1 | 487 |
1 row | 0.45s runtime
```

## Data Lake of Unstructured Data

Scala

```
// Conta o número total de elencos na tabela DimCast, através da coluna "Cast".  
val countCast = DimCast  
  .agg(count("Cast").alias("Cast Count"))  
display(countCast)
```

(3) Spark Jobs

countCast: org.apache.spark.sql.DataFrame = [Cast Count: long]

Cast Count
2000

1 row | 0.73s runtime

Refreshed 4 minutes ago

Scala

```
// Conta o número total de gênero na tabela DimGenre, através da coluna "Genre".  
val countGenre = DimGenre  
  .agg(count("Genre").alias("Genre Count"))  
display(countGenre)
```

(3) Spark Jobs

countGenre: org.apache.spark.sql.DataFrame = [Genre Count: long]

Genre Count
23

1 row | 0.41s runtime

Refreshed 1 hour ago

Scala

```
// Realiza uma junção entre DimGenre e BridgeGenre com base no GenreID e, em seguida, conta o número de filmes por gênero, ordenando o resultado em ordem decrescente.  
val aux = DimGenre.join(BridgeGenre, BridgeGenre("GenreID") === DimGenre("GenreID"), "inner").groupBy("Genre").agg(count(lit("1")).alias("NumberOffilms")).orderBy(col("NumberOffilms").desc)  
display(aux)
```

(7) Spark Jobs

aux: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Genre: string, NumberOffilms: long]

Genre	NumberOffilms
Drama	531
Comedy	208
Crime	150
Adventure	136
Action	127
Romance	101
Thriller	94
Biography	81
Mystery	74
Sci-Fi	57
Animation	49
Fantasy	47
History	39
Family	36
War	34

23 rows | 0.72s runtime

Refreshed 16 minutes ago

Scala

```
// Seleciona os 10 filmes mais votados, unindo as tabelas "FactFilmsManagement" e "dimfilm" através de "filmID",  
// selecionando as colunas "Title", "Date", "Rating", "Score" e "Votes", e ordenando por "Rating" em ordem decrescente  
val mostVotedFilms = FactFilmsManagement  
  .join(dimfilm, "filmID")  
  .select("Title", "Date", "Rating", "Score", "Votes")  
  .orderBy(desc("Rating"))  
  .limit(10)  
display(mostVotedFilms)
```

(5) Spark Jobs

mostVotedFilms: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Title: string, Date: integer ... 3 more fields]

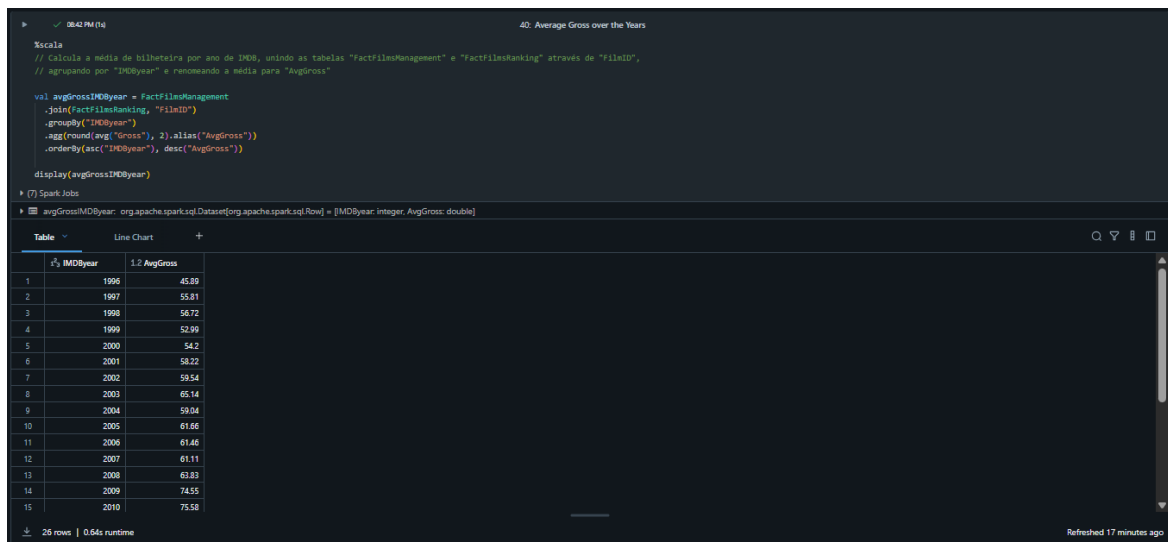
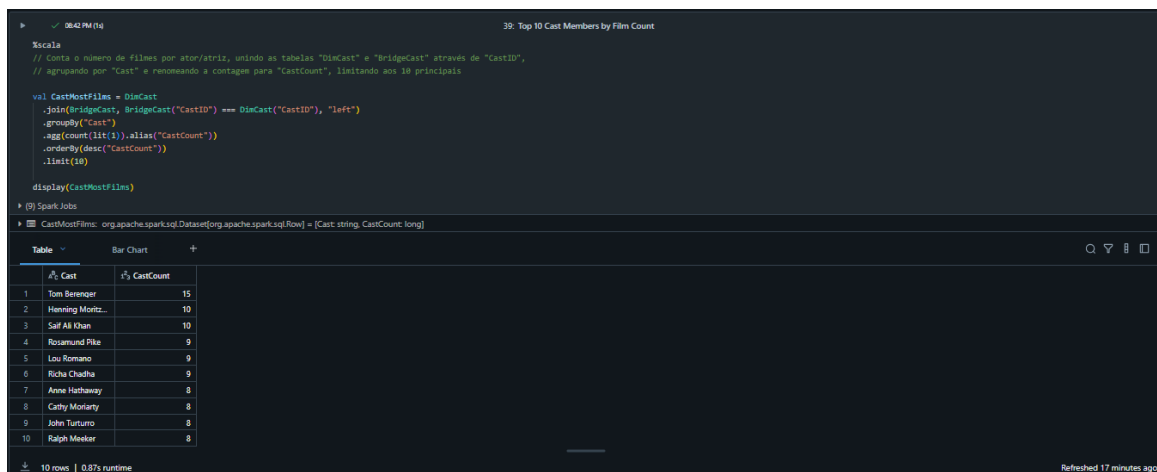
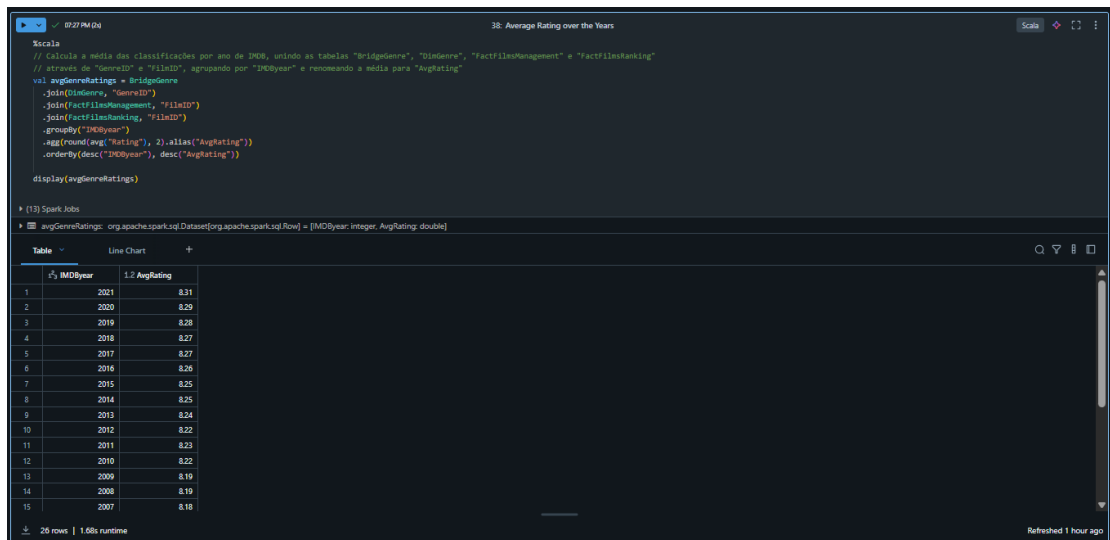
Title	Date	Rating	Score	Votes
The Shawshank Redemption	1994	9.3	80	2539673
Jai Bhim	2021	9.3	80	169009
The Godfather	1972	9.2	100	1741574
The Dark Knight	2008	9	84	2409130
The Godfather: Part II	1974	9	90	1208326
12 Angry Men	1957	9	96	747260
Pulp Fiction	1994	8.9	94	1948662
Schindler's List	1993	8.9	94	1202510
The Lord of the Rings: The Return of the Ki...	2003	8.9	94	1745769
Forrest Gump	1994	8.8	82	1952353

10 rows | 0.60s runtime

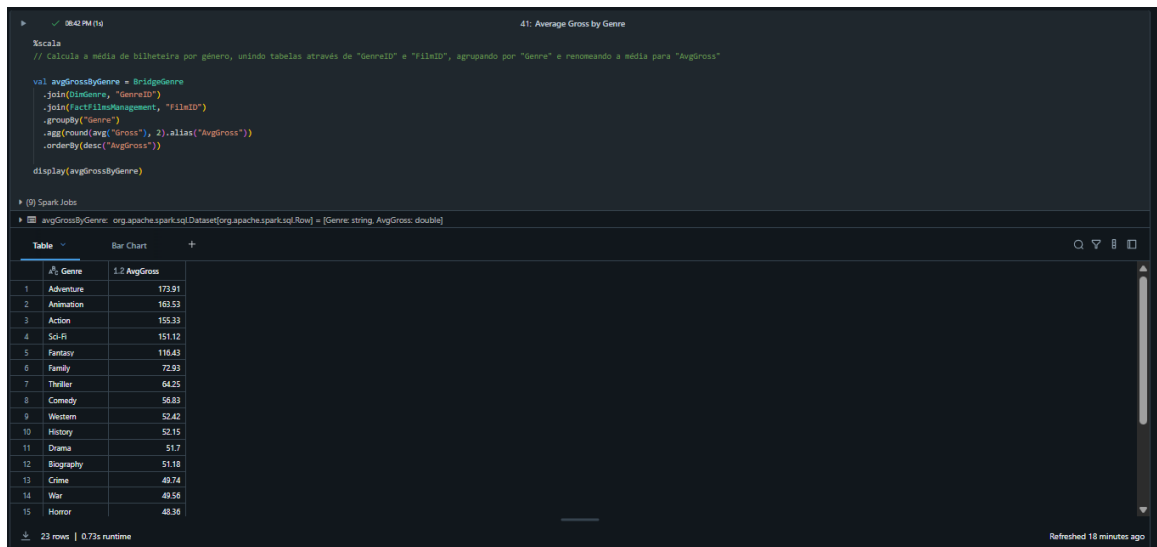
Refreshed 1 hour ago



# Data Lake of Unstructured Data



## Data Lake of Unstructured Data



7. Por fim, incluímos as análises que considerámos mais relevantes no *Dashboard* ('G06\_DLUD13\_Dashboard'), utilizando gráficos e tabelas para apresentar os principais resultados de forma clara e intuitiva.

