

MVP – Engenharia de Dados

Marcos Fernando de Moraes Siliprandi

Descrição

Este trabalho tem como objetivo a construção de um pipeline de dados utilizando tecnologias na nuvem, com foco na plataforma Databricks Community Edition. O pipeline abrangerá todo o ciclo de vida dos dados, incluindo busca, coleta, modelagem, carga e análise. A base de dados utilizada contém informações sobre partidas do Campeonato Brasileiro, estatísticas dos clubes, gols e cartões, permitindo a extração de insights relevantes sobre o desempenho dos times e jogadores ao longo da competição.

Objetivo

O objetivo deste MVP é estruturar e transformar dados brutos do Campeonato Brasileiro em um formato otimizado para análise, possibilitando a extração de insights que possam responder a perguntas como:

- Quais clubes mais finalizaram ao longo do campeonato?
- Quais clubes mais receberam cartões amarelos e vermelhos no campeonato?
- Quais jogadores mais marcaram gols?
- Quais intervalos de tempo mais ocorrem gols?
- Como o fator "mandante" influencia nos resultados das partidas?

A partir dessa estruturação, será possível garantir a qualidade, consistência e acessibilidade dos dados para futuras análises, facilitando a tomada de decisão baseada em dados.

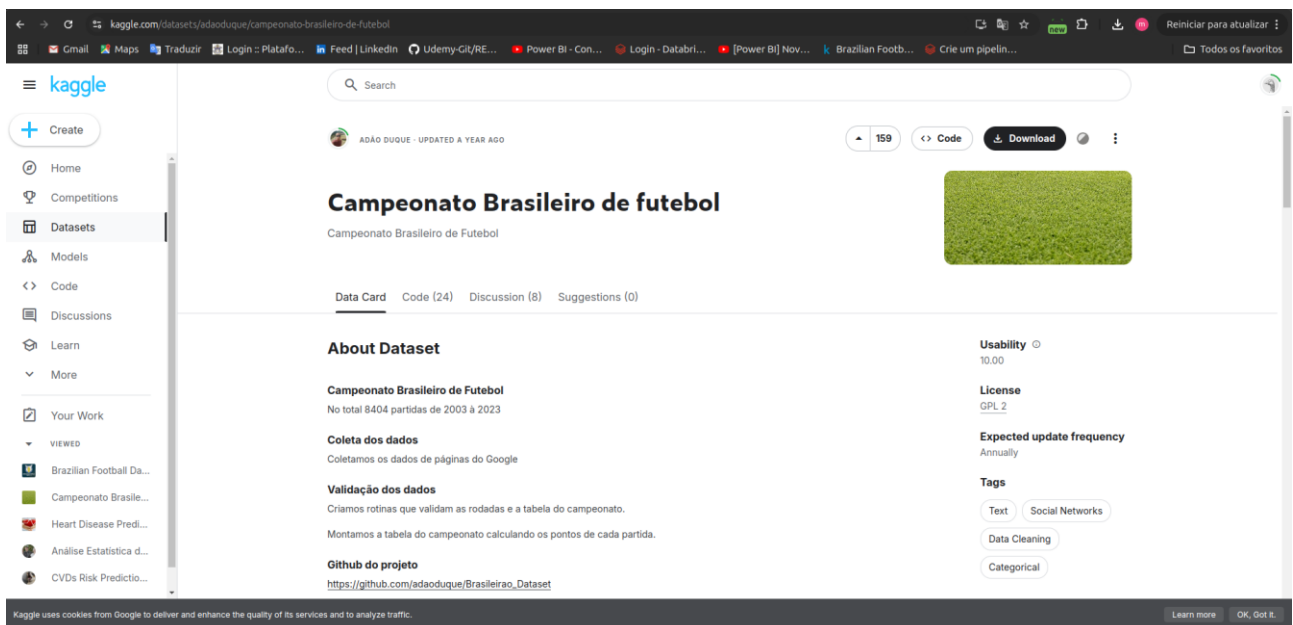
Detalhamento

1 - BUSCA PELOS DADOS

Para atender aos objetivos deste MVP, optei por utilizar a base de dados "**Campeonato Brasileiro de Futebol**", disponível gratuitamente no Kaggle. Essa base contém informações detalhadas sobre partidas, estatísticas dos clubes, gols e cartões, permitindo uma análise abrangente do desempenho das equipes ao longo do campeonato.

A base pode ser acessada através do link:

[Campeonato Brasileiro de Futebol - Kaggle](https://www.kaggle.com/adaodduque/campeonato-brasileiro-de-futebol)



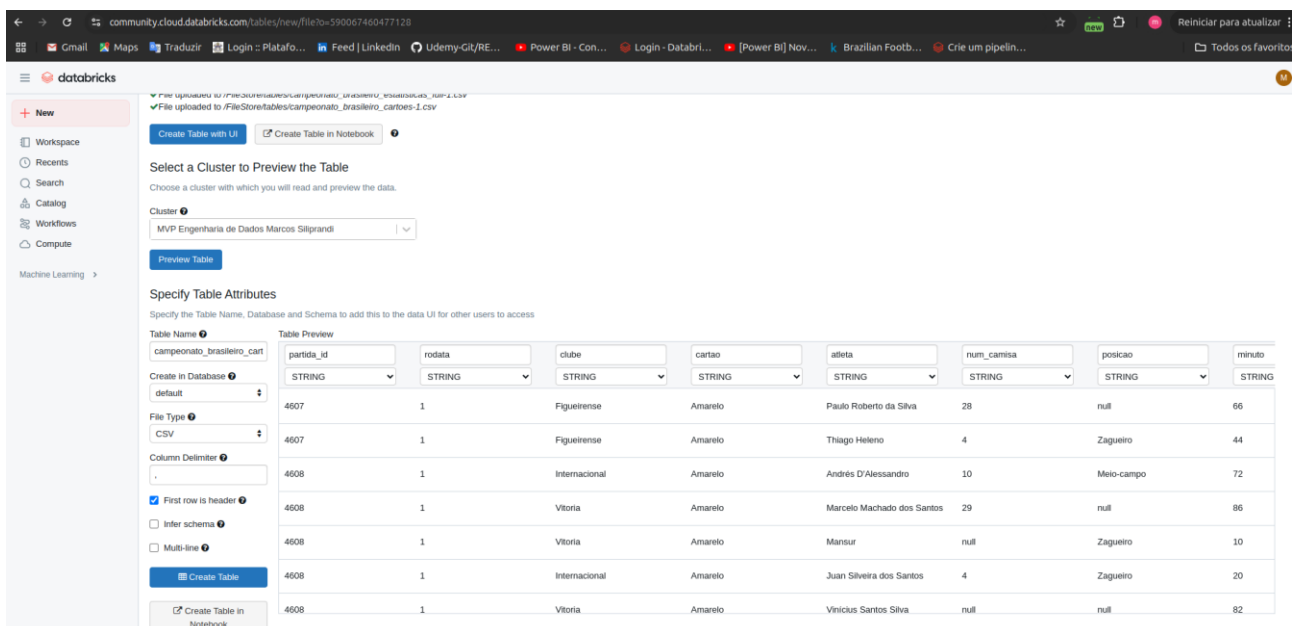
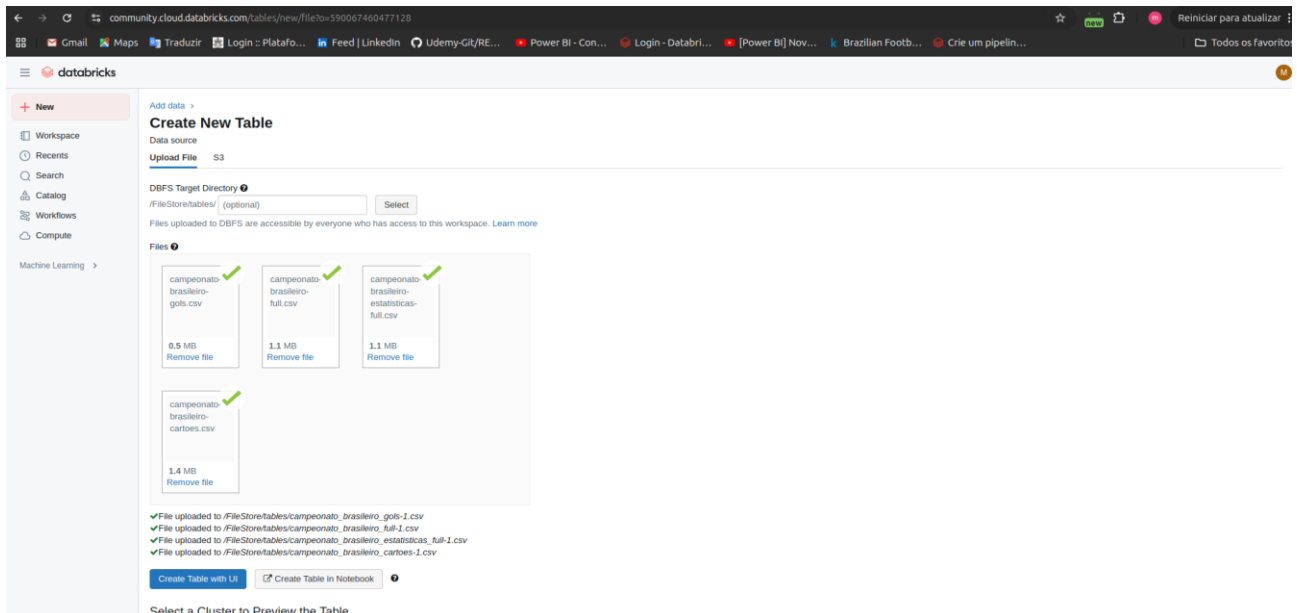
The screenshot shows the Kaggle dataset page for "Campeonato Brasileiro de Futebol" by user ADÃO DUQUE. The page is viewed on a desktop browser with multiple tabs open. The left sidebar shows the Kaggle navigation menu with "Datasets" selected. The main content area displays the dataset title, a search bar, and a "Download" button. Below the title, there are tabs for "Data Card", "Code (24)", "Discussion (8)", and "Suggestions (0)". The "Data Card" tab is active, showing an "About Dataset" section with details about the data collection (8404 matches from 2003 to 2023), data collection method (Google pages), and data validation. A "Github do projeto" link is also provided. On the right side, there are sections for "Usability" (10.00), "License" (GPL 2), "Expected update frequency" (Annually), and "Tags" (Text, Social Networks, Data Cleaning, Categorical).

2 – COLETA DOS DADOS

A base de dados escolhida possui 4 arquivos em formato CSV e optei por baixar os arquivos e realizar o upload deles através do modo gráfico do Databricks salvando-os no caminho /FileStore/tables/.

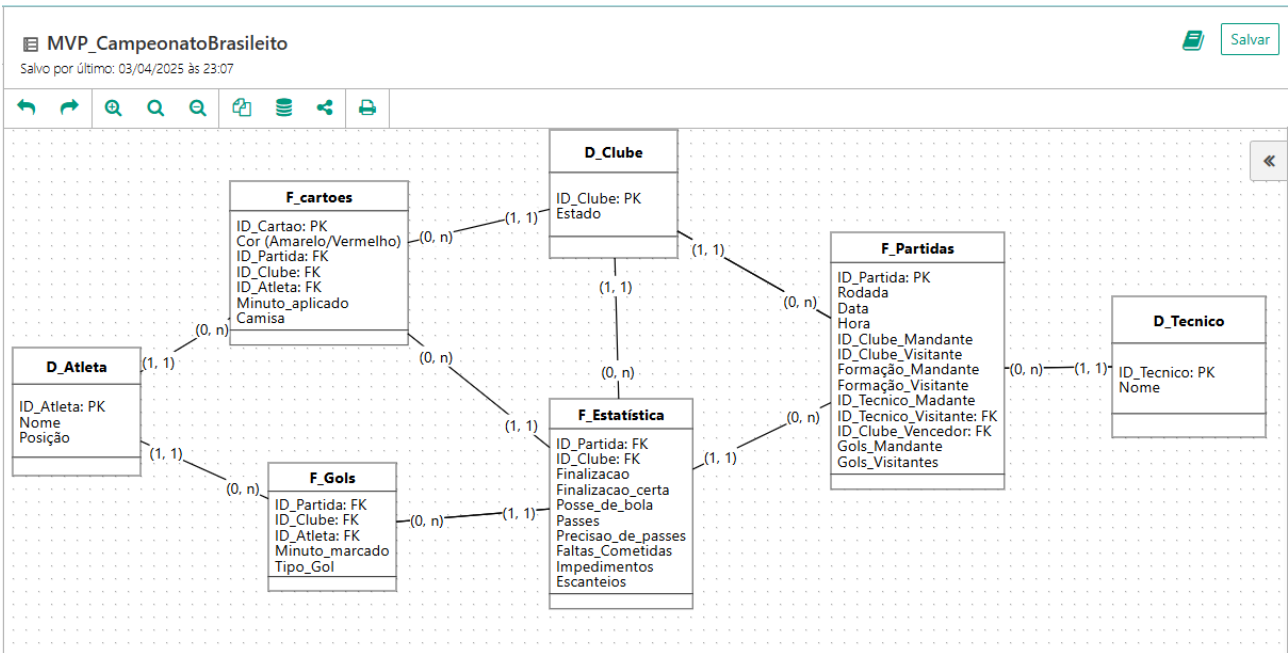
Nome dos arquivos:

- campeonato_brasileiro_full.csv – Contém os dados das partidas.
- campeonato_brasileiro_estatisticas_full.csv – Contém os dados estatísticos
- campeonato_brasileiro_gols.csv – Contém os dados de gols marcados.
- campeonato_brasileiro_cartoes.csv – Contém os dados de cartões aplicados.



3 – MODELAGEM

Analisando as tabelas, percebi que os dados têm diferentes níveis de detalhe, então faz sentido usar mais de uma tabela fato. Essas tabelas podem se ligar tanto às dimensões quanto entre si, dependendo do caso. Por isso, cheguei à conclusão de que o modelo mais adequado aqui é o Esquema Estrela com múltiplas tabelas fato.



3.1 Catálogo de dados

Tabela Fato: F_Partidas

Coluna	Descrição	Tipo	Domínio
ID_Partida	Identificador único da partida	INT	-
Rodada	Rodada do campeonato	INT	1 - 38
Data	Data do jogo	DATE	-
Hora	Horário do jogo	TIME	-
ID_Clube_Mandante	Clube mandante (chave estrangeira)	INT	-
ID_Clube_Visitante	Clube visitante (chave estrangeira)	INT	-
Formação_Mandante	Formação do time mandante (ex: 4-4-2)	STRING	-
Formação_Visitante	Formação do time visitante	STRING	-
ID_Tecnico_Mandante	Técnico do time mandante	INT	-
ID_Tecnico_Visitante	Técnico do time visitante	INT	-
ID_Clube_Vencedor	Clube vencedor da partida	INT	- / NULL (empate)
Gols_Mandante	Gols marcados pelo mandante	INT	0 - 10
Gols_Visitante	Gols marcados pelo visitante	INT	0 - 10

Tabela Fato: F_Cartoes

Coluna	Descrição	Tipo	Domínio
ID_cartoes	Identificador único do evento de cartão	INT	-
Cor	Cor do cartão aplicado	STRING	Amarelo, Vermelho
ID_Partida	Identificador da partida	INT	-
ID_Clube	Clube do atleta penalizado	INT	-
ID_Atleta	Atleta penalizado	INT	-
Minuto_aplicado	Minuto da partida em que o cartão foi dado	INT	0 - 100
Camisa	Número da camisa do jogador	INT	1 - 99 (pode haver ausentes)

Tabela Fato: F_Estatistica

Coluna	Descrição	Tipo	Domínio
ID_Partida	Identificador da partida	INT	-
ID_Clube	Identificador do clube	INT	-
Finalizacao	Total de finalizações	INT	0 - 40
Finalizacao_certa	Finalizações no alvo	INT	0 - 30
Posse_bola	Posse de bola (%)	FLOAT	0 - 100
Passes	Total de passes	INT	0 - 1000
Precisao_passes	Precisão dos passes (%)	FLOAT	0 - 100
Faltas_cometidas	Número de faltas cometidas	INT	0 - 50
Impedimentos	Número de impedimentos	INT	0 - 10
Escanteios	Número de escanteios	INT	0 - 15

Tabela Fato: F_Gols

Coluna	Descrição	Tipo	Domínio
ID_Partida	Identificador da partida	INT	-
ID_Clube	Clube do atleta que marcou o gol	INT	-
ID_Atleta	Atleta que marcou o gol	INT	-
Minuto_marcado	Minuto do jogo em que o gol foi marcado	INT	0 - 100
Tipo_Gol	Tipo de gol	STRING	Normal, Pênalti, Gol Contra

Tabela Dimensão: D_Clube

Coluna	Descrição	Tipo	Domínio
ID_Clube	Identificador do clube	INT	-
Nome	Nome do clube	STRING	-
Estado	Estado de origem do clube	STRING	UF (ex: SP)

Tabela Dimensão: D_Atleta

Coluna	Descrição	Tipo	Domínio
ID_Atleta	Identificador do atleta	INT	-
Nome	Nome do atleta	STRING	-
Posição	Posição do jogador	STRING	Goleiro, Zagueiro, Meio-campo, Atacante

Tabela Dimensão: D_Tecnico

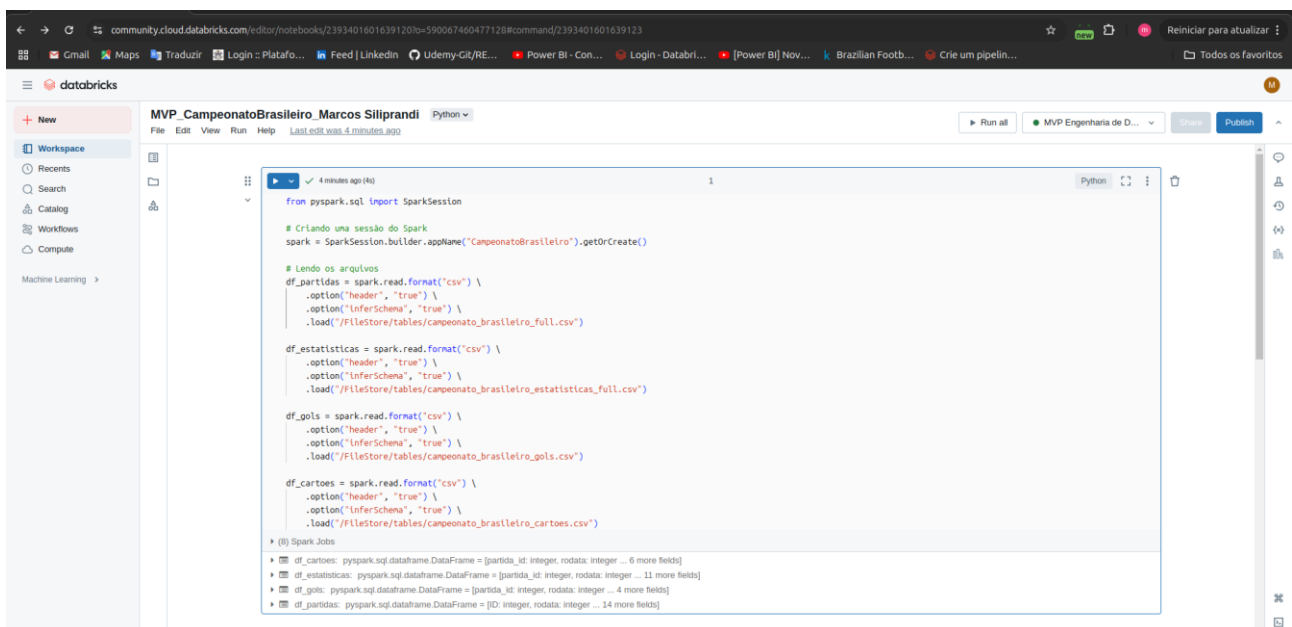
Coluna	Descrição	Tipo	Domínio
ID_Tecnico	Identificador do técnico	INT	-
Nome	Nome do técnico	STRING	-

4 – PIPELINE DE ETL

Nessa etapa, vou fazer a **extração, transformação e carga dos dados (ETL)**, explicando como tudo será armazenado e organizado dentro do Databricks.

4.1 – Extração

Como comentei lá na parte de **coleta**, os dados vêm de **arquivos CSV**. Usando o **Databricks com Spark**, estou lendo esses arquivos direto e começando a preparar os dados. Nesse momento, o modelo de armazenamento que estou usando é mais no estilo **Data Lake**, porque os dados ainda estão em um formato mais bruto, sem tratamento ou estrutura de banco tradicional.



```
from pyspark.sql import SparkSession

# Criando uma sessão do Spark
spark = SparkSession.builder.appName("CampeonatoBrasileiro").getOrCreate()

# Lendo os arquivos
df_partidas = spark.read.format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .load("/FileStore/tables/campeonato_brasileiro_full.csv")

df_estatisticas = spark.read.format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .load("/FileStore/tables/campeonato_brasileiro_estatisticas_full.csv")

df_gols = spark.read.format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .load("/FileStore/tables/campeonato_brasileiro_gols.csv")

df_cartoes = spark.read.format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .load("/FileStore/tables/campeonato_brasileiro_cartoes.csv")
```

(t) Spark Jobs

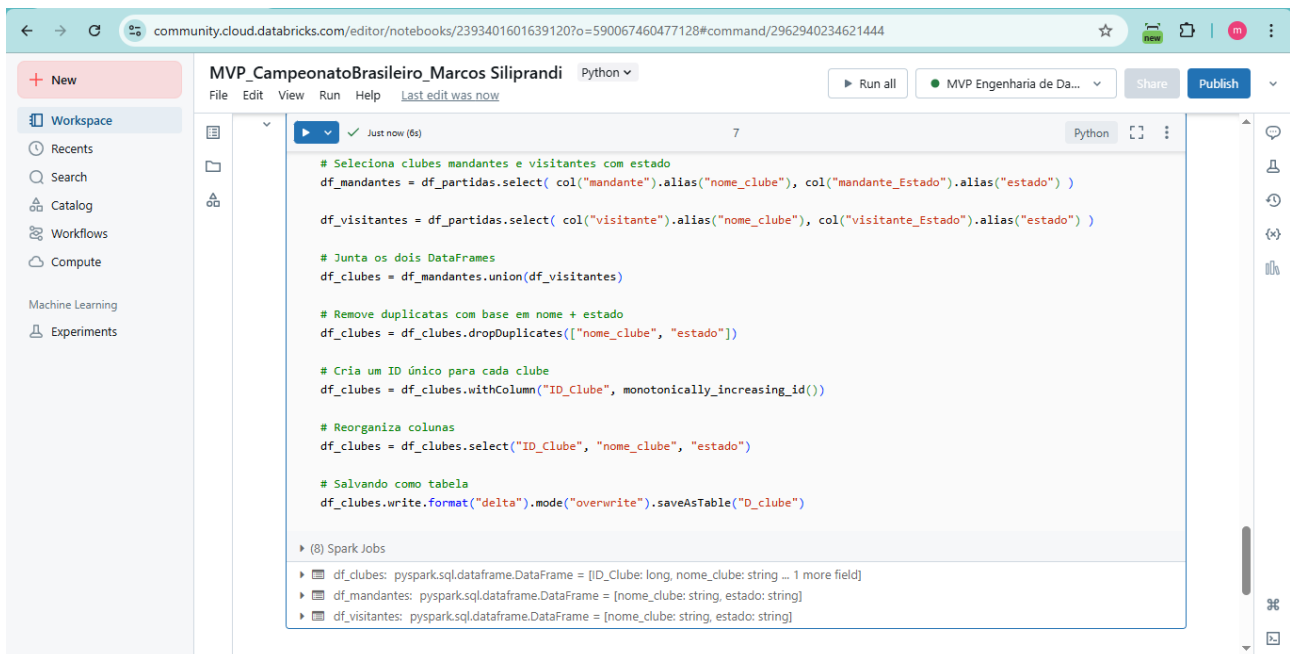
- df_cartoes: pyspark.sql.dataframe.DataFrame = [partida_id: integer, rodada: integer ... 6 more fields]
- df_estatisticas: pyspark.sql.dataframe.DataFrame = [partida_id: integer, rodada: integer ... 11 more fields]
- df_gols: pyspark.sql.dataframe.DataFrame = [partida_id: integer, rodada: integer ... 4 more fields]
- df_partidas: pyspark.sql.dataframe.DataFrame = [ID: integer, rodada: integer ... 14 more fields]

4.2 - Transformação

Aqui começo a aplicar o modelo de dados para Data Warehouse, transformando as tabelas brutas em tabelas dimensão e fato. A transformação será feita seguindo essa ordem, começando pelas dimensões, já que as tabelas fato dependem dos atributos das dimensões, que serão utilizados como chaves estrangeiras para manter os relacionamentos entre os dados.

- TABELA DIMENSÃO CLUBE(D_CLUBE)

Para criar a tabela D_Clube, utilizei como base o DataFrame `df_partidas`, que contém os atributos `mandante`, `visitante`, `mandante_Estado` e `visitante_Estado`. Os clubes foram unificados com base na combinação entre nome e estado, permitindo manter registros distintos para clubes com o mesmo nome, mas localizados em estados diferentes. Além desses atributos, foi criado o campo `ID_Clube`, que é gerado automaticamente com um identificador único para cada clube.



```
# Seleciona clubes mandantes e visitantes com estado
df_mandantes = df_partidas.select( col("mandante").alias("nome_clube"), col("mandante_Estado").alias("estado") )

df_visitantes = df_partidas.select( col("visitante").alias("nome_clube"), col("visitante_Estado").alias("estado") )

# Junta os dois DataFrames
df_clubes = df_mandantes.union(df_visitantes)

# Remove duplicatas com base em nome + estado
df_clubes = df_clubes.dropDuplicates(["nome_clube", "estado"])

# Cria um ID único para cada clube
df_clubes = df_clubes.withColumn("ID_Clube", monotonically_increasing_id())

# Reorganiza colunas
df_clubes = df_clubes.select("ID_Clube", "nome_clube", "estado")

# Salvando como tabela
df_clubes.write.format("delta").mode("overwrite").saveAsTable("D_Clube")
```

(8) Spark Jobs

- df_clubes: pyspark.sql.dataframe.DataFrame = [ID_Clube: long, nome_clube: string ... 1 more field]
- df_mandantes: pyspark.sql.dataframe.DataFrame = [nome_clube: string, estado: string]
- df_visitantes: pyspark.sql.dataframe.DataFrame = [nome_clube: string, estado: string]

- TABELA DIMENSÃO TÉCNICO (D_TECNICO)

A tabela D_Tecnico foi criada a partir do DataFrame `df_partidas`, utilizando os atributos `tecnico_mandante` e `tecnico_visitante`. Os nomes dos técnicos foram unificados e duplicatas removidas. Além disso, foi criado o campo `ID_Tecnico`, gerado automaticamente como um identificador único para cada técnico.

The screenshot shows a Databricks notebook interface. The top bar indicates the notebook is named 'MVP_CampeonatoBrasileiro_Marcos Siliprandi' and is written in Python. The left sidebar shows the workspace navigation menu. The main area displays a code cell with the following Python code:

```
# Seleciona os técnicos (mandante e visitante)
df_tecnicos_mandante = df_partidas.select(col("tecnico_mandante").alias("nome")).filter(col("tecnico_mandante").isNotNull())
df_tecnicos_visitante = df_partidas.select(col("tecnico_visitante").alias("nome")).filter(col("tecnico_visitante").isNotNull())

# Junta os dois e remove duplicatas
df_tecnicos = df_tecnicos_mandante.union(df_tecnicos_visitante).dropDuplicates()

# Cria ID único para cada técnico
df_tecnicos = df_tecnicos.withColumn("ID_Tecnico", monotonically_increasing_id())

# Organiza colunas
df_tecnicos = df_tecnicos.select("ID_Tecnico", "nome")

# Salva como tabela Delta
df_tecnicos.write.format("delta").mode("overwrite").saveAsTable("D_tecnico")
```

Below the code, the Spark Jobs section shows the execution of 7 jobs. The output of the jobs is displayed, showing the schema of the DataFrames created:

```
df_tecnicos: pyspark.sql.dataframe.DataFrame = [ID_Tecnico: long, nome: string]
df_tecnicos_mandante: pyspark.sql.dataframe.DataFrame = [nome: string]
df_tecnicos_visitante: pyspark.sql.dataframe.DataFrame = [nome: string]
```

- TABELA DIMENSÃO ATLETA (D_ATLETA)

A tabela D_Atleta foi criada a partir dos DataFrames `df_cartoes` e `df_gols`, que contêm nomes de atletas e, quando disponível, a posição em que jogaram. Os nomes foram unificados, as duplicatas removidas e os dados padronizados. O campo `ID_Atleta` foi criado como identificador único para cada jogador, e servirá como chave para as tabelas fato que envolvem cartões e gols.

The screenshot shows a Databricks notebook interface. The top bar indicates the notebook is named 'MVP_CampeonatoBrasileiro_Marcos Siliprandi' and is written in Python. The left sidebar shows the workspace navigation menu. The main area displays a code cell with the following Python code:

```
# Do df_cartoes: tem atleta + posição
df_atletas_cartoes = df_cartoes.select(col("atleta").alias("nome_atleta"), col("posicao"))
    .filter(col("nome_atleta").isNotNull())

# Do df_gols: tem atleta, mas sem posição - então adiciona coluna nula
df_atletas_gols = df_gols.select(col("atleta").alias("nome_atleta")).filter(col("nome_atleta").isNotNull()) \
    .withColumn("posicao", lit(None))

# Junta os dois e remove duplicatas
df_atletas = df_atletas_cartoes.unionByName(df_atletas_gols).dropDuplicates(["nome_atleta"])

# Cria ID único
df_atletas = df_atletas.withColumn("ID_Atleta", monotonically_increasing_id())

# Organiza colunas
df_atletas = df_atletas.select("ID_Atleta", "nome_atleta", "posicao")

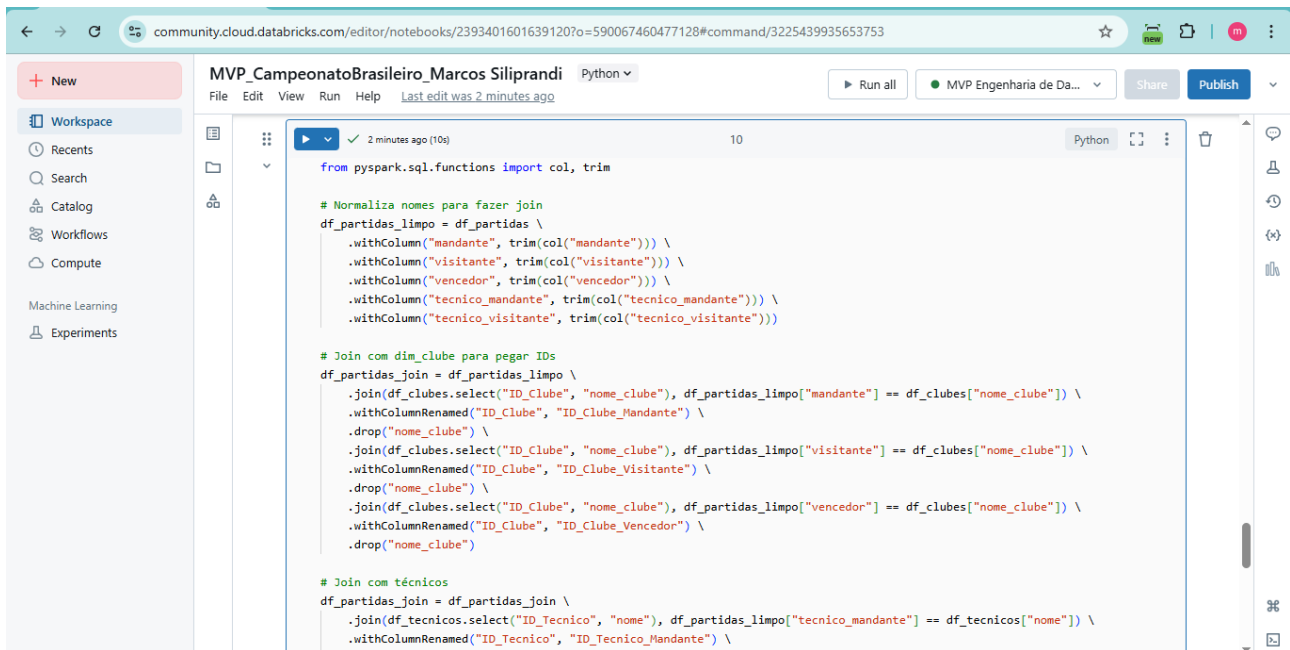
# Salva como tabela Delta
df_atletas.write.format("delta").mode("overwrite").option("overwriteSchema", "true").saveAsTable("dim_atleta")
```

Below the code, the Spark Jobs section shows the execution of 7 jobs. The output of the jobs is displayed, showing the schema of the DataFrames created:

```
df_atletas: pyspark.sql.dataframe.DataFrame = [ID_Atleta: long, nome_atleta: string ... 1 more field]
df_atletas_cartoes: pyspark.sql.dataframe.DataFrame = [nome_atleta: string, posicao: string]
df_atletas_gols: pyspark.sql.dataframe.DataFrame = [nome_atleta: string, posicao: void]
```

- TABELA FATO PARTIDAS (F_PARTIDAS)

A tabela F_Partidas foi construída a partir do DataFrame df_partidas, contendo as informações de cada jogo realizado no campeonato. Foram realizados joins com as tabelas D_Clube e D_Tecnico para substituir os nomes por seus respectivos IDs. Essa tabela contém dados como data, hora, rodada, formação tática, gols marcados e as chaves estrangeiras que ligam cada partida aos clubes e técnicos envolvidos.

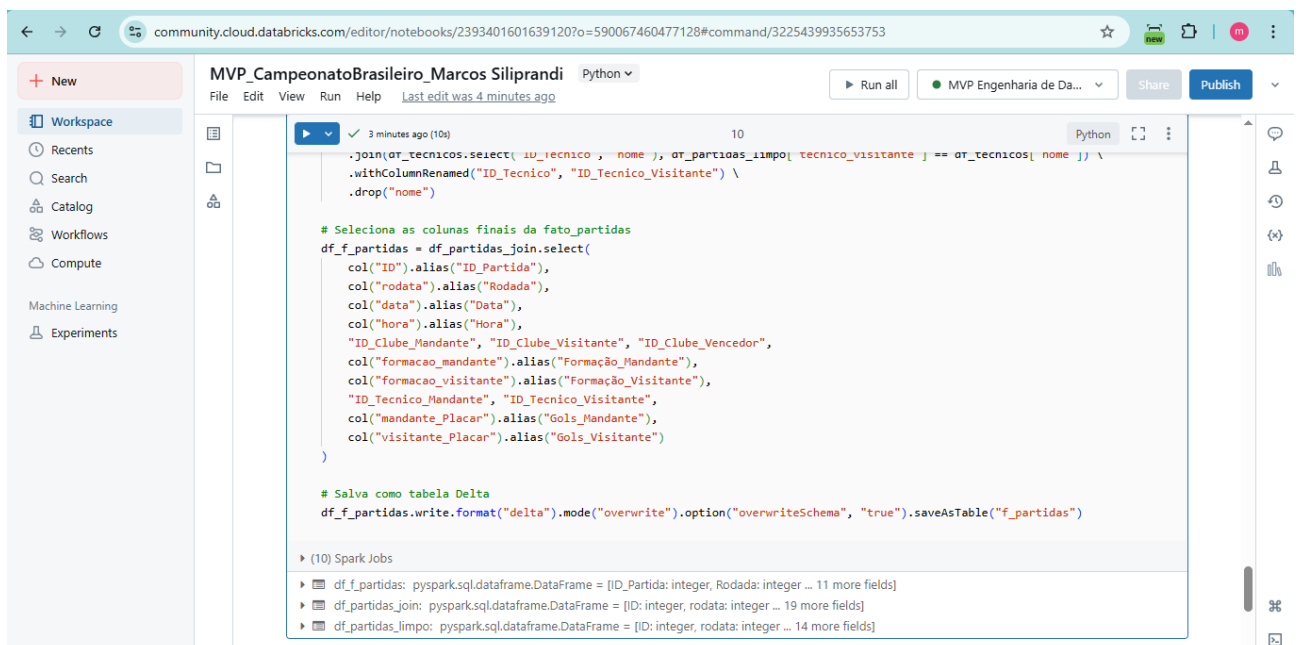


```
from pyspark.sql.functions import col, trim

# Normaliza nomes para fazer join
df_partidas_limpo = df_partidas \
    .withColumn("mandante", trim(col("mandante"))) \
    .withColumn("visitante", trim(col("visitante"))) \
    .withColumn("vencedor", trim(col("vencedor"))) \
    .withColumn("tecnico_mandante", trim(col("tecnico_mandante"))) \
    .withColumn("tecnico_visitante", trim(col("tecnico_visitante")))

# Join com dim_clube para pegar IDs
df_partidas_join = df_partidas_limpo \
    .join(df_clubes.select("ID_Clube", "nome_clube"), df_partidas_limpo["mandante"] == df_clubes["nome_clube"]) \
    .withColumnRenamed("ID_Clube", "ID_Clube_Mandante") \
    .drop("nome_clube") \
    .join(df_clubes.select("ID_Clube", "nome_clube"), df_partidas_limpo["visitante"] == df_clubes["nome_clube"]) \
    .withColumnRenamed("ID_Clube", "ID_Clube_Visitante") \
    .drop("nome_clube") \
    .join(df_clubes.select("ID_Clube", "nome_clube"), df_partidas_limpo["vencedor"] == df_clubes["nome_clube"]) \
    .withColumnRenamed("ID_Clube", "ID_Clube_Vencedor") \
    .drop("nome_clube")

# Join com técnicos
df_partidas_join = df_partidas_join \
    .join(df_tecnicos.select("ID_Tecnico", "nome"), df_partidas_limpo["tecnico_mandante"] == df_tecnicos["nome"]) \
    .withColumnRenamed("ID_Tecnico", "ID_Tecnico_Mandante") \
    .drop("nome")
```



```
# Seleciona as colunas finais da fato_partidas
df_f_partidas = df_partidas_join.select(
    col("ID").alias("ID_Partida"),
    col("rodada").alias("Rodada"),
    col("data").alias("Data"),
    col("hora").alias("Hora"),
    "ID_Clube_Mandante", "ID_Clube_Visitante", "ID_Clube_Vencedor",
    col("formacao_mandante").alias("Formação_Mandante"),
    col("formacao_visitante").alias("Formação_Visitante"),
    "ID_Tecnico_Mandante", "ID_Tecnico_Visitante",
    col("mandante_Placar").alias("Gols_Mandante"),
    col("visitante_Placar").alias("Gols_Visitante")
)

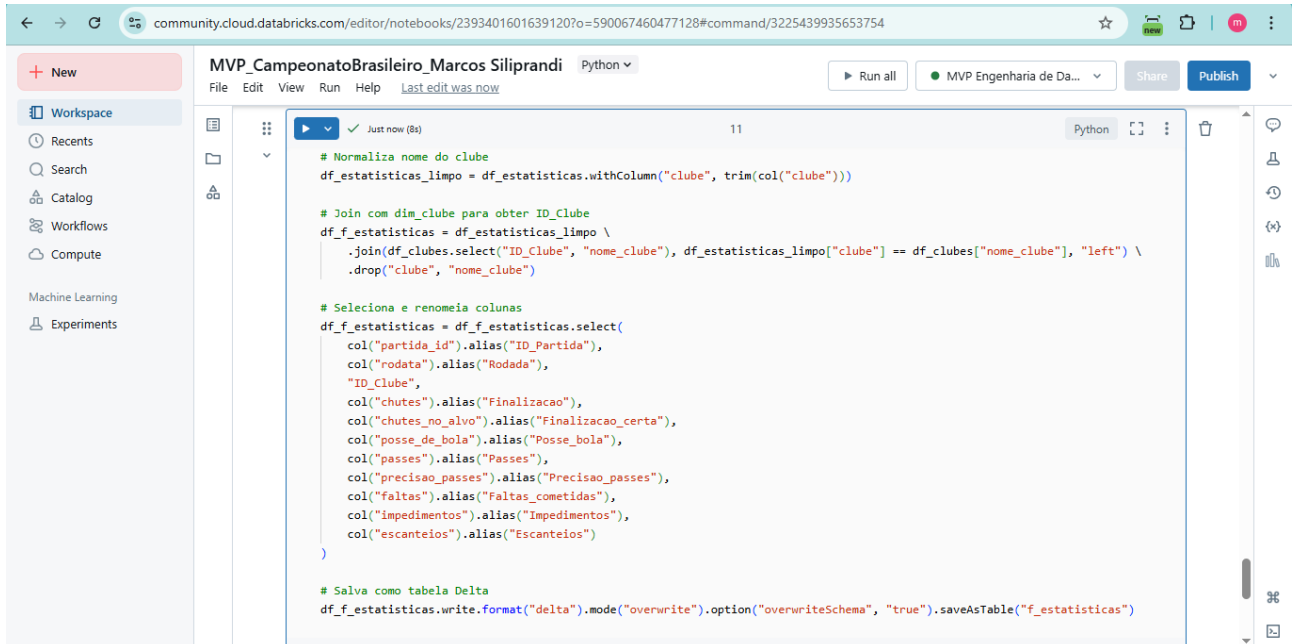
# Salva como tabela Delta
df_f_partidas.write.format("delta").mode("overwrite").option("overwriteSchema", "true").saveAsTable("f_partidas")
```

(10) Spark Jobs

- df_f_partidas: pyspark.sql.dataframe.DataFrame = [ID_Partida: integer, Rodada: integer ... 11 more fields]
- df_partidas_join: pyspark.sql.dataframe.DataFrame = [ID: integer, rodada: integer ... 19 more fields]
- df_partidas_limpo: pyspark.sql.dataframe.DataFrame = [ID: integer, rodada: integer ... 14 more fields]

- Tabela Fato Estatísticas (F_Estatísticas)

A tabela F_Estatísticas foi criada a partir do DataFrame df_estatisticas, contendo os dados estatísticos dos clubes em cada partida. Os nomes dos clubes foram substituídos pelos seus respectivos IDs da dimensão D_Clube. A tabela inclui indicadores como número de finalizações, posse de bola, passes, faltas, impedimentos e escanteios, ligados a cada partida por meio da chave ID_Partida.



```
community.cloud.databricks.com/editor/notebooks/2393401601639120?o=590067460477128#command/3225439935653754

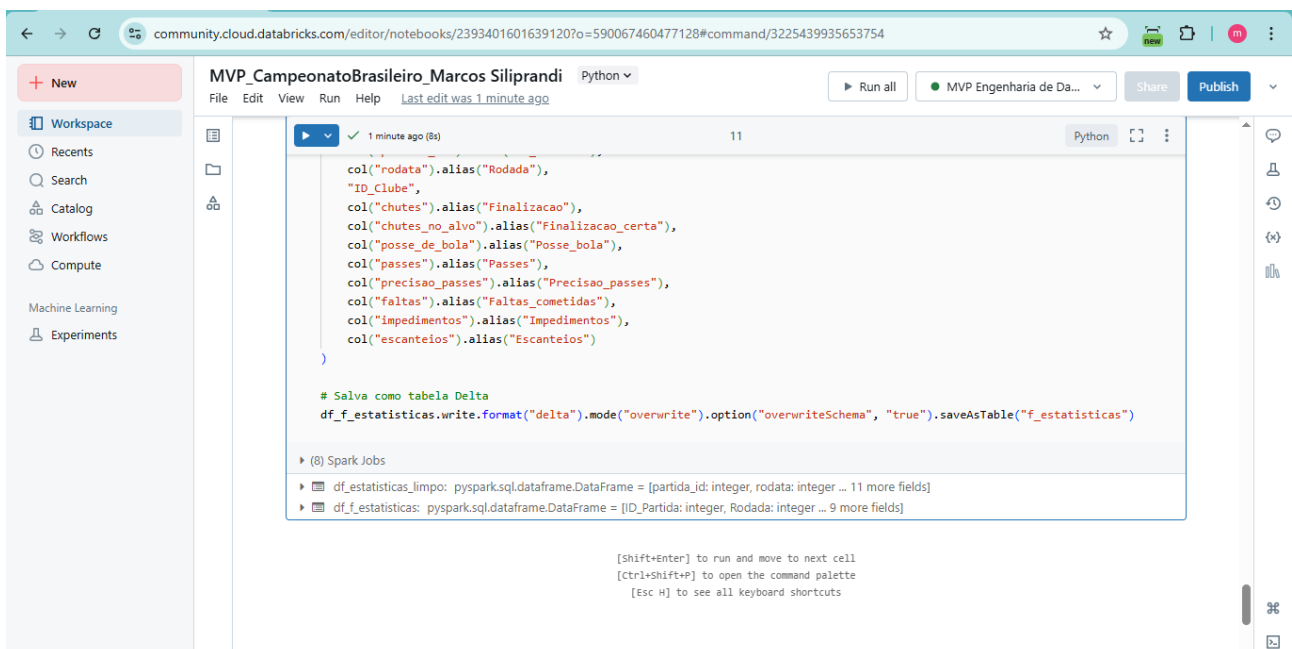
MVP_CampeonatoBrasileiro_Marcos Siliprandi Python
File Edit View Run Help Last edit was now

# Normaliza nome do clube
df_estatisticas_limpo = df_estatisticas.withColumn("clube", trim(col("clube")))

# Join com dim_clube para obter ID_Clube
df_f_estatisticas = df_estatisticas_limpo \
    .join(df_clubes.select("ID_Clube", "nome_clube"), df_estatisticas_limpo["clube"] == df_clubes["nome_clube"], "left") \
    .drop("clube", "nome_clube")

# Seleciona e renomeia colunas
df_f_estatisticas = df_f_estatisticas.select(
    col("partida_id").alias("ID_Partida"),
    col("rodada").alias("Rodada"),
    "ID_Clube",
    col("chutes").alias("Finalizacao"),
    col("chutes_no_alvo").alias("Finalizacao_certa"),
    col("posse_de_bola").alias("Posse_bola"),
    col("passes").alias("Passes"),
    col("precisao_passes").alias("Precisao_passes"),
    col("faltas").alias("Faltas_cometidas"),
    col("impedimentos").alias("Impedimentos"),
    col("escanteios").alias("Escanteios")
)

# Salva como tabela Delta
df_f_estatisticas.write.format("delta").mode("overwrite").option("overwriteSchema", "true").saveAsTable("f_estatisticas")
```



```
community.cloud.databricks.com/editor/notebooks/2393401601639120?o=590067460477128#command/3225439935653754

MVP_CampeonatoBrasileiro_Marcos Siliprandi Python
File Edit View Run Help Last edit was 1 minute ago

col("rodada").alias("Rodada"),
"ID_Clube",
col("chutes").alias("Finalizacao"),
col("chutes_no_alvo").alias("Finalizacao_certa"),
col("posse_de_bola").alias("Posse_bola"),
col("passes").alias("Passes"),
col("precisao_passes").alias("Precisao_passes"),
col("faltas").alias("Faltas_cometidas"),
col("impedimentos").alias("Impedimentos"),
col("escanteios").alias("Escanteios")
)

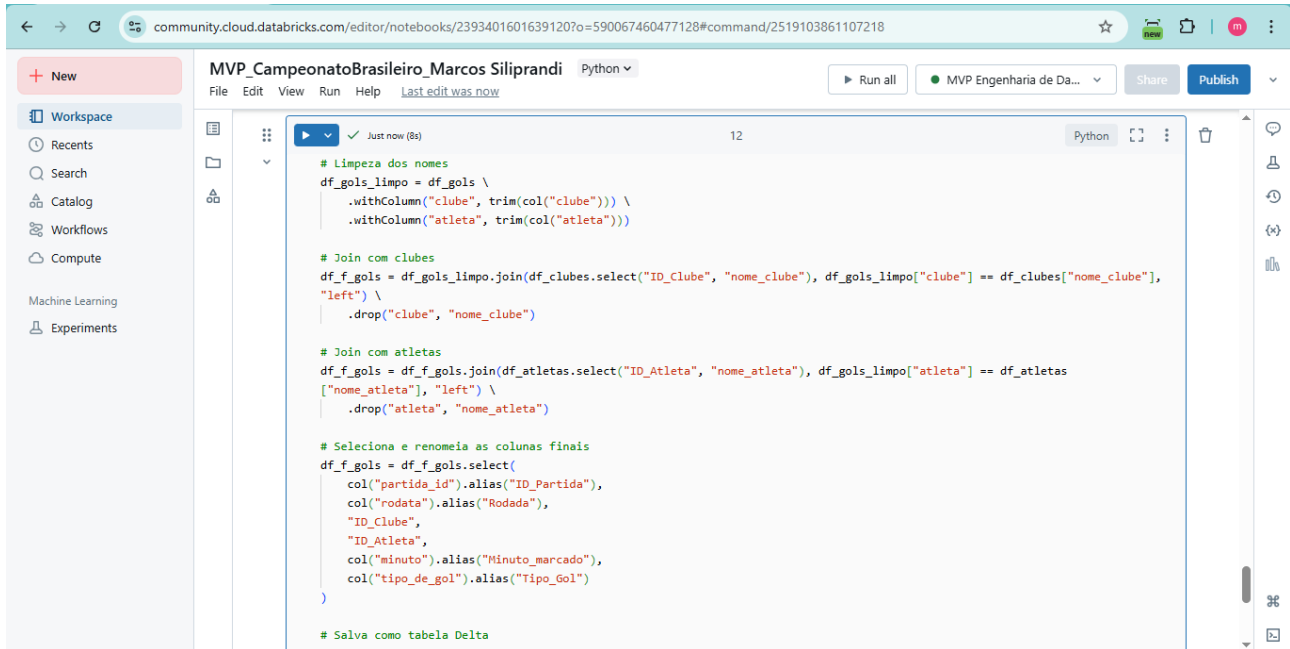
# Salva como tabela Delta
df_f_estatisticas.write.format("delta").mode("overwrite").option("overwriteSchema", "true").saveAsTable("f_estatisticas")

(8) Spark Jobs
df_estatisticas_limpo: pyspark.sql.dataframe.DataFrame = [partida_id: integer, rodada: integer ... 11 more fields]
df_f_estatisticas: pyspark.sql.dataframe.DataFrame = [ID_Partida: integer, Rodada: integer ... 9 more fields]

[Shift+Enter] to run and move to next cell
[Ctrl+Shift+P] to open the command palette
[Esc H] to see all keyboard shortcuts
```

- TABELA FATO GOLS (F_GOLS)

A tabela F_Gols foi criada a partir do DataFrame df_gols, com o objetivo de registrar cada gol marcado nas partidas. Para isso, foi feita a substituição dos nomes dos clubes e jogadores pelos seus respectivos IDs das dimensões D_Clube e D_Atleta. Cada linha representa um gol, contendo o minuto em que foi marcado, o tipo de gol (normal, pênalti, contra, etc.) e os identificadores de clube, atleta e partida.



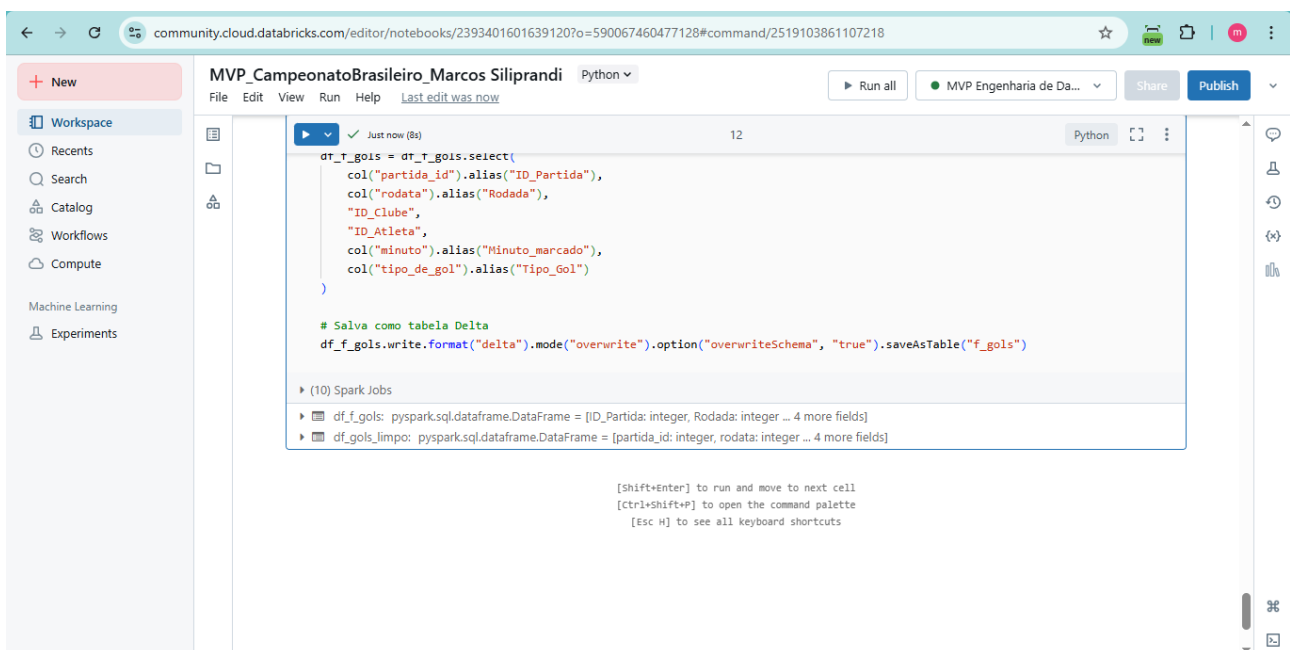
```
# Limpeza dos nomes
df_gols_limpo = df_gols \
    .withColumn("clube", trim(col("clube"))) \
    .withColumn("atleta", trim(col("atleta")))

# Join com clubes
df_f_gols = df_gols_limpo.join(df_clubes.select("ID_Clube", "nome_clube"), df_gols_limpo["clube"] == df_clubes["nome_clube"],
    "left") \
    .drop("clube", "nome_clube")

# Join com atletas
df_f_gols = df_f_gols.join(df_atletas.select("ID_Atleta", "nome_atleta"), df_gols_limpo["atleta"] == df_atletas
    ["nome_atleta"], "left") \
    .drop("atleta", "nome_atleta")

# Seleciona e renomeia as colunas finais
df_f_gols = df_f_gols.select(
    col("partida_id").alias("ID_Partida"),
    col("rodada").alias("Rodada"),
    "ID_Clube",
    "ID_Atleta",
    col("minuto").alias("Minuto_marcado"),
    col("tipo_de_gol").alias("Tipo_Gol")
)

# Salva como tabela Delta
```



```
# Salva como tabela Delta
df_f_gols.write.format("delta").mode("overwrite").option("overwriteSchema", "true").saveAsTable("f_gols")
```

▶ (10) Spark Jobs

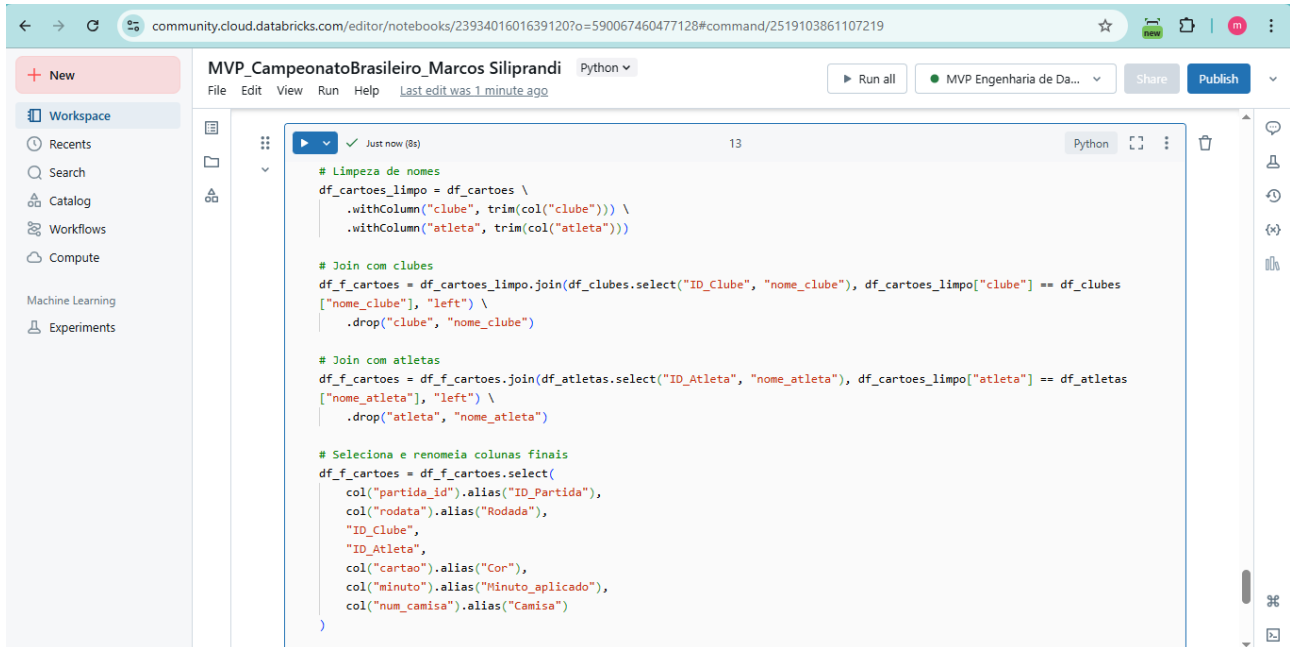
▶ df_f_gols: pyspark.sql.dataframe.DataFrame = [ID_Partida: integer, Rodada: integer ... 4 more fields]

▶ df_gols_limpo: pyspark.sql.dataframe.DataFrame = [partida_id: integer, rodada: integer ... 4 more fields]

[Shift+Enter] to run and move to next cell
[Ctrl+Shift+P] to open the command palette
[Esc H] to see all keyboard shortcuts

- TABELA FATO CARTÕES (F_CARTOES)

A tabela F_Cartoes foi criada a partir do DataFrame df_cartoes, contendo os registros de cartões aplicados aos jogadores durante as partidas. Foram feitos joins com as dimensões D_Clube e D_Atleta para substituir os nomes por seus respectivos identificadores. A tabela inclui informações como o tipo de cartão (amarelo ou vermelho), o minuto da aplicação, o número da camisa e os vínculos com partida, clube e atleta.



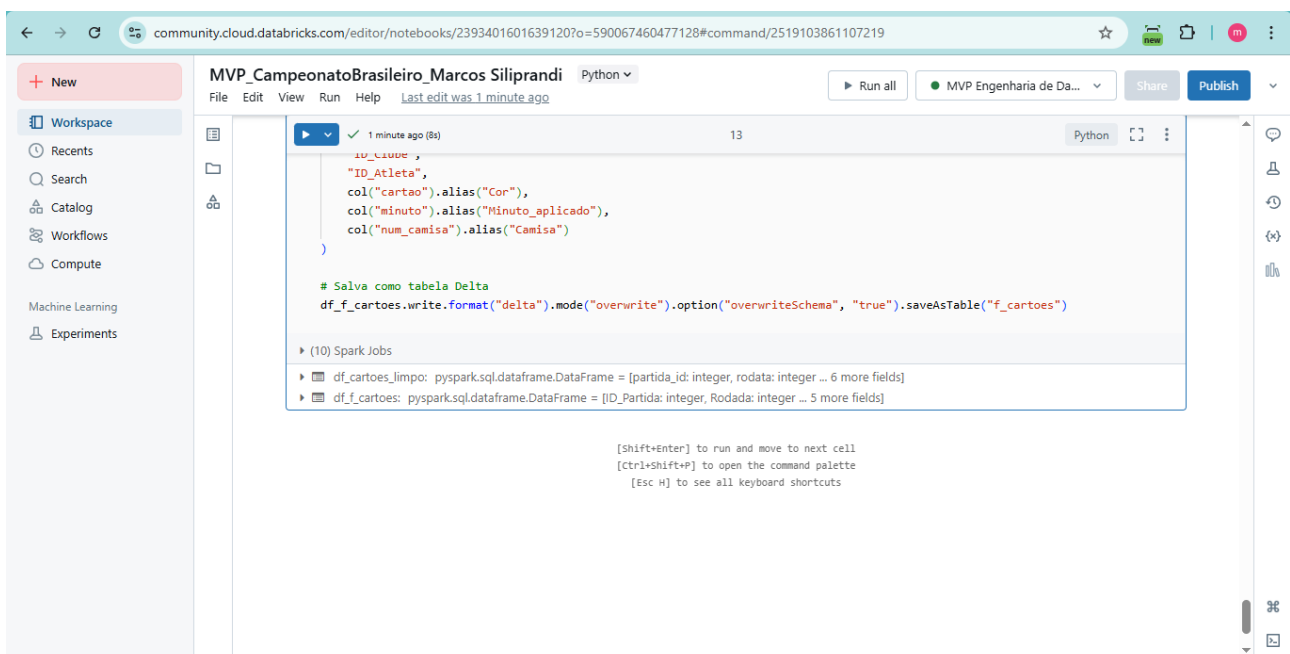
The screenshot shows a Databricks notebook interface. The top bar includes navigation icons, a URL, and a 'Run all' button. The left sidebar shows the workspace structure with 'Workspace', 'Recents', 'Search', 'Catalog', 'Workflows', 'Compute', 'Machine Learning', and 'Experiments'. The main editor area displays Python code for data processing. The code includes comments in Portuguese and uses PySpark syntax for DataFrame operations.

```
# Limpeza de nomes
df_cartoes_limpo = df_cartoes \
    .withColumn("clube", trim(col("clube"))) \
    .withColumn("atleta", trim(col("atleta")))

# Join com clubes
df_f_cartoes = df_cartoes_limpo.join(df_clubes.select("ID_Clube", "nome_clube"), df_cartoes_limpo["clube"] == df_clubes["nome_clube"], "left") \
    .drop("clube", "nome_clube")

# Join com atletas
df_f_cartoes = df_f_cartoes.join(df_atletas.select("ID_Atleta", "nome_atleta"), df_f_cartoes["atleta"] == df_atletas["nome_atleta"], "left") \
    .drop("atleta", "nome_atleta")

# Seleciona e renomeia colunas finais
df_f_cartoes = df_f_cartoes.select(
    col("partida_id").alias("ID_Partida"),
    col("rodada").alias("Rodada"),
    "ID_Clube",
    "ID_Atleta",
    col("cartao").alias("Cor"),
    col("minuto").alias("Minuto_aplicado"),
    col("num_camisa").alias("Camisa")
)
```



This screenshot shows the continuation of the Databricks notebook. The code defines the final schema for the cleaned dataframes and saves them to a Delta table. The interface shows the execution progress with a 'Spark Jobs' section at the bottom.

```
# Salva como tabela Delta
df_f_cartoes.write.format("delta").mode("overwrite").option("overwriteSchema", "true").saveAsTable("f_cartoes")

# df_cartoes_limpo: pyspark.sql.dataframe.DataFrame = [partida_id: integer, rodada: integer ... 6 more fields]
# df_f_cartoes: pyspark.sql.dataframe.DataFrame = [ID_Partida: integer, Rodada: integer ... 5 more fields]
```

Below the code, the 'Spark Jobs' section shows the execution status of the code blocks. At the bottom, there are keyboard shortcuts: [Shift+Enter] to run and move to next cell, [Ctrl+Shift+P] to open the command palette, and [Esc H] to see all keyboard shortcuts.

4.3 – Carga

Com as tabelas já transformadas no formato que defini no modelo estrela, agora foi realizada a carga dos dados para o ambiente do Databricks. Nesta etapa, utilizei o formato Delta para armazenar as tabelas, o que permite melhor performance e flexibilidade na hora de fazer as consultas em SQL.

A carga foi feita primeiro pelas tabelas de dimensão (`dim_clube`, `dim_tecnico` e `dim_atleta`), pois elas são utilizadas como base pelas tabelas fato. Em seguida, carreguei as tabelas fato (`f_partidas`, `f_estatisticas`, `f_gols` e `f_cartoes`), que já fazem referência às dimensões por meio dos seus respectivos IDs.

A estrutura foi pensada para manter os relacionamentos entre as tabelas e facilitar futuras análises e cruzamentos de informações. Agora os dados estão organizados, limpos e prontos para a próxima fase do projeto.

Desta forma, observo que há a flexibilidade do Data Lake aliada a organização do Data Warehouse. Sendo assim, concluo que o modelo de dados utilizado neste MVP é o Data LakeHouse.

5 – Análise

A. QUALIDADE DOS DADOS

Para verificar a qualidade dos dados, realizei a verificação utilizando scripts específicos para cada tabela, analisando valores nulos, duplicatas e possíveis valores fora do padrão. Antes de partir para as tabelas fato, analisei as tabelas de dimensão para garantir que estavam organizadas e prontas para serem usadas nas análises.

Abaixo, fiz um resumo do que encontrei:

- **dim_clube:** Não encontrei nenhum valor nulo e não apareceu nenhum clube repetido com o mesmo nome e estado. Os estados estão todos no padrão certo (siglas como SP, RJ, MG...), então essa dimensão já estava correta e não precisou de ajustes.
- **dim_tecnico:** Nesta dimensão também não foi necessário fazer nenhuma alteração, pois não havia valores faltando nem nomes duplicados. Os nomes estão padronizados, com iniciais e sobrenomes.
- **dim_atleta:** Já nesta dimensão apareceram muitos valores nulos na coluna posição, o que era esperado, já que os dados de gols não trazem essa informação. Fiz uma melhoria usando os dados de cartões para preencher a posição sempre que possível. Também foi encontrado um valor incoerente, onde a posição "Zagueira" foi substituída por "Zagueiro", deixando os dados padronizados. Os nulos que ainda restaram são de atletas que realmente não tinham essa informação, e a solução para esses casos pode ser analisada conforme a necessidade de cada análise.

Com isso, considero que todas as tabelas de dimensão estão com boa qualidade e já podem ser utilizadas nas tabelas fato e nas consultas que vou construir na próxima etapa.

Depois de analisar as tabelas de dimensão, realizei a verificação da qualidade dos dados nas tabelas fato, observando possíveis valores nulos, inconsistências ou dados fora do padrão. Abaixo, segue o resumo do que foi encontrado e ajustado:

- **f_partidas:** A tabela está completa, sem valores nulos ou inconsistências. Todos os IDs de referência (clubes e técnicos) estão preenchidos corretamente e não foram encontrados valores negativos nos placares. Nenhum ajuste foi necessário.
- **f_estatisticas:** Também não apresentou problemas com os IDs de referência. Os valores de posse de bola e precisão de passes estão dentro do intervalo esperado (0 a 100), e as colunas numéricas não apresentaram valores negativos. Apenas alguns registros apresentaram posse_bola e precisao_passes nulos, o que é aceitável, já que esses dados não estavam disponíveis em alguns jogos.
- **f_gols:** A tabela também está íntegra. Não foram encontrados nulos nas colunas principais (ID_Partida, ID_Clube, ID_Atleta, Minuto_marcado). A coluna Tipo_Gol apresentava valores nulos, que foram substituídos por "Normal" para manter a consistência.
- **f_cartoes:** Inicialmente, havia 6 registros com ID_Atleta nulo, que foram excluídos por representarem uma quantidade muito pequena e não impactarem nas análises. Também foi feita uma correção no valor " Amarelo" (com espaço) da coluna Cor, padronizando para "Amarelo". Fora isso, todos os demais dados estavam coerentes

B. SOLUÇÃO DO PROBLEMA

- Quais clubes mais finalizaram ao longo do campeonato?

```
Just now (<1s) 29 Python

# Executa SQL e armazena resultado em um DataFrame Spark
df_finalizacoes = spark.sql("""
    SELECT
        c.nome_clube,
        COUNT(*) AS total_jogos,
        SUM(e.Finalizacao) AS total_finalizacoes,
        ROUND(SUM(e.Finalizacao) / COUNT(*), 2) AS media_finalizacoes
    FROM f_estatisticas e
    JOIN d_clube c
    ON e.ID_Clube = c.ID_Clube
    WHERE e.Finalizacao IS NOT NULL
    GROUP BY c.nome_clube
    ORDER BY media_finalizacoes DESC
""")

df_finalizacoes: pyspark.sql.dataframe.DataFrame = [nome_clube: string, total_jogos: long ... 2 more fields]
```

```
1 minute ago (2s) 30 Python

# Converte para Pandas
pdf = df_finalizacoes.toPandas()

# Ordena pelos que têm maior média de finalizações
pdf = pdf.sort_values(by="media_finalizacoes", ascending=False)

# Mantém apenas os 15 primeiros (opcional)
top15 = pdf.head(15)

(5) Spark Jobs
```



```

community.cloud.databricks.com/editor/notebooks/2393401601639120?o=590067460477128#command/1497139172212410

MVP_CampeonatoBrasileiro_Marcos Siliprandi Python
File Edit View Run Help Last edit was now

Run all MVP Engenharia de Da... Share Publish

Workspace
Recents
Search
Catalog
Workflows
Compute
Machine Learning
Experiments

# Plota o gráfico duplo: Média de Finalizações vs Total de Jogos
import matplotlib.pyplot as plt
import seaborn as sns

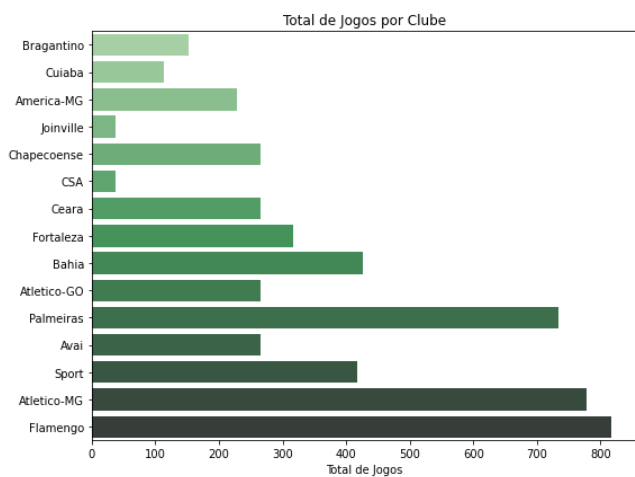
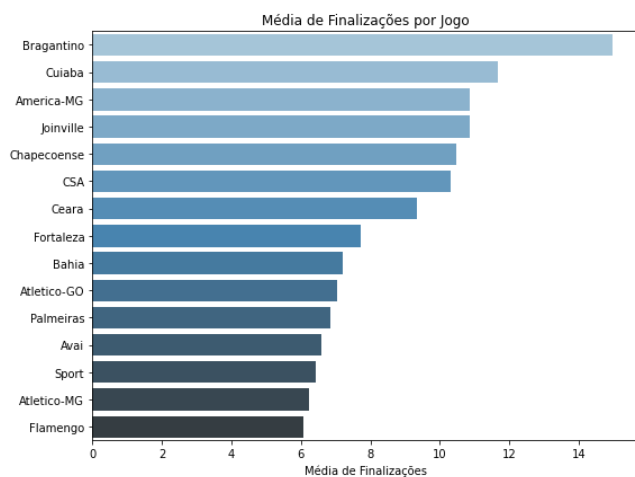
plt.figure(figsize=(16, 6))

# Gráfico 1 - Média de Finalizações
plt.subplot(1, 2, 1)
sns.barplot(data=top15, y="nome_clube", x="media_finalizacoes", palette="Blues_d")
plt.title("Média de Finalizações por Jogo")
plt.xlabel("Média de Finalizações")
plt.ylabel("")

# Gráfico 2 - Total de Jogos
plt.subplot(1, 2, 2)
sns.barplot(data=top15, y="nome_clube", x="total_jogos", palette="Greens_d")
plt.title("Total de Jogos por Clube")
plt.xlabel("Total de Jogos")
plt.ylabel("")

plt.tight_layout()
plt.show()

```

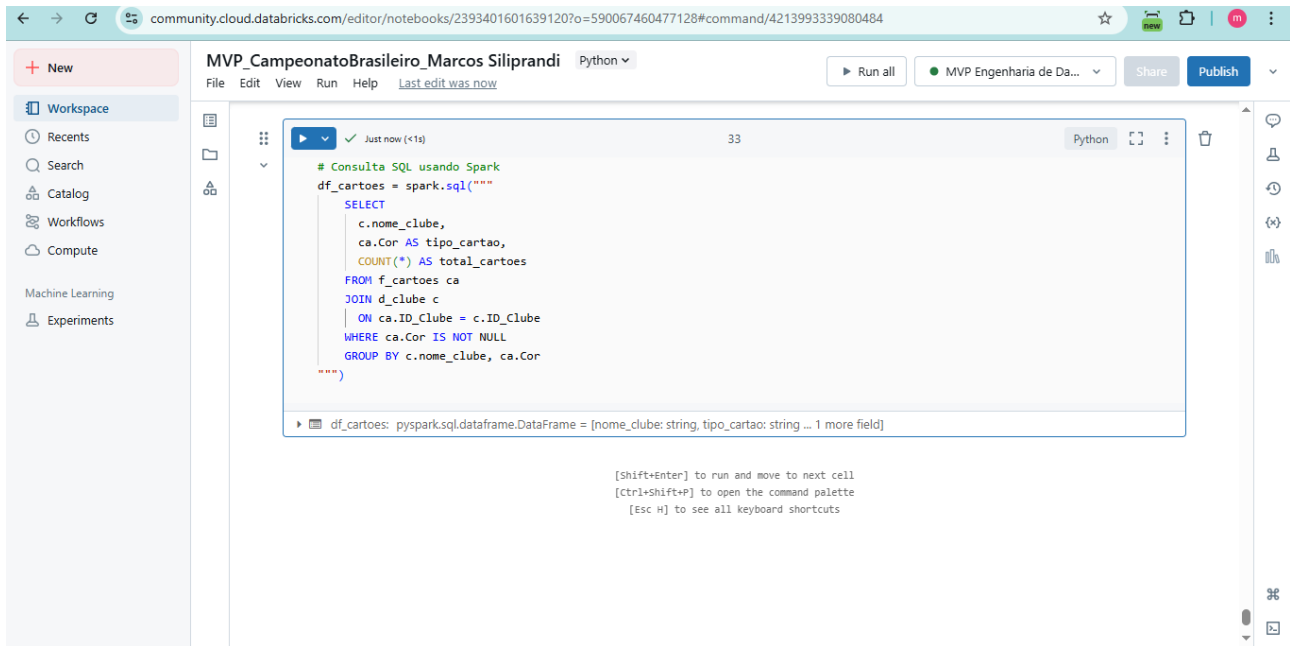


Ao analisar quais clubes mais finalizaram por partida, percebi que alguns times com menos jogos, como Bragantino, Cuiabá e América-MG, se destacaram com médias bem altas de finalizações. Eles aproveitaram bem o tempo em campo e mostraram um estilo mais ofensivo.

Já os clubes tradicionais, como Fluminense, Palmeiras e Atlético-MG, aparecem com médias um pouco menores, mas é importante lembrar que eles disputaram muitas temporadas, o que acaba deixando a média mais estável.

No geral, a média de finalizações ajuda a mostrar o quanto um time pressiona o adversário, e quando comparamos isso com o número de jogos, fica mais fácil entender se o desempenho foi consistente ou só em alguns campeonatos.

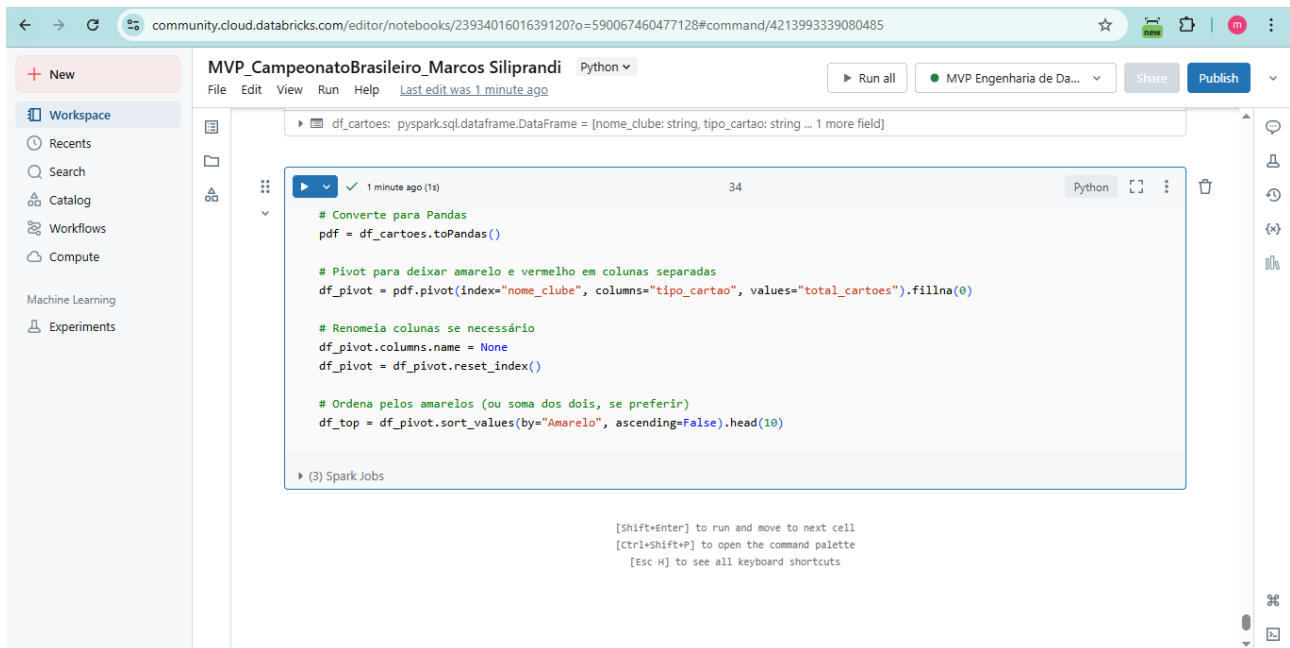
- Quais clubes mais receberam cartões amarelos e vermelhos no campeonato?



The screenshot shows a Databricks notebook interface. The top bar indicates the notebook is titled "MVP_CampeonatoBrasileiro_Marcos Siliprandi" and is written in Python. The left sidebar shows the workspace navigation menu. The main area contains a code cell that has been executed, as indicated by a green checkmark and the text "Just now (<1s)". The code is a SQL query using Spark to select club names, card colors, and the count of cards. The output of the query is displayed below the code cell, showing a DataFrame with columns for club name and card color.

```
# Consulta SQL usando Spark
df_cartoes = spark.sql("""
SELECT
  c.nome_clube,
  ca.Cor AS tipo_cartao,
  COUNT(*) AS total_cartoes
FROM f_cartoes ca
JOIN d_clube c
  ON ca.ID_Clube = c.ID_Clube
WHERE ca.Cor IS NOT NULL
GROUP BY c.nome_clube, ca.Cor
""")
```

df_cartoes: pyspark.sql.dataframe.DataFrame = [nome_clube: string, tipo_cartao: string ... 1 more field]



The screenshot shows the same Databricks notebook interface, but with a new code cell added. This cell has also been executed, as indicated by a green checkmark and the text "1 minute ago (1s)". The code converts the Spark DataFrame to a Pandas DataFrame, pivots the data to show the count of cards by club and color, and then sorts the results by the number of yellow cards in descending order. The output of the code is displayed below the code cell, showing a DataFrame with columns for club name and the count of yellow and red cards.

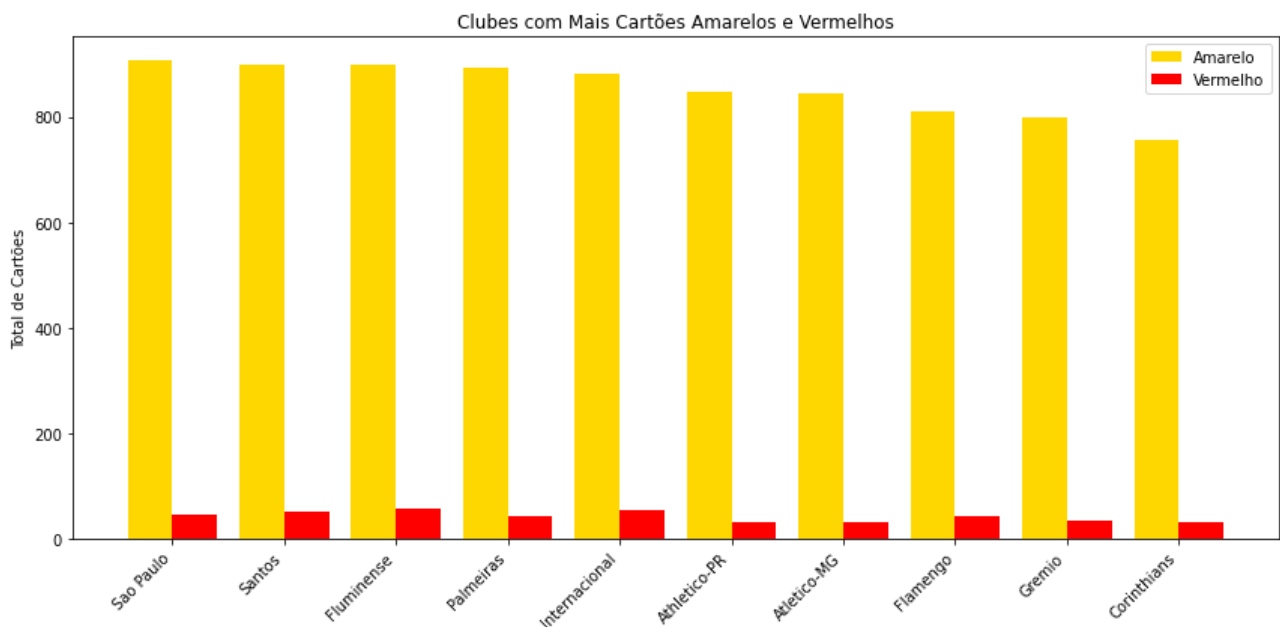
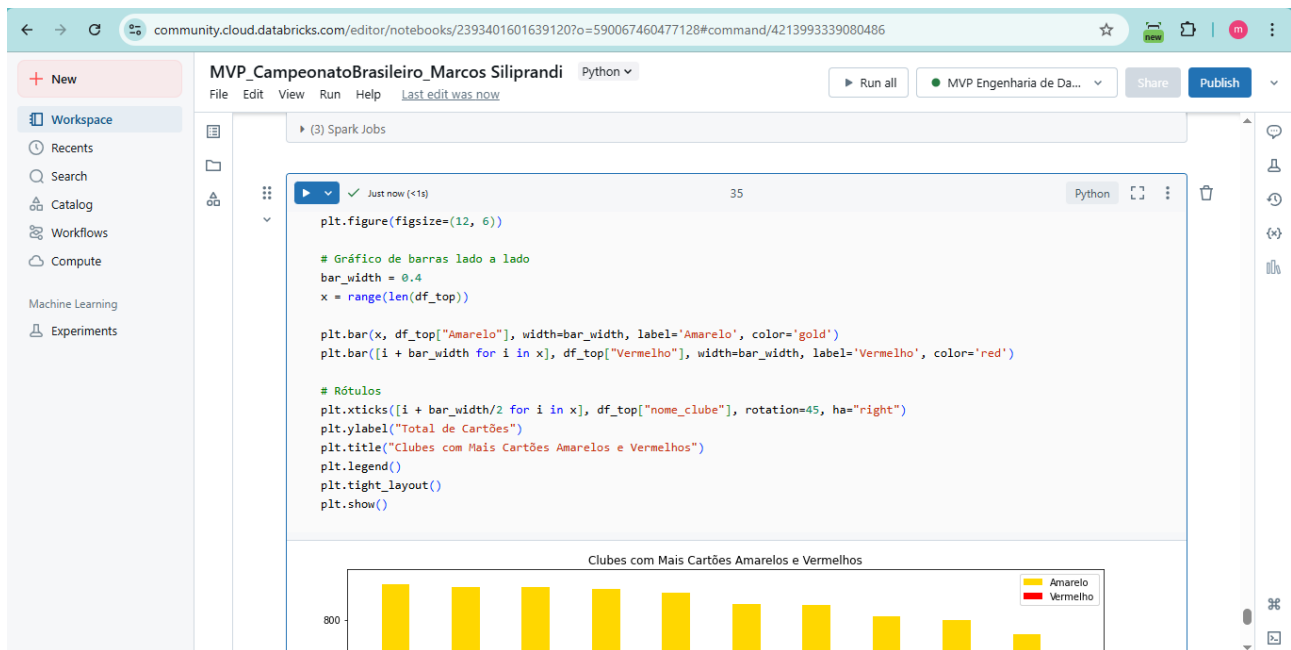
```
# Converte para Pandas
pdf = df_cartoes.toPandas()

# Pivot para deixar amarelo e vermelho em colunas separadas
df_pivot = pdf.pivot(index="nome_clube", columns="tipo_cartao", values="total_cartoes").fillna(0)

# Renomeia colunas se necessário
df_pivot.columns.name = None
df_pivot = df_pivot.reset_index()

# Ordena pelos amarelos (ou soma dos dois, se preferir)
df_top = df_pivot.sort_values(by="Amarelo", ascending=False).head(10)
```

(3) Spark Jobs



Nessa análise, levantei os clubes que mais receberam cartões amarelos e vermelhos durante o campeonato. O São Paulo lidera com 906 cartões amarelos, seguido bem de perto por Fluminense e Santos, todos com mais de 890 amarelos.

Já nos cartões vermelhos, o Fluminense também aparece no topo com 58 expulsões, seguido por Internacional (54) e Santos (52).

De forma geral, os clubes com mais participações também estão entre os que mais receberam cartões, o que é esperado. Mas o destaque vai pro Fluminense, que aparece no topo tanto nos amarelos quanto nos vermelhos, indicando um perfil disciplinar mais agressivo ou com mais faltas táticas ao longo do tempo.

- Quais jogadores mais marcaram gols?

community.cloud.databricks.com/editor/notebooks/2393401601639120?o=590067460477128#command/4213993339080487

MVP_CampeonatoBrasileiro_Marcos Siliprandi Python

File Edit View Run Help Last edit was now

Run all MVP Engenharia de Da... Share Publish

Workspace

Recents

Search

Catalog

Workflows

Compute

Machine Learning

Experiments

```
df_artilheiros = spark.sql("""
SELECT
  a.nome_atleta,
  COUNT(*) AS total_gols
FROM f_gols g
JOIN dim_atleta a
  ON g.ID_Atleta = a.ID_Atleta
GROUP BY a.nome_atleta
ORDER BY total_gols DESC
""")
```

df_artilheiros: pyspark.sql.dataframe.DataFrame = [nome_atleta: string, total_gols: long]

[Shift+Enter] to run and move to next cell
[Ctrl+Shift+P] to open the command palette
[Esc H] to see all keyboard shortcuts

community.cloud.databricks.com/editor/notebooks/2393401601639120?o=590067460477128#command/4213993339080489

MVP_CampeonatoBrasileiro_Marcos Siliprandi Python

File Edit View Run Help Last edit was now

Run all MVP Engenharia de Da... Share Publish

Workspace

Recents

Search

Catalog

Workflows

Compute

Machine Learning

Experiments

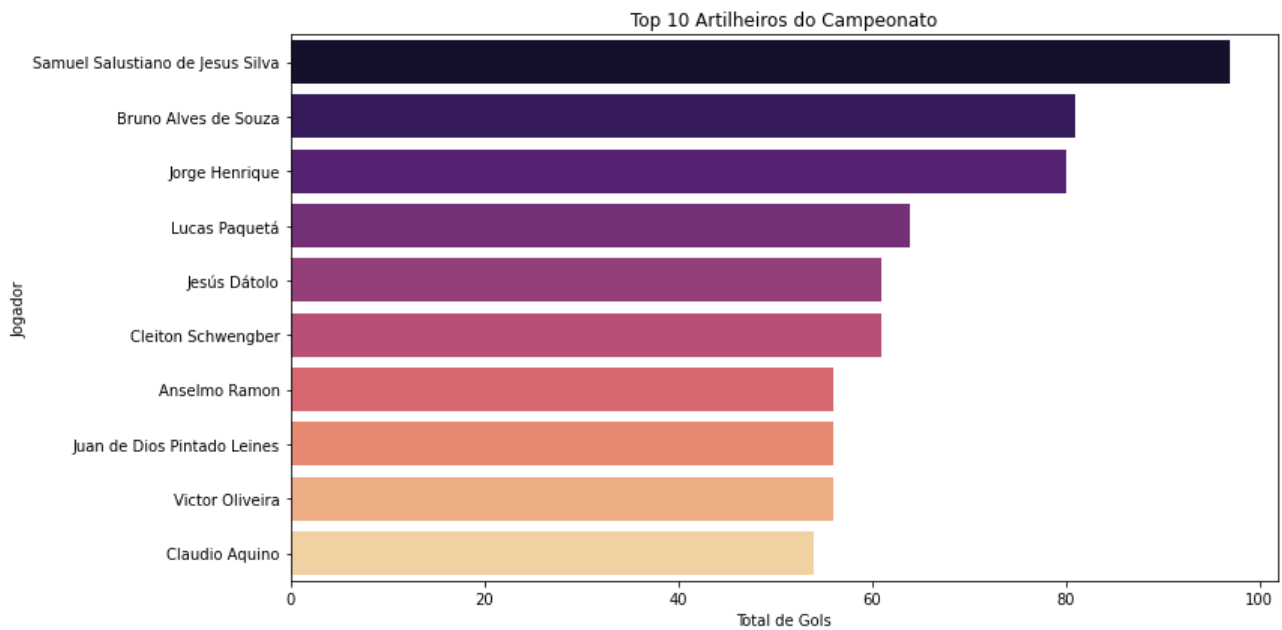
```
# Converte para Pandas e pega os top 10
top_artilheiros = df_artilheiros.toPandas().head(10)

# Gráfico em Python - Top 10 artilheiros
plt.figure(figsize=(12, 6))
sns.barplot(data=top_artilheiros, x="total_gols", y="nome_atleta", palette="magma")
plt.title("Top 10 Artilheiros do Campeonato")
plt.xlabel("Total de Gols")
plt.ylabel("Jogador")
plt.tight_layout()
plt.show()
```

(7) Spark Jobs

Top 10 Artilheiros do Campeonato

Jogador	Total de Gols
Samuel Salustiano de Jesus Silva	12
Bruno Alves de Souza	11
Jorge Henrique	10
Lucas Paquetá	9
Jesús Dátolo	8



Nesta análise, levantei os jogadores que mais marcaram gols no campeonato. O ranking foi feito com base na quantidade total de gols por atleta, considerando todos os registros da base.

O gráfico mostra claramente quem são os principais artilheiros, e pode ser usado como base pra entender o impacto individual dos jogadores no desempenho ofensivo dos clubes.

Esse tipo de análise é útil não só pra identificar os destaques, mas também pra avaliar se os gols estão concentrados em poucos nomes ou bem distribuídos entre os atletas.

- Quais intervalos de tempo mais ocorrem gols?

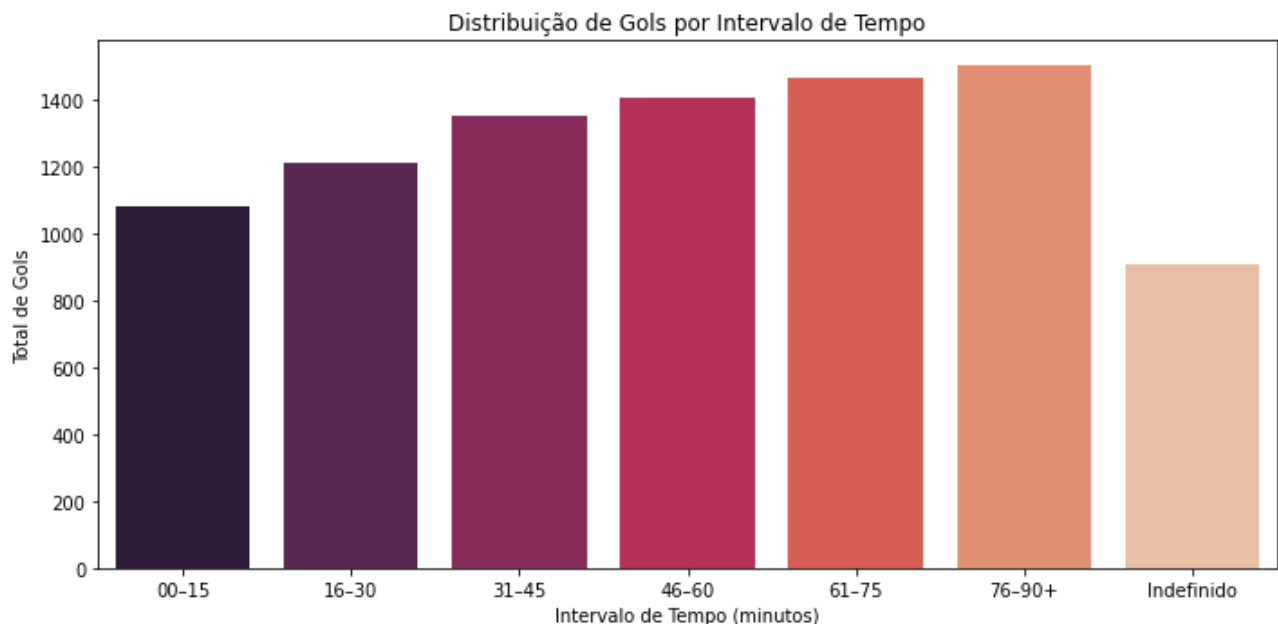
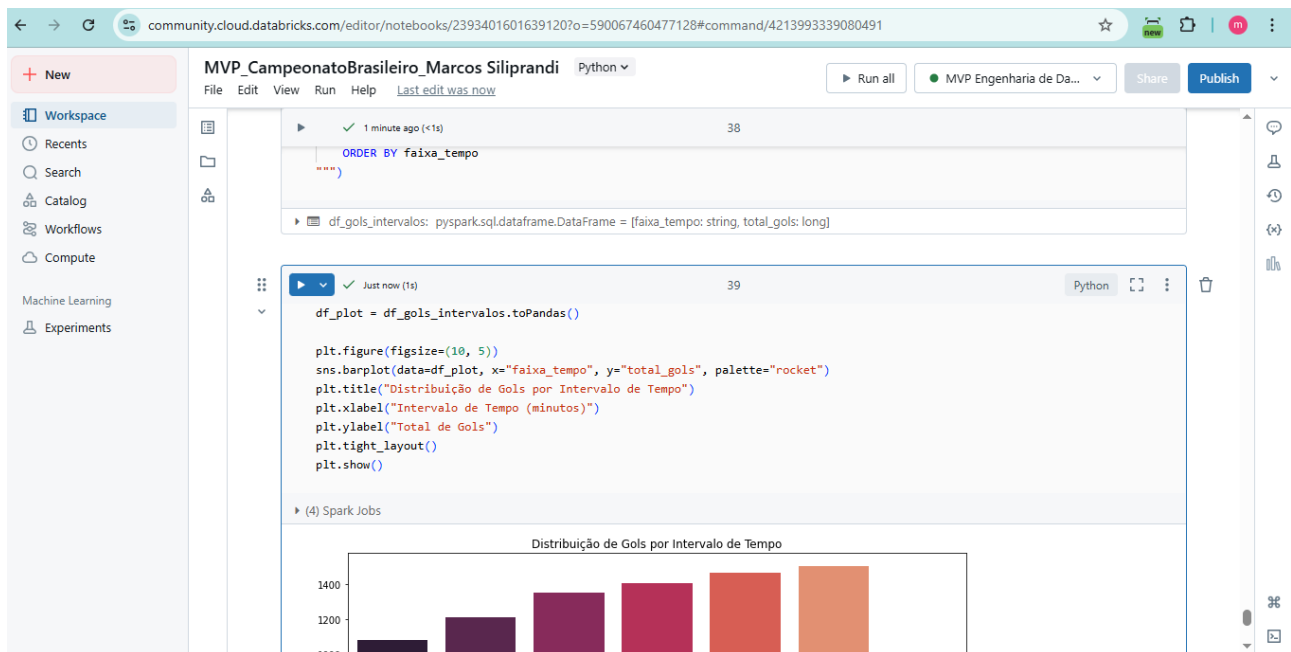
```
community.cloud.databricks.com/editor/notebooks/2393401601639120?o=590067460477128#command/4213993339080490
```

MVP_CampeonatoBrasileiro_Marcos Siliprandi Python

```
df_gols_intervalos = spark.sql("""
SELECT
CASE
WHEN Minuto_marcado BETWEEN 0 AND 15 THEN '00-15'
WHEN Minuto_marcado BETWEEN 16 AND 30 THEN '16-30'
WHEN Minuto_marcado BETWEEN 31 AND 45 THEN '31-45'
WHEN Minuto_marcado BETWEEN 46 AND 60 THEN '46-60'
WHEN Minuto_marcado BETWEEN 61 AND 75 THEN '61-75'
WHEN Minuto_marcado >= 76 THEN '76-90+'
ELSE 'Indefinido'
END AS faixa_tempo,
COUNT(*) AS total_gols
FROM f_gols
WHERE Minuto_marcado IS NOT NULL
GROUP BY faixa_tempo
ORDER BY faixa_tempo
""")

df_gols_intervalos: pyspark.sql.dataframe.DataFrame = [faixa_tempo: string, total_gols: long]
```

[Shift+Enter] to run and move to next cell
[ctrl+Shift+P] to open the command palette
[Esc H] to see all keyboard shortcuts



Dividi os gols em faixas de tempo de 15 em 15 minutos para entender melhor em quais momentos das partidas eles costumam acontecer.

A maioria dos gols ocorre na **reta final do jogo**, entre os **76 e 90+ minutos**, com mais de 1.500 gols registrados nesse intervalo. Isso faz sentido, já que é quando os times geralmente se lançam mais ao ataque ou estão tentando virar ou segurar o placar.

Também percebi um aumento gradual ao longo do jogo, os gols vão aumentando conforme o tempo passa, principalmente do meio pro fim.

A faixa "Indefinido" representa registros que vieram sem o minuto exato, o que pode ser comum em dados mais antigos ou incompletos.

- Como o fator "mandante" influencia nos resultados das partidas?

community.cloud.databricks.com/editor/notebooks/2393401601639120?o=590067460477128#command/4213993339080493

+ New

Workspace

Recents

Search

Catalog

Workflows

Compute

Machine Learning

Experiments

MVP_CampeonatoBrasileiro_Marcos Siliprandi Python

File Edit View Run Help Last edit was now

Run all MVP Engenharia de Da... Share Publish

```
df_mandante = spark.sql("""
SELECT
CASE
  WHEN ID_Clube_Vencedor = ID_Clube_Mandante THEN 'Mandante'
  WHEN ID_Clube_Vencedor = ID_Clube_Visitante THEN 'Visitante'
  ELSE 'Empate'
END AS resultado,
COUNT(*) AS total
FROM f_partidas
GROUP BY resultado
""")
```

df_mandante: pyspark.sql.dataframe.DataFrame = [resultado: string, total: long]

[Shift+Enter] to run and move to next cell
[Ctrl+Shift+P] to open the command palette
[Esc H] to see all keyboard shortcuts

community.cloud.databricks.com/editor/notebooks/2393401601639120?o=590067460477128#command/4213993339080494

+ New

Workspace

Recents

Search

Catalog

Workflows

Compute

Machine Learning

Experiments

MVP_CampeonatoBrasileiro_Marcos Siliprandi Python

File Edit View Run Help Last edit was now

Run all MVP Engenharia de Da... Share Publish

```
df_mandante = spark.sql("""
SELECT
CASE
  WHEN ID_Clube_Vencedor = ID_Clube_Mandante THEN 'Mandante'
  WHEN ID_Clube_Vencedor = ID_Clube_Visitante THEN 'Visitante'
  ELSE 'Empate'
END AS resultado,
COUNT(*) AS total
FROM f_partidas
GROUP BY resultado
""")
```

df_mandante: pyspark.sql.dataframe.DataFrame = [resultado: string, total: long]

```
df_plot = df_mandante.toPandas()

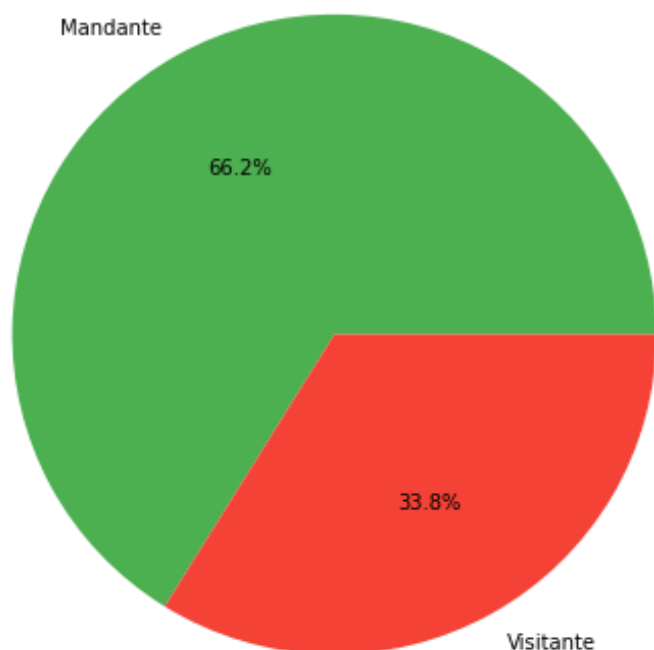
plt.figure(figsize=(6, 6))
plt.pie(df_plot["total"], labels=df_plot["resultado"], autopct='%1.1f%%', colors=["#4CAF50", "#F44336", "#FFC107"])
plt.title("Distribuição de Resultados - Mandante x Visitante")
plt.tight_layout()
plt.show()
```

(3) Spark Jobs

Distribuição de Resultados - Mandante x Visitante

Mandante 66.2%

Distribuição de Resultados - Mandante x Visitante



Olhando para os dados, dá pra ver que jogar em casa realmente faz diferença. Os times mandantes venceram 1839 partidas, contra 940 vitórias dos visitantes.

Isso mostra uma vantagem bem clara pra quem joga em casa com o apoio da torcida, familiaridade com o campo ou até cansaço do time visitante.

O curioso é que não houve empates registrados na tabela `f_partidas` após a transformação, provavelmente porque a coluna de vencedor foi convertida em ID e os empates acabaram ficando como nulos. Isso é algo que vale ficar atento se for necessário um dado mais completo.

Autoavaliação

Durante esse projeto consegui entender muito mais sobre como montar uma pipeline de dados do início ao fim. Achei que seria só jogar os arquivos e pronto, mas na real tem bastante detalhe, principalmente na parte de transformar e limpar os dados.

Usar o Databricks com Spark foi novidade pra mim, mas aos poucos fui pegando o jeito. Vi na prática como organizar as tabelas em fatos e dimensões ajuda na análise.

O mais legal foi ver que, depois de todo o trabalho, consegui gerar insights e responder as perguntas que tinha definido no começo. Isso mostrou que tô conseguindo aplicar o que venho aprendendo de verdade.