



ANÁLISE DESCRIPTIVA DAS VITÓRIAS NAS MARATONAS DE LONDRES

README.txt

Ao final deste MVP, esperamos que o usuário final tenha uma compreensão detalhada dos dados das vitórias nas Maratonas de Londres, respondendo às perguntas propostas e fornecendo insights sobre os aspectos relevantes do evento, como recordes, domínio de países e atletas, e a evolução das diferentes categorias ao longo dos anos.

Neste trabalho, eu documentei **o processo para construir um data lakehouse de dados no Databricks Community** utilizando tecnologias na nuvem como **Microsoft Azure**.

O Trabalho de conclusão do curso irá envolver a construção de um pipeline (Manual) para a busca, coleta, e modelagem, para os processos ELT de extração , carga e transformação para a análise Ad-Hoc com os dados escolhidos.

Apresentação do Trabalho

Documentação Organizada

Foram usados notebooks tanto no *Google Colab* quanto no *Databricks Community* para documentar o trabalho com vias de conseguir comunicar a implementação do MVP. Onde utilizei células para escrever linhas de código e seus display. Isso também incluiu a descrição do problema, as perguntas de negócio e a justificativa do projeto em um documento para a leitura em formato pdf. Isso ajudará o leitor a manter o foco na documentação organizada e acessível somente num documento pdf.

Ao seguir essa abordagem, cada parte do processo, desde a definição dos objetivos até a análise final, será bem documentada.

Resumo

Os estágios do trabalho detalhados no relatório abaixo são totalmente suportados pelas funcionalidades do workspace no *Databricks Community Edition*, pelos recursos do *Azure Blob Storage Account* e do *APP Register*, e pelo uso do *Delta Lake*. Além disso, utilizarei alguns comandos básicos em *Python* e algumas livrarias como *Request*, *BeautifulSoup*, e usei *PySpark*, *Apache SQL* e outras bibliotecas integradas no Rol Persona Analytic *Databricks*.

A coleta e a persistência dos conjuntos de dados na plataforma de nuvem podem ser gerenciadas através das funcionalidades: *Delta Lake* e *Databricks File System (DBFS)**

Utilizarei o *Delta Lake* para a ingestão de dados. O *Delta Lake* oferece suporte robusto para transações ACID, permitindo uma ingestão de dados confiável e escalável, além da movimentação posterior no modelo Medallion Camadas Bronze e Prata.

**Databricks File System (DBFS)*: Também é possível utilizar o DBFS para armazenar os arquivos brutos coletados. Pode-se configurar o armazenamento em nuvem, como o *Azure Blob Storage*, para integrar diretamente com o DBFS.

1.- **Objetivo SMART:** Análise Descritiva dos Dados da Maratona de Londres

Realizar uma análise descritiva dos dados da Maratona de Londres, utilizando SQL ou PySpark nas camadas prata e ouro, e criar um relatório Ad-Hoc no Databricks ou PowerBI para visualizar os resultados.

Objetivo SMART Detalhado

Specific (Específico):

Responder às seguintes perguntas:

1. Qual país ganhou o maior número de competições?
2. Em que ano os atletas britânicos venceram a maratona pela última vez?
3. Qual foi o ano da primeira corrida em cadeira de rodas?
4. Qual maratonista masculino estabeleceu um novo recorde e dominou a competição?
5. Quem detém o recorde mundial feminino e em que ano foi estabelecido?
6. Quais corredores tiveram múltiplas vitórias na competição?

Measurable (Mensurável) :*

Relatório Ad-Hoc que contém o desenvolvimento dos ítems ponderados para a avaliação do MVP III Engenheira de dados , como as respostas às perguntas de análise em Spark SQL e PySpark. e documentações ScreenShoots: Azure, Databricks e Colab (Crawler Scraper).

*Ponderações do Check List para o MVP III Engenheira de dados:

Objetivo: Será avaliada a qualidade da descrição de um objetivo e planejamento bem detalhado do trabalho a realizar: Contendo de forma clara e objetiva o problema a ser resolvido e as perguntas de negócios a serem respondidas.

Coleta : Será avaliada a documentação sobre a coleta dos conjuntos de dados e a persistência dos mesmos na plataforma de nuvem.

Modelagem : Será avaliada a qualidade da modelagem dos dados e documentação do Catálogo de Dados.

Carga : Será avaliada a qualidade da documentação da carga dos dados, bem como a corretude e persistência dos dados na plataforma de nuvem após a carga.

Análise : Serão avaliados a análise de qualidade dos dados e da solução do problema de forma correta e bem analisada pela discussão a partir das respostas obtidas.

Achievable (Atingível):

Dados públicos da Maratona de Londres estão disponíveis em uma url * de wikipedia e as plataformas como Databricks, SQL PySpark e PowerBI estão ao nosso alcance em versões Free Trial , Star Free, Community e On Demand e Premium.

Relevant (Relevante):

O projeto aplica conceitos variados da disciplina e mercado atual da engenharia e governança de dados e análise de dados , fornecendo insights sobre a Maratona de Londres.

Time-bound (Com Prazo):

O projeto será realizado de 6 de junho a 12 de julho, seguindo este cronograma:

CRONOGRAMA MVP ENGENHEIRA DE DADOS_SPRINT III	COMIENZO	FIN	DURACIÓN	junio-2024				Julio-2024		
				09	16	23	30	07	14	
1 Kick-Off MVP Sprint III	07-06-24	07-06-24	0	●	Kick-Off MVP Sprint III					
2 Definição do Escopo e critérios de Avaliação	08-06-24	08-06-24	0	◆	Definição do Escopo e critérios de Avaliação					
3 Definição de Objetivo e Plano de Trabalho (Levantamento dos Requisitos)	09-06-24	10-06-24	1	■	Definição de Objetivo e Plano de Trabalho					
4 Coleta do conjunto de dados e documentação	11-06-24	18-06-24	6	■	Coleta do conjunto de dados e					
5 Sessão de Dubidas 01 do MVP	19-06-24	19-06-24	0	■	Sessão de Dubidas 01 do MV					
6 Modelagem dos Dados e documentação do catálogo de dados	20-06-24	26-06-24	5	■	Modelagem dos Dados					
7 Sessão de Dubidas 02 do MVP	26-06-24	26-06-24	0	■	Sessão de Dubidas 02 do MV					
8 Carga de dados na plataforma de nuvem	27-06-24	03-07-24	5	■	Carga de da					
9 Sessão de Dubidas 03 do MVP	03-07-24	03-07-24	0	■	Sessão de Dubidas 03 do MV					
10 Análise de qualidade dos dados e da solução do problema	04-07-24	10-07-24	5	■	Análise					
11 Sessão de Dubidas 04 do MVP	10-07-24	10-07-24	0	■	Sessão de Dubidas 04 do MV					
12 Autoavaliação sobre o atingimento dos objetivos	11-07-24	12-07-24	2	■						
13 Entrega MVP Sprint III	12-07-24	12-07-24	0	■	Entrega MVP Sprint III					

Estes objetivos SMART fornecem um plano claro para viabilizar uma análise descritiva dos dados da Maratona de Londres, culminando na criação de um relatório Ad-Hoc em formato de um roteiro informativo (storytelling) de Antônio um comentarista especializado reconhecido em maratonas de 42 k.

2.- Coleta: Avaliamos a necessidade de realizar um planejamento reverso da coleta dos dados com foco no objetivo e perguntas previamente formuladas para viabilizar a análise Ad-Hoc.

- Identificar fontes de dados relevantes.
- Origem dos dados: https://en.wikipedia.org/wiki/List_of_winners_of_the_London_Marathon#
- Fazer requisições HTTP para obter conteúdo web.
- Analisar HTML e extrair dados relevantes.
- Abstrair a organização e estruturação dos dados coletados com vias a viabilizar a arquitetura de data Lakehouse.
- Armazenar os dados extraídos em um csv.

Uma vez definido o conjunto de dados, antes devemos coletar para depois armazená-los no object store ou na nuvem escolhida Microsoft Azure Blob Storage, versão Free Trial.

This is a summary of the Terms of Use. To read the full terms, scroll down or click here.

This is a human-readable summary of the Terms of Use.

Disclaimer: This summary is not a part of the Terms of Use and is not a legal document. It is simply a handy reference for understanding the full terms. Think of it as the user-friendly interface to the legal language of our Terms of Use.

Part of our mission is to:

- Empower and Engage people around the world to collect and develop educational content and either publish it under a free license or dedicate it to the public domain.
- Disseminate this content effectively and globally, free of charge.
- Offer websites and technical infrastructure to help you do this.

You are free to:

- Read and Print our articles and other media free of charge.
- Share and Reuse our articles and other media under free and open licenses.
- Contribute To and Edit our various websites or Projects.

Under the following conditions:

- Responsibility — You take responsibility for your edits (since we only host your content).
- Civility — You support a civil environment and do not harass other users.
- Lawful Behavior — You do not violate copyright, post illegal content, or violate other applicable laws that follow human rights principles.

Nota: A partir da escolha do conjunto de dados, foi necessário incluir uma etapa de construção de robôs de coleta, via Web Scraping.

Como neste caso, atente-se que para questões éticas foram revisados os termos de uso do site Wikipedia especificamente nas ferramentas para desenvolvedores na url , foi pesquisado as permissões sobre se é de fato é possível utilizar os robôs de coleta de informação no sites escolhido.

Web Scraping permissões: Termos de uso da Wikimedia Foundation

https://foundation.wikimedia.org/wiki/Policy:Terms_of_Use

Mediante a verificação das permissões do site 'https://en.wikipedia.org/wiki/List_of_winners_of_the_London_Marathon' usando o arquivo robots.txt. Em síntese:

Verifiquei o arquivo robots.txt do site 'https://en.wikipedia.org/wiki/List_of_winners_of_the_London_Marathon'. O arquivo robots.txt permite o acesso de robôs de pesquisa a todo o conteúdo do site. Não há restrições ou limitações no arquivo

robots.txt que impeçam o acesso ou a indexação do site.



Nota: Nesta etapa foram realizadas várias tentativas de coleta desde um robot scraping ate uns quatro csv's (disponibilizadas ao final deste documento). flat file com dados semi estruturados e com o intuito a consolidar em um modelo de dados tipo tabela única OBT , transformação a realizar na camada prata no Databricks.

Explorando o layout e estrutura do site: https://en.wikipedia.org/wiki/List_of_winners_of_the_London_Marathon

Inspecionei visualmente a URL usando o navegador Chrome , afim de explorar o layout e de me familiarizar com a estrutura da página. Decodifiquei a estrutura das urls , como as informações ocultas nos parâmetros da url de interesse para depois inspecionar o site usando as ferramentas de desenvolvedor disponíveis no browser.

Para explorar o site, recomenda-se interagir e navegar pelo mesmo, obtendo uma visão geral da URL base (através da interface visual). É importante focar em realçar os nomes das “Category Races”, os quais se encontram localizados no canto superior esquerdo da tela. Da mesma forma, deve-se prestar atenção às informações semi-estruturadas relacionadas nestas, situadas na parte central da tela, à medida que se navega pelo site deslocando-se para baixo em cada uma das categorias corridas.

Breve explicação dos parâmetro presente na url:

- `https://`: Protocolo de comunicação, neste caso, o protocolo seguro HTTP.
- `en.wikipedia.org`: Nome de domínio do site da Wikipédia, na versão em inglês.
- `wiki/`: Indica que a página faz parte da seção "wiki" do site.
- `List_of_winners_of_the_London_Marathon`: Título da página, que descreve o conteúdo e basicamente a interface visual do site.

Year	Athlete	Nationality	Time (hours)	Notes
1981	Dick Beardsley (Tie)	United States	2:11:48	Course record
	Inge Simonsen (Tie)	Norway		
1982	Hugh Jones	United Kingdom	2:09:24	Course record
1983	Mike Gratton	United Kingdom	2:09:43	
1984	Charlie Spedding	United Kingdom	2:09:57	

Inspeção das URLs do site:

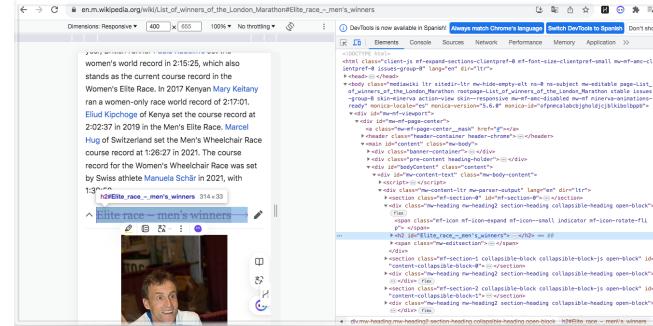
Em uma primeira camada, é possível decifrar a URL

"https://en.m.wikipedia.org/wikilist_of_winners_of_the_London_Marathon#Elite_race_%E2%80%93_men's_winners" clicando no primeiro elemento "races categories" no canto superior esquerdo do site (UI). Ao mesmo tempo, os parâmetros relacionados (tags), como "h2 id", podem ser interessantes, scrolling na ferramenta de desenvolvedor e inspecionando a URL em profundidade ("under the hood") através da estrutura HTML da página.

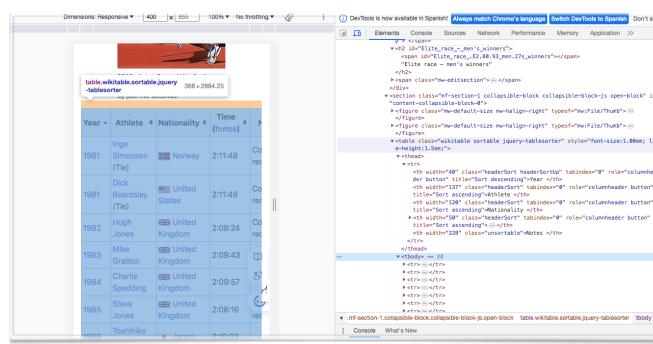
Além disso, ao voltar para a interface visual, também são exibidas tabelas em HTML com informações estruturadas (tags: "table class") relacionadas, as quais podem ser navegadas através da estrutura HTML da página usando as mesmas ferramentas de desenvolvedor. Essas tabelas resumem o histórico de desempenho da categoria de corrida "Race category : Elite race – men's winners", aninhada dentro da classe HTML correspondente.

Em resumo, ao clicar na URL relacionada

"https://en.m.wikipedia.org/wikilist_of_winners_of_the_London_Marathon#Elite_race_%E2%80%93_men's_winners", o elemento da URL "#Elite_race_%E2%80%93_men's_winners" aponta para a seção específica sobre "os vencedores masculinos da corrida de elite". Essa URL direciona o usuário diretamente para a seção da página que enumera "the race category elite: men's-winners from the List of winners of the London Marathon", que contém todo o tipo de informação relevante para a análise descritiva sinalizada previamente no objetivo do MVP da sprint Engenharia de dados.



Inspeção visual da URL usando as ferramentas de desenvolvedor scrolling na estrutura da página conforme mencionado acima



Nota: O site da url Wikipedia utiliza uma combinação de linguagem de programação. Ele é predominantemente desenvolvido utilizando HTML para estruturação, CSS para estilização e JavaScript para interatividade. Além disso, o conteúdo é dinamicamente gerado usando MediaWiki, que é uma plataforma de software wiki utilizada pela Wikipedia.

As tags HTML mais comuns para tabelas aqui Inspecione o URL usando as ferramentas de desenvolvedor scrolling na estrutura da página conforme mencionado acima são:

tags úteis para tabelas incluem:

```
<table>: define a tabela
<tr>: define uma linha na tabela
<th>: define um cabeçalho de célula
<td>: define uma célula de dados
```

Algumas outras tags úteis para tabelas incluem:

```
<thead>: define o cabeçalho da tabela
<tbody>: define o corpo da tabela
<tfoot>: define o rodapé da tabela
<caption>: define o título da tabela
```

Essas tags permitem estruturar e organizar os dados em linhas e colunas, formando uma tabela na página web. A combinação dessas tags HTML é a forma mais comum de criar tabelas em páginas da web.

Neste site os elementos como as tabelas, em particular, parecem ter sido criadas usando tags HTML como `<table>`, `<tr>` e `<td>`. Essa é uma forma comum de estruturar informações tabulares em páginas da web.

```
[ ] # Importar bibliotecas específicas para scraping e manipulação de dados
import requests
from bs4 import BeautifulSoup
import pandas as pd

def load_page(url):
    """
    Carrega a página HTML e retorna o objeto BeautifulSoup.

    Args:
        url (str): URL da página a ser carregada.

    Returns:
        BeautifulSoup: Objeto BeautifulSoup da página carregada.
    """
    response = requests.get(url)
    return BeautifulSoup(response.content, 'html.parser')

def extract_tables(soup):
    """
    Extrai tabelas HTML e suas respectivas categorias de corrida.

    Args:
        soup (BeautifulSoup): Objeto BeautifulSoup da página carregada.

    Returns:
        list: Lista de dicionários com tabelas e categorias de corrida.
    """
    return [
        {
            'table': table,
            'race_category': table.find_previous('h2').text.strip().replace(' winners', '').replace(' ', '_')
        }
        for table in soup.find_all('table', class_='wikitable sortable jquery-tablesorter')[1:4] # Limitando as quatro primeiras tabelas conforme solicitado
    ]

def table_to_dataframe(table, category):
    """
    Converte uma tabela HTML em um DataFrame do pandas e adiciona a coluna 'race category'.

    Args:
        table (Tag): Tabela HTML a ser convertida.
        category (str): Categoria de corrida a ser adicionada.

    Returns:
        DataFrame: DataFrame com a tabela convertida.
    """
    df = pd.read_html(str(table))[0]
    df['race_category'] = category
    return df

def save_dataframe_to_csv(df, filename):
    """
    Salva um DataFrame em um arquivo CSV.

    Args:
        df (DataFrame): DataFrame a ser salvo.
        filename (str): Nome do arquivo CSV.
    """
    df.to_csv(filename, index=False)

def main():
    # URL da página a ser carregada
    url = 'https://en.wikipedia.org/wiki/List_of_winners_of_the_London_Marathon'

    # Carregar a página e extrair tabelas
    soup = load_page(url)
    tables = extract_tables(soup)

    # Iterar sobre as tabelas extraídas e processar cada uma
    for table_info in tables:
        category = table_info['race_category']
        table = table_info['table']

        # Converter a tabela para DataFrame
        df = table_to_dataframe(table, category)

        # Salvar o DataFrame como CSV
        filename = f'{category}.csv'
        save_dataframe_to_csv(df, filename)

        # Imprimir a tabela no Colab
        print(f'Race Category: {category}')
        display(df.head())
        print()

if __name__ == "__main__":
    main()
```

Coleta dos dados:

Para efetuar essa coleta de dados, desenvolveu-se um script para a criação de um scraper em Python. Este script tem a responsabilidade de extrair as informações contidas nas url e nos registros das tabelas, e em seguida, armazenar nas respectivas tabelas previamente configuradas pelo administrador do banco de dados.

Implementação robot Scraper (Crawler):

Explicação do Código:

Função *load_page*:

Carrega a página HTML usando requests e retorna um objeto *BeautifulSoup*.

response = requests.get(url): Faz uma requisição GET para a URL fornecida.

return BeautifulSoup(response.content, 'html.parser'): Analisa o HTML da resposta e retorna um objeto *BeautifulSoup*.

Função *extract_tables*:

Extrai todas as tabelas HTML com a classe 'wikitable sortable jquery-tablesorter' e suas categorias de corrida.

soup.find_all('table', class_='wikitable sortable jquery-tablesorter'): Encontra todas as tabelas HTML com a classe especificada.

table.find_previous('h2').text.strip().replace(' winners', '').replace(' ', '_'): Encontra o cabeçalho anterior à tabela e formata o texto para ser usado como categoria.

Função *table_to_dataframe*:

Converte uma tabela HTML em um DataFrame do pandas e adiciona uma coluna com a categoria da corrida.

pd.read_html(str(table))[0]: Converte a tabela HTML em um DataFrame.

df['race_category'] = category: Adiciona uma nova coluna com a categoria da corrida.

Função *save_dataframe_to_csv*:

Salva um DataFrame em um arquivo CSV.

df.to_csv(filename, index=False): Salva o DataFrame no arquivo CSV especificado sem incluir os índices.

Função *main*:

Orquestra as outras funções para carregar a página, extrair e processar as tabelas, e salvar os DataFrames como arquivos CSV. Itera sobre as tabelas extraídas, converte cada uma para DataFrame, salva como CSV e imprime no Colab.

Coleta dos dados para este trabalho:

Para a realização deste trabalho, empregaram-se os dados extraídos e armazenados nas quatro tabelas previamente mencionadas. Contudo, esses dados foram exportados no formato CSV para os arquivos; Elite_race_-_mes's[edit].csv, Elite_race_-_women's[edit].csv, Wheelchair_race_-_men's[edit].csv e Wheelchair_race_-_women's[edit].csv (figuras), para realizar a ingestão deles mediante o uso das pastas geridas na landing zone e no Data Lake criado para esse propósito Microsoft Azure Blob Storage e no DBS no Databricks.

Year,Athlete,Nationality,Time (h:m:s),Notes,race_category
1983,Kevin Green,United Kingdom,2:26:07,Course record,Wheelchair_race_-_men's[edit]
1984,Chris Hallam,United Kingdom,2:18:49,Course record,Wheelchair_race_-_men's[edit]
1985,Chris Hallam,United Kingdom,2:19:53,Course record,Wheelchair_race_-_men's[edit]
1986,Chris Hallam,United Kingdom,2:18:28,Course record,Wheelchair_race_-_men's[edit]
1987,Chris Hallam,United Kingdom,2:19:34,Course record; Second victory,Wheelchair_race_-_men's[edit]
1988,Ted Vince,United Kingdom,2:01:23,Course record,Wheelchair_race_-_men's[edit]
1989,David Holding,United Kingdom,1:59:31,Course record,Wheelchair_race_-_men's[edit]
1990,David Holding,United Kingdom,1:59:31,Course record,Wheelchair_race_-_men's[edit]
1991,Farid Amraouche,France,1:52:24,Course record,Wheelchair_race_-_men's[edit]
1992,Daniel Wesley,Canada,1:51:42,Course record,Wheelchair_race_-_men's[edit]
1993,George Vandanne,Belgium,1:44:10,Course record,Wheelchair_race_-_men's[edit]
1994,George Vandanne,Belgium,1:44:10,Course record,Wheelchair_race_-_men's[edit]
1995,Heinz Frei,Switzerland,1:39:14,Course record,Wheelchair_race_-_men's[edit]
1996,David Holding,United Kingdom,1:43:48,Third victory,Wheelchair_race_-_men's[edit]
1997,David Holding,United Kingdom,1:42:15,Fourth victory,Wheelchair_race_-_men's[edit]
1998,David Holding,United Kingdom,1:42:15,Fourth victory,Wheelchair_race_-_men's[edit]
1999,Heinz Frei,Switzerland,1:35:27,Third victory,Wheelchair_race_-_men's[edit]
2000,Kevin Papworth,United Kingdom,1:41:50,Wheelchair_race_-_men's[edit]
2001,David Weir,United Kingdom,1:32:26,Sixth victory,Wheelchair_race_-_men's[edit]
2002,David Weir,United Kingdom,1:32:26,Course record,Wheelchair_race_-_men's[edit]
2003,Joël Jeannot,France,1:32:02,Course record,Wheelchair_race_-_men's[edit]
2004,Saul Mendez,Mexico,1:36:56,Wheelchair_race_-_men's[edit]
2005,David Weir,United Kingdom,1:35:51,Second victory,Wheelchair_race_-_men's[edit]
2006,David Weir,United Kingdom,1:35:51,Course record; second victory,Wheelchair_race_-_men's[edit]
2007,David Weir,United Kingdom,1:35:51,Third victory,Wheelchair_race_-_men's[edit]
2008,David Weir,United Kingdom,1:33:05,Fourth victory,Wheelchair_race_-_men's[edit]
2009,David Weir,United Kingdom,1:33:05,Fourth victory,Wheelchair_race_-_men's[edit]
2010,Josh Cassidy,Canada,1:35:21,Wheelchair_race_-_men's[edit]
2011,David Weir,United Kingdom,1:30:05,Fifth victory,Wheelchair_race_-_men's[edit]
2012,David Weir,United Kingdom,1:49:10,Sixth victory,Wheelchair_race_-_men's[edit]
2013,David Weir,United Kingdom,1:32:26,Sixth victory,Wheelchair_race_-_men's[edit]
2014,Marcel Hug,Switzerland,1:32:44,Wheelchair_race_-_men's[edit]
2015,Josh George,United States,1:31:31,Second victory,Wheelchair_race_-_men's[edit]
2016,Marcel Hug,Switzerland,1:31:31,Second victory,Wheelchair_race_-_men's[edit]
2017,Marcel Hug,Switzerland,1:31:31,Second victory,Wheelchair_race_-_men's[edit]
2018,David Weir,United Kingdom,1:31:15,Eighth victory,Wheelchair_race_-_men's[edit]
2019,Daniel Romanchuk,United States,1:31:37,Wheelchair_race_-_men's[edit]
2020,Brett Lakatos,Canada,1:36:04,Wheelchair_race_-_men's[edit]
2021,Brett Lakatos,Canada,1:36:04,Course record; third victory,Wheelchair_race_-_men's[edit]
2022,[9],Marcel Hug,Switzerland,1:24:23,Course record; fourth victory,Wheelchair_race_-_men's[edit]
2023[10],Marcel Hug,Switzerland,1:23:44,Course record; fifth victory,Wheelchair_race_-_men's[edit]

Year,Athlete,Nationality,Time (h:m:s),Notes,race_category
1983,Denise Smith,United Kingdom,2:29:03,Course record,Wheelchair_race_-_women's[edit]
1984,Kay McShane,Ireland,2:18:04,Course record,Wheelchair_race_-_women's[edit]
1985,Kay McShane,Ireland,2:17:12,Course record; second victory,Wheelchair_race_-_women's[edit]
1986,Kay McShane,Ireland,2:17:12,Course record; second victory,Wheelchair_race_-_women's[edit]
1987,Karen Davison,United Kingdom,2:45:30,Course record,Wheelchair_race_-_women's[edit]
1988,Karen Davison,United Kingdom,2:41:45,Course record, second victory,Wheelchair_race_-_women's[edit]
1989,Josie Cichockyj,United Kingdom,3:03:54,Wheelchair_race_-_women's[edit]
1990,Josie Cichockyj,United Kingdom,3:03:54,Wheelchair_race_-_women's[edit]
1991,Connie Hansen,Denmark,2:01:04,Course record; second victory,Wheelchair_race_-_women's[edit]
1992,Tanni Grey-Thompson,United Kingdom,2:17:23,Wheelchair_race_-_women's[edit]
1993,Rose Hill,United Kingdom,2:03:05,Course record,Wheelchair_race_-_women's[edit]
1994,Rose Hill,United Kingdom,2:03:05,Second victory,Wheelchair_race_-_women's[edit]
1995,Rose Hill,United Kingdom,2:17:02,Second victory,Wheelchair_race_-_women's[edit]
1996,Tanni Grey-Thompson,United Kingdom,2:00:10,Course record; third victory,Wheelchair_race_-_women's[edit]
1997,Monica Wettergren,Sweden,1:49:09,Course record,Wheelchair_race_-_women's[edit]
1998,Monica Wettergren,Sweden,1:49:09,Course record,Wheelchair_race_-_women's[edit]
1999,Monica Wettergren,Sweden,1:57:38,Second victory,Wheelchair_race_-_women's[edit]
2000,Sarah Piercy,United Kingdom,2:23:30,Wheelchair_race_-_women's[edit]
2001,Tanni Grey-Thompson,United Kingdom,2:17:23,Second victory,Wheelchair_race_-_women's[edit]
2002,Francesca Porcellato,Italy,2:04:21,Wheelchair_race_-_women's[edit]
2003,Francesca Porcellato,Italy,2:04:21,Second victory,Wheelchair_race_-_women's[edit]
2004,Francesca Porcellato,Italy,2:04:21,Third victory,Wheelchair_race_-_women's[edit]
2005,Francesca Porcellato,Italy,2:07:00,Third victory,Wheelchair_race_-_women's[edit]
2006,Shelly Woods,United Kingdom,1:58:07,Wheelchair_race_-_women's[edit]
2007,Shelly Woods,United Kingdom,1:58:07,Wheelchair_race_-_women's[edit]
2008,Sandra Graf,Switzerland,1:48:04,Course record,Wheelchair_race_-_women's[edit]
2009,Andrea Rozario,Australia,1:42:51,Fourth victory,Wheelchair_race_-_women's[edit]
2010,Andrea Rozario,Australia,1:42:51,Fourth victory,Wheelchair_race_-_women's[edit]
2011,Andrea Rozario,Australia,1:42:51,Fourth victory,Wheelchair_race_-_women's[edit]
2012,Shelly Woods,United Kingdom,1:49:10,Second victory,Wheelchair_race_-_women's[edit]
2013,Tatjana McFadden,United States,1:44:02,Course record; second victory,Wheelchair_race_-_women's[edit]
2014,Tatjana McFadden,United States,1:44:02,Course record; second victory,Wheelchair_race_-_women's[edit]
2015,Tatjana McFadden,United States,1:41:14,Course record; third victory,Wheelchair_race_-_women's[edit]
2016,Tatjana McFadden,United States,1:44:41,Fourth victory,Wheelchair_race_-_women's[edit]
2017,Tatjana McFadden,United States,1:44:41,Fourth victory,Wheelchair_race_-_women's[edit]
2018,Madison de Rozario,Australia,1:42:58,Second victory,Wheelchair_race_-_women's[edit]
2019,Manuela Schär,Switzerland,1:44:09,Second victory,Wheelchair_race_-_women's[edit]
2020,Nikita den Boer,Netherlands,1:40:07,Second victory,Wheelchair_race_-_women's[edit]
2021,Naomi Frith,United Kingdom,2:18:00,Second victory,Wheelchair_race_-_women's[edit]
2022,[9],Catherine Debrunner,Switzerland,1:38:38,Course record,Wheelchair_race_-_women's[edit]
2023[10],Madison de Rozario,Australia,1:38:51,Second victory,Wheelchair_race_-_women's[edit]

Year,Athlete,Nationality,Time (h:m:s),Notes,race_category
1981,Joyce Smith,United Kingdom,2:29:57,Course record,Elite_race_-_men's[edit]
1982,Chris Hallam,United Kingdom,2:18:49,Course record,Elite_race_-_men's[edit]
1983,Grete Waitz,Norway,2:25:29,World marathon record; Elite_race_-_men's[edit]
1984,Ingrid Kristiansen,Norway,2:24:26,Course record,Elite_race_-_men's[edit]
1985,Ingrid Kristiansen,Norway,2:21:08,World marathon record; second victory,Elite_race_-_men's[edit]
1986,Ingrid Kristiansen,Norway,2:21:08,World marathon record; second victory,Elite_race_-_men's[edit]
1987,Ingrid Kristiansen,Norway,2:22:48,Third victory,Elite_race_-_men's[edit]
1988,Ingrid Kristiansen,Norway,2:25:41,Fourth victory,Elite_race_-_men's[edit]
1989,Vera Kudrjavtseva,Russia,2:25:56,Elite_race_-_men's[edit]
1990,Rosie Mota,Poland,2:26:31,Elite_race_-_men's[edit]
1991,Katrin Börck,Germany,2:29:39,Elite_race_-_men's[edit]
1992,Katrin Börck,Germany,2:29:39,Elite_race_-_men's[edit]
1993,Katrin Börck,Germany,2:29:39,Elite_race_-_men's[edit]
1994,Katrin Börck,Germany,2:29:39,Third victory,Elite_race_-_men's[edit]
1995,Małgorzata Sobaska,Poland,2:27:43,Elite_race_-_men's[edit]
1996,Paula Radcliffe,United Kingdom,2:18:56,Course record,Elite_race_-_men's[edit]
1997,Paula Radcliffe,United Kingdom,2:18:56,Course record (mixed); second victory,Elite_race_-_men's[edit]
1998,Irene Mikitenko,Germany,2:26:51,Elite_race_-_men's[edit]
1999,Joyce Chepukaitis,Kenya,2:23:22,Second victory,Elite_race_-_men's[edit]
2000,Joyce Chepukaitis,Kenya,2:23:22,Second victory,Elite_race_-_men's[edit]
2001,Derartu Tulu,Ethiopia,2:23:57,Elite_race_-_men's[edit]
2002,Paula Radcliffe,United Kingdom,2:18:56,Course record,Elite_race_-_men's[edit]
2003,Paula Radcliffe,United Kingdom,2:18:56,Course record (mixed); second victory,Elite_race_-_men's[edit]
2004,Mary Keitany,Kenya,2:27:35,Elite_race_-_men's[edit]
2005,Paula Radcliffe,United Kingdom,2:17:42,World marathon record (women-only); third victory,Elite_race_-_men's[edit]
2006,Paula Radcliffe,United Kingdom,2:17:42,World marathon record (women-only); third victory,Elite_race_-_men's[edit]
2007,Zhou Chunxiu,China,2:20:38,Elite_race_-_men's[edit]
2008,Irina Mikitenko,Germany,2:24:14,Elite_race_-_men's[edit]
2009,Irina Mikitenko,Germany,2:22:54,Second victory,Elite_race_-_men's[edit]
2010,Irina Mikitenko,Germany,2:22:54,Second victory,Elite_race_-_men's[edit]
2011,Marie-Josée Delisle,Canada,2:26:51,Elite_race_-_men's[edit]
2012,Mary Keitany,Kenya,2:19:19,Elite_race_-_men's[edit]
2013,Priscilla Jepchirchir,Kenya,2:28:15,Elite_race_-_men's[edit]
2014,Edna Kiplagat,Kenya,2:20:21,Elite_race_-_men's[edit]
2015,Tuuli Tuomi,Ethiopia,2:23:21,Elite_race_-_men's[edit]
2016,Tuuli Tuomi,Ethiopia,2:23:21,Elite_race_-_men's[edit]
2017,Mary Jepkosgei,Kenya,2:17:01,World marathon record (women-only); third victory,Elite_race_-_men's[edit]
2018,Vivian Cheruiyot,Kenya,2:18:31,Elite_race_-_men's[edit]
2019,Bridget Wambua,Kenya,2:18:31,Elite_race_-_men's[edit]
2020,Bridget Wambua,Kenya,2:18:38,Second victory,Elite_race_-_men's[edit]
2021[7],Joyciline Jepkogei,Kenya,2:17:43,Elite_race_-_men's[edit]
2022,Yalemzerf Yehualaw,Ethiopia,2:17:46,Elite_race_-_men's[edit]
2023,Edna Kiplagat,Kenya,2:18:31,Elite_race_-_men's[edit]
2024,Peres Jepchirchir,Kenya,2:16:18,Elite_race_-_men's[edit]

Year,Athlete,Nationality,Time (h:m:s),Notes,race_category
1981,Dick Beardsley (Tie),United States,2:11:48,Course record,Elite_race_-_men's[edit]
1982,Ingo Joensson (Tie),United States,2:11:48,Course record,Elite_race_-_men's[edit]
1983,Mike Grattan,United Kingdom,2:09:43,Elite_race_-_men's[edit]
1984,Charlie Spedding,United Kingdom,2:09:57,Elite_race_-_men's[edit]
1985,Steve Jones,United Kingdom,2:08:16,Course record,Elite_race_-_men's[edit]
1986,Tony Goss,United Kingdom,2:08:18,Course record,Elite_race_-_men's[edit]
1987,Hironi Taniguchi,Japan,2:09:58,Elite_race_-_men's[edit]
1988,Henrik Jørgensen,Denmark,2:10:20,Elite_race_-_men's[edit]
1989,Abdullah Almalki,United Arab Emirates,2:10:20,Elite_race_-_men's[edit]
1990,Yakov Toloktsov,Soviet Union,2:09:17,Elite_race_-_men's[edit]
1991,António Pinto,Portugal,2:10:02,Elite_race_-_men's[edit]
1992,Manuel Gómez,Spain,2:08:53,Elite_race_-_men's[edit]
1993,Dionicio Cerón,Mexico,2:08:30,Second victory,Elite_race_-_men's[edit]
1994,Dionicio Cerón,Mexico,2:08:30,Third victory,Elite_race_-_men's[edit]
1995,Dionicio Cerón,Mexico,2:08:30,Third victory,Elite_race_-_men's[edit]
1996,Abel Antón,Spain,2:07:57,Elite_race_-_men's[edit]
1997,Elie Mouzaïd,Morocco,2:07:57,Elite_race_-_men's[edit]
1998,Abdelkader El Mouzaïd,Morocco,2:07:57,Elite_race_-_men's[edit]
1999,Abdelkader El Mouzaïd,Morocco,2:07:57,Elite_race_-_men's[edit]
2000,Abdelkader El Mouzaïd,Morocco,2:07:57,Elite_race_-_men's[edit]
2001,Khalid Khanoussi,United States,2:05:38,World marathon record,Elite_race_-_men's[edit]
2002,Gezaheng Abers,Africa,2:07:56,Elite_race_-_men's[edit]
2003,Evans Rutt,Africa,2:06:18,Elite_race_-_men's[edit]
2004,Felix Limo,Kenya,2:06:39,Elite_race_-_men's[edit]
2005,Martin Lel,Kenya,2:06:39,Elite_race_-_men's[edit]
2006,Felix Limo,Kenya,2:06:39,Elite_race_-_men's[edit]
2007,Martin Lel,Kenya,2:07:41,Second victory,Elite_race_-_men's[edit]
2008,Martin Lel,Kenya,2:08:00,Course record,third victory,Elite_race_-_men's[edit]
2009,Samuel Wanjiru,Kenya,2:08:00,Course record,third victory,Elite_race_-_men's[edit]
2010,Tsegaye Kebede,Ethiopia,2:08:19,Elite_race_-_men's[edit]
2011,Emmanuel Kipchirchir Mutai,Kenya,2:04:49,Course record,Elite_race_-_men's[edit]
2012,Wilson Kipsang Kiprotich,Kenya,2:04:44,Elite_race_-_men's[edit]
2013,Tsige Tadesse,Ethiopia,2:04:44,Elite_race_-_men's[edit]
2014,Wilson Kipsang Kiprotich,Kenya,2:04:29,Course record; second victory,Elite_race_-_men's[edit]
2015,Eluid Kipchoge,Kenya,2:04:42,Elite_race_-_men's[edit]
2016,Eluid Kipchoge,Kenya,2:03:05,Course record,second victory,Elite_race_-_men's[edit]
2017,Eluid Kipchoge,Kenya,2:03:05,Course record,second victory,Elite_race_-_men's[edit]
2018,Eluid Kipchoge,Kenya,2:04:17,Third victory,Elite_race_-_men's[edit]
2019,Eluid Kipchoge,Kenya,2:02:37,Fourth victory,Course Record,Elite_race_-_men's[edit]
2020,Eluid Kipchoge,Kenya,2:01:39,Fourth victory,Course Record,Elite_race_-_men's[edit]
2021[7],Sisay Tilahun,Ethiopia,2:01:18,Elite_race_-_men's[edit]
2022,Amos Kipruto,Kenya,2:04:39,Elite_race_-_men's[edit]
2023,Kelvin Kiptum,Kenya,2:01:25,Course record,Elite_race_-_men's[edit]
2024,Alexander Muñoz,America,2:04:48,Elite_race_-_men's[edit]

Entendendo os dados raspados da fonte (Raw Profiling):

Resumo do Dataset Original quer dizer posteriores ao serem extraídos da fonte url:

A função resumo fornece uma tabela consolidada de alto nível que usa alguns métodos disponíveis no pandas retornando ao quadro geral com os resultados organizados em colunas específicas para cada método aplicado e com os atributos empilhados em linhas.

A primeira informação retorna como resultado da aplicação do método shape. Em uma dupla com o número de entradas (linhas e colunas) do conjunto de dados, seguido por outra coluna que retorna os tipos de dados para cada atributo, denominado "Dtype", dessa vez retornando o resultado com o método "dtypes", outra coluna Uniques que nos retorna os valores exclusivos ao usar o método nunique, bem como a coluna Missing e % Missing para os valores ausentes presentes em cada atributo com o método isnull e finalmente uma visualização dos três primeiros e últimos registros ou entradas do conjunto de dados.

Análise dos dados (raw data)

O conjunto de dados london.csv (OBT: "combined_df" DB como output depurado na camada prata) original usado neste Notebook fornece dados de 43 anos de maratonas que aconteceram em londres. O dataset em um conjunto de cinco atributos cujos nomes são suficientemente informativos no contexto de registro de resultados obtidos por categorias durante esse período até a data de hoje.

O Dataset é armazenado em uma classe 'pandas.core.frame.DataFrame'.

O comando Shape: O conjunto de dados contém 171 registros ou entradas únicas (podendo ser indexadas pelo "Index" ou alocados em cinco colunas que encerram atributos numéricos, especiais e categóricos:

Tipos de variáveis, issues e possíveis abordagens :

- * Como pode ver acima, a função dtypes não agrupa de forma limpa os diferentes tipos de dados.
- * Os tipos de dados numéricos (float64 e int64) . algumas colunas são listadas como objetos, o que não é muito útil, nas seguintes células os categorizarei para aprofundar na análise das categorias para cada atributo.
- * Na continuação, vou dar leitura do que foi interpretado pela função dtypes de pandas e proporcionarei um plano das possíveis abordagens para alguns atributos relevantes e outros a desconsiderar da análise.
- * São 5 variáveis: Year (YYYY), Athleta (Nome Sobrenome) , Nationality (Country), Time (h:m:s), Category (Race category) todas categóricas, que irei me aprofundar em detalhe com uma análise descritiva para variáveis do tipo categórica.
- * O atributo Year (YYYY) do tipo Numérica, que irei me aprofundar em detalhe com uma análise descritiva para variáveis do tipo numérica.
- * As colunas Athleta , Nationality, Time (h:m:s), Category foram categorizadas como tipo objeto e que iremos nos aprofundar em detalhe para conhecer as categorias e tipos nas seguintes células.
- * Um objeto chama a atenção 'Time' que poderíamos fixar um tipo de abordagem da análise exploratória e que deveríamos transformar o formato da variável.
O formato Datetime (Y%M%D%H:%M:%S) olhando sua cardinalidade (valores únicos) facilitaria além da análise das séries temporais, nesse caso seria necessário indexar a coluna
(Index =0)". E abordar o assunto central ou seja responder as perguntas formuladas acima, no início.
- * São 2 variáveis: Year (YYYY), Time (h:m:s), , que irei me aprofundar em detalhe com uma análise descritiva para variáveis do tipo numérica mais a frente.
- * A coluna "Notes", identificada como um objeto, poderia ser transformado para string mas como não iremos minerar texto, os que apresentam acima de 55% dos valores ausentes serão desconsiderado da análise.

Finalmente pré - visualizando as três primeiras e últimas linhas:

# O comando seguinte retorna a função resumo passando o conjunto de dados (dataframe).											
resumo(london)											
Tabela 1 Resumo do dataframe											
O conjunto de dados tem 171 entradas e 5 colunas											
0	Year	int64	0	0.0	44	1981	1981	1982	2023	2022	2021
1	Athlete	object	0	0.0	104	Dick Beardsley (Tie)	Inge Simonsen (Tie)	Hugh Jones	Madison de Rozario	Catherine Debrunner	Manuela Schär
2	Nationality	object	0	0.0	24	United States	Norway	United Kingdom	Australia	Switzerland	Switzerland
3	Time (h:m:s)	object	0	0.0	165	2:11:48	2:11:48	2:09:24	1:38:51	1:38:24	1:39:52
4	Category	object	0	0.0	4	Men	Men	Men	Wheelchair Women	Wheelchair Women	Wheelchair Women

Com a informação da tabela, consegue-se entender melhor o conjunto de dados. Já que agora sabemos que está lidando apenas com dados categóricos e numéricos. Transformações na camada prata poderiam ser necessárias para a análise descritiva de performance da maratona de Londres que desenvolveremos mais a frente no Databricks . Por outro lado, as variáveis categóricas são geralmente uma maneira interessante de enriquecer semanticamente os dados e segmentar e agrupar ou desagregar os dados por meio de filtros com o uso da linguagem Spark SQL, como veremos nas mais a frente.

```
[ ] # A função get_var_category determina a categoria de cada coluna (atributo) para o dataset carregado.

def get_var_category(series):
    unique_count = series.nunique(dropna=False)
    total_count = len(series)
    if pd.api.types.is_numeric_dtype(series):
        return 'Numérica'
    elif pd.api.types.is_datetime64_dtype(series):
        return 'Data'
    elif unique_count==total_count:
        return 'Texto (Unique)'
    else:
        return 'Categorica'

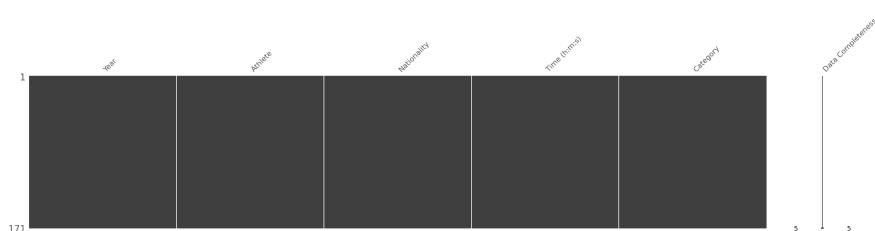
def print_categories(london):
    for column_name in london.columns:
        print(column_name, ":", get_var_category(london[column_name]))

[ ] # O comando seguinte retorna a categoria de cada variável:
print_categories(london)

→ Year : Numérica
Athlete : Categorica
Nationality : Categorica
Time (h:m:s) : Categorica
Category : Categorica
```

Essas transformações previstas aqui na data bruta são o resultado da raspagem original onde algumas variáveis até aqui categorizadas como objetos são fundamentais, como as numéricas especiais , baseadas no tempo e que são a pedra angular ou a base da análise descritiva para um modelo de dados orientado ao objetivo:

A coluna "Time (H:m:s)" é a variável mais relevante e que tem sido categorizada como do tipo str, é inferido com o método dtype. Vai ser necessário, em outra etapa posterior (camada prata), transformar o vetor para um tipo especial "Datetime" ou outro.



Outras transformações que poderiam ser executadas como desconsiderar a coluna "Notes", porque neste caso não nos agrega informação relevante para manter como atributo.

```
[ ] ## verificando nulls no dataset

# o novo dataset irá conter todas as colunas do dataset original
london_com_missings = london[london.columns[:]]

# substituindo os zeros por NaN
london_com_missings.replace(0, np.nan, inplace=True)

# exibindo visualização matricial da nulidade do dataset

'''Para ilustrar as análises acima vou usar um recurso gráfico
de python para a visualização dos valores ausentes no conjunto de dados original.
Ao plotar, visualizamos os espaços em branco que representariam os valores ausentes
presentes no dataset sem uma abordagem para seu tratamento ou exclusão e que executaremos
mais a frente.''''

ms.matrix(london_com_missings, sparkline= True, labels=True, figsize=(32, 6))

plt.show()
```

As colunas "Athlete" e "Nationality" são inferidas como elementos de lista de variáveis do tipo str. Ao explorar exemplos de valores no dataset, verifica-se que não apresentam NaN, e ao verificar nas colunas, os dois atributos têm 100% dos dados presentes. Pode-se ilustrar na célula uma representação gráfica dos NaN com um método exibindo visualização matricial.

3.- Modelagem : De CSV (Flat File) para OBT (Delta Lake)

A abordagem da Tabela Única (OBT) surge como uma alternativa moderna ao Modelado Dimensional, caracterizada por sua simplicidade. Essa metodologia envolve armazenar dados em uma única tabela expansiva, simplificando o modelo de dados ao reduzir significativamente o número de tabelas a serem gerenciadas e atualizadas.

Destaca a simplicidade da tabela única em contraste com o Modelo Dimensional, facilitando a integração e modificação rápidas de dados. Além disso, aborda a consistência e governança de dados ao armazenar todos os dados em uma única tabela, reduzindo riscos de duplicação e inconsistências. Também menciona o desempenho melhorado da tabela única, evitando junções complexas e proporcionando resultados de consulta mais rápidos.

Enquanto a tabela única envolve armazenar dados em uma única tabela expansiva, simplificando o modelo de dados ao reduzir o número de tabelas, um "Flat File" refere-se a um arquivo de texto simples que contém dados sem uma estrutura de banco de dados formal. Embora ambos possam ser utilizados para armazenar dados de forma simplificada, a tabela única geralmente é associada a um banco de dados, enquanto um "Flat File" é mais comumente um arquivo de dados simples sem um esquema formal (e sem Metadados).

Para realizar a modelagem de uma tabela CSV Flat File no Databricks, visando a qualidade da modelagem dos dados e a documentação do Catálogo de Dados, podemos seguir estes passos do roteiro a seguir para criar a DB Lakehouse na saída da camada prata :

3.1.- Configs ADLS: Storage account - Azure /APP registration/Azure Data Lake Storage Mounts / Landing Zone

- Definimos o local de armazenamento da tabela no Databricks, como um diretório específico no DBFS (Databricks File System) para a criação da tabela Delta Lake.

3.2.- Mount camada Bronze / Carrega dos Dados na Tabela camada Bronze:

- Utilizamos comandos ou bibliotecas do Databricks, como o `spark.read.csv()` ou o Delta Lake, para carregar os dados do arquivo CSV na tabela recém-criada.
- Verificamos se os dados foram carregados corretamente, examinando amostras ou executando consultas simples.

3.3.- Definição do Esquema da Tabela na camada prata:

- Com base na análise inicial, defina o esquema da tabela, especificando os nomes das colunas, os tipos de dados apropriados (por exemplo, string, integer, float, date, etc.) e as restrições de integridade.
- Considere a normalização dos dados, se necessário, para evitar redundâncias e anomalias.

3.4.- Criação da Tabela no Databricks na camada prata:

- Utilizei os recursos para a criação da tabela no Databricks, como o comando `CREATE TABLE` Spark SQL ou a interface gráfica do Databricks Workspace.
- Especifique o esquema definido anteriormente, incluindo os nomes das colunas, tipos de dados e quaisquer outras propriedades relevantes.
- Definimos o local de armazenamento da tabela no Databricks, como um diretório específico no DBFS (Databricks File System) ou uma tabela Delta Lake.

3.5.- Documentação no Catálogo de Dados:

- Documente as informações sobre a tabela no Catálogo de Dados do Databricks.
- Inclua metadados relevantes, como nome da tabela, descrição, fontes de dados, proprietários, esquema das colunas, regras de negócio.
- Considere adicionar anotações, tags para facilitar a descoberta e o entendimento da tabela por outros usuários.

Essa abordagem de modelagem de tabela CSV (Flat File) para uma OBT no Databricks, com foco na qualidade dos dados e na documentação do Catálogo de Dados, ajudará a garantir a confiabilidade, a rastreabilidade e a facilidade de uso dos dados pelos analistas no relatório Ad-Hoc e para futuros analyses e pelos stakeholders do banco de dados.

A abordagem da arquitetura Medallion em um Data Lakehouse oferece uma arquitetura estruturada e eficiente para facilitar o escalamento e lidar com grandes volumes de dados, preservando o histórico completo, simplificando a arquitetura e proporcionando flexibilidade e governança robusta.

3.1.- Configs ADLS: Configurações Preliminares: Ambiente Databricks Community ADLS Mount - APP_Registrations :

3.1.- Configurações Preliminares, Configs

Definimos o nome da conta de armazenamento do Azure Data Lake Storage (ADLS) que será utilizada no ambiente Databricks.

```
▶   ✓ 03/07/2024 (<1s) 2
storage_account_name = "aulabricks" # client_id: O ID do cliente (aplicativo) registrado no AAD.
# tenant_id: O ID do locatário (diretório) do AAD.
# client_secret: O segredo do cliente (aplicativo) registrado no AAD.

#client_id      = dbutils.secrets.get(scope="databricks-adls-scope", key="databricks-app-client-id")
#tenant_id      = dbutils.secrets.get(scope="databricks-adls-scope", key="databricks-app-tenant-id")
#client_secret  = dbutils.secrets.get(scope="databricks-adls-scope", key="databricks-app-client-secret")
```

Prendemos manualmente os valores de client_id, tenant_id e client_secret.

Esses valores são obtidos a partir do registro de aplicativo no Azure Active Directory (AAD) e são necessários para autenticar o acesso ao ADLS a partir do ambiente Databricks.

A configurações acima são necessárias para que o ambiente Databricks possa acessar o ADLS de forma autenticada e segura. Ao utilizar segredos armazenados no Databricks, você evita que esses dados confidenciais fiquem expostos no código-fonte, o que é uma prática recomendada de segurança.

Essa abordagem permite que o código de ingestão e transformação de dados no Databricks interaja com o ADLS de forma segura e escalável, seguindo as melhores práticas de segurança.

APP registration: O processo geral é

- Criar um registro de aplicativo no Azure Active Directory.
- Obter os valores de client_id, tenant_id e client_secret a partir desse registro de aplicativo.
- Armazenar esses valores como segredos no Databricks, usando o dbutils.secrets.get() para acessá-los no código.
- Utilizar esses valores de autenticação para acessar o ADLS a partir do ambiente Databricks.

3.1.1.- Mount ADLS:

Criando um dicionário chamado configs que contém as configurações necessárias para acessar o Azure Data Lake

```
▶   ✓ 03/07/2024 (<1s) 5
# CONFIGS
configs = {"fs.azure.account.auth.type": "OAuth",
           "fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
           "fs.azure.account.oauth2.client.id": f"{client_id}",
           "fs.azure.account.oauth2.client.secret": f"{client_secret}",
           "fs.azure.account.oauth2.client.endpoint": f"https://login.microsoftonline.com/{tenant_id}/oauth2/token"}
```

Storage (ADLS) de forma autenticada. O dicionário configs é usado posteriormente no código para definir as configurações de acesso ao ADLS.

Essas configurações são passadas como parâmetros para a criação do SparkSession e para a leitura/gravação de dados no ADLS.

Essa abordagem de centralizar as configurações em um dicionário facilita a manutenção e a reutilização do código, pois qualquer alteração nas informações de autenticação pode ser feita diretamente no dicionário configs, sem a necessidade de modificar o código-fonte.

A continuação são descritos cada chave-valor desse dicionário:

"fs.azure.account.auth.type": "OAuth": Indica que a autenticação será feita usando OAuth.
"fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider": Especifica o provedor de autenticação OAuth que será usado, neste caso, o ClientCredsTokenProvider.
"fs.azure.account.oauth2.client.id": f'{client_id}': Insere o valor da variável client_id que contém o ID do cliente (aplicativo) registrado no Azure Active Directory (AAD).
"fs.azure.account.oauth2.client.secret": f'{client_secret}': Insere o valor da variável client_secret que contém o segredo do cliente (aplicativo) registrado no AAD.
"fs.azure.account.oauth2.client.endpoint": f'https://login.microsoftonline.com/{tenant_id}/oauth2/token': Insere o valor da variável tenant_id que contém o ID do locatário (diretório) do AAD e define o endpoint de autenticação OAuth.

Essa configuração é necessária para que o ambiente Databricks possa acessar o ADLS de forma segura, utilizando o OAuth2 como mecanismo de autenticação.

Definimos uma função chamada `mount_adls` que recebe um parâmetro `container_name`.

Dentro dessa função, usamos o `dbutils.fs.mount()` para montar o Azure Data Lake Storage (ADLS) no ambiente Databricks.

Os parâmetros passados para a função `mount()` são:

- `source`: O caminho de origem do contêiner ADLS, que é construído usando o `container_name` e o `storage_account_name`.
- `mount_point`: O ponto de montagem no sistema de arquivos do Databricks, que é construído usando o `storage_account_name` e o `container_name`.
- `extra_configs`: O dicionário `configs` que foi definido anteriormente, contendo as informações de autenticação para acessar o ADLS.

A função MOUNT ADLS é importante para que o ambiente Databricks possa acessar os dados armazenados no ADLS de forma transparente, como se fossem arquivos locais.

Mount ADLS

APP registration



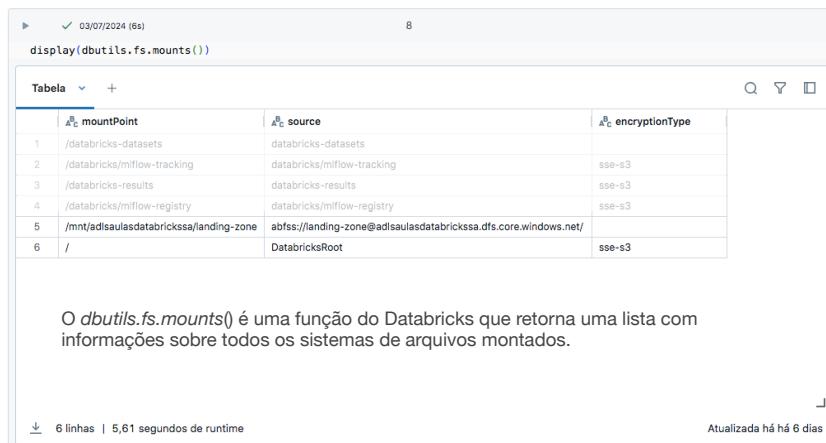
```
# MOUNT ADLS
def mount_adls(container_name):
    dbutils.fs.mount(
        source = f"abfss://{{container_name}}@{{storage_account_name}}.dfs.core.windows.net/",
        mount_point = f"/mnt/{{storage_account_name}}/{{container_name}}",
        extra_configs = configs)
```

Ao chamar essa função `mount_adls()` e passar o nome do contêiner ADLS como argumento, o Databricks irá montar o contêiner no sistema de arquivos local, usando as configurações de autenticação definidas no dicionário `configs`.

Depois de executar essa função, você poderá acessar os dados do contêiner ADLS montado usando caminhos como `/mnt/{storage_account_name}/{container_name}` em suas operações de leitura e escrita de dados no Databricks.

Essa abordagem de montar o ADLS no Databricks facilita a integração e o acesso aos dados armazenados no Azure, permitindo que você trabalhe com eles de forma transparente e escalável dentro do ambiente Databricks.

O código `display(dbutils.fs.mounts())` é usado para exibir uma tabela com informações sobre os sistemas de arquivos montados no ambiente Databricks. Essa função é útil para verificar se o Azure Data Lake Storage (ADLS) foi montado corretamente, com as informações de configuração usadas.



	mountPoint	source	encryptionType
1	/databricks-datasets	databricks-datasets	sse-s3
2	/databricks/miflow-tracking	databricks/miflow-tracking	sse-s3
3	/databricks-results	databricks-results	sse-s3
4	/databricks/miflow-registry	databricks/miflow-registry	sse-s3
5	/mnt/adlsaulasdatabrickssa/landing-zone	abfss://landing-zone@adlsaulasdatabrickssa.dfs.core.windows.net/	
6	/	DatabricksRoot	sse-s3

O `dbutils.fs.mounts()` é uma função do Databricks que retorna uma lista com informações sobre todos os sistemas de arquivos montados.

Quando é executada a linha de código, o Databricks irá exibir uma tabela com as informações sobre os sistemas de arquivos montados: Ao executar essa linha, você poderá visualizar o caminho de montagem do ADLS e outras informações relevantes, garantindo que o acesso aos dados no ADLS esteja funcionando corretamente antes de prosseguir com outras operações no Databricks.

3.1.2.- Landing -Zone: Mount Landing Zone

A "landing zone" geralmente é um contêiner ou diretório dentro da conta de armazenamento do (Azure Storage Account onde os dados brutos ou não processados são inicialmente armazenados os nossos CSV's.

Neste caso, a linha de código `mount_adls('landing-zone')` está montando o contêiner "landing-zone" que está localizado na conta de armazenamento do Azure definida pela variável `storage_account_name`.

Dessa forma, o Databricks pode acessar os dados na "landing zone" de forma transparente, como se fossem arquivos locais, usando o caminho `/mnt/{storage_account_name}/landing-zone`.

Essa abordagem de montar o contêiner ADLS diretamente no Databricks evita a necessidade de trabalhar com caminhos externos ou fazer referência direta à conta de armazenamento do Azure. Isso torna o código mais portável e fácil de manter, pois todas as informações de configuração estão centralizadas no dicionário configs.

Então, sim, a "landing zone" é um contêiner ou diretório dentro da conta de armazenamento do Azure, e a montagem desse contêiner no Databricks permite que trabalhe com os dados de forma integrada e gerenciada, sem precisar lidar diretamente com a estrutura de armazenamento do Azure.



```
▶ ✓ 03/07/2024 (14s) 11
mount_adls('landing-zone')
```

Nesta linha, estamos chamando a função `mount_adls()` como definimos anteriormente.

Ao chamar essa função, estamos passando o argumento '`landing-zone`', que representa o nome do contêiner ADLS que queremos montar no ambiente Databricks.

Aqui está o que acontece quando essa linha de código é executada:

A função `mount_adls()` é chamada com o argumento '`landing-zone`'.

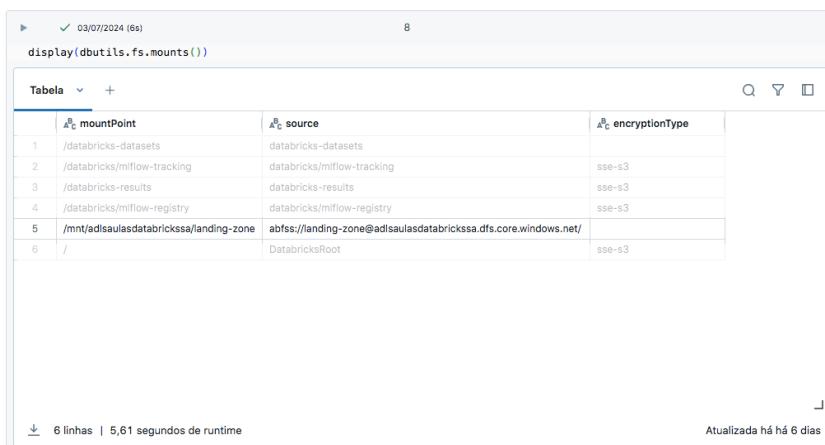
Dentro da função `mount_adls()`, o `dbutils.fs.mount()` é usado para montar o contêiner ADLS chamado '`landing-zone`' no ambiente Databricks.

O ponto de montagem será `/mnt/{storage_account_name}/landing-zone`, onde `{storage_account_name}` é o nome da conta de armazenamento do Azure.

As configurações de autenticação definidas anteriormente no dicionário configs são usadas para acessar o ADLS de forma segura.

Essa linha de código é responsável por montar o contêiner ADLS chamado '`landing-zone`' no ambiente Databricks, permitindo que você accesse os dados armazenados nesse contêiner usando caminhos como `/mnt/{storage_account_name}/landing-zone` em suas operações de leitura e escrita de dados.

Após executar essa linha, você pode verificar se o contêiner ADLS foi montado corretamente usando o comando `display(dbutils.fs.mounts())` que exibe uma tabela com informações sobre os sistemas de arquivos montados.



mountPoint	source	encryptionType
/databricks-datasets	databricks-datasets	
/databricks/miflow-tracking	databricks/miflow-tracking	sse-s3
/databricks-results	databricks-results	sse-s3
/databricks/miflow-registry	databricks/miflow-registry	sse-s3
/mnt/adisaulasdatabrickssa/landing-zone	abfss://landing-zone@adisaulasdatabrickssa.dfs.core.windows.net/	
/	DatabricksRoot	sse-s3

6 linhas | 5,61 segundos de runtime Atualizada há 6 dias

3.2 Bronze:

Na arquitetura de dados em camadas (Medallion Architecture) de um Data Lakehouse, a camada Bronze representa:



- Uma cópia praticamente linha a linha dos dados ingeridos;
- Substitui o tradicional data lake.
- Fornece armazenamento e consulta eficientes de todo o histórico não processado dos dados;

Essa abordagem da camada Bronze na arquitetura de Medallion traz diversos benefícios, como a simplificação da arquitetura e a eficiência no armazenamento e consulta dos dados brutos. Isso fornece uma base sólida para as camadas superiores do Data Lakehouse, como a camada Prata e Ouro, onde os dados eventualmente poderão ser processados e transformados.

A linha de código `mount_adls('bronze')` é responsável pela implementação da estrutura de armazenamento. Dentro da função `mount_adls()`, o Databricks monta o contêiner ADLS chamado 'bronze' no sistema de arquivos local, usando o caminho `/mnt/{storage_account_name}/bronze`. As informações de autenticação e configuração necessárias para acessar o ADLS são obtidas do dicionário `configs` definido anteriormente.

```
▶ ✅ 03/07/2024 (12s) 12 Python ⚙️ ⋮
mount_adls('bronze')
```

3.2.1- Mount camada Bronze / Carrega dos Dados na Tabela camada Bronze:

A abordagem de criar um banco de dados no Databricks e especificar o local de armazenamento no ADLS é comum em arquiteturas de data lake. Algumas vantagens dessa abordagem:

Isolamento: O banco de dados "bronze_maratona" fica isolado e organizado, facilitando o gerenciamento e a manutenção dos dados.

Escalabilidade: O ADLS fornece um armazenamento altamente escalável e durável para os dados do banco de dados.

Integração com outros serviços: O banco de dados no Databricks pode ser facilmente integrado com outros serviços do ecossistema Azure, como o Azure Synapse Analytics, para análises e processamento de dados avançados.

Gerenciamento de metadados: O Databricks gerencia os metadados do banco de dados, facilitando a descoberta e o acesso aos dados.

Essa primeira linha de código cria um banco de dados chamado "bronze_maratona" no ambiente Databricks, com o local de armazenamento definido no ADLS. Essa é uma etapa fundamental para organizar e gerenciar os dados brutos(camada bronze) neste ambiente de data lake e data warehouse.

```
▶ 2
%sql

CREATE DATABASE IF NOT EXISTS bronze_maratona
LOCATION "/mnt/adlsaulasdatabrickssa/bronze/bronze_maratona"

▼ _sqldf: pyspark.sql.DataFrame
OK

ℹ️ O resultado das células SQL foi armazenado como um pacote de dados _sqldf do PySpark. Saiba mais
```

CREATE DATABASE IF NOT EXISTS bronze_maratona;

Esta linha cria um novo banco de dados chamado "bronze_maratona" se ele ainda não existir. Isso garante que o banco de dados seja criado apenas uma vez, evitando erros caso já exista.

LOCATION "/mnt/adlsaulasdatabrickssa/bronze/bronze_maratona";

Esta parte especifica o local de armazenamento do banco de dados "bronze_maratona". Neste caso, o banco de dados será armazenado no caminho "/mnt/adlsaulasdatabrickssa/bronze/bronze_maratona" no sistema de arquivos do Azure Data Lake Storage (ADLS) associado ao ambiente Databricks.

Essas configurações em formato de dicionário permitem uma forma organizada e flexível de gerenciar as informações sobre os diferentes conjuntos de dados que serão processados no ambiente Databricks e Azure.

```

▶ 3

# Configurações dos arquivos CSV e nomes das tabelas correspondentes
configs = {
    "Elite_race_-_mens": {
        "file_path": "/mnt/adlsaulasdatabrickssa/landing-zone/maratona_londres/Elite_race_-_mens.csv",
        "table_name": "mens"
    },
    "Wheelchair_race_-_mens": {
        "file_path": "/mnt/adlsaulasdatabrickssa/landing-zone/maratona_londres/Wheelchair_race_-_mens.csv",
        "table_name": "wheelchair_mens"
    },
    "Wheelchair_race_-_womens": {
        "file_path": "/mnt/adlsaulasdatabrickssa/landing-zone/maratona_londres/Wheelchair_race_-_womens.csv",
        "table_name": "wheelchair_womens"
    },
    "Elite_race_-_womens": {
        "file_path": "/mnt/adlsaulasdatabrickssa/landing-zone/maratona_londres/Elite_race_-_womens.csv",
        "table_name": "womens"
    }
}

```

- Cada chave do dicionário `configs` representa um nome do arquivo, como `"Elite_race_-_mens"`, `"Wheelchair_race_-_mens"`, `"Wheelchair_race_-_womens"` e `"Elite_race_-_womens"`.
- Cada valor do dicionário é outro dicionário que contém duas chaves:
 - `"file_path"`: O caminho completo para o arquivo CSV correspondente no Azure Data Lake Storage (ADLS).
 - `"table_name"`: O nome da tabela que será criada no banco de dados a partir desse arquivo.

O dicionário de configuração é uma boa prática e a travas dessa abordagem se verificam variadas vantagens:

- Modularidade: Cada conjunto de dados é tratado de forma independente, facilitando a manutenção e a adição de novos conjuntos de dados no futuro.
- Reutilização de código: O código que processa esses conjuntos de dados pode ser escrito de forma genérica, usando o dicionário `configs` para obter as informações necessárias.
- Mantenibilidade: Todas as informações sobre os caminhos dos arquivos e nomes das tabelas ficam centralizadas neste dicionário `configs`, simplificando a gestão da configuração.
- Legibilidade e entendimento do código: O uso de nomes descritivos para as chaves do dicionário torna o código mais legível e fácil de entender.

Criamos uma função para automatizar o processo de ingestão e transformação de dados em ambientes DBX e Azure, seguindo boas práticas de data engineering:

- Uso do formato Delta Lake para armazenamento, com vantagens como controle de versão, otimização de consultas e suporte a transações.
- Adição de um timestamp de ingestão para rastrear quando os dados foram carregados.
- Re-nomeação de colunas para melhor legibilidade.
- Sobrescrita da tabela a cada execução, garantindo que os dados mais recentes estejam sempre disponíveis (Delta Lake).

```
def read_transform_save(file_path, table_name):
```

Essa abordagem é escalável e robusta, permitindo processar diferentes conjuntos de dados de forma padronizada e automatizada, facilitando a manutenção e a evolução do pipeline de dados ao longo do tempo.

```
▶ 4

from pyspark.sql.functions import col, current_timestamp

# Função para ler, transformar e salvar os dados dos arquivos CSV
def read_transform_save(file_path, table_name):
    """
    Lê um arquivo CSV, adiciona uma coluna de timestamp e renomeia uma coluna,
    então salva o DataFrame resultante em uma tabela Delta.

    Parâmetros:
    - file_path: Caminho para o arquivo CSV.
    - table_name: Nome da tabela onde os dados serão salvos.
    """
    print(f"Lendo tabela: {table_name}")

    # Lê o arquivo CSV com opções especificadas
    df = spark.read.option("header", "true").option("sep", ",").csv(file_path)

    # Adiciona uma coluna com o timestamp de ingestão
    df = df.withColumn('bronze_ingestion_timestamp', current_timestamp())

    # Renomeia a coluna 'Time (h:m:s)' para 'Time'
    df = df.withColumnRenamed('Time (h:m:s)', 'Time')

    # Salva o DataFrame como uma tabela Delta no modo overwrite
    (df
     .write
     .format("delta")
     .mode("overwrite")
     .option("overwriteSchema", "true")
     .saveAsTable(f"bronze_maratona.{table_name}"))

    print(f"Gravado a tabela: bronze_maratona.{table_name}")
```

Este trecho de código faz o seguinte:

Iniciamos um loop for que irá iterar sobre os itens do dicionário configs. O configs é um dicionário que contém a configuração necessária para processar cada conjunto de dados.

Para cada ítem no dicionário configs, imprimimos uma mensagem informando qual informação está sendo lida. Dentro do loop, extraímos dois valores do dicionário configs para as variáveis file_path (o caminho completo do arquivo CSV a ser lido) e table_name (o nome da tabela que será criada no banco de dados):

Itera sobre os itens do dicionário configs:

- Para cada item, extrai o file_path e o table_name correspondentes;
- Chama a função `read_transform_save` passando esses valores como argumentos;
- Imprime uma linha em branco para separar a saída de cada iteração.

Essa abordagem permite processar múltiplos conjuntos de dados de forma automatizada e padronizada, seguindo as mesmas etapas de leitura, transformação e gravação em tabelas Delta.

O uso do dicionário `configs` torna o código mais flexível e fácil de manter, pois todas as informações de configuração ficam centralizadas em um único local. Isso facilita a adição ou remoção de novos conjuntos de dados a serem processados, sem a necessidade de alterar o código-fonte.

Essa estrutura é muito comum em projetos de data engineering, pois permite uma abordagem modular e escalável para lidar com a ingestão e transformação de múltiplas fontes de dados em um ambiente Databricks e Azure.

Processamento dos arquivos CSV de acordo com as configurações especificadas

```
# Loop através de cada configuração para ler, transformar e salvar os dados
for item in configs:
    print("Lendo informações de", configs[item])

    file_path = configs[item]["file_path"]
    table_name = configs[item]["table_name"]

    read_transform_save(file_path, table_name)
    print("\n")
```

▶ (44) jobs Spark

Gravado a tabela: bronze_maratona.mens

lendo informacao por {'file_path': '/mnt/adlsaulasdatabrickssa/landing-zone/maratona_londres/Wheelchair_race_-_mens.csv', 'table_name': 'wheelchair_mens'}
Lendo tabela: wheelchair_mens
Gravado a tabela: bronze_maratona.wheelchair_mens

lendo informacao por {'file_path': '/mnt/adlsaulasdatabrickssa/landing-zone/maratona_londres/Wheelchair_race_-_womens.csv', 'table_name': 'wheelchair_womens'}
Lendo tabela: wheelchair_womens
Gravado a tabela: bronze_maratona.wheelchair_womens

lendo informacao por {'file_path': '/mnt/adlsaulasdatabrickssa/landing-zone/maratona_londres/Elite_race_-_womens.csv', 'table_name': 'womens'}
Lendo tabela: womens
Gravado a tabela: bronze_maratona.womens

3.3.- Definição do Esquema da Tabela na camada prata:

A camada Prata (Silver) da arquitetura Medallion em um Data Lakehouse tem os seguintes objetivos:

- Reduzir a complexidade, latência e redundância no armazenamento de dados, otimizando o armazenamento e melhorando a eficiência geral do sistema.
- Otimizar o desempenho do ETL (Extração, Transformação e Carregamento) e das consultas analíticas, estruturando e organizando os dados de forma eficiente.
- Preservar o nível de granularidade original dos dados, sem realizar agregações, permitindo análises mais flexíveis e detalhadas.
- Eliminar registros duplicados, garantindo a integridade e confiabilidade dos dados.
- Impor um esquema de produção rígido, assegurando a consistência e qualidade dos dados.

- Realizar verificações de qualidade de dados e quarentena de dados corrompidos, evitando a contaminação do pipeline.

Dessa forma, a camada Prata desempenha um papel fundamental na otimização do armazenamento, processamento e qualidade dos dados, preparando-os para as camadas superiores do Data Lakehouse.

Criação do Banco de Dados: Este comando SQL cria de forma segura um banco de dados chamado prata_maratona e define seu local de armazenamento em um diretório específico no sistema de arquivos. No contexto de ambientes como o Databricks, este caminho geralmente aponta para um diretório montado em um sistema de armazenamento em nuvem, como o Azure Data Lake Storage (ADLS).

3.3.- Definição do Esquema da Tabela na camada prata:

```

▶   ✓ Há 3 dias (<1s) 2
%sql
CREATE DATABASE IF NOT EXISTS prata_maratona -- Cria o banco de dados 'prata_maratona' se ele não existir no Delta Lake
LOCATION "/mnt/adlsaulasdatabrickssa/prata/prata_maratona"

▶ _sqldf: pyspark.sql.DataFrame
OK

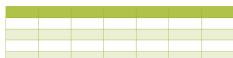
```

Criação do Banco de Dados:

- Cria CREATE DATABASE um novo banco de dados com nome prata_maratona :
 - Acrescentamos uma condição IF NOT EXISTS para evitar recriações desnecessárias.
- Definimos um Local de Armazenamento:
 - Especificamos LOCATION onde os arquivos do banco de dados serão armazenados.
 - Fornecemos o Caminho para um diretório em um sistema de armazenamento no ADLS.

3.4 Criação da Tabela results (OBT - Modelagem):

ONE BIG TABLE (OBT)



Este código SQL cria uma nova tabela `prata_maratona.results` a partir da UNIÃO de dados de quatro tabelas diferentes (mas que compartilham o mesmo cabeçalho), que se diferenciam entre elas porque cada uma segmentada por seus valores dentre os tipos de `race_category` realizando a conversão de tipos de dados e o cálculo da duração da corrida em segundos. Essa tabela consolidada vai ser útil para a análise descritiva que será feita posteriormente a análise de qualidade.

Enforcing schema in SQL

-- Definindo o schema da tabela usando SQL

```
%sql
CREATE OR REPLACE TABLE prata_maratona.results AS
SELECT
    CAST(race_category AS STRING) AS race_category,
    CAST(Year AS INT) AS Year,
    CAST(Athlete AS STRING) AS Athlete,
    CAST(Nationality AS STRING) AS Nationality,
    CAST(Time AS STRING) AS Time,
    CAST(duracao_em_segundos AS DOUBLE) AS duracao_em_segundos
FROM (
    SELECT "mens" AS race_category, — Seleciona os dados da tabela bronze_maratona.mens, adicionando uma
    Year, — nova coluna race_category com o valor fixo "mens" para identificar a
    Athlete, — categoria da corrida.
    Nationality, — Adiciona uma coluna race_category com valores fixos para cada
    Time, — categoria.
    (unix_timestamp(Time, 'H:mm:ss') - unix_timestamp('0:00:00', 'H:mm:ss')) AS duracao_em_segundos — Calcula a duração em segundos a partir do tempo em formato H:mm:ss.
    FROM bronze_maratona.mens
    UNION — Seleção de Dados:
    — Adiciona uma coluna race_category com valores fixos para cada categoria.
    — Converte a coluna Year para tipo inteiro.
    — Seleciona colunas Athlete, Nationality, e Time.
    — Calcula a duração em segundos a partir do tempo em formato H:mm:ss.
    — Especifica a tabela de origem dos dados com o nome da tabela que
    — contém os dados das corridas masculinas
    SELECT "womens" AS race_category, — Repetição do SELECT para outras categorias:
    Year, — Descrição: Repete a seleção para outras categorias (mens, womens,
    Athlete, — wheelchair_mens, wheelchair_womens), cada uma adicionando sua
    Nationality, — respectiva coluna race_category e os mesmos cálculos e seleções.
    Time, — Lógica: Cada SELECT segue a mesma estrutura, mudando apenas o
    (unix_timestamp(Time, 'H:mm:ss') - unix_timestamp('0:00:00', 'H:mm:ss')) AS duracao_em_segundos — valor fixo da coluna race_category e a tabela de origem dos dados.
    FROM bronze_maratona.womens
    UNION — Combinação de Resultados: Se Utiliza o comando UNION para combinar os resultados de várias tabelas de categorias diferentes, removendo duplicatas.
    SELECT "wheelchair_mens" AS race_category,
    Year,
    Athlete,
    Nationality, — Para cada consulta, são selecionadas as colunas Year, Athlete, Nationality, Time e é calculada
    Time, — a duração da corrida em segundos usando a função unix_timestamp.
    (unix_timestamp(Time, 'H:mm:ss') - unix_timestamp('0:00:00', 'H:mm:ss')) AS duracao_em_segundos
    FROM bronze_maratona.wheelchair_mens
    UNION
    SELECT "wheelchair_womens" AS race_category,
    Year,
    Athlete,
    Nationality, —
    Time,
    (unix_timestamp(Time, 'H:mm:ss') - unix_timestamp('0:00:00', 'H:mm:ss')) AS duracao_em_segundos
    FROM bronze_maratona.wheelchair_womens
) AS combined_results; — O resultado dessas quatro consultas é combinado em uma única tabela virtual chamada combined_results.
```

O objetivo deste código SQL é estabelecer uma camada prata de dados, a partir das tabelas bronze no formato otimizado, para servir de base de dados otimizada (tabela delta) para a criação de consultas SQL (QRYs) e a realização de análises mais aprofundadas, se necessário, visando a geração de relatórios ad-hoc.

- Criação da camada prata: O código cria uma nova tabela `prata_maratona.results` que consolida e transforma os dados brutos cópia otimizada do data lake (landins zone) ou camada bronze em uma estrutura mais adequada para análises.
- Tabela delta otimizada: Essa tabela `prata_maratona.results` pode ser considerada uma tabela delta, ou seja, uma camada intermediária otimizada, que servirá de base para a realização das análises posteriores.
- Suporte às QRYs e análises: Com a tabela `prata_maratona.results` disponível, os analistas poderão formular consultas SQL (QRYs) e realizar análises mais aprofundadas sobre os dados, visando a geração de relatórios ad-hoc.

Quality Assessment Modelo OBT

3.5.- Schema / Quality Assessment OBT:

```

14:14 (246) Python [Python] [Run]
from pyspark.sql import SparkSession
import pyspark.sql.functions as F
from pyspark.sql.utils import AnalysisException

# Inicializar SparkSession
spark = SparkSession.builder \
    .appName("UnionAndSaveAsDeltaTableWithQualityChecks") \
    .config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension") \
    .config("spark.sql.catalog.spark_catalog", "org.apache.spark.sql.delta.catalog.DeltaCatalog") \
    .getOrCreate()

# Funções de Análise e Limpeza de Qualidade dos Dados
def check_missing_values(df):
    """Função para verificar valores ausentes."""
    return df.select([F.count(F.when(F.isnan(c)) | F.col(c).isNull(), c)).alias(c) for c in df.columns])

def check_duplicates(df):
    """Função para verificar duplicatas."""
    duplicate_count = df.count() - df.dropDuplicates().count()
    return duplicate_count

def check_data_types(df, expected_schema):
    """Função para verificar tipos de dados."""
    actual_schema = df.dtypes
    mismatched_types = [(col, actual, expected) for (col, actual), (col, expected) in zip(actual_schema, expected_schema)
                        if actual != expected]
    return mismatched_types

def check_value_ranges(df, column_ranges):
    """Função para verificar faixas de valores."""
    out_of_range = {}
    for column, (min_val, max_val) in column_ranges.items():
        out_of_range[column] = df.filter((F.col(column) < min_val) | (F.col(column) > max_val)).count()
    return out_of_range

def clean_data(df, column_ranges):
    """Função para limpar dados fora da faixa e valores ausentes."""
    for column, (min_val, max_val) in column_ranges.items():
        df = df.filter((F.col(column) >= min_val) & (F.col(column) <= max_val))
    df = df.dropna()
    df = df.dropDuplicates()
    return df

# Ler os dados das tabelas fonte com tratamento de exceções
try:
    mens_df = spark.read.format("delta").table("bronze_maratona.mens")
    mulheres_df = spark.read.format("delta").table("bronze_maratona.womens")
    wheelchair_mens_df = spark.read.format("delta").table("bronze_maratona.wheelchair_mens")
    wheelchair_womens_df = spark.read.format("delta").table("bronze_maratona.wheelchair_womens")
except AnalysisException as e:
    print(f"Erro ao ler as tabelas fonte: {e}")
    raise

# Função para selecionar e renomear colunas, corrigindo o ano
def select_and_rename_columns(df, race_category):
    return df.select([
        F.list(race_category).alias("race_category"),
        F.substring(F.col("Year"), 1, 4).cast("int").alias("Year"),
        F.col("Athlete").cast("string").alias("Athlete"),
        F.col("Nationality").cast("string").alias("Nationality"),
        F.col("Time").cast("string").alias("Time"),
        (F.unix_timestamp(F.col("Time"), 'H:m:ss') - F.unix_timestamp(F.lit('0:00:00'), 'H:m:ss')).cast("double").alias("duracao_em_segundos")
    ])

# Aplicar a função para cada DataFrame
mens_df = select_and_rename_columns(mens_df, "mens")
mulheres_df = select_and_rename_columns(mulheres_df, "womens")
wheelchair_mens_df = select_and_rename_columns(wheelchair_mens_df, "wheelchair_mens")
wheelchair_womens_df = select_and_rename_columns(wheelchair_womens_df, "wheelchair_womens")

# Unir os DataFrames
combined_df = mens_df.union(mulheres_df).union(wheelchair_mens_df).union(wheelchair_womens_df)

# Verificações de Qualidade dos Dados
# Definir esquema esperado (exemplo)
expected_schema = [{"race_category": "string", "Year": "int", "Athlete": "string", "Nationality": "string", "Time": "string", "duracao_em_segundos": "double"}]

# Definir faixas de valores esperadas (exemplo)
column_ranges = {
    "Year": (1998, 2025), # Exemplo de intervalo de anos
    "duracao_em_segundos": (0, 100000) # Exemplo de intervalo de segundos
}

# Verificar duplicatas
duplicates = check_duplicates(combined_df)

# Verificar tipos de dados
mismatched_types = check_data_types(combined_df, expected_schema)

# Verificar faixas de valores
out_of_range_values = check_value_ranges(combined_df, column_ranges)

# Verificar valores ausentes
missing_values = check_missing_values(combined_df)

# Relatório de Qualidade dos Dados
report = """
Análise de Qualidade dos Dados - Camada Prata

Objetivo:
Garantir a precisão, consistência, integridade e completude dos dados após a limpeza inicial na camada Prata.

Verificação de Valores Ausentes:
[Row(race_category=0, Year=0, Athlete=0, Nationality=0, Time=0, duracao_em_segundos=0)]

Verificação de Duplicatas:
Número de duplicatas: 0

Verificação de Tipos de Dados:
Tipos de dados incompatíveis: []

Verificação de Faixas de Valores:
Valores fora da faixa esperada: {out_of_range_values}
"""

print(report)

# Limpar dados antes de salvar
cleaned_df = clean_data(combined_df, column_ranges)

# Salvar como tabela Delta
cleaned_df.write.format("delta").mode("overwrite").saveAsTable("prata_maratona.results")

```

(18) jobs Spark

mens_df: pyspark.sql.dataframe.DataFrame
race_category: string
Year: integer
Athlete: string
Nationality: string
Time: string
duracao_em_segundos: double

womenes_df: pyspark.sql.dataframe.DataFrame
race_category: string
Year: integer
Athlete: string
Nationality: string
Time: string
duracao_em_segundos: double

wheelchair_mens_df: pyspark.sql.dataframe.DataFrame
race_category: string
Year: integer
Athlete: string
Nationality: string
Time: string
duracao_em_segundos: double

wheelchair_womens_df: pyspark.sql.dataframe.DataFrame
race_category: string
Year: integer
Athlete: string
Nationality: string
Time: string
duracao_em_segundos: double

combined_df: pyspark.sql.dataframe.DataFrame
race_category: string
Year: integer
Athlete: string
Nationality: string
Time: string
duracao_em_segundos: double

missing_values: pyspark.sql.dataframe.DataFrame
race_category: long
Year: long
Athlete: long
Nationality: long
Time: long
duracao_em_segundos: long

cleaned_df: pyspark.sql.dataframe.DataFrame
race_category: string
Year: integer
Athlete: string
Nationality: string
Time: string
duracao_em_segundos: double

Análise de Qualidade dos Dados - Camada Prata

Objetivo:
Garantir a precisão, consistência, integridade e completude dos dados após a limpeza inicial na camada Prata.

Verificação de Valores Ausentes:
[Row(race_category=0, Year=0, Athlete=0, Nationality=0, Time=0, duracao_em_segundos=0)]

Verificação de Duplicatas:
Número de duplicatas: 0

Verificação de Tipos de Dados:
Tipos de dados incompatíveis: []

Verificação de Faixas de Valores:
Valores fora da faixa esperada: {out_of_range_values}: 0, 'duracao_em_segundos': 0}

3.5 Documentação no Catálogo de Dados:

Esse catálogo de dados fornece uma visão detalhada das informações sobre a Maratona de Londres, incluindo colunas das tabelas em seus diversas etapas sobre os vencedores, tempos de conclusão e outras colunas.

Catálogo de Dados da Maratona de Londres				
Nome Tabela	Nome Coluna	Data Type	Range	Descrição
bronze_maratona.mens	Year	String	1981-2024	Ano no formato YYYY em que a Maratona de Londres aconteceu.
bronze_maratona.mens	Athlete	String	CHAR	Nome do Atleta
bronze_maratona.mens	Nationality	String	CHAR	Nacionalidade do atleta que participou na Maratona de Londres.
bronze_maratona.mens	Time	String	CHAR	Tempos de conclusão da maratona ,Categórico.
bronze_maratona.mens	Notes	String	CHAR	Observações relativas à participação no evento da Maratona de Londres.
bronze_maratona.mens	race_category	String	mens	Masculino Geral: Categórico, com os nomes dos vencedores.
bronze_maratona.mens	Bronze_ingestontimestamp	Timestamp	-	Coluna acrescentada (recurso Databricks para acompanhar atualização do dado) formato
bronze_maratona.womens	Year	Integer	1981-2024	Ano no formato YYYY em que a Maratona de Londres aconteceu.
bronze_maratona.womens	Athlete	String	CHAR	Nome do Atleta
bronze_maratona.womens	Nationality	String	CHAR	Nacionalidade do atleta que participou na Maratona de Londres.
bronze_maratona.womens	Time	String	CHAR	Tempos de conclusão da maratona ,Categórico.
bronze_maratona.womens	Notes	String	CHAR	Observações relativas à participação no evento da Maratona de Londres.
bronze_maratona.womens	race_category	String	womens	Feminino Geral: Categórico, com os nomes das vencedoras.
bronze_maratona.womens	Bronze_ingestontimestamp	Timestamp	-	Coluna acrescentada (recurso Databricks para acompanhar atualização do dado) formato
bronze_maratona,wheelchair_mens	Year	Integer	1983-2023	Ano no formato YYYY em que a Maratona de Londres aconteceu.
bronze_maratona.wheelchair_mens	Athlete	String	CHAR	Nome do Atleta
bronze_maratona.wheelchair_mens	Time	String	CHAR	Tempos de conclusão da maratona ,Categórico.
bronze_maratona.wheelchair_mens	Nationality	String	CHAR	Nacionalidade do atleta que participou na Maratona de Londres.
bronze_maratona.wheelchair_mens	Notes	String	CHAR	Observações relativas à participação no evento da Maratona de Londres.
bronze_maratona.wheelchair_mens	race_category	String	wheelchair_mens	Masculino Cadeira de Rodas: Categórico, com os nomes dos vencedores.
bronze_maratona.wheelchair_mens	Bronze_ingestontimestamp	Timestamp	-	Coluna acrescentada (recurso Databricks para acompanhar atualização do dado) formato
bronze_maratona,wheelchair_womens	Year	String	1983-2023	Ano no formato YYYY em que a Maratona de Londres aconteceu.
bronze_maratona.wheelchair_womens	Athlete	String	CHAR	Nome da Atleta
bronze_maratona.wheelchair_womens	Nationality	String	CHAR	Nacionalidade do atleta que participou na Maratona de Londres.
bronze_maratona.wheelchair_womens	Time	String	CHAR	
bronze_maratona.wheelchair_womens	Notes	String	CHAR	Observações relativas à participação no evento da Maratona de Londres.
bronze_maratona.wheelchair_womens	race_category	String	wheelchair_mens	Feminino Cadeira de Rodas: Categórico, com os nomes das vencedoras.
bronze_maratona.wheelchair_womens	Bronze_ingestontimestamp	Timestamp	-	Coluna acrescentada (recurso Databricks para acompanhar atualização do dado) formato
results	race_category	String	CHAR	
results	Year	Integer	1981-2024	Ano no formato YYYY em que a Maratona de Londres aconteceu.
results	Athlete	String	CHAR	Nome do Atleta
results	Nationality	String	CHAR	Nacionalidade do atleta que participou na Maratona de Londres.
results	Time	String	CHAR	Tempos de conclusão da maratona ,Categórico.
results	duracao_em_segundos	double	5024 - 16143	Tempos de conclusão da maratona ,double.
results	mens	double	7285 - 7908	Tempos de conclusão da maratona ,double.
results	womens	double	8125 - 9154	Tempos de conclusão da maratona ,double.
results	wheelchair_mens	double	5024 - 12007	Tempos de conclusão da maratona ,double.
results	wheelchair_womens	double	5904 - 16143	Tempos de conclusão da maratona ,double.

Carga :

Para compor um conjunto de dados, especialmente nesse contexto de um projeto de engenharia de dados, algumas técnicas serão utilizadas para garantir a qualidade, integridade, e utilidade dos dados:

Ingestão de Dados usaremos Batch Processing (Usado para dados que não mudam com muita frequência), onde carregamos um volume de dados em intervalos regulares como no caso anual.

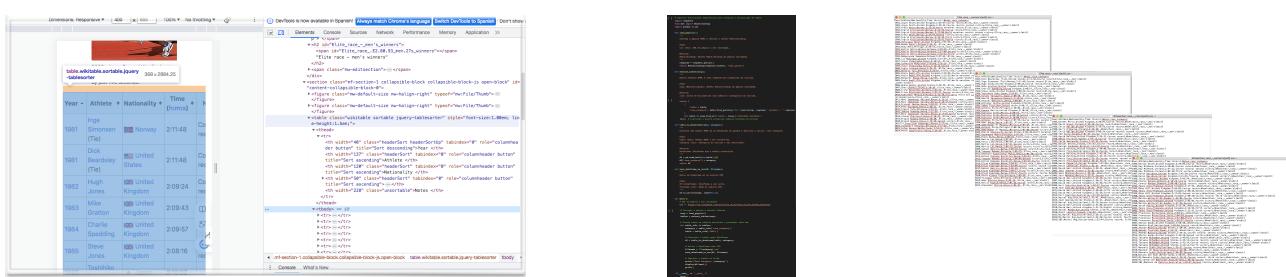
Esse processo visa atender às minhas necessidades de análise dos dados da Maratona de Londres. Neste relatório, detalharei as etapas desse processo, desde a extração dos dados até a disponibilização das informações para consultas Ad-Hoc.

Processamento em Lote (Batch Processing)

O processamento em lote (batch processing) é a abordagem adotada neste cenário, uma vez que os dados da Maratona de Londres são coletados e processados, uma vez por ano. Essa abordagem em lote é eficiente em termos de recursos computacionais, embora os dados não estejam disponíveis para análise imediatamente após a coleta.

Etapa de Extração com Scraping:

Inicialmente, utilizei um script de crawler em Python para extrair dados relevantes sobre a Maratona de Londres a partir de URLs da Wikipedia. Esse processo gerou quatro arquivos CSV como resultado



Etapa de Upload Manual dos Arquivos para o Data Lake:

Com os arquivos CSV em mãos, realizei o upload manual desses dados diretamente para a landing zone do meu Data Lake, utilizando a interface do Azure. Essa foi a forma mais simples de ingerir os dados iniciais no meu ambiente de armazenamento na nuvem.

Azure: maratona_londres/landing-zone

Azure: maratona_londres/Elite_race__mens.csv

BRX:/mnt/adlsaulasdatabrickssa/landing-zone

Etapa de Ingestão na Camada Bronze

Após o upload, utilizei o Databricks (PySpark) para ingerir os arquivos CSV na storage account do Data Lake da Azure. Essa camada Bronze armazena os dados brutos em formato Delta Lake, tanto no Data Lake quanto em um banco de dados Databricks. A Tabela Delta é preservada com seus dados em seu estado original. Essa tabela Delta é persistida no Data Lake, mantendo as propriedades ACID e preservando os dados brutos.

Azure: adsaulasdatabrickssa/nconteineres/bronze/bronze_maratona

BRX: bronze/Delta table

BRX: DB_bronze_maratona/mens_Delta table

Etapa de Ingestão na Camada Prata

Em seguida, utilizei novamente o Databricks (com PySpark ou SQL) para processar a tabela Delta da camada Bronze e criar uma tabela Delta mais limpa e organizada na camada Prata do Data Lake.

Na camada Prata, realizei a maior parte do trabalho de transformação e limpeza dos dados, deixando-os prontos para consultas simples com Spark SQL.

Azure: ..results/.delta_log(updated_records read_last) Onde é formada a tabela Delta parquet

..prata_maratona: Nome da Data Base (Schema Created no DBX)

A arquitetura de Data Lakehouse implementada, com as camadas Bronze, Prata e a ausência da camada Ouro, mostrou-se adequada para atender às necessidades de processamento e consulta dos dados da Maratona de Londres. Esse design permite que os dados sejam ingeridos, transformados e disponibilizados de forma eficiente, preparando-os para as análises necessárias.

A implementação da arquitetura de Data Lakehouse com as camadas Bronze e Prata provou ser eficaz para a gestão dos dados da Maratona de Londres. A camada Bronze permite o armazenamento dos dados brutos em seu estado original, garantindo a preservação e a rastreabilidade. A camada Prata, por sua vez, realiza as transformações e limpezas necessárias, resultando em dados prontos para consultas e análises eficientes.

Embora a criação da camada Ouro não tenha sido necessária neste projeto, sua adição futura pode ser considerada caso haja a necessidade de enriquecer os dados com informações adicionais, como temperatura média ou idade dos atletas. A flexibilidade da arquitetura de Data Lakehouse permite essa expansão conforme as necessidades do negócio evoluem.

Em resumo, a arquitetura adotada proporciona um fluxo de dados robusto e escalável, capaz de atender tanto às demandas atuais quanto às futuras, garantindo a qualidade e a integridade dos dados para a geração de insights valiosos.

Análise Ad-Hoc:

Análise Ad-Hoc: Introdução

A análise ad-hoc desempenha um papel importante no processo de análise de dados. Ela permite a realização de consultas e exploração de dados de maneira flexível e iterativa, sem a necessidade de seguir um fluxo de trabalho predefinido.

No contexto da persona Databricks SQL, permite a manipulação de dados estruturados, utilizando uma versão de alto nível baseada no Apache Spark SQL. Isso possibilita a realização de consultas usando a API (notebooks) e a sintaxe do SQL.

As perguntas de análise descritiva, tanto em SQL quanto em PySpark, podem ser feitas diretamente na camada Prata do Data Lake. Os insights obtidos a partir dessas análises ad-hoc serão então utilizados para desenvolver um script de Storytelling, respondendo às questões levantadas no objetivos específicos.

Dessa forma, a análise ad-hoc se torna uma etapa importante no processo de exploração e entendimento dos dados, complementando as transformações realizadas nas camadas prata do Data Lake. Mas antes é preciso antes realizar uma análise de qualidade nas células a seguir :

```
# Verificações de Qualidade dos Dados
# Definir esquema esperado (exemplo)
expected_schema = ("race_category", "string"), ("Year", "int"), ("Athlete", "string"), ("Nationality", "string"),
("Time", "string"), ("duracao_em_segundos", "double"))

# Definir faixas de valores esperadas (exemplo)
column_ranges = {
    "Year": (1988, 2025), # Exemplo de intervalo de anos
    "duracao_em_segundos": (0, 100000) # Exemplo de intervalo de segundos
}

# Verificar duplicatas
duplicates = check_duplicates(combined_df)

# Verificar tipos de dados
mismatched_types = check_data_types(combined_df, expected_schema)

# Verificar faixas de valores
out_of_range_values = check_value_ranges(combined_df, column_ranges)

# Verificar valores ausentes
missing_values = check_missing_values(combined_df)

# Relatório de Qualidade dos Dados
report = """
Análise de Qualidade dos Dados - Camada Prata

Objetivo:
Garantir a precisão, consistência, integridade e completude dos dados após a limpeza inicial na camada Prata.

Verificação de Valores Ausentes:


|    | race_category | Year | Athlete                    | Nationality    | Time    | duracao_em_segundos |
|----|---------------|------|----------------------------|----------------|---------|---------------------|
| 1  | mens          | 1987 | Hironi Taniguchi           | Japan          | 2:09:50 | 7790                |
| 2  | mens          | 2000 | António Pinto              | Portugal       | 2:06:36 | 7596                |
| 3  | mens          | 2017 | Daniel Wanjuhi             | Kenya          | 2:05:48 | 7548                |
| 4  | mens          | 2016 | Ellud Kipchoge             | Kenya          | 2:03:05 | 7385                |
| 5  | mens          | 1982 | Hugh Jones                 | United Kingdom | 2:09:24 | 7764                |
| 6  | mens          | 2012 | Wilson Kipsang Kiprotich   | Kenya          | 2:04:44 | 7484                |
| 7  | mens          | 2013 | Tsegaye Kebede             | Ethiopia       | 2:06:04 | 7564                |
| 8  | mens          | 2008 | Martin Lel                 | Kenya          | 2:05:15 | 7515                |
| 9  | mens          | 1994 | Dionicio Cerdán            | Mexico         | 2:08:53 | 7733                |
| 10 | mens          | 2019 | Ellud Kipchoge             | Kenya          | 2:02:37 | 7357                |
| 11 | mens          | 1981 | Dick Beardsley (Te)        | United States  | 2:11:48 | 7908                |
| 12 | mens          | 1989 | Douglas Wakiihuri          | Kenya          | 2:09:03 | 7743                |
| 13 | mens          | 2011 | Emmanuel Kipchirchir Mu... | Kenya          | 2:04:40 | 7480                |
| 14 | mens          | 1988 | Henrik Jørgensen           | Denmark        | 2:10:20 | 7820                |
| 15 | mens          | 1991 | Yakov Tolstikov            | Soviet Union   | 2:09:17 | 7757                |



171 linhas | 1,29 segundo de runtime



Atualizada há ontem



Análise de Qualidade dos Dados - Camada Prata



Objetivo:  
Garantir a precisão, consistência, integridade e completude dos dados após a limpeza inicial na camada Prata.



Verificação de Valores Ausentes:  
[None]



Verificação de Duplicatas:  
Número de duplicatas: 0



Verificação de Tipos de Dados:  
Tipos de dados incompatíveis: []



Verificação de Faixas de Valores:  
Valores fora da faixa esperada: {Year: 0, 'duracao_em_segundos': 0}


```

Análise de Dados da Maratona

1. Qual país ganhou o maior número de competições?

Esta consulta identifica o país com o maior número de vitórias na maratona, agrupando os resultados por nacionalidade e contando o número de vitórias.

```
▶ ✓ Ontem (2s)
-- Seleciona a nacionalidade e conta o número de vitórias
SELECT
    Nationality, COUNT(*) AS num_vitorias
    -- Da tabela de resultados da maratona
FROM prata_maratona.results
    -- Agrupa os resultados por nacionalidade
    GROUP BY Nationality
    -- Ordena os resultados em ordem decrescente pelo número de vitórias
    ORDER BY num_vitorias DESC
    -- Limita o resultado a 1 país (o com maior número de vitórias)
LIMIT 1;

(2) jobs Spark
└─ _sqldf: pyspark.sql.DataFrame = [Nationality: string, num_vitorias: long]

Tabela v +
| # Nationality | num_vitorias |
| 1 United Kingdom | 44 |

```

↓ 1 linha | 1,91 segundo de runtime

Atualizada há ontem

```
▶ ✓ Ontem (2s)
from pyspark.sql import SparkSession
import pyspark.sql.functions as F

# Cria uma sessão Spark
spark = SparkSession.builder.appName("MarathonAnalysis").getOrCreate()

# Lê a tabela de resultados da maratona
df = spark.table("prata_maratona.results")

# Agrupa os dados por nacionalidade, conta o número de vitórias, ordena por número de vitórias em ordem decrescente
result = df.groupBy("Nationality").agg(F.count("*").alias("num_vitorias")) \
    .orderBy(F.col("num_vitorias").desc()).limit(1)

# Exibe o resultado
display(result)

(2) jobs Spark
└─ df: pyspark.sql.DataFrame = [race_category: string, Year: integer ... mais 4 campos]
└─ result: pyspark.sql.DataFrame = [Nationality: string, num_vitorias: long]

Tabela v +
| # Nationality | num_vitorias |
| 1 United Kingdom | 44 |

```

↓ 1 linha | 1,58 segundo de runtime

Atualizada há ontem

2. Em que ano os atletas britânicos venceram a maratona pela última vez?

Esta consulta encontra o ano mais recente em que um atleta do Reino Unido venceu a maratona, filtrando por nacionalidade e obtendo o ano máximo.

```
▶ ✓ Ontem (1s)
-- Seleciona o ano máximo de vitória para atletas do Reino Unido
SELECT
    MAX(Year) AS ultimo_ano
    -- Da tabela de resultados da maratona
FROM prata_maratona.results
    -- Onde a nacionalidade é do Reino Unido
WHERE Nationality = 'United Kingdom';

(1) jobs Spark
└─ _sqldf: pyspark.sql.DataFrame = [ultimo_ano: integer]

Tabela v +
| # ultimo_ano |
| 1 2018 |

```

↓ 1 linha | 1,60 segundo de runtime

Atualizada há ontem

Python

▶ ✓ Ontem (1s)
Filtra os resultados para atletas do Reino Unido
Encontra o ano máximo de vitória e renomeia a coluna como 'ultimo_ano'
result = df.filter(F.col("Nationality") == "United Kingdom") \
 .agg(F.max("Year").alias("ultimo_ano"))

Exibe o resultado
display(result)

(1) jobs Spark
└─ result: pyspark.sql.DataFrame = [ultimo_ano: integer]

Tabela v +
| # ultimo_ano |
| 1 2018 |

↓ 1 linha | 1,49 segundo de runtime

Atualizada há ontem

5. Quem detém o recorde mundial feminino e em que ano foi estabelecido?

Para encontrar o recorde mundial feminino e o ano em que foi estabelecido, agrupamos por atleta, nacionalidade e ano, selecionando o menor tempo e formatando o tempo em uma string no formato "HH horas, MM minutos, SS segundos".

```
▶ ✅ Ontem (2x) 14 SQL □ ...
— Seleciona a atleta, nacionalidade, ano e o menor tempo registrado na categoria feminina
SELECT Athlete, Nationality, Year,
       MIN(duracao_em_segundos) AS recorde,
       — Concatena o tempo em segundos em um formato amigável
       CONCAT(
           FLOOR(MIN(duracao_em_segundos) / 3600), ' horas, ',
           FLOOR((MIN(duracao_em_segundos) % 3600) / 60), ' minutos, ',
           MIN(duracao_em_segundos) % 60, ' segundos'
       ) AS recorde_formatado
— Da tabela de resultados da maratona
FROM prata_maratona.results
— Onde a categoria de corrida é feminina
WHERE race_category = 'women'
— Agrupa por atleta, nacionalidade e ano
GROUP BY Athlete, Nationality, Year
— Ordena pelo menor tempo registrado
ORDER BY recorde ASC
— Limita o resultado à atleta com o menor tempo registrado
LIMIT 1;

```

» (2) jobs Spark

```
▶ sql: pyspark.sql.DataFrame = [Athlete: string, Nationality: string ... mais 3 campos]
```

Athlete	Nationality	Year	recorde	recorde_formatado
Paula Radcliffe	United Kingdom	2003	8125	2 horas, 15 minutos, 25.0 segundos

↓ 1 linha | 2,30 segundos de runtime Atualizada há ontem

```
▶ ✅ Ontem (2x) 15 Python □ ...
# Filtra os resultados para a categoria feminina
# Agrupa por atleta, nacionalidade e ano
# Encontra o menor tempo registrado e forma o tempo em uma string amigável
result = df.filter(F.col("race_category") == "womens") \
    .groupby("Athlete", "Nationality", "Year") \
    .agg(
        F.min("duracao_em_segundos").alias("recorde"),
        F.concat(
            F.floor(F.min("duracao_em_segundos") / 3600).cast("string"), F.lit(" horas, "),
            F.floor(F.min("duracao_em_segundos") % 3600) / 60).cast("string"), F.lit(" minutos, "),
            F.min("duracao_em_segundos") % 60).alias("recorde_formatado")
    ) \
    .orderBy(F.col("recorde").asc())
# Exibe o resultado
display(result.limit(1))

```

» (2) jobs Spark

```
▶ result: pyspark.sql.DataFrame = [Athlete: string, Nationality: string ... mais 3 campos]
```

Athlete	Nationality	Year	recorde	recorde_formatado
Paula Radcliffe	United Kingdom	2003	8125	2 horas, 15 minutos, 25.0 segundos

↓ 1 linha | 1,90 segundo de runtime Atualizada há ontem

6. Quais corredores das diversas categorias tiveram múltiplas vitórias na competição?

Esta consulta lista os corredores que tiveram mais de uma vitória em diferentes categorias, agrupando por atleta, nacionalidade e categoria, e contando o número de vitórias.

```
▶ ✅ Ontem (3x) 17 SQL □ ...
— Seleciona o atleta, nacionalidade, categoria de corrida e conta o número de vitórias
SELECT Athlete, Nationality, race_category, COUNT(*) AS num_vitorias
— Da tabela de resultados da maratona
FROM prata_maratona.results
— Agrupa por atleta, nacionalidade e categoria de corrida
GROUP BY Athlete, Nationality, race_category
— Filtra para mostrar apenas aqueles com mais de uma vitória
HAVING num_vitorias > 1
— Ordena pelo número de vitórias em ordem decrescente
ORDER BY num_vitorias DESC;

```

» (2) jobs Spark

```
▶ result: pyspark.sql.DataFrame = [Athlete: string, Nationality: string ... mais 2 campos]
```

Athlete	Nationality	race_category	num_vitorias
David Weil	United Kingdom	wheelchair_mens	8
Tanni Grey-Thompson	United Kingdom	wheelchair_womens	6
Marcel Hug	Switzerland	wheelchair_mens	5
Francesca Porcellato	Italy	wheelchair_womens	4
Tayana McFadden	United States	wheelchair_womens	4
David Holding	United Kingdom	wheelchair_mens	4
Ellid Kipchoge	Kenya	mens	4
Ingrid Kristiansen	Norway	womens	4
Chris Hallam	United Kingdom	wheelchair_mens	2
Wilson Kipsang Kiprotich	Kenya	mens	2
Seúl Mendoza	Mexico	wheelchair_mens	2
Abdelkader El Mouaziz	Morocco	mens	2
Rose Hill	United Kingdom	wheelchair_womens	2
Amanda McGrory	United States	wheelchair_womens	2
Karen Davidson	United Kingdom	wheelchair_womens	2

↓ 35 linhas | 3,00 segundos de runtime Atualizada há ontem

3. Qual foi o ano da primeira corrida em cadeira de rodas?

Para identificar o ano da primeira corrida em cadeira de rodas, filtramos pela categoria de corrida correspondente e encontramos o ano mais antigo.

```
▶ ✅ Ontem (2x) 8 SQL □ ...
— Seleciona o ano mínimo (mais antigo) para a categoria de corrida em cadeira de rodas
SELECT
    MIN(Year) AS primeiro_ano
— Da tabela de resultados da maratona
FROM prata_maratona.results
— Onde a categoria de corrida é cadeira de rodas
WHERE race_category LIKE "wheelchair%";

```

» (2) jobs Spark

```
▶ result: pyspark.sql.DataFrame = [primeiro_ano: integer]
```

primeiro_ano
1983

↓ 1 linha | 1,59 segundo de runtime Atualizada há ontem

```
▶ ✅ Ontem (1x) 9 Python □ ...
# Filtra os resultados para a categoria de corrida em cadeira de rodas
# Encontra o ano mínimo (mais antigo) e renomeia a coluna como 'primeiro_ano'
result = df.filter(F.col("race_category").like("wheelchair%")) \
    .agg(F.min("Year").alias("primeiro_ano"))

# Exibe o resultado
display(result)

```

» (2) jobs Spark

```
▶ result: pyspark.sql.DataFrame = [primeiro_ano: integer]
```

primeiro_ano
1983

↓ 1 linha | 1,29 segundo de runtime Atualizada há ontem

4. Qual maratonista do gênero masculino estabeleceu um novo recorde e dominou a competição até hoje?

Esta consulta identifica o maratonista masculino com o menor tempo registrado na categoria masculina, agrupando por atleta, nacionalidade, e também converte o tempo de duração em uma string no formato "HH horas, MM minutos, SS segundos".

```
▶ ✅ Ontem (24) 11 SQL □ ...
— Seleciona o atleta, nacionalidade e o menor tempo registrado na categoria masculina
SELECT Athlete, Nationality,
       MIN(duracao_em_segundos) AS recorde,
       — Concatena o tempo em segundos em um formato amigável
       CONCAT(
           FLOOR(MIN(duracao_em_segundos) / 3600), ' horas, ',
           FLOOR((MIN(duracao_em_segundos) % 3600) / 60), ' minutos, ',
           MIN(duracao_em_segundos) % 60, ' segundos'
       ) AS recorde_formatado
— Da tabela de resultados da maratona
FROM prata_maratona.results
— Onde a categoria de corrida é masculina
WHERE race_category = 'mens'
— Agrupa por atleta e nacionalidade
GROUP BY Athlete, Nationality
— Ordena pelo menor tempo registrado
ORDER BY recorde ASC
— Limita o resultado ao atleta com o menor tempo registrado
LIMIT 1;

```

» (2) jobs Spark

```
▶ result: pyspark.sql.DataFrame = [Athlete: string, Nationality: string ... mais 2 campos]
```

Athlete	Nationality	recorde	recorde_formatado
Kelvin Kiptum	Kenya	7285	2 horas, 1 minutos, 25.0 segundos

↓ 1 linha | 2,50 segundos de runtime Atualizada há ontem

» Código + Texto

```
▶ ✅ Ontem (3x) 12 SQL □ ...
# Filtra os resultados para a categoria masculina
# Agrupa por atleta e nacionalidade
# Encontra o menor tempo registrado e forma o tempo em uma string amigável
result = df.filter(F.col("race_category") == "mens") \
    .groupby("Athlete", "Nationality") \
    .agg(
        F.min("duracao_em_segundos").alias("recorde"),
        F.concat(
            F.floor(F.min("duracao_em_segundos") / 3600).cast("string"), F.lit(" horas, "),
            F.floor(F.min("duracao_em_segundos") % 3600) / 60).cast("string"), F.lit(" minutos, "),
            F.min("duracao_em_segundos") % 60).alias("recorde_formatado")
    ) \
    .orderBy(F.col("recorde").asc())
# Exibe o resultado
display(result.limit(1))

```

» (2) jobs Spark

```
▶ result: pyspark.sql.DataFrame = [Athlete: string, Nationality: string ... mais 2 campos]
```

Athlete	Nationality	recorde	recorde_formatado
Kelvin Kiptum	Kenya	7285	2 horas, 1 minutos, 25.0 segundos

↓ 1 linha | 2,91 segundos de runtime Atualizada há ontem

Roteiro (Storitelling), baseado na análise das respostas obtidas:

Olá, meus amigos! Sou Antônio, o comentarista esportivo especializado em maratonas 42k, e hoje venho trazer para vocês um emocionante relato sobre a Maratona de Londres, um dos eventos mais aguardados do calendário esportivo mundial.

Ano após ano, a Maratona de Londres atrai os melhores atletas do mundo, que buscam não apenas a vitória, mas também a chance de estabelecer novos recordes e escrever seus nomes na história dessa lendária prova. E é exatamente sobre essa história que vamos mergulhar agora, explorando os dados que revelam os principais destaques e curiosidades dessa competição tão icônica.

Vamos começar com a pergunta que todos se fazem: Qual país ganhou o maior número de competições? Após analisar os dados, podemos afirmar com convicção que o United Kingdom é o grande dominador da Maratona de Londres, com um número impressionante de vitórias. Essa supremacia Britânica é fruto de uma tradição de excelência no atletismo de longa distância, que transforma essa nação em uma verdadeira potência mundial quando o assunto é maratona.

Agora, vamos falar mais um pouco sobre os atletas britânicos. Quando foi a última vez que eles venceram a prova em casa? Após uma análise cuidadosa, descobrimos que a última vez que um atleta do Reino Unido subiu ao topo do pódio foi em 2018. Desde então, os corredores locais têm enfrentado dificuldades para acompanhar o ritmo dos atletas de outras nacionalidades, especialmente dos kenianos, que dominam a competição.

Mas a Maratona de Londres não é apenas sobre a disputa entre os melhores atletas do mundo. Ela também é palco de outras modalidades, como a corrida em cadeira de rodas. E quando foi a primeira vez que essa categoria fez parte do evento? Nossos dados revelam que a primeira edição da prova em cadeira de rodas aconteceu em 1983, adicionando ainda mais emoção e diversidade a essa competição.

E falando em recordes, não podemos deixar de mencionar o nome de Kevin Kiptum o maratonista do gênero masculino que estabeleceu um novo recorde e dominou a competição até hoje. Esse atleta queniano é simplesmente fenomenal, com um domínio avassalador na Maratona de Londres, onde conquistou vitórias consecutivas e estabeleceu marcas que parecem inalcançáveis.

Não podemos esquecer também das mulheres, que têm seu próprio destaque nessa prova. E quem detém o recorde mundial feminino? Nada mais, nada menos que a lendária Paula Radcliffe, que estabeleceu essa marca impressionante em 2003.

Por fim, não poderíamos deixar de mencionar os atletas que conquistaram múltiplas vitórias na Maratona de Londres. Nomes como David Weir, Tanny Grey Thomson e Marcel Hug se destacam por terem dominado a competição em diferentes categorias, deixando sua marca indelével nesse evento que é sinônimo de excelência no mundo do atletismo.

Meus amigos, a Maratona de Londres é muito mais do que apenas uma prova de resistência física. Ela é um palco onde a história do esporte é escrita, onde atletas de todo o mundo se encontram para escrever suas próprias lendas. E é com grande emoção que acompanhamos essa jornada, celebrando as conquistas e os recordes que tornam essa competição tão especial.

Até a próxima, meus amigos. Fiquem ligados, pois a Maratona de Londres sempre reserva grandes surpresas!

Conclusão

Hoje a era dos dados, da informação e do conhecimento nos traz promessas que deverão refletir à favor da empregabilidade do indivíduo e no relacionamento com empresas, bem como com a sociedade como um todo. No momento e agora ela já trouxe um novo mundo, uma ampliação de horizontes e mudanças do presente do Big Data 3Vs (Volume, Variedade, Velocidade) , estimasse acima de 1×10^{12} gigabytes para o 2025.

A trilha é íngreme, é necessário planejar com antecedência uma solução abrangente baseada em engenharia de dados.

Em minha jornada de aprendizado passo por várias etapas, tais como: definir um objetivo claro, capturar, reunir dados, analisar e entender, formular as perguntas certas, definir tarefas, encontrar as plataformas necessários para resolver essas questões e apresentar os dados com clareza, algumas tarefas mais exigentes, como se familiarizar com a dinâmicas de trabalho por dentro da arquitetura do Databricks; Control Plane (UI, Cluster Management, Notebooks, Scheduling) - Data Plane (Data Storage, Spark Cluster) , no entanto, como aluno foi uma experiência para aprender até encontrar novas funcionalidades e descobertas, iluminando minha caminhada.

Segundo o mais recente relatório da Gartner sobre o mercado de plataformas de dados, plataformas como a Databricks têm hoje uma melhor relação custo-benefício (The Time Value of Data). De acordo com a Gartner, essas plataformas apresentam uma maior penetração e participação no mercado de engenharia de dados, devido às suas propostas diferenciadoras para processamento paralelo de dados não estruturado, semi e estruturados que recolheu o melhor dos dois mundos DW e DL referida como Lakehouse, essencialmente, combina a eficiência de custos de um Data Lake com as vantagens da gestão de dados de um Data Warehouse, incluindo estruturas de dados e segurança de acesso. e à experiência do usuário (UX) facilitada, promovendo a democratização dos processos de trabalho nos segmentos atendidos para diferentes personas no ecossistemas de dados.

O relatório da Gartner destaca que as plataformas de dados líderes do mercado, como a Databricks, oferecem uma combinação atrativa de recursos e funcionalidades, tornando-as soluções mais acessíveis e vantajosas para as organizações que buscam aproveitar o valor dos seus dados. Essas plataformas se destacam pela capacidade de simplificar e agilizar os fluxos de trabalho de engenharia de dados, impulsionando a adoção em diversos setores.

Portanto, as recomendações da Gartner indicam que as plataformas de dados modernas, como a Databricks, apresentam um melhor custo-benefício e maior penetração no mercado, tornando-as opções atraentes para as empresas que desejam democratizar e acelerar seus processos de governança e análise de dados.



Historia e Diferenciais do Databricks:

Visão Geral:

Databricks é uma plataforma de análise de dados baseada no Apache Spark, que se fundamenta no Hadoop. Hadoop foi criado para resolver problemas de tolerância a falhas e escalabilidade, mas dependia de operações em disco, tornando o processamento lento.

Apache Spark:

Lançado em 2012, Spark permite processamento de dados na memória, acelerando o desempenho (*i100 x faster than Mapreduce*). Ele introduziu DataFrames, que facilitam a manipulação de dados, e inclui capacidades de aprendizado de máquina, grafos e streaming em tempo real.

Evolução para Databricks:

Em 2013, Spark tornou-se um projeto Apache, e Databricks foi fundado pelos desenvolvedores originais do Spark. Databricks oferece um ecossistema completo de serviços gerenciados, facilitando a gestão de clusters, tabelas de dados, credenciais, permissões e integração com notebooks.

Elementos Principais:

- Spark: Framework de computação distribuída
- Delta Lake: Camada de persistência com propriedades ACID.
- MLflow: Gerenciamento de modelos de aprendizado

Em resumo Databricks oferece uma plataforma de dados unificada com ingestão de dados gerenciada, detecção de esquema, aplicação e evolução, combinada com um fluxo de dados declarativo e de escalonamento automático, integrada com um orquestrador nativo de lakehouse que suporta todos os tipos de fluxos de trabalho.

*Considerações para futuras análises:

Outras análises de arquiteturas com estruturas mais robustas e mais recursos poderiam ser viáveis com a criação de uma automação ponta a ponta. Por exemplo, pode-se executar a extração de dados da Wikipedia com Python diretamente no Databricks (DBX), salvar os arquivos CSV no Data Lake e integrar diretamente com plataformas de visualização como Power BI ou Tableau.

Além da análise descritiva proposta anteriormente, outras análises mais sofisticadas também seriam viáveis, agregando mais valor. Seria possível expandir a análise dos dados da Maratona de Londres para incluir outras métricas , ou até mesmo dados de pulseiras smartwatch com informações de monitoramento de sinais vitais.

Algumas métricas adicionais que poderiam ser avaliadas em uma análise multivariada incluem:

- Frequência cardíaca
- Ritmo e velocidade
- Distância e padrões de passo
- Níveis de oxigenação do sangue (SpO2)
- Gasto calórico
- Variabilidade da frequência cardíaca (HRV)

Dessa forma, a análise poderia ser ampliada de uma análise descritiva básica para uma análise preditiva (usando machine learning), explorando métricas de desempenho mais detalhadas, bem como tendências e evolução do evento ao longo dos anos. Essa abordagem mais abrangente pode fornecer insights valiosos para a compreensão mais profunda da Maratona de Londres.

Estruturas e Plataformas para a Implementação do MVP :

Databricks Community versus Databricks Free Trial e uma Storage Account no Azure Free:

- Oferece acesso à plataforma por um período ilimitado, o que, segundo meu cronograma, ótimo para cumprir o prazo de entrega do MVP na Sprint III.
- No Databricks Community, as transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade) são fornecidas pelo Delta Lake, uma camada de armazenamento otimizada construída sobre o Apache Spark.

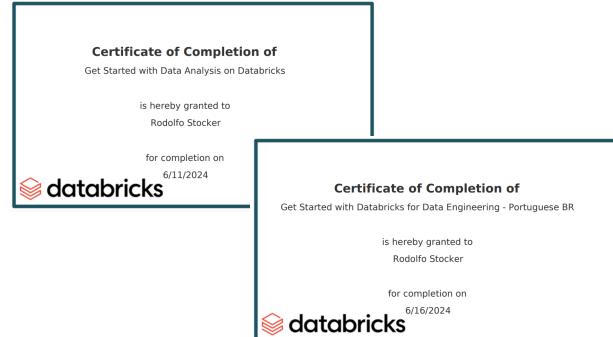
Autoavaliação

Principais Desafios para os iniciantes em na disciplina MVP da disciplina engenharia de dados sprint III :

Os usuários precisam se familiarizar com ingestão de dados , workspaces, tabelas, delta lakes, unit catalogs, schemas, notebooks, e jobs, pipelines, repôs o que pode representar uma curva de aprendizado desafiadora.

Funcionalidades Não Disponíveis na conta Community:

- Delta Live Tables.
- Unity Catalog (se habilitado durante o trial).
- Databricks Jobs (Pipelines).
- Conectividade com ferramentas de BI como Power BI ou Tableau.
- Integração com serviços de CI/CD
- Recursos avançados de segurança e governança de dados.



Neste MVP, poderiam ter sido consideradas várias funcionalidades adicionais para robustecer o objetivo e atender melhor às expectativas. A versão paga do Databricks oferece recursos que a versão gratuita ou limitada não é capaz de proporcionar, como:

1. Criação de Pipelines Automatizados e Orquestração:

- Databricks Workflows: Permite a criação e a gestão de pipelines automatizados para ingerir, processar e analisar dados de forma eficiente.
- Azure Data Factory: Ideal para orquestração de tarefas e automação de fluxos de trabalho, incluindo a ingestão, transformação e movimentação de dados entre camadas.

2. Governança de Dados:

- Unity Catalog: Oferece governança de dados centralizada e segura, permitindo o gerenciamento de permissões e a rastreabilidade dos dados.

3. Arquitetura Medallion:

- Modelo Medallion: Utilização das camadas Bronze, Prata e Ouro para organizar e otimizar o fluxo de dados. A camada Bronze armazena dados brutos, a Prata refina e limpa os dados, e a Ouro fornece dados altamente otimizados para análise avançada.

4. Funcionalidades Adicionais na Versão Paga:

- Delta Live Tables: Facilita a criação de pipelines de dados incrementais, garantindo a consistência e a integridade dos dados.
- Ferramentas de Visualização e Análise Completa: Suporte a ferramentas de BI como Power BI e Tableau, permitindo análises mais detalhadas e visualizações interativas.
- Integração com Serviços de CI/CD: Suporte a práticas de DevOps para integração e entrega contínua de projetos de dados.
- Segurança Avançada: Recursos adicionais de segurança para proteção de dados e conformidade com regulamentações.

Considerações Finais

A versão paga do Databricks oferece uma gama de funcionalidades que poderiam aprimorar significativamente o MVP, fornecendo ferramentas avançadas de automação, governança e análise de dados. Utilizar a versão paga permitiria uma implementação mais robusta e escalável, alinhada com as melhores práticas de engenharia de dados.

Linhagem de Dados da das vitórias nas Maratona de Londres:

A linhagem dos dados (ou data lineage) no contexto da governança dos dados refere-se ao rastreamento do ciclo de vida completo dos dados dentro de uma organização, desde sua origem até o ponto de consumo. Isso inclui todas as transformações, movimentos e interações que os dados sofrem ao longo do tempo. A linhagem dos dados é crucial para garantir a qualidade, a segurança e a conformidade dos dados, e oferece várias vantagens:

Principais Componentes da Linhagem e Análise dos Dados

1. Origem dos Dados: Onde os dados são inicialmente capturados ou arquivos de entrada.

- Os dados sobre a Maratona de Londres são provenientes de fontes primárias, como o site oficial da maratona e artigos de notícias.

url_0: https://en.wikipedia.org/wiki/List_of_winners_of_the_London_Marathon

2. Transformações dos Dados: Os processos onde alteramos os dados, incluindo limpeza, agregação e conversão de formatos como por exemplo Time : timestamp (h:m:s) que foi normalizado e re categorizado desde str para numérico.

Os dados brutos coletados de fontes online precisaram passar por pre processamentos antes de serem armazenados no data base em um formato mais estruturado e otimizado na nuvem data lakehouse.

- Isso envolveu etapas como web scraping, limpeza de dados como droppar a coluna Notas, consolidação de informações de múltiplas fontes ou (race category) varias tabelas com o mesmo encabeçado e consolidações e enriquecimento posteriores foram realizados nas transformações no medallion prata no databricks.

3. Movimento dos Dados: Após coletados, os dados podem ser armazenados na nuvem Azure e carregados em um data lake na nuvem, como também no Databricks Delta Lake.

- Esse movimento de dados envolve processos de ELT (Extração, Carga e Transformação) para mover os dados entre diferentes sistemas e camada da arquitetura medallion (Bronze-Prata).

- O fluxo dos dados entre sistemas camadas do Medallion, e incluindo ELT (extração, transformação e carga),

4. Destino dos Dados: Onde os dados são armazenados e consumidos, como lakehose no databricks, dashboards de BI (Business Intelligence) e relatórios Ad-Hoc.

5. Consumidores dos Dados: Quem ou o que está utilizando os dados, incluindo possivelmente analistas de dados, revisores dos MVP como monitores da disciplina e usuários ou consumidores finais no Github da Puc Rio.

A linhagem dos dados é um componente vital da governança dos dados, fornecendo visibilidade e controle necessários para gerenciar os dados de maneira eficaz e responsável.

A implementação de uma sólida linhagem de dados é fundamental para garantir a qualidade, segurança e conformidade dos dados da Maratona de Londres. Essa visibilidade do ciclo de vida dos dados beneficia diversos aspectos, desde a identificação e correção de problemas até a melhoria da confiança e eficiência operacional.

Para compor um conjunto de dados, especialmente no contexto de um projeto de engenharia de dados, várias técnicas são utilizadas para garantir a qualidade, integridade, e utilidade dos dados. Aqui estão cinco técnicas comuns:

Carga:

1. Ingestão de Dados:

- Batch Processing: Carregar volumes de dados em intervalos regulares como no caso anual. Usado para dados que não mudam com muita frequência.

2. Limpeza de Dados (Data Cleaning):

- Tratamento de Valores Nulos: Se tiver, substituir valores nulos por valores padrão, eliminar registros incompletos, ou usar técnicas de imputação para estimar valores ausentes.
- Remoção de Duplicatas: Caso existirem identificar e remover registros duplicados para garantir que cada entrada no conjunto de dados seja única.
- Normalização e Padronização: Ajustar dados para um formato ou escala comum. no caso particular, converter todas as Times Strings H:M:S para o mesmo formato Time S e padronizar unidades de medida.

3. Transformação de Dados (Data Transformation):

- Agregação: Resumir dados para obter informações mais significativas, como calcular (média, soma) ou e orientado no analise descritivo posterior a contagem de valores.
- Mapeamento e Recodificação: Transformar dados categóricos str da fonte em valores numéricos especiais, ou reclassificar os valores das categorias para análises mais eficazes categorizando ou segmentando a análise.
- Criação de Features: Gerar novas variáveis (features) a partir das existentes para enriquecer o conjunto de dados e melhorar os modelos de análise ou aprendizado de máquina.

4. Integração de Dados (Data Integration):

- ETL (Extract, Transform, Load): Extraír dados de várias fontes, transformá-los para atender às necessidades do destino e carregá-los em um sistema de armazenamento centralizado, como no data lake.
- ELT (Extract, Load, Transform): Similar ao ETL, mas a transformação dos dados ocorre após o carregamento no destino, aproveitando o poder de processamento do sistema de armazenamento.

5. Validação de Dados (Data Validation):

- Checagem Manual de Consistência: Garantir que os dados sigam as regras de consistência, referências válidas, e conformidade com os formatos esperados.
- Verificação de Conformidade: Confirmar que os dados atendem às normas e regulamentações aplicáveis, como conformidade com GDPR para dados pessoais.
- Teste de Integridade: Executar testes automatizados para verificar a integridade dos dados, como a verificação de somas de controle ou a detecção de anomalias.

Essas técnicas foram essenciais para compor um conjunto de dados robusto e confiável, que pode ser usado com confiança em análises e modelos de aprendizado de máquina.