

Pontifícia Universidade Católica do Rio de Janeiro - PUC Rio
Pós-Graduação Data Science e Analytics

MVP - SPRINT ENGENHARIA DE DADOS

Leonardo José Lopes da Silva

Julho de 2024

Sumário

Objetivos.....	3
Busca e Coleta de Dados.....	4
Modelagem.....	8
Carga e Transformações.....	12
Análise.....	32
Autoavaliação.....	40

Objetivos

Este trabalho tem o intuito de exercitar três principais habilidades importantes relacionadas a área de Engenharia de Dados:

- A pesquisa de dados abertos na internet e exploração de diversas fontes, simulando situações reais de busca por informações.
- Aplicar o processo de Extração, Transformação e Carga (ETL) de dados em uma plataforma consolidada de nuvem.
- Explorar questões e motivações específicas relacionadas aos dados obtidos através de consultas (SQL, NoSQL etc).

Busca e Coleta de Dados

A busca por dados na internet pode ser uma tarefa desafiadora devido à abundância de fontes com baixa qualidade e confiabilidade. No contexto deste trabalho, o primeiro desafio foi explorar e identificar fontes de dados abertos, públicas e confiáveis. A seguir, são listadas algumas das fontes de dados encontradas:

- **Portal Brasileiro de Dados Abertos:** Plataforma do governo brasileiro que disponibiliza uma ampla gama de dados públicos de diferentes áreas. Link: <https://dados.gov.br/home>.
- **Dados Abertos do Governo dos Estados Unidos:** Portal do governo dos Estados Unidos que oferece acesso a diversos conjuntos de dados sobre saúde, educação, clima, entre outros. Link: <https://data.gov/>.
- **Dados Abertos da União Europeia:** Base de dados da União Europeia, contendo informações econômicas, sociais e ambientais sobre os países membros. Link: <https://data.europa.eu/en>
- **Earth Data:** Site com dados abertos coletados pela NASA de observações da Terra (atmosfera, biosfera, superfície terrestre, oceanos, hidrosfera etc). Link: <https://www.earthdata.nasa.gov/>
- **GitHub:** Plataforma de desenvolvimento colaborativo que hospeda muitos repositórios de dados abertos e projetos de código aberto. Link: <https://github.com/search?q=dataset&type=repositories>

Dados Escolhidos

Dentre o grande volume e variedade de dados disponíveis nas fontes apresentada, o conjunto de dados escolhido foi: a população de carros elétricos registrados no estado de Washington (EUA), disponível através do link: <https://catalog.data.gov/dataset/electric-vehicle-population-data>.

Este conjunto de dados mostra os Veículos Elétricos a Bateria (BEVs) e os Veículos Elétricos Híbridos Plug-in (PHEVs) que estão atualmente registrados no Departamento de Licenciamento do Estado de Washington.

A escolha deste conjunto de dados foi baseada na motivação de compreender melhor a adoção de veículos elétricos na região. Compreender a

distribuição e as preferências dos consumidores, o que pode fornecer insights valiosos para políticas públicas e estratégias de marketing.

Os principais questionamentos que serão respondidos por meio deste trabalho são:

- Quais fabricantes possuem a maior quantidade de carros registrados?
- Quais os modelos de carros com maior número de registros?
- Quais os modelos de carros com maior autonomia elétrica?
- Quais os tipos de carros mais com maior quantidade de registros?
- Quais as cidades com maior número de carros registrados?
- Qual a porcentagem dos registros são de estados diferentes de Washington?

Licença

O uso dos dados escolhidos é regido pela licença Open Database License (ODbL) 1.0. Abaixo são destacados os principais pontos contemplados por esta modalidade de licenciamento:

- **Notificações:** Ao compartilhar publicamente o banco de dados ou derivativos, é necessário incluir uma cópia da licença e manter intactos os avisos de direitos autorais.
- **Aviso de uso:** Ao utilizar o conteúdo publicamente, deve-se informar que os dados são provenientes do banco de dados licenciado.
- **Compartilhamento igual:** Qualquer banco de dados derivado deve ser licenciado sob termos iguais ou compatíveis.
- **Acesso a Derivados:** Deve-se oferecer acesso gratuito ou a custo razoável ao banco de dados derivado ou às alterações feitas.
- **Medidas tecnológicas:** Não se pode impor restrições tecnológicas que alterem os termos da licença.

A versão completa dos termos da licença pode ser acessada através do link: <https://opendatacommons.org/licenses/odbl/1-0/>.

Coleta

O processo de coleta dos dados envolveu o download de um arquivo no formato CSV diretamente no site apresentado. Este formato de arquivo foi escolhido devido à sua facilidade de manuseio e compatibilidade com diversas ferramentas modernas de análise de dados.

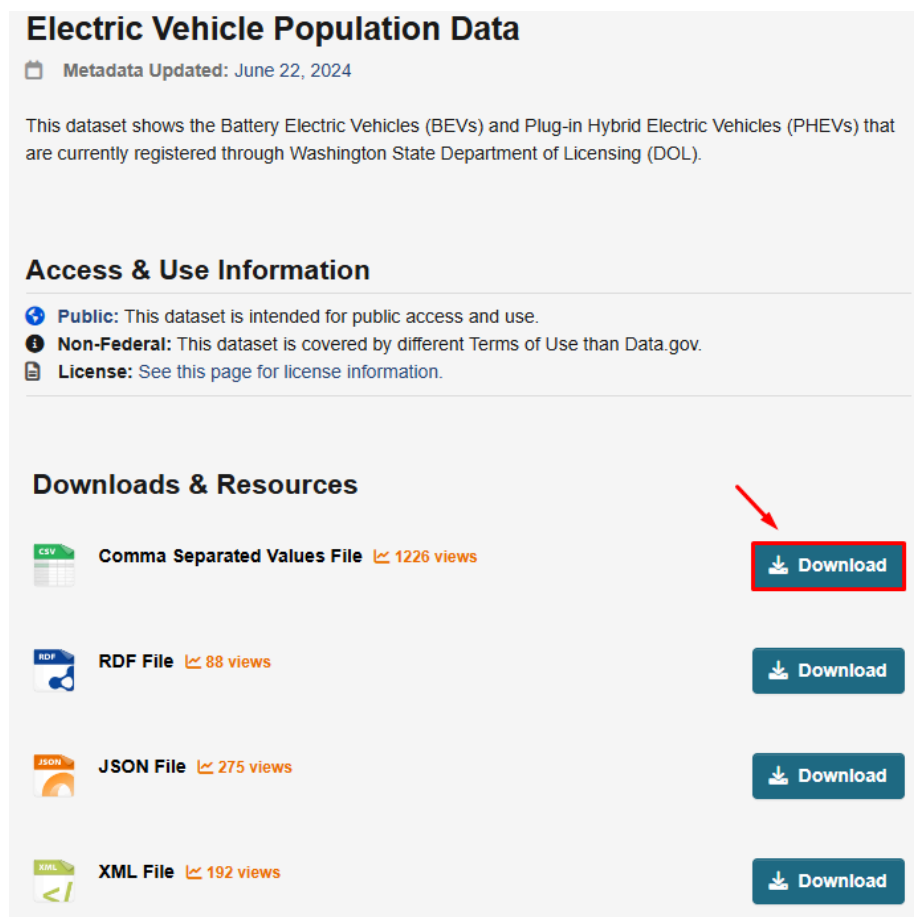


Figura 1 - Botão para download do arquivo CSV

Outros Dados

Para contextualizar o processo envolvido na escolha do conjunto de dados, serão apresentadas as principais alternativas de dados e questionamentos que foram consideradas durante a etapa de busca e pesquisa:

- **Banco de Dados de Reclamações de Consumidores (EUA):** Encontrar as empresas que possuem a maior quantidade de registros de reclamações de consumidores, assim como quais são os principais motivos das reclamações. Também seria possível obter as empresas que possuem a

maior quantidade de reclamações atrasadas (não respondidas ou respondidas após o prazo estabelecido).

Link: <https://catalog.data.gov/dataset/consumer-complaint-database>.

- **Estatísticas e dados sobre energia (EUA):** Encontrar os estados americanos que possuem a maior emissão de poluentes (CO₂, NO_x e SO₂) durante o consumo de energia elétrica para produção de combustíveis. Também seria possível avaliar qual o combustível emite a maior quantidade de poluentes.

Link: <https://catalog.data.gov/dataset/electricity-data-and-statistics-application-programming-interface-api>.

- **Dados do Cadastro Nacional da Pessoa Jurídica (BR):** Utilizar informações das empresas brasileiras para a análise preliminar da viabilidade de um produto destinado a um setor específico. A proposta consistia em filtrar as empresas com base em seus serviços (código CNAE), coletando dados como porte, localização, entre outros, para um estudo de viabilidade direcionado à construção civil ou indústria. Essa abordagem visava identificar o público-alvo do produto e estabelecer seus limites potenciais.

Link: <https://dados.gov.br/dados/conjuntos-dados/cadastro-nacional-da-pessoa-juridica---cnpj>.

Modelo de Dados

Ao iniciar o processo de análise dos dados obtidos, observou-se que o arquivo CSV baixado consiste essencialmente em uma única tabela de dados, onde cada linha representa uma entrada única e cada coluna representa um atributo específico. Diante dessa estrutura simplificada e direta, o modelo de dados mais adequado é o de *Flat Table*.

O modelo de *Flat Table*, ou tabela plana, é caracterizado pela organização dos dados em uma estrutura bidimensional simples, onde todas as informações são mantidas em uma única tabela. Esse modelo é ideal para conjuntos de dados que não exigem hierarquias complexas ou relacionamentos detalhados entre entidades.

A seguir, é apresentada uma visão geral da estrutura da tabela original baixada, destacando suas colunas e tipos de dados:

Tabela 1 - Estrutura/Catálogo original da tabela

Coluna	Descrição	Tipo
VIN (1-10)	Os primeiros 10 caracteres do Número de Identificação do Veículo (VIN) de cada veículo.	Texto
County	Região geográfica de um estado onde o proprietário do veículo está registrado como residente. Veículos registrados no estado de Washington podem estar localizados em outros estados.	Texto
City	A cidade na qual o proprietário registrado reside.	Texto
State	O estado do país associado ao registro.	Texto
Postal Code	O código postal de 5 dígitos no qual o proprietário registrado reside.	Texto
Model Year	O ano do modelo do veículo.	Texto
Make	O fabricante do veículo.	Número
Model	O modelo do veículo.	Número
Electric Vehicle Type	Distingue o veículo como totalmente elétrico (BEV) ou híbrido plug-in (PHEV).	Texto

Clean Alternative Fuel (CAFV) Eligibility	Categoriza o veículo como Veículo de Combustível Alternativo Limpo (CAFV) com base nos requisitos de combustível e na exigência de autonomia elétrica estabelecidos no House Bill 2042 aprovado na sessão legislativa de 2019.	Texto
Electric Range	Distância que um veículo pode viajar puramente com sua carga elétrica (em milhas).	Texto
Base MSRP	Este é o menor preço, em dólares, sugerido pelo fabricante (MSRP) para qualquer nível de acabamento do modelo em questão.	Número
Legislative District	A seção específica do estado de Washington onde o proprietário do veículo reside.	Texto
DOL Vehicle ID	Número único atribuído a cada veículo pelo Departamento de Licenciamento para fins de identificação.	Texto
Vehicle Location	O centro do CEP (Código de Endereçamento Postal) para o veículo registrado.	Ponto
Electric Utility	Essa é a área de territórios de serviço de varejo de energia elétrica que atende ao endereço do veículo registrado.	Texto
2020 Census Tract	O identificador de setor censitário é uma combinação dos códigos de estado, condado e setor censitário atribuídos pelo United States Census Bureau no censo de 2020, também conhecido como Identificador Geográfico (GEOID).	Texto

É importante notar que as colunas "Electric Range" e "Base MSRP" contêm o valor zero quando a autonomia elétrica do veículo não foi determinada e o preço sugerido pelo fabricante não foi fornecido, respectivamente.

Catálogo de Dados

Para melhor adequar os dados à análise proposta neste trabalho, identificou-se que algumas colunas da tabela original não contribuem diretamente com informações relevantes. Como resultado, foi desenvolvido um catálogo de dados adaptado, onde colunas não significativas foram removidas e as restantes foram renomeadas para facilitar a compreensão e interpretação dos resultados.

A tabela a seguir ilustra a correspondência entre as colunas originais e suas contrapartes no catálogo adaptado, destacando seus tipos de dados e eventuais limitações:

Tabela 2 - Catálogo dos dados

Coluna original	Coluna do catálogo	Tipo	Notas
VIN (1-10)	Removida	-	-
County	Condado	Texto	Não existem restrições específicas.
City	Cidade	Texto	Não existem restrições específicas.
State	Estado	Texto	Não existem restrições específicas.
Postal Code	Removida	-	-
Model Year	Ano	Número inteiro	O ano de fabricação deve ser maior do que zero e inferior ao ano de 2025.
Make	Fabricante	Texto	Não existem restrições específicas.
Model	Modelo	Texto	Não existem restrições específicas.
Electric Vehicle Type	Tipo	Categórico	As entradas podem assumir apenas os valores “BEV” ou “PHEV” (ou nulo no caso de valores não fornecidos)
Clean Alternative Fuel (CAEV) Eligibility	Elegibilidade CAEV	Categórico	As entradas podem assumir apenas os valores “Sim” ou “Não” (ou nulo no caso de valores não fornecidos).
Electric Range	Autonomia Elétrica	Número inteiro	As entradas não podem ser negativas.
Base MSRP	Valor de Mercado	Número inteiro	As entradas não podem ser negativas.
Legislative District	Removida	-	-
DOL Vehicle ID	Id	Texto	Os caracteres das entradas devem ser apenas numéricos.

Vehicle Location	Removida	-	-
Electric Utility	Removida	-	-
2020 Census Tract	Removida	-	-

Para atingir as alterações apresentadas no catálogo, os dados brutos passaram por transformações consideráveis, não se limitando apenas à mudança de nomenclatura. Essas transformações serão mostradas em detalhe na próxima seção.

Plataforma

Para realizar o processamento e análise dos dados neste trabalho, utilizou-se a plataforma Databricks. O Databricks é uma plataforma analítica unificada e aberta projetada para construir, implantar, compartilhar e manter análises de dados empresariais e soluções de inteligência artificial em escala.

Optou-se pela versão Community do Databricks para este projeto, visando minimizar custos dentro do escopo definido. Apesar das limitações desta versão, é possível desenvolver e validar protótipos de análises relativamente complexas, demonstrando a robustez e a versatilidade da plataforma para cenários acadêmicos e de pesquisa.

Carga dos Dados

Para iniciar o processo de carga dos dados, foi criado um cluster, através do menu Compute dentro do ambiente Databricks, utilizando a versão de Runtime 15.2 (Scala 2.12, Spark 3.5.0), conforme ilustrado na imagem abaixo.

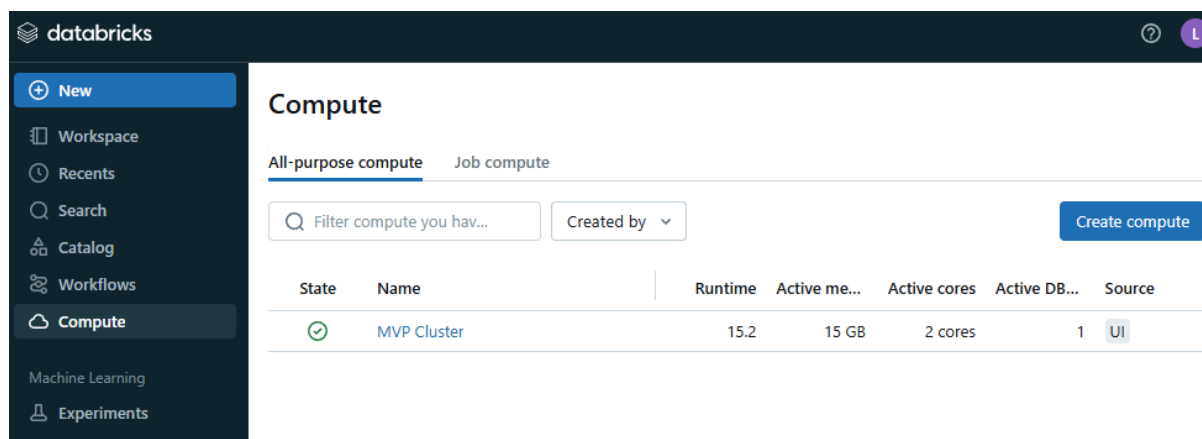


Figura 2 - Cluster criado utilizando Runtime 15.2

Em seguida, utilizando a opção "New > Table", procedeu-se com a carga do arquivo CSV baixado para o sistema de arquivos em nuvem do Databricks (DBFS). Após esse processo, o arquivo ficou armazenado no seguinte caminho dentro do DBFS: "FileStore/tables/Electric_Vehicle_Population_Data.csv". Esta abordagem permitiu a rápida disponibilidade e acesso aos dados para as etapas subsequentes de processamento e análise.

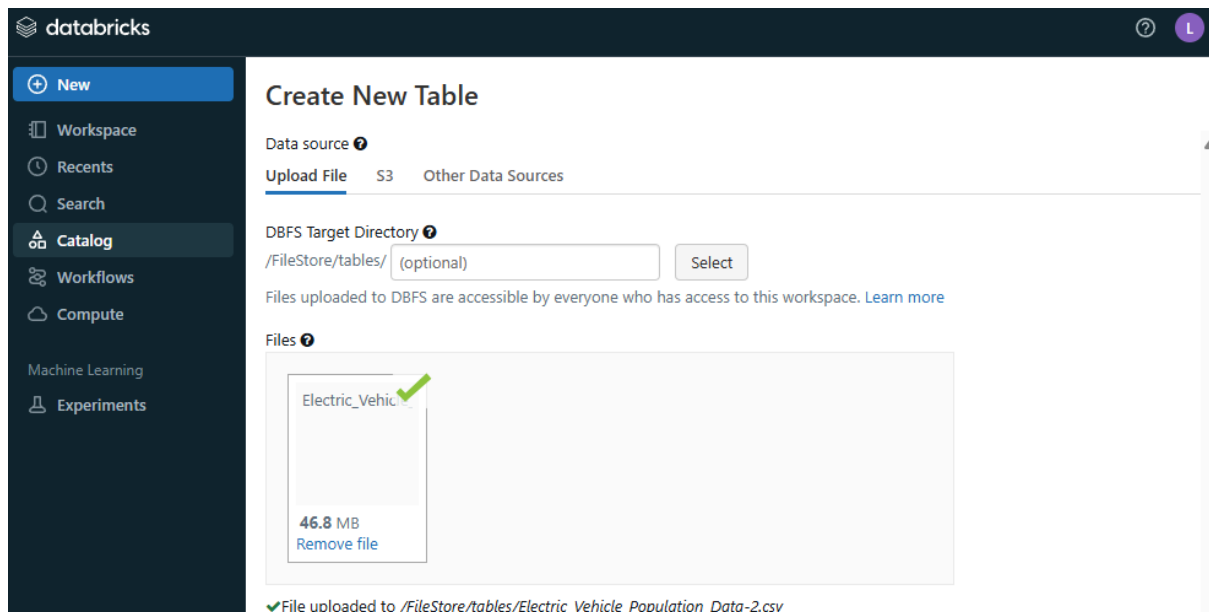


Figura 3 - Upload do arquivo CSV para o DBFS

Processamento dos Dados

Para realizar o processamento dos dados, foi criado um *notebook* dentro da plataforma Databricks. Um *notebook* é um ambiente de desenvolvimento interativo que permite escrever e executar código, visualizar resultados e documentar o processo de forma integrada. Ele é amplamente utilizado em análises de dados por sua flexibilidade e facilidade de uso.

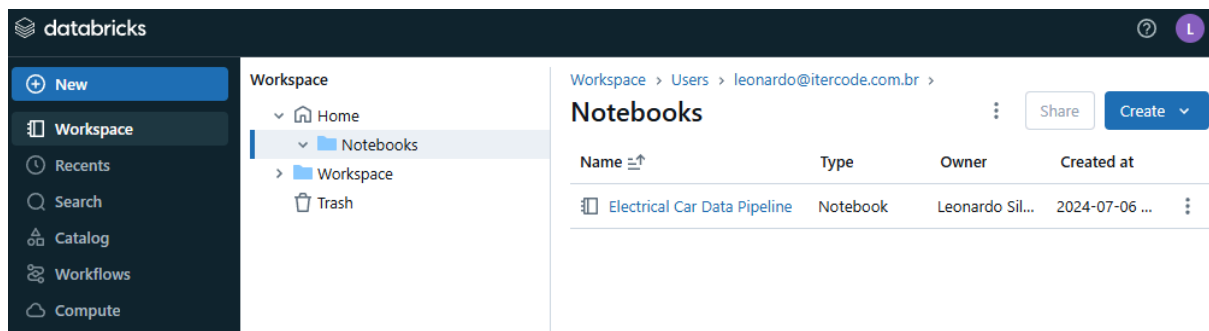


Figura 4 - Notebook utilizado para o processamento dos dados

No *notebook*, utilizou-se a linguagem Python para realizar as transformações necessárias sobre os dados brutos. Além disso, foi utilizado o framework PySpark, uma API Python para o Apache Spark. O PySpark combina a facilidade de aprendizado e uso do Python com o poder do Apache Spark para possibilitar o processamento e a análise de dados de qualquer tamanho.

As transformações realizadas nos dados brutos tiveram como objetivo construir uma tabela mais limpa e coesa para a etapa subsequente de análises. Para melhor ilustrar a organização e fluxo das transformações, a seguir, serão detalhadas em subseções cada uma das etapas de transformação realizadas.

1. Leitura do arquivo CSV

A primeira etapa consistiu na leitura do arquivo CSV carregado no DBFS do Databricks. Para isso, criou-se uma sessão Spark e utilizou-se o método `read()`, para criar um *Data Frame*, especificando o caminho do arquivo.

Os *Data Frames* do Apache Spark oferecem um conjunto abrangente de funções (selecionar colunas, filtrar, unir, agregar) que permitem resolver eficientemente problemas comuns de análise de dados.

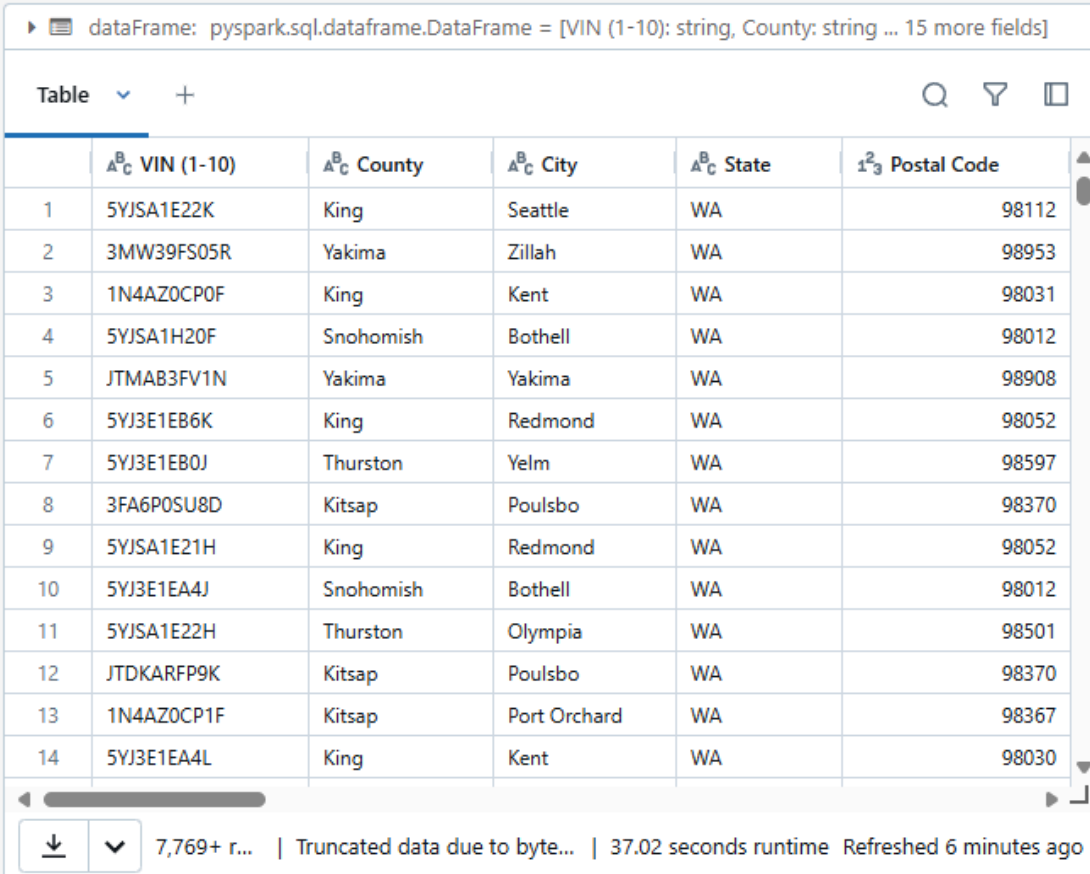
```
1 # =====
2 # Leitura do arquivo CSV
3 # =====
4
5 # Diretivas de importação
6 from pyspark.sql import SparkSession
7 from pyspark.sql.functions import *
8
9 # Localização do arquivo CSV com os dados brutos
10 fileName = "Electric_Vehicle_Population_Data.csv"
11 fileRootDirectory = "/FileStore/tables/"
12 fileLocation = fileRootDirectory + fileName
13
14 # Construção de um DataFrame a partir do arquivo CSV
15 sparkSession = SparkSession.builder.getOrCreate()
16 dataframe = sparkSession.read.option("sep", ",") \
17 |.option("header", True) \
18 |.option("inferSchema", True) \
19 |.csv(fileLocation)
20
21 # Display da tabela original
22 display(dataframe)
23
```

Figura 5 - Leitura do arquivo CSV e criação do *Data Frame*

Algumas configurações adicionais foram informadas ao método `read` para garantir a leitura correta dos dados:

- **Separador de Dados:** Especificou-se que o separador dos dados no arquivo CSV é a vírgula, através da opção `sep=","`.
- **Cabeçalho:** A opção `header=True` foi utilizada para informar ao leitor de arquivos do PySpark que a primeira linha do arquivo contém o nome das colunas da tabela. Esta opção é bastante conveniente para arquivos CSV, pois simplifica a identificação das colunas.
- **Dedução de Esquema:** A opção `inferSchema=True` foi utilizada para criar um esquema padrão de tabela. Sabendo que os dados não são muito complexos, permitiu-se que o próprio algoritmo do PySpark deduzisse o esquema a partir dos dados.

Abaixo apresenta-se um trecho da tabela, em sua configuração original, gerada através do *Data Frame* utilizando o método `display()`.



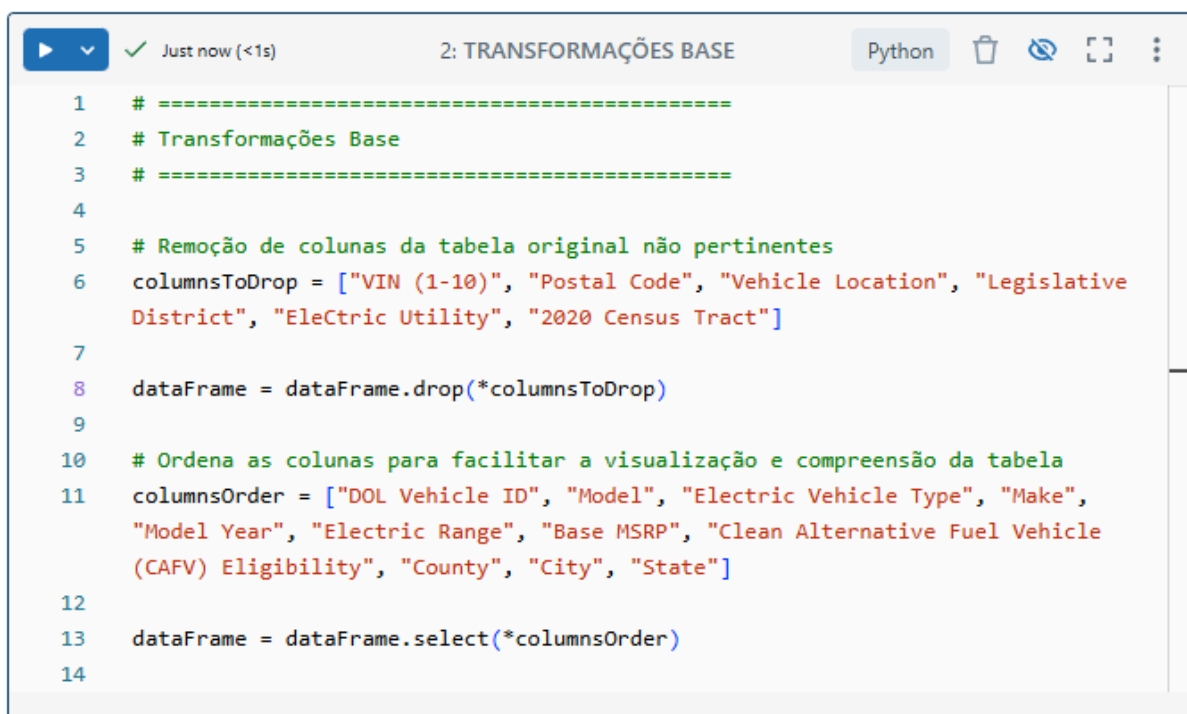
	^A _C VIN (1-10)	^A _C County	^A _C City	^A _C State	¹ ₃ Postal Code
1	5YJSA1E22K	King	Seattle	WA	98112
2	3MW39FS05R	Yakima	Zillah	WA	98953
3	1N4AZ0CP0F	King	Kent	WA	98031
4	5YJSA1H20F	Snohomish	Bothell	WA	98012
5	JTMAB3FV1N	Yakima	Yakima	WA	98908
6	5YJ3E1EB6K	King	Redmond	WA	98052
7	5YJ3E1EB0J	Thurston	Yelm	WA	98597
8	3FA6P0SU8D	Kitsap	Poulsbo	WA	98370
9	5YJSA1E21H	King	Redmond	WA	98052
10	5YJ3E1EA4J	Snohomish	Bothell	WA	98012
11	5YJSA1E22H	Thurston	Olympia	WA	98501
12	JTDKARFP9K	Kitsap	Poulsbo	WA	98370
13	1N4AZ0CP1F	Kitsap	Port Orchard	WA	98367
14	5YJ3E1EA4L	King	Kent	WA	98030

Figura 6 - Trecho da tabela original

2. Transformações Gerais

Em seguida, foram aplicadas duas transformações gerais sobre o *Data Frame*:

- **Remoção de Colunas:** Remoção de algumas colunas da tabela original que não adicionavam valor significativo à análise.
- **Reordenamento das Colunas:** Após a remoção das colunas irrelevantes, as colunas restantes foram reordenadas para melhorar a visualização e a compreensão da tabela. O reordenamento ajuda a destacar as colunas mais importantes e a seguir uma lógica que facilita a interpretação dos dados.



```
1 # =====
2 # Transformações Base
3 # =====
4
5 # Remoção de colunas da tabela original não pertinentes
6 columnsToDrop = ["VIN (1-10)", "Postal Code", "Vehicle Location", "Legislative
   District", "EleCtric Utility", "2020 Census Tract"]
7
8 dataframe = dataframe.drop(*columnsToDrop)
9
10 # Ordena as colunas para facilitar a visualização e compreensão da tabela
11 columnsOrder = ["DOL Vehicle ID", "Model", "Electric Vehicle Type", "Make",
   "Model Year", "Electric Range", "Base MSRP", "Clean Alternative Fuel Vehicle
   (CAFV) Eligibility", "County", "City", "State"]
12
13 dataframe = dataframe.select(*columnsOrder)
14
```

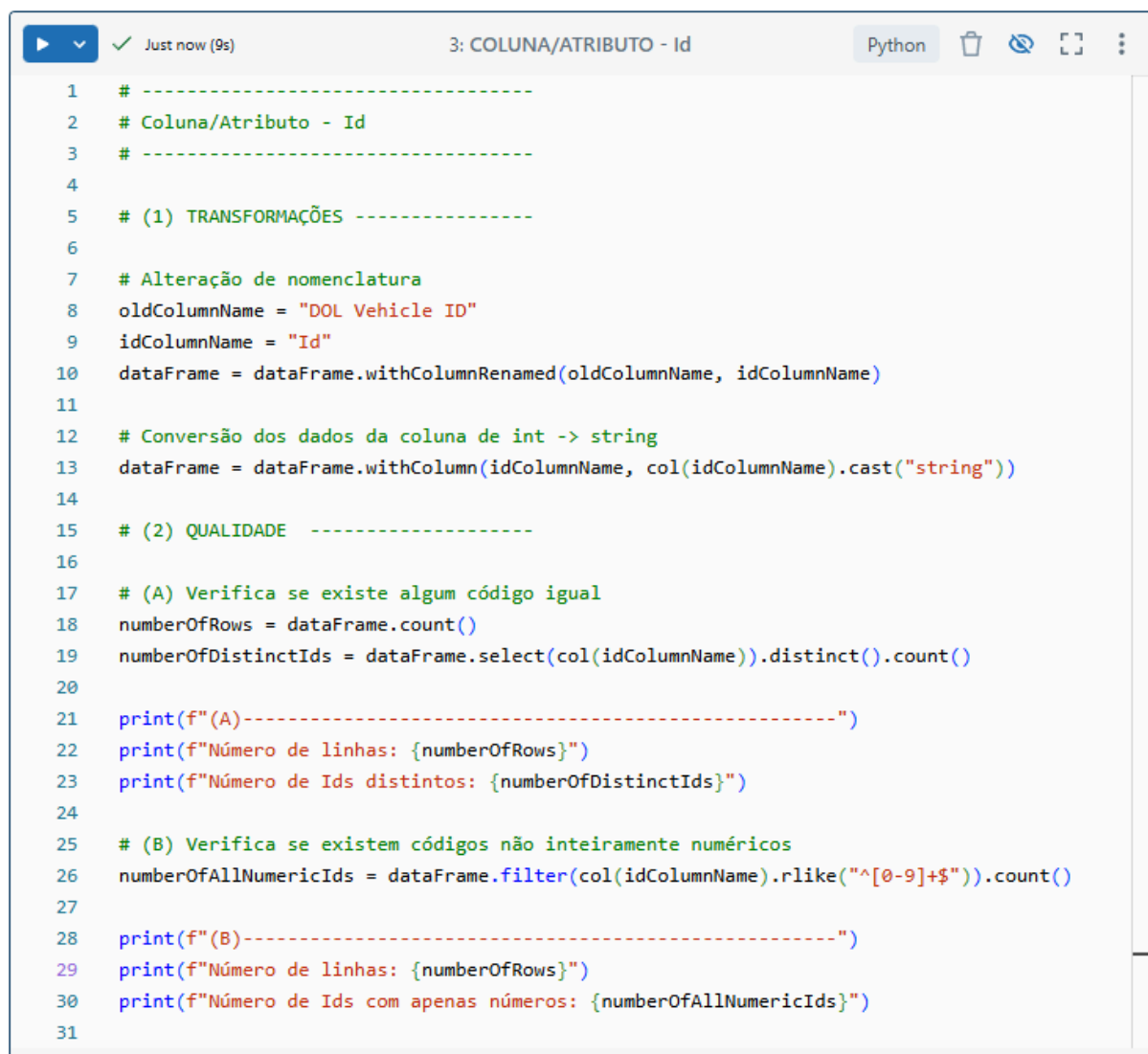
Figura 7 - Transformações gerais

Nas próximas subseções, serão apresentadas as transformações específicas realizadas sobre cada uma das colunas remanescentes da tabela, detalhando os ajustes e melhorias feitos para garantir a qualidade e relevância dos dados.

3. Coluna - DOL Vehicle ID

A coluna "DOL Vehicle ID" passou por duas transformações principais. Primeiramente, o nome da coluna foi alterado para "Id" com o objetivo de simplificar a nomenclatura e facilitar a referência a essa coluna nas etapas subsequentes.

Em seguida, o tipo dos dados da coluna foi alterado para *string*. Durante a leitura do arquivo CSV, com a opção `inferSchema`, os dados dessa coluna foram reconhecidos como números inteiros (*int*). No entanto, para manter a coerência com a tabela original, decidiu-se converter estes valores de *int* para *string*.



```
1 # -----
2 # Coluna/Atributo - Id
3 # -----
4
5 # (1) TRANSFORMAÇÕES -----
6
7 # Alteração de nomenclatura
8 oldColumnName = "DOL Vehicle ID"
9 idColumnName = "Id"
10 dataframe = dataframe.withColumnRenamed(oldColumnName, idColumnName)
11
12 # Conversão dos dados da coluna de int -> string
13 dataframe = dataframe.withColumn(idColumnName, col(idColumnName).cast("string"))
14
15 # (2) QUALIDADE -----
16
17 # (A) Verifica se existe algum código igual
18 numberOfRows = dataframe.count()
19 numberOfDistinctIds = dataframe.select(col(idColumnName)).distinct().count()
20
21 print(f"(A)-----")
22 print(f"Número de linhas: {numberOfRows}")
23 print(f"Número de Ids distintos: {numberOfDistinctIds}")
24
25 # (B) Verifica se existem códigos não inteiramente numéricos
26 numberOfAllNumericIds = dataframe.filter(col(idColumnName).rlike("^[0-9]+$")).count()
27
28 print(f"(B)-----")
29 print(f"Número de linhas: {numberOfRows}")
30 print(f"Número de Ids com apenas números: {numberOfAllNumericIds}")
31
```

Figura 8 - Transformações e validações da coluna Id

Destaca-se que esta foi a única coluna que necessitou passar por uma transformação nos tipos dos dados. A opção `inferSchema`, para todas as outras colunas, deduziu o tipo esperado.

Após estas transformações, a qualidade dos dados foi verificada para evitar que isso influencie nas análises posteriores. Primeiramente, utilizou-se o método `distinct()` para garantir que todos os valores na coluna "Id" sejam únicos, assegurando que cada registro possui um identificador exclusivo.

Além disso, foi verificado se todos os valores desta coluna possuem apenas caracteres numéricos, uma vez que o código de identificação deve conter exclusivamente esses caracteres.

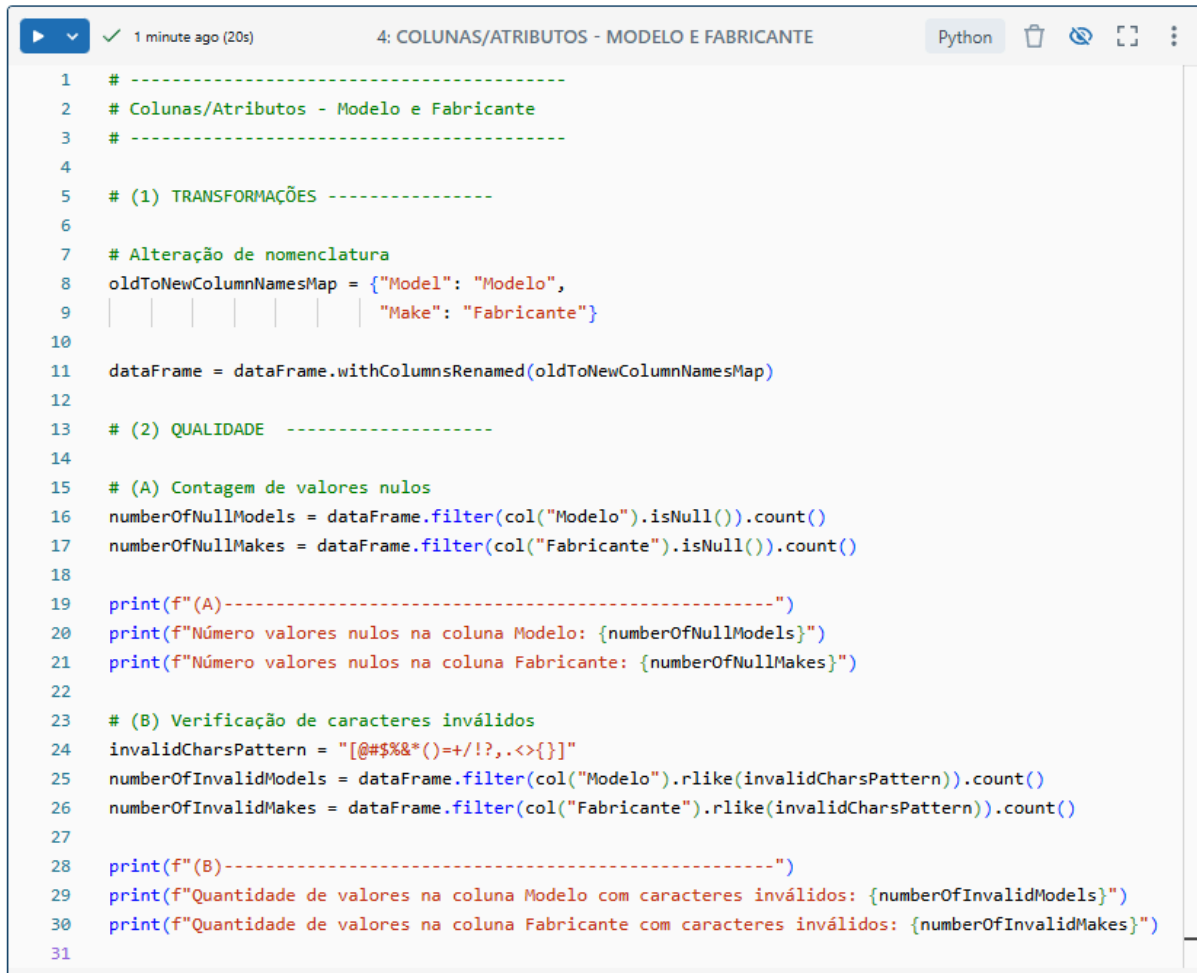
```
dataFrame: pyspark.sql.dataframe.DataFrame = [Id: string, Model: string ... 9 more fields]
(A)-----
Número total de linhas: 191407
Número de Ids distintos: 191407
(B)-----
Número total de linhas: 191407
Número de Ids com apenas números: 191407
```

Figura 9 - Resultados das verificações de qualidade da coluna Id

Conforme ilustrado na Figura 9, a integridade dos dados nesta coluna é assegurada, uma vez que o número total de linhas na tabela corresponde ao número de IDs únicos, bem como ao número de linhas contendo apenas caracteres numéricos.

4. Colunas - Model e Make

Estas colunas passaram apenas por uma mudança de nomenclatura de “Model” e “Make” para “Modelo” e “Fabricante”, respectivamente.



```
1 # -----
2 # Colunas/Atributos - Modelo e Fabricante
3 # -----
4
5 # (1) TRANSFORMAÇÕES -----
6
7 # Alteração de nomenclatura
8 oldToNewColumnNameMap = {"Model": "Modelo",
9 | | | | | | | "Make": "Fabricante"}
10
11 dataframe = dataframe.withColumnsRenamed(oldToNewColumnNameMap)
12
13 # (2) QUALIDADE -----
14
15 # (A) Contagem de valores nulos
16 numberOfNullModels = dataframe.filter(col("Modelo").isNull()).count()
17 numberOfNullMakes = dataframe.filter(col("Fabricante").isNull()).count()
18
19 print(f"(A)-----")
20 print(f"Número valores nulos na coluna Modelo: {numberOfNullModels}")
21 print(f"Número valores nulos na coluna Fabricante: {numberOfNullMakes}")
22
23 # (B) Verificação de caracteres inválidos
24 invalidCharsPattern = "[@#$$%&*()=+/?!,.<>{}]"
25 numberOfInvalidModels = dataframe.filter(col("Modelo").rlike(invalidCharsPattern)).count()
26 numberOfInvalidMakes = dataframe.filter(col("Fabricante").rlike(invalidCharsPattern)).count()
27
28 print(f"(B)-----")
29 print(f"Quantidade de valores na coluna Modelo com caracteres inválidos: {numberOfInvalidModels}")
30 print(f"Quantidade de valores na coluna Fabricante com caracteres inválidos: {numberOfInvalidMakes}")
31
```

Figura 10 - Transformações e validações das colunas Modelo e Fabricante

Para garantir a qualidade dos dados destas colunas, primeiramente, foi contada a quantidade de valores nulos, conforme ilustrado na Figura 10. Adicionalmente, verificou-se a presença de caracteres inválidos como "!", "?", ">", encontrando 4.337 ocorrências desses caracteres na coluna Modelo e 5 na coluna Fabricante.

```

▶ dataframe: pyspark.sql.dataframe.DataFrame = [Id: string, Modelo: string ... 9 more fields]
(A)-----
Número valores nulos na coluna Modelo: 0
Número valores nulos na coluna Fabricante: 0
(B)-----
Quantidade de valores na coluna Modelo com caracteres inválidos: 4337
Quantidade de valores na coluna Fabricante com caracteres inválidos: 5

```

Figura 11 - Resultados das verificações das colunas Modelo e Fabricante

A partir disso, decidiu-se investigar com maior profundidade estes valores para garantir que não irão causar problemas. A investigação foi feita através do *display* do *Data Frame* com os mesmos filtros aplicados anteriormente e da contagem dos valores distintos com inconsistências.

Para a coluna Modelo, encontrou-se apenas 1 valor distinto com os caracteres buscados, este sendo o modelo "ID.4" da fabricante Volkswagen. Ao realizar uma busca rápida na internet, constatou-se que este modelo realmente existe, logo, não há problemas com relação aos dados desta coluna.

```

32 # (C) Verificação dos Modelos com caracteres inválidos
33 invalidModelsDF = dataframe.filter(col("Modelo").rlike(invalidCharsPattern))
34 numberOfInvalidDistinctModels = invalidModelsDF.select("Modelo").distinct().count()
35 print(f"(C)-----")
36 print(f"Número de Modelos distintos com caracteres inválidos: {numberOfInvalidDistinctModels}")
37
38 display(invalidModelsDF)

```

▶ (14) Spark Jobs

```

▶ dataframe: pyspark.sql.dataframe.DataFrame = [Id: string, Modelo: string ... 9 more fields]
▶ invalidModelsDF: pyspark.sql.dataframe.DataFrame = [Id: string, Modelo: string ... 9 more fields]
(C)-----
Número de Modelos distintos com caracteres inválidos: 1

```

	^A _C Id	^A _C Modelo	^A _C Tipo	^A _C Fabricante	¹ ₃ Ano	¹ ₃ Autonomia_Eletrica	¹ ₃ Val
5	194504103	ID.4	BEV	VOLKSWAGEN	2021	null	
6	161892538	ID.4	BEV	VOLKSWAGEN	2021	null	
7	225963825	ID.4	BEV	VOLKSWAGEN	2022	null	
8	251946978	ID.4	BEV	VOLKSWAGEN	2023	null	
9	253078997	ID.4	BEV	VOLKSWAGEN	2023	null	

4,337 rows | 14.72 seconds runtime Refreshed 1 minute ago

Figura 12 - Investigação de valores na coluna Modelo

Quanto à coluna Fabricante, todos os cinco valores inconsistentes encontrados correspondem ao fabricante "TH!NK", que é um fabricante existente. Portanto, não há preocupações em relação à qualidade desta coluna.

```

40 # (D) Verificação dos Fabricantes com caracteres inválidos
41 invalidMakesDF = dataframe.filter(col("Fabricante").rlike(invalidCharsPattern))
42 display(invalidMakesDF)

```

▶ (14) Spark Jobs

- ▶ dataframe: pyspark.sql.dataframe.DataFrame = [Id: string, Modelo: string ... 9 more fields]
- ▶ invalidModelsDF: pyspark.sql.dataframe.DataFrame = [Id: string, Modelo: string ... 9 more fields]
- ▶ invalidMakesDF: pyspark.sql.dataframe.DataFrame = [Id: string, Modelo: string ... 9 more fields]

Table ▾ + 🔍 🏠

	^A _C Id	^A _C Modelo	^A _C Electric Vehicle Type	^A _C Fabricante	¹ ₃ Model Year	¹ ₃ Electric Range
1	350464278	CITY	Battery Electric Vehicle (BEV)	TH!NK	2011	
2	156401885	CITY	Battery Electric Vehicle (BEV)	TH!NK	2011	
3	477078717	CITY	Battery Electric Vehicle (BEV)	TH!NK	2011	
4	240738235	CITY	Battery Electric Vehicle (BEV)	TH!NK	2011	
5	272385039	CITY	Battery Electric Vehicle (BEV)	TH!NK	2011	

◀ 5 rows | 14.95 seconds runtime Refreshed now ▶

Figura 13 - Investigação de valores na coluna Fabricante

5. Coluna - Electric Vehicle Type

A coluna "Electric Vehicle Type" também passou por uma mudança de nomenclatura, sendo renomeada para "Tipo".

Ademais, os valores desta coluna foram padronizados em apenas duas categorias: "BEV" e "PHEV". Na tabela original, os dados possuíam os valores "Battery Electric Vehicle (BEV)" ou "Plug-in Hybrid Electric Vehicle (PHEV)". Optou-se pela utilização apenas das abreviações dos tipos de carros elétricos por questões de simplicidade, visto que a essência da informação ainda se mantém.

The screenshot shows a Databricks notebook interface. The top bar indicates the notebook is named "5: COLUNA/ATRIBUTO - TIPO" and was last updated "Just now (3s)". The code editor displays the following Python code:

```
1 # -----
2 # Coluna/Atributo - Tipo
3 # -----
4
5 # (1) QUALIDADE -----
6 oldColumnName = "Electric Vehicle Type"
7 distinctTypeValuesDF = dataframe.select(oldColumnName).distinct()
8 display(distinctTypeValuesDF)
9
10 # (2) TRANSFORMAÇÕES -----
11
12 # Alteração de nomenclatura
13 typeColumnName = "Tipo"
14 dataframe = dataframe.withColumnRenamed(oldColumnName, typeColumnName)
15
16 # Padronização de valores categóricos: apenas BEV, PHEV ou Null
17 dataframe = dataframe.withColumn(typeColumnName, \
18     when(col(typeColumnName)=="Battery Electric Vehicle (BEV)", "BEV") \
19     .when(col(typeColumnName)=="Plug-in Hybrid Electric Vehicle (PHEV)", "PHEV") \
20     .otherwise(None))
21
```

Below the code editor, the "Spark Jobs" section shows the execution of the code. It lists two jobs: "distinctTypeValuesDF: pyspark.sql.dataframe.DataFrame = [Electric Vehicle Type: string]" and "dataframe: pyspark.sql.dataframe.DataFrame = [Id: string, Modelo: string ... 9 more fields]".

At the bottom, a table view displays the results of the transformation. The table has two columns: "Table" and "Electric Vehicle Type". It shows two rows of data:

Table	Electric Vehicle Type
1	Plug-in Hybrid Electric Vehicle (PHEV)
2	Battery Electric Vehicle (BEV)

The table view also indicates "2 rows" and "2.75 seconds runtime". A "Refreshed now" button is visible in the bottom right corner.

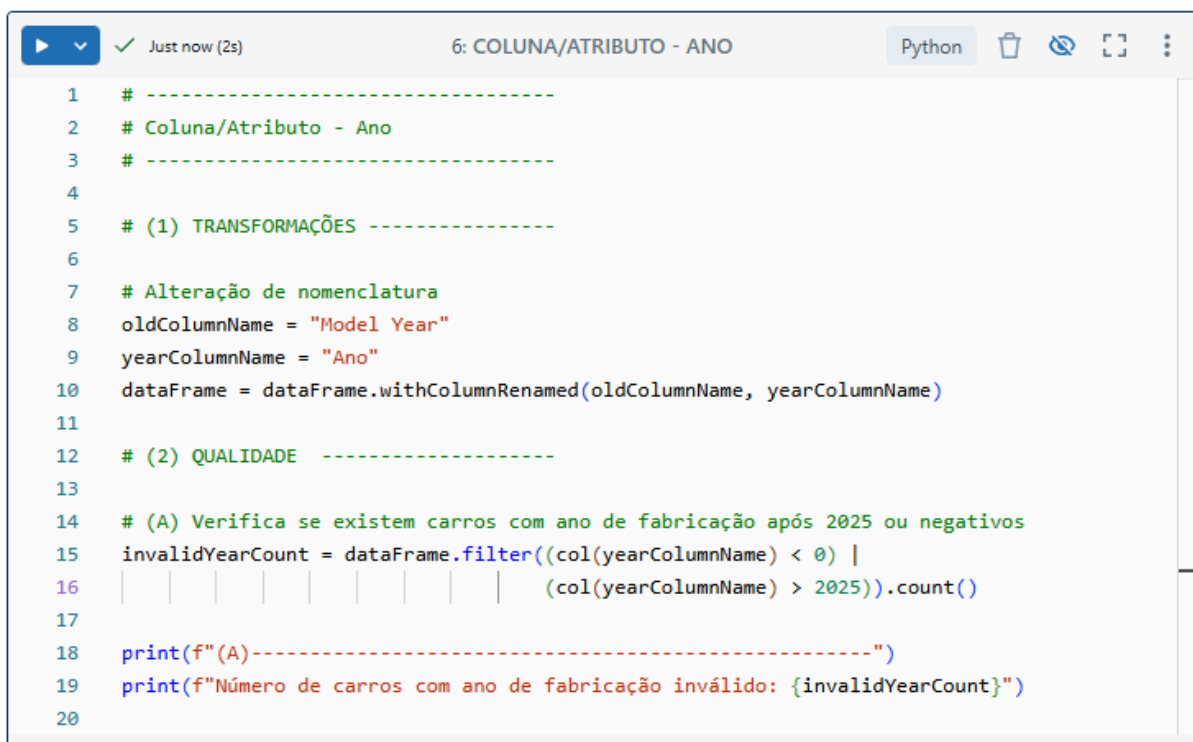
Figura 14 - Transformações e validações da coluna Tipo

Para obter uma visão clara dos diferentes tipos de veículos elétricos presentes no conjunto de dados, foi realizada uma consulta para identificar os valores distintos na coluna "Tipo", conforme ilustrado na Figura 14.

Isso ocorreu, porque decidiu-se primeiramente comprovar que a coluna "Tipo" continha apenas dois valores distintos antes de aplicar as transformações de simplificação. Vale destacar que esta foi uma das únicas colunas nas quais optou-se por realizar a validação de qualidade dos dados antes das transformações.

6. Coluna - Model Year

Esta coluna foi renomeada de "Model Year" para "Ano", simplificando a nomenclatura para melhor clareza no contexto do trabalho.



```
1 # -----
2 # Coluna/Atributo - Ano
3 # -----
4
5 # (1) TRANSFORMAÇÕES -----
6
7 # Alteração de nomenclatura
8 oldColumnName = "Model Year"
9 yearColumnName = "Ano"
10 dataframe = dataframe.withColumnRenamed(oldColumnName, yearColumnName)
11
12 # (2) QUALIDADE -----
13
14 # (A) Verifica se existem carros com ano de fabricação após 2025 ou negativos
15 invalidYearCount = dataframe.filter((col(yearColumnName) < 0) |
16 | | | | | | | | | | (col(yearColumnName) > 2025)).count()
17
18 print(f"(A)-----")
19 print(f"Número de carros com ano de fabricação inválido: {invalidYearCount}")
20
```

Figura 15 - Transformações e validações da coluna Ano

Em seguida, foram realizadas verificações para garantir a integridade dos dados. Especificamente, buscou-se valores que fossem negativos ou maiores que 2025, visto que, dada a data de confecção deste trabalho, não há a possibilidade de existirem carros com ano de fabricação superior a este valor.

```

14 # (A) Verifica se existem carros com ano de fabricação após 2025 ou negativos
15 invalidYearCount = dataframe.filter((col(yearColumnName) < 0) |
16 | | | | | | | | | | (col(yearColumnName) > 2025)).count()
17
18 print(f"(A)-----")
19 print(f"Número de carros com ano de fabricação inválido: {invalidYearCount}")
20
▶ (2) Spark Jobs
▶ dataframe: pyspark.sql.dataframe.DataFrame = [Id: string, Modelo: string ... 9 more fields]
(A)-----
Número de carros com ano de fabricação inválido: 0

```

Figura 16 - Resultado da verificação de qualidade da coluna Ano

7. Coluna - Electric Range

A coluna "Electric Range" foi submetida a várias transformações para torná-la mais adequada ao contexto do trabalho.

```

1 # -----
2 # Coluna/Atributo - Autonomia Elétrica
3 # -----
4
5 # (1) TRANSFORMAÇÕES -----
6
7 # Alteração de nomenclatura
8 oldColumnName = "Electric Range"
9 elecRangeColumnName = "Autonomia_Eletrica"
10 dataframe = dataframe.withColumnRenamed(oldColumnName, elecRangeColumnName)
11
12 # Converte os valores de milhas para km
13 conversionFactor = 1.60934
14 dataframe = dataframe.withColumn(elecRangeColumnName,
15 | | | | | | | | | | (col(elecRangeColumnName) * conversionFactor).cast("int"))
16
17 # Troca todos os valores 0 para Nulo
18 dataframe = dataframe.withColumn(elecRangeColumnName, when(col(elecRangeColumnName) == 0,
19 | | | | | | | | | | None) \
20 | | | | | | | | | | .otherwise(col(elecRangeColumnName)))
21
22 # (2) QUALIDADE -----
23
24 # Verifica se existem carros com autonomia elétrica negativa
25 carsWithNegativeRange = dataframe.filter(col(elecRangeColumnName) < 0).count()
26
27 print(f"(A)-----")
28 print(f"Número de carros com autonomia elétrica negativa: {carsWithNegativeRange}")

```

Figura 17 - Transformações e validações da coluna Autonomia_Eletrica

Primeiramente, a coluna foi renomeada para "Autonomia_Eletrica". Em seguida, seus valores foram convertidos de milhas para quilômetros. Esta conversão foi realizada para tornar os dados mais tangíveis, uma vez que no Brasil é mais comum apresentar distâncias em quilômetros. A conversão utilizou o fator de multiplicação de 1,60934, que é a equivalência de uma milha em quilômetros.

Por fim, todos os valores zero foram convertidos para nulo. Esta transformação foi feita porque considera-se mais coerente que um dado ausente ou não fornecido seja representado como nulo, em vez de zero.

Além disso, foi realizada uma única verificação para identificar valores inferiores a zero. Esta análise foi feita devido à falta de sentido prático de um carro elétrico possuir uma autonomia elétrica negativa.

```
23 # Verifica se existem carros com autonomia elétrica negativa
24 carsWithNegativeRange = dataframe.filter(col(elecRangeColumnName) < 0).count()
25
26 print(f"(A)-----")
27 print(f"Número de carros com autonomia elétrica negativa: {carsWithNegativeRange}")
28
```

▶ (2) Spark Jobs

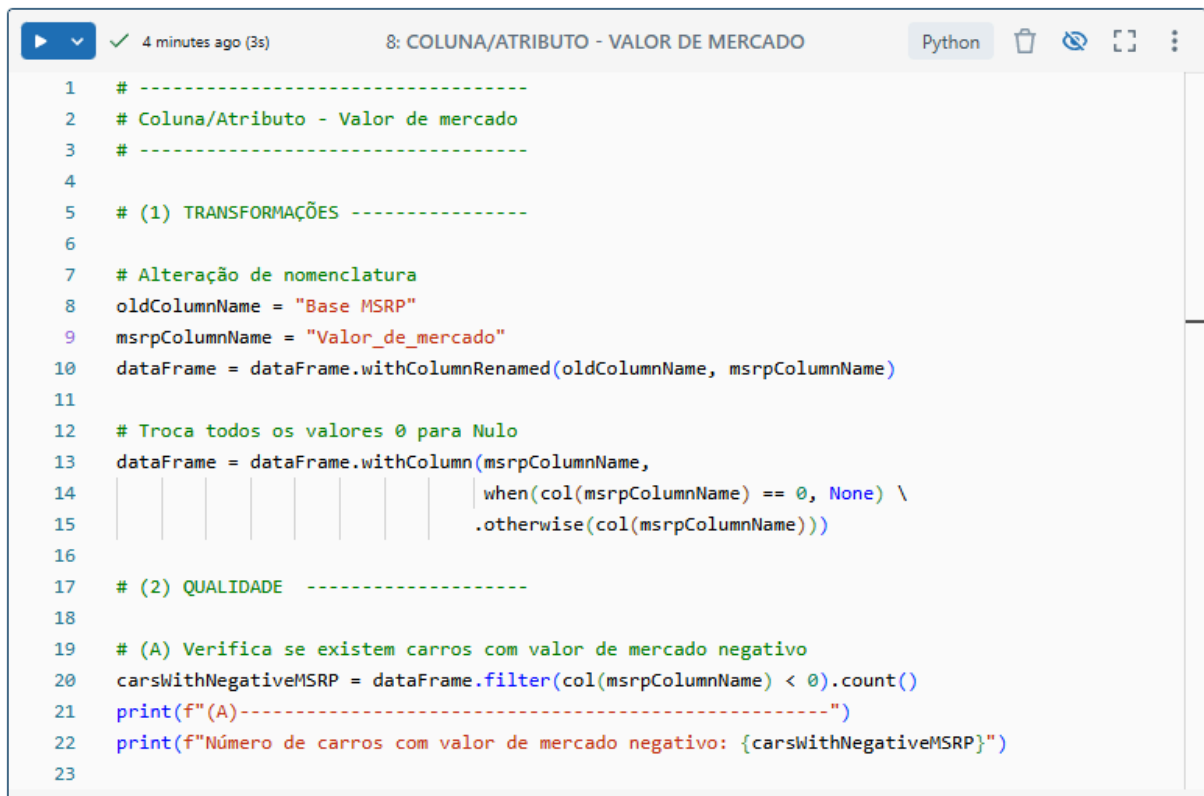
▶ dataframe: pyspark.sql.dataframe.DataFrame = [Id: string, Modelo: string ... 9 more fields]

(A)-----
Número de carros com autonomia elétrica negativa: 0

Figura 18 - Resultado da verificação de qualidade da coluna Autonomia_Eletrica

8. Coluna - Base MSRP

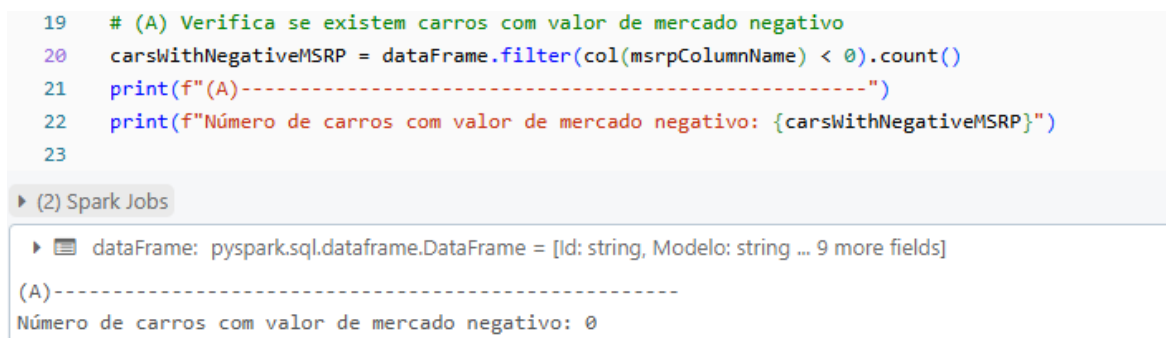
A coluna "Base MSRP" foi renomeada para "Valor_de_mercado" e, considerando a mesma lógica adotada para a coluna "Autonomia_Eletrica", os valores zero foram convertidos para nulo.



```
1 # -----
2 # Coluna/Atributo - Valor de mercado
3 # -----
4
5 # (1) TRANSFORMAÇÕES -----
6
7 # Alteração de nomenclatura
8 oldColumnName = "Base MSRP"
9 msrpColumnName = "Valor_de_mercado"
10 dataframe = dataframe.withColumnRenamed(oldColumnName, msrpColumnName)
11
12 # Troca todos os valores 0 para Nulo
13 dataframe = dataframe.withColumn(msrpColumnName,
14 |                               when(col(msrpColumnName) == 0, None) \
15 |                               .otherwise(col(msrpColumnName)))
16
17 # (2) QUALIDADE -----
18
19 # (A) Verifica se existem carros com valor de mercado negativo
20 carsWithNegativeMSRP = dataframe.filter(col(msrpColumnName) < 0).count()
21 print(f"(A)-----")
22 print(f"Número de carros com valor de mercado negativo: {carsWithNegativeMSRP}")
23
```

Figura 19 - Transformações e validações da coluna Valor_de_mercado

Ademais, a única verificação de qualidade que se considerou pertinente foi avaliar se existem carros com valor de mercado inferior a zero.



```
19 # (A) Verifica se existem carros com valor de mercado negativo
20 carsWithNegativeMSRP = dataframe.filter(col(msrpColumnName) < 0).count()
21 print(f"(A)-----")
22 print(f"Número de carros com valor de mercado negativo: {carsWithNegativeMSRP}")
23
```

▶ (2) Spark Jobs

▶ dataframe: pyspark.sql.dataframe.DataFrame = [Id: string, Modelo: string ... 9 more fields]

(A)-----
Número de carros com valor de mercado negativo: 0

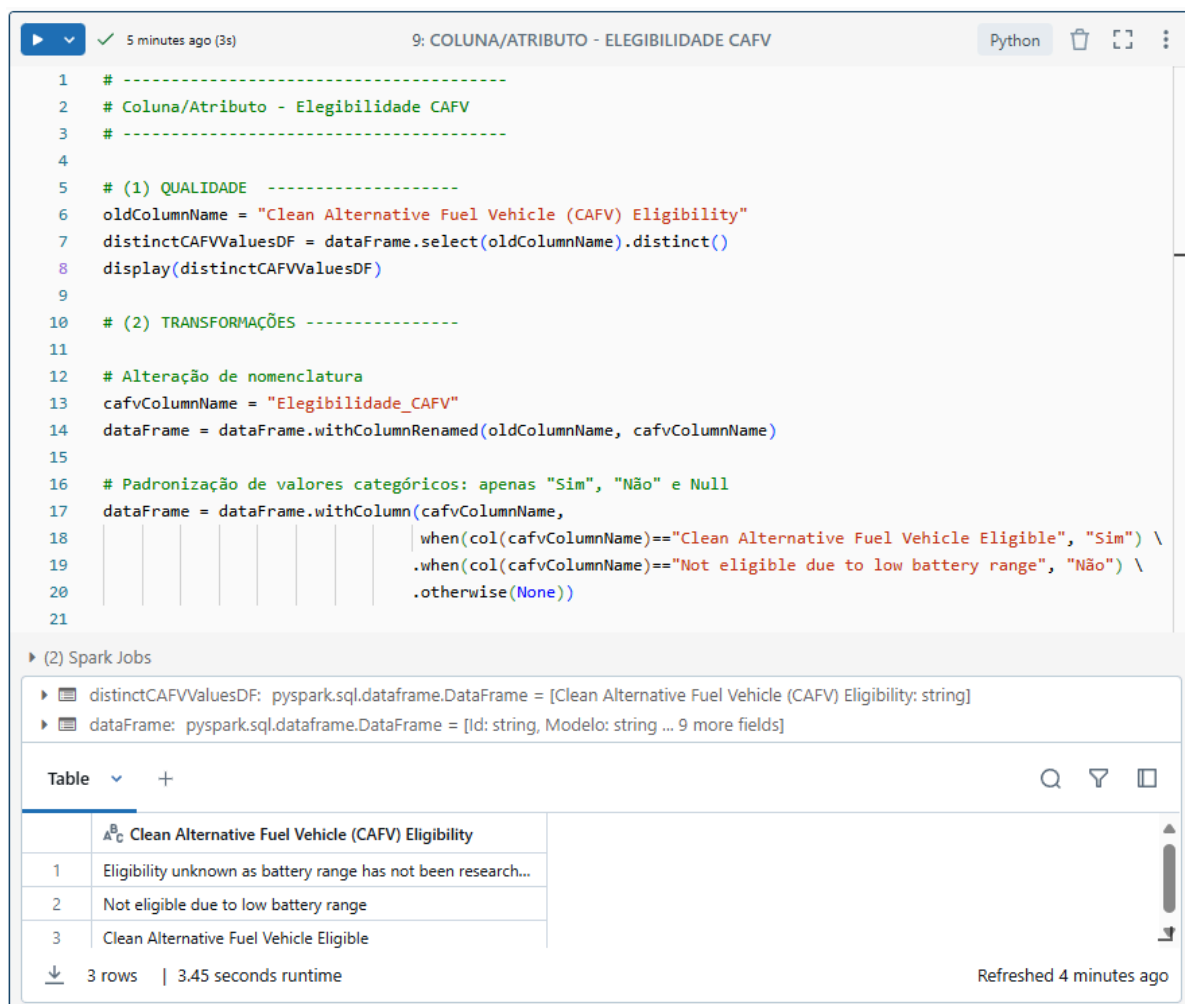
Figura 20 - Resultado da verificação de qualidade da coluna Valor_de_mercado

9. Coluna - Clean Alternative Fuel (CAFV) Eligibility

Esta coluna foi renomeada para "Elegibilidade_CAFV", alinhando-se com a convenção de nomenclatura simplificada adotada para as outras colunas.

Além disso, seus valores foram simplificados e padronizados em "Sim", "Não" ou nulo. Esta simplificação foi realizada para tornar mais direta a interpretação dos dados desta coluna. A seguir detalha-se as conversões realizadas para consolidar a padronização:

- "Clean Alternative Fuel Vehicle Eligible" foi convertido para "Sim".
- "Not eligible due to low battery range" foi convertido para "Não".
- "Eligibility unknown as battery range has not been researched" foi convertido para nulo.



```
1 # -----
2 # Coluna/Atributo - Elegibilidade CAFV
3 # -----
4
5 # (1) QUALIDADE -----
6 oldColumnName = "Clean Alternative Fuel Vehicle (CAFV) Eligibility"
7 distinctCAFVValuesDF = dataframe.select(oldColumnName).distinct()
8 display(distinctCAFVValuesDF)
9
10 # (2) TRANSFORMAÇÕES -----
11
12 # Alteração de nomenclatura
13 cafvColumnName = "Elegibilidade_CAFV"
14 dataframe = dataframe.withColumnRenamed(oldColumnName, cafvColumnName)
15
16 # Padronização de valores categóricos: apenas "Sim", "Não" e Null
17 dataframe = dataframe.withColumn(cafvColumnName,
18 | | | | | | | | when(col(cafvColumnName)=="Clean Alternative Fuel Vehicle Eligible", "Sim") \
19 | | | | | | | | .when(col(cafvColumnName)=="Not eligible due to low battery range", "Não") \
20 | | | | | | | | .otherwise(None))
21
```

▶ (2) Spark Jobs

▶ distinctCAFVValuesDF: pyspark.sql.dataframe.DataFrame = [Clean Alternative Fuel Vehicle (CAFV) Eligibility: string]

▶ dataframe: pyspark.sql.dataframe.DataFrame = [Id: string, Modelo: string ... 9 more fields]

Table ▾ + 🔍 🔍 📄

	⚙️ Clean Alternative Fuel Vehicle (CAFV) Eligibility
1	Eligibility unknown as battery range has not been research...
2	Not eligible due to low battery range
3	Clean Alternative Fuel Vehicle Eligible

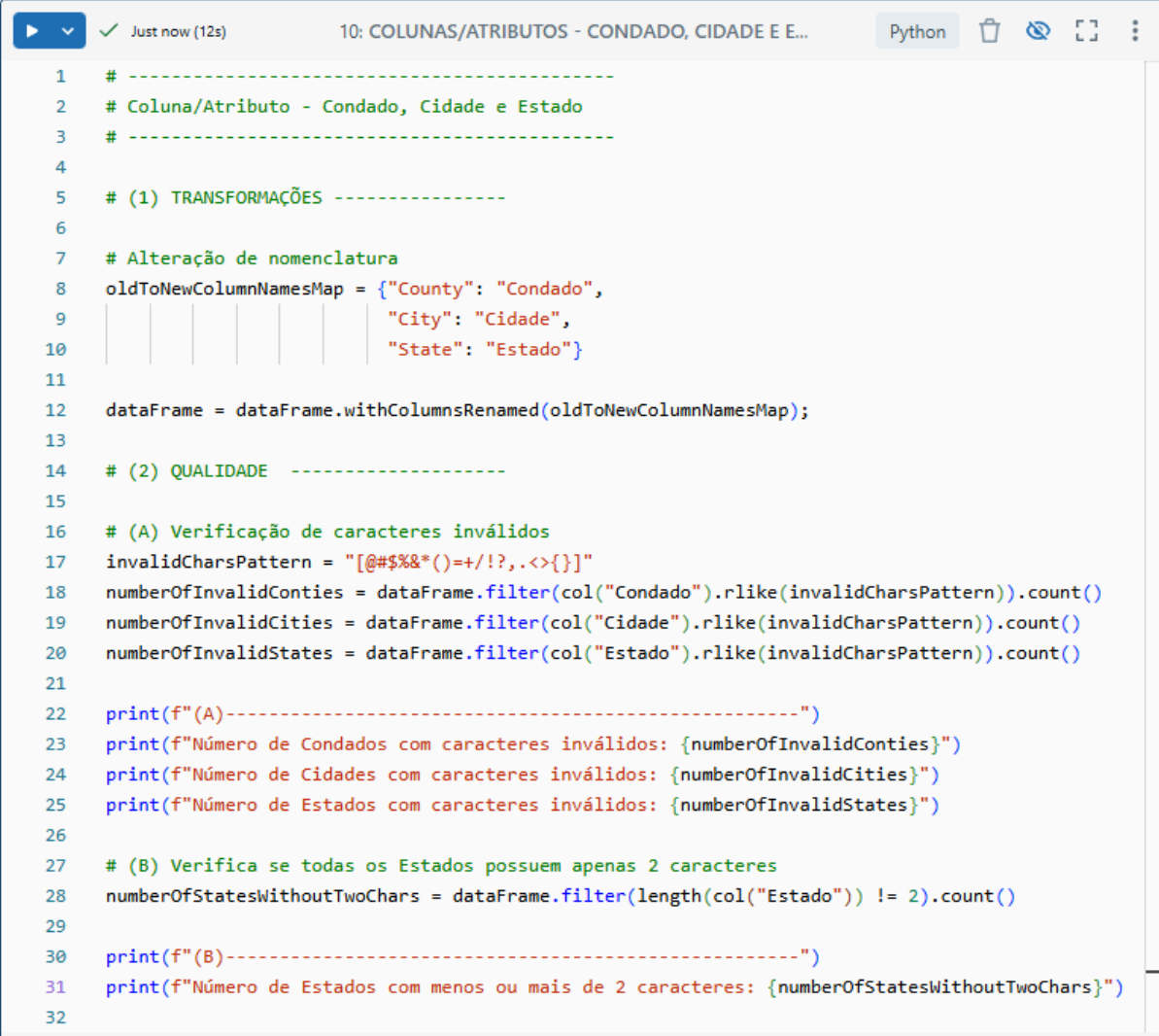
⬇ 3 rows | 3.45 seconds runtime Refreshed 4 minutes ago

Figura 21 - Transformações e validações da coluna Elegibilidade_CAFV

Assim como na coluna "Tipo", a verificação de qualidade na coluna "Elegibilidade_CAFV" foi realizada antes das transformações para comprovar seus três valores distintos, conforme apresentado na Figura 21.

10. Colunas - County, City e State

As últimas colunas da tabela "County", "City" e "State" foram renomeadas para "Condado", "Cidade" e "Estado", respectivamente.



```
1 # -----
2 # Coluna/Atributo - Condado, Cidade e Estado
3 # -----
4
5 # (1) TRANSFORMAÇÕES -----
6
7 # Alteração de nomenclatura
8 oldToNewColumnNamesMap = {"County": "Condado",
9                             "City": "Cidade",
10                             "State": "Estado"}
11
12 dataframe = dataframe.withColumnsRenamed(oldToNewColumnNamesMap);
13
14 # (2) QUALIDADE -----
15
16 # (A) Verificação de caracteres inválidos
17 invalidCharsPattern = "[@#$$%&*()=+/?!,.<>{}]"
18 numberOfInvalidConties = dataframe.filter(col("Condado").rlike(invalidCharsPattern)).count()
19 numberOfInvalidCities = dataframe.filter(col("Cidade").rlike(invalidCharsPattern)).count()
20 numberOfInvalidStates = dataframe.filter(col("Estado").rlike(invalidCharsPattern)).count()
21
22 print(f"(A)-----")
23 print(f"Número de Condados com caracteres inválidos: {numberOfInvalidConties}")
24 print(f"Número de Cidades com caracteres inválidos: {numberOfInvalidCities}")
25 print(f"Número de Estados com caracteres inválidos: {numberOfInvalidStates}")
26
27 # (B) Verifica se todas os Estados possuem apenas 2 caracteres
28 numberOfStatesWithoutTwoChars = dataframe.filter(length(col("Estado")) != 2).count()
29
30 print(f"(B)-----")
31 print(f"Número de Estados com menos ou mais de 2 caracteres: {numberOfStatesWithoutTwoChars}")
32
```

Figura 22 - Transformações e validações das colunas Condado, Cidade e Estado

As verificações de qualidade para estas colunas constituíram na busca por caracteres inválidos, como "!", "?," e ">" e, apenas na coluna "Estado", foi validado se

todos os valores possuíam exatamente dois caracteres, correspondendo à abreviação padrão dos estados americanos.

```

▶ dataframe: pyspark.sql.dataframe.DataFrame = [Id: string, Modelo: string ... 9 more fields]

(A)-----
Número de Condados com caracteres inválidos: 4
Número de Cidades com caracteres inválidos: 0
Número de Estados com caracteres inválidos: 0
(B)-----
Número de Estados com menos ou mais de 2 caracteres: 0

```

Figura 23 - Resultado da verificação de qualidade das colunas Condado, Cidade e Estado

Como ilustrado na Figura 23, foram identificados quatro condados com caracteres inválidos. Ao inspecioná-los utilizando o método `display()`, verificou-se que eles correspondem aos condados de St. Charles, St. Mary's e St. Lawrence. Todos esses condados são válidos, portanto, não há problemas com os dados.

```

33 # (C) Verificação de Condados com caracteres inválidos
34 invalidCharsPattern = "[@#$$%&*()=+/?,<>{}]"
35 invalidCounties = dataframe.filter(col("Condado").rlike(invalidCharsPattern))
36 display(invalidCounties)
37

```

▶ (11) Spark Jobs

▶ dataframe: pyspark.sql.dataframe.DataFrame = [Id: string, Modelo: string ... 9 more fields]
 ▶ invalidCounties: pyspark.sql.dataframe.DataFrame = [Id: string, Modelo: string ... 9 more fields]

Table ▾ + 🔍 🏠

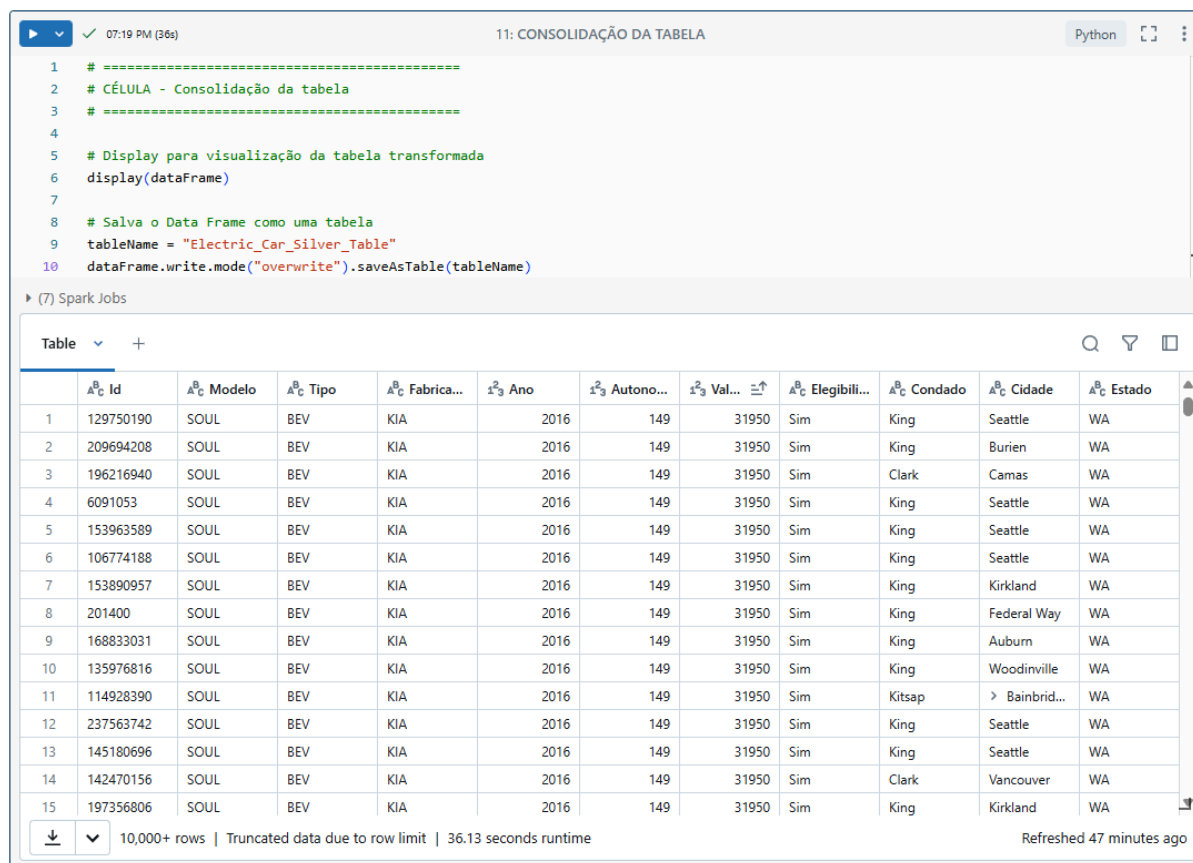
		¹ / ₃ Valor de mercado	^A / _C Elegibilidade CAFV	^A / _C Condado	^A / _C Cidade	^A / _C Estado
1	11	null	null	St. Charles	O Fallon	MO
2	46	null	Sim	St. Mary's	Lexington Park	MD
3	11	null	null	St. Mary's	Lexington Park	MD
4	11	null	null	St. Lawrence	Ogdensburg	NY

4 rows | 16.10 seconds runtime Refreshed 2 minutes ago

Figura 24 - Investigação de valores na coluna Condado

11. Consolidação da Tabela

Após realizar todas as transformações e verificações de qualidade nas colunas da tabela, o *Data Frame* resultante foi apresentado utilizando o método `display()`. Este passo permitiu uma visualização clara e direta da estrutura final dos dados, facilitando a revisão e validação das transformações aplicadas.



The screenshot shows a Databricks notebook interface. The top section displays a Python script with the following code:

```
1 # =====  
2 # CÉLULA - Consolidação da tabela  
3 # =====  
4  
5 # Display para visualização da tabela transformada  
6 display(dataFrame)  
7  
8 # Salva o Data Frame como uma tabela  
9 tableName = "Electric_Car_Silver_Table"  
10 dataFrame.write.mode("overwrite").saveAsTable(tableName)
```

Below the code, the notebook shows a table view with 15 rows and 12 columns. The columns are: Id, Modelo, Tipo, Fabrica..., Ano, Autono..., Val..., Elegibili..., Condado, Cidade, and Estado. The data represents electric cars, mostly from KIA, with various models and locations in Washington state.

	Id	Modelo	Tipo	Fabrica...	Ano	Autono...	Val...	Elegibili...	Condado	Cidade	Estado
1	129750190	SOUL	BEV	KIA	2016	149	31950	Sim	King	Seattle	WA
2	209694208	SOUL	BEV	KIA	2016	149	31950	Sim	King	Burien	WA
3	196216940	SOUL	BEV	KIA	2016	149	31950	Sim	Clark	Camas	WA
4	6091053	SOUL	BEV	KIA	2016	149	31950	Sim	King	Seattle	WA
5	153963589	SOUL	BEV	KIA	2016	149	31950	Sim	King	Seattle	WA
6	106774188	SOUL	BEV	KIA	2016	149	31950	Sim	King	Seattle	WA
7	153890957	SOUL	BEV	KIA	2016	149	31950	Sim	King	Kirkland	WA
8	201400	SOUL	BEV	KIA	2016	149	31950	Sim	King	Federal Way	WA
9	168833031	SOUL	BEV	KIA	2016	149	31950	Sim	King	Auburn	WA
10	135976816	SOUL	BEV	KIA	2016	149	31950	Sim	King	Woodinville	WA
11	114928390	SOUL	BEV	KIA	2016	149	31950	Sim	Kitsap	> Bainbrid...	WA
12	237563742	SOUL	BEV	KIA	2016	149	31950	Sim	King	Seattle	WA
13	145180696	SOUL	BEV	KIA	2016	149	31950	Sim	King	Seattle	WA
14	142470156	SOUL	BEV	KIA	2016	149	31950	Sim	Clark	Vancouver	WA
15	197356806	SOUL	BEV	KIA	2016	149	31950	Sim	King	Kirkland	WA

At the bottom of the table view, it indicates "10,000+ rows | Truncated data due to row limit | 36.13 seconds runtime" and "Refreshed 47 minutes ago".

Figura 25 - Trecho da tabela após as transformações

Para garantir a persistência dos dados processados e possibilitar futuras consultas e análises, o *Data Frame* foi salvo como uma tabela no Databricks utilizando o método `saveAsTable()`. Esse método permite que a tabela seja armazenada de forma eficiente dentro do ambiente do Databricks, facilitando o acesso, compartilhamento e manipulação dos dados processados.

A tabela foi nomeada "Electric_Car_Silver_Table", consolidando assim todo o trabalho de preparação e limpeza dos dados realizado ao longo do processo.

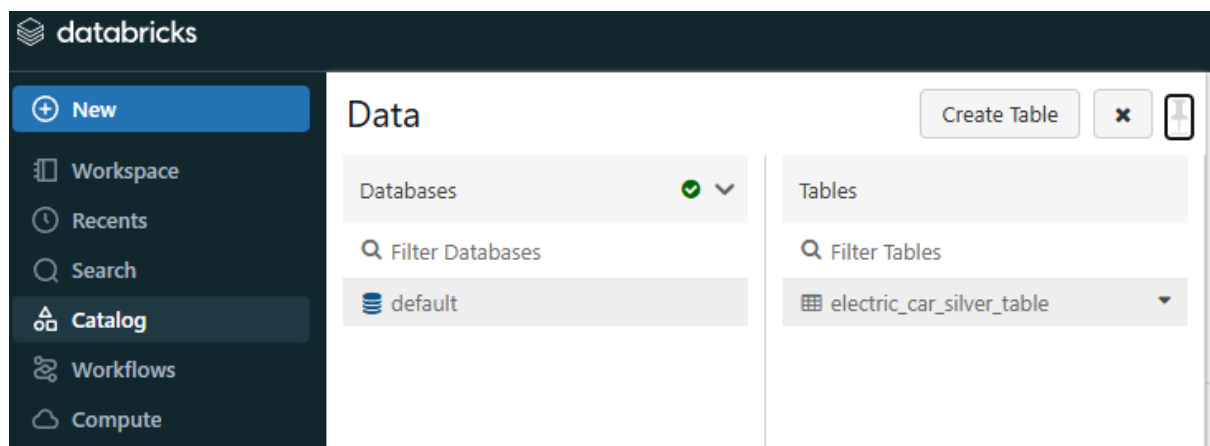
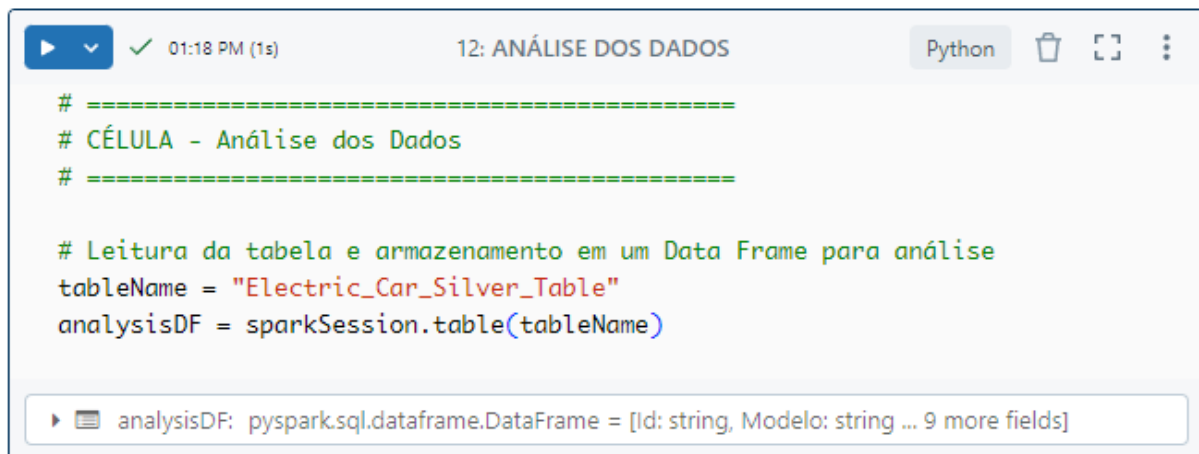


Figura 26 - Tabela salva no catálogo do Databricks

Análise

Para a etapa de análise dos dados, optou-se por continuar utilizando o PySpark devido à sua flexibilidade e simplicidade de aprendizado. O PySpark facilita a implementação de consultas complexas e a manipulação eficiente dos dados, garantindo uma análise robusta e eficiente.

Dessa forma, a etapa de análise foi iniciada pela leitura da tabela salva no catálogo do Databricks ao final da etapa de transformações. Utilizou-se o método `table()`, especificando o nome da tabela "Electric_Car_Silver_Table".



```
# =====  
# CÉLULA - Análise dos Dados  
# =====  
  
# Leitura da tabela e armazenamento em um Data Frame para análise  
tableName = "Electric_Car_Silver_Table"  
analysisDF = sparkSession.table(tableName)
```

▶ analysisDF: pyspark.sql.dataframe.DataFrame = [Id: string, Modelo: string ... 9 more fields]

Figura 27 - Leitura da tabela para análise

É importante ressaltar que embora as linhas da tabela representem os registros de carros elétricos no Departamento de Licenciamento de Washington, optou-se por interpretar esses registros como vendas. Esta abordagem foi adotada com base na premissa de que o registro implica na venda prévia dos veículos, antes de sua licenciatura.

Essa interpretação é válida no contexto analisado, pois reflete o momento em que os carros elétricos foram comercializados aos consumidores, independentemente de quando foram formalmente registrados. No entanto, é importante comentar que um grande número de carros elétricos não licenciados poderia prejudicar as conclusões das análises.

A seguir, serão apresentadas todas as respostas das perguntas e reflexões elencadas anteriormente, assim como análises e comentários a respeito das mesmas.

(1) Quais fabricantes possuem a maior quantidade de carros registrados?

Através deste questionamento busca-se entender quais os fabricantes de veículos possuem maior popularidade no estado de Washington. Este tipo de análise pode fornecer insights valiosos sobre o sucesso e a competitividade dos fabricantes de veículos na região, auxiliando na compreensão das preferências dos consumidores e das tendências do mercado local.

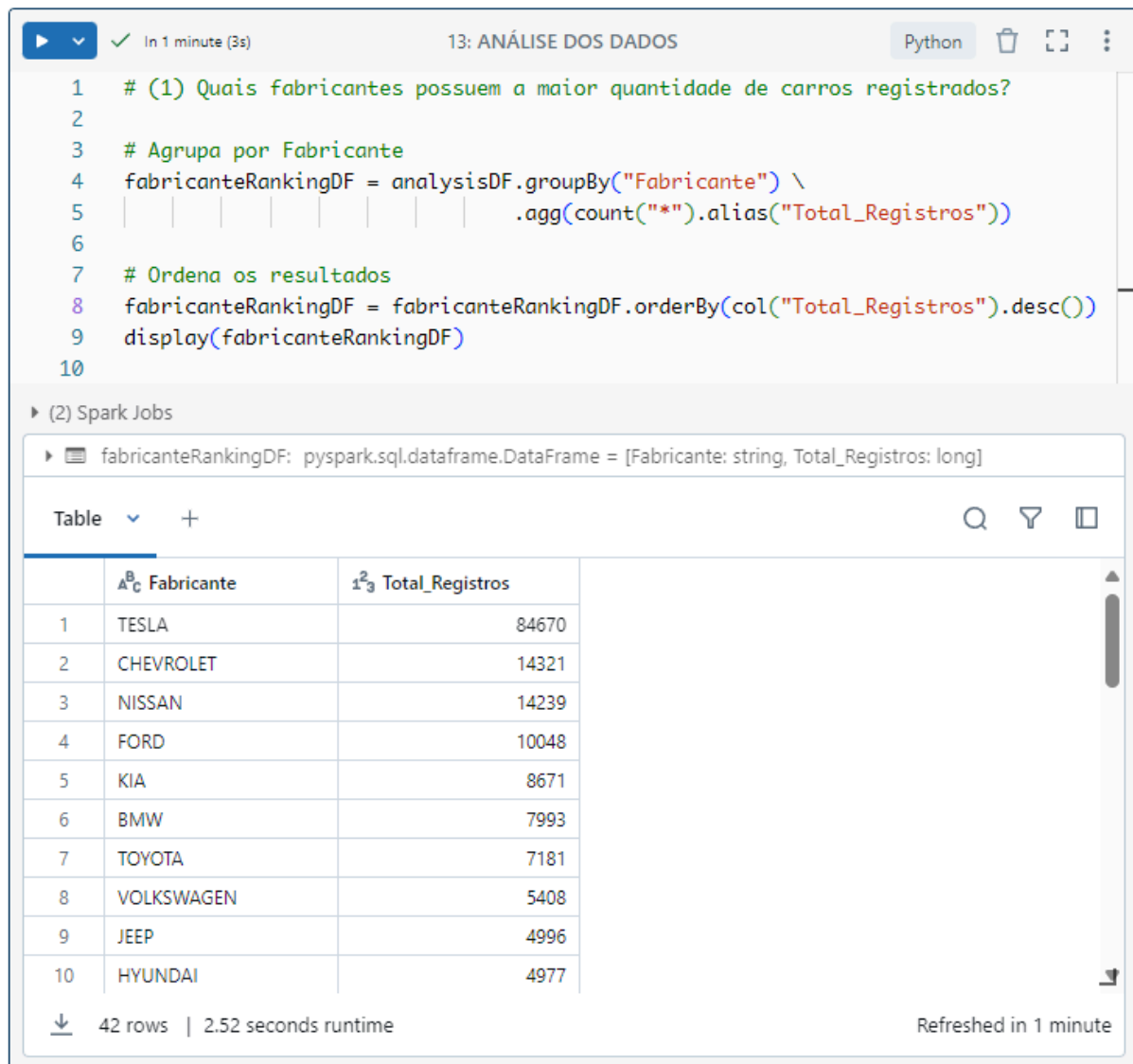


Figura 28 - Ranking de registros de carros por Fabricante (apenas 10 primeiros)

Os resultados revelam que a fabricante Tesla demonstra um sucesso significativo no estado de Washington, sugerindo uma possível superioridade de sua linha de carros elétricos em comparação com outros fabricantes. Além disso, fatores como a confiabilidade e posicionamento de mercado da marca também

podem estar envolvidos nesta liderança. A investigação com profundidade desta predominância não está dentro do escopo deste trabalho, porém é importante reconhecer que esses dados podem ser integrados a uma base maior de informações para tirar conclusões sobre a venda de carros elétricos.

(2) Quais os modelos de carros com maior número de registros?

O objetivo deste questionamento é similar ao anterior: analisar as preferências da população do estado de Washington em relação aos modelos de carros e também a variedade de modelos de sucesso de cada fabricante.

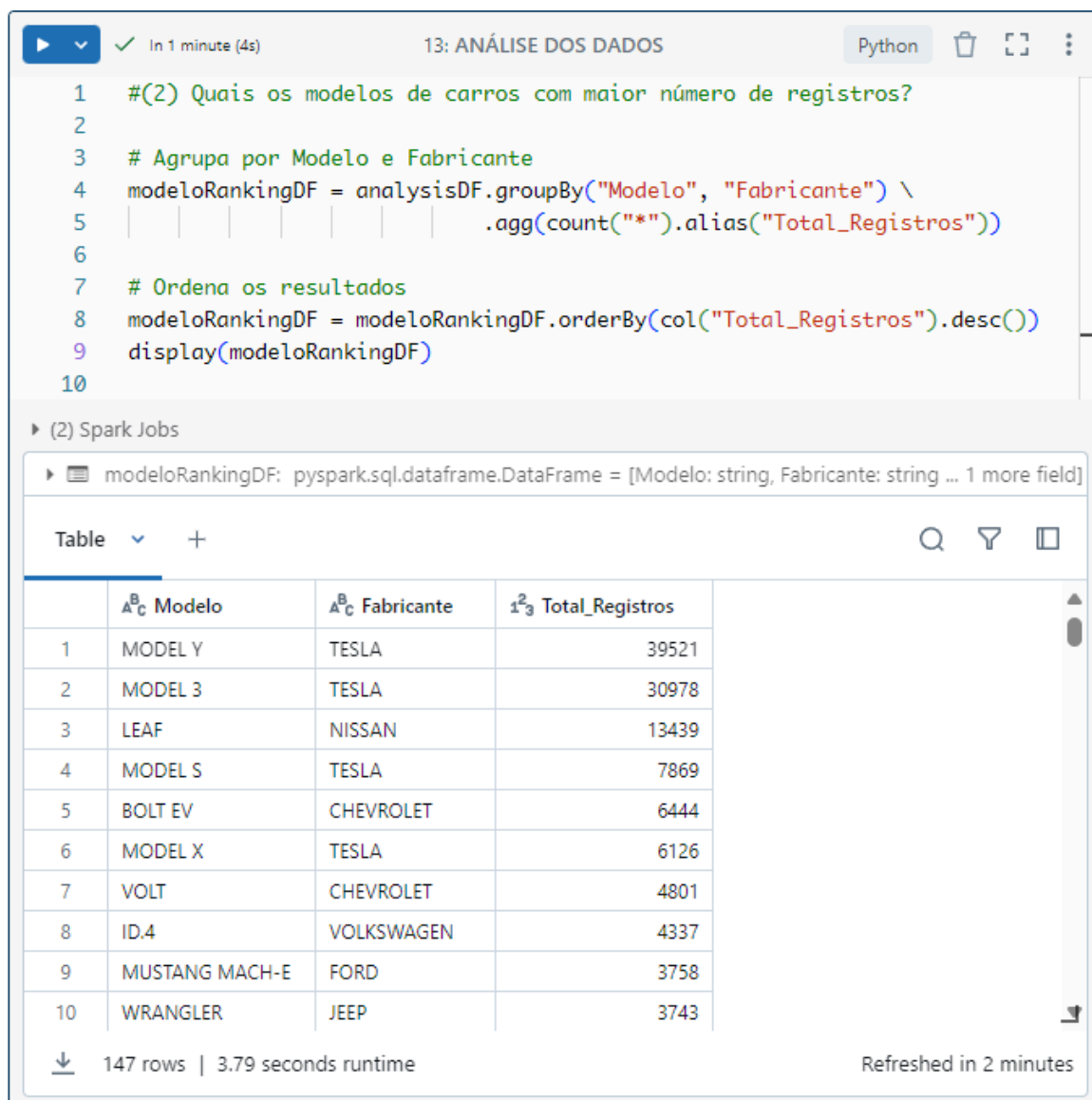


Figura 28 - Ranking de registros de carros por Modelo (apenas 10 primeiros)

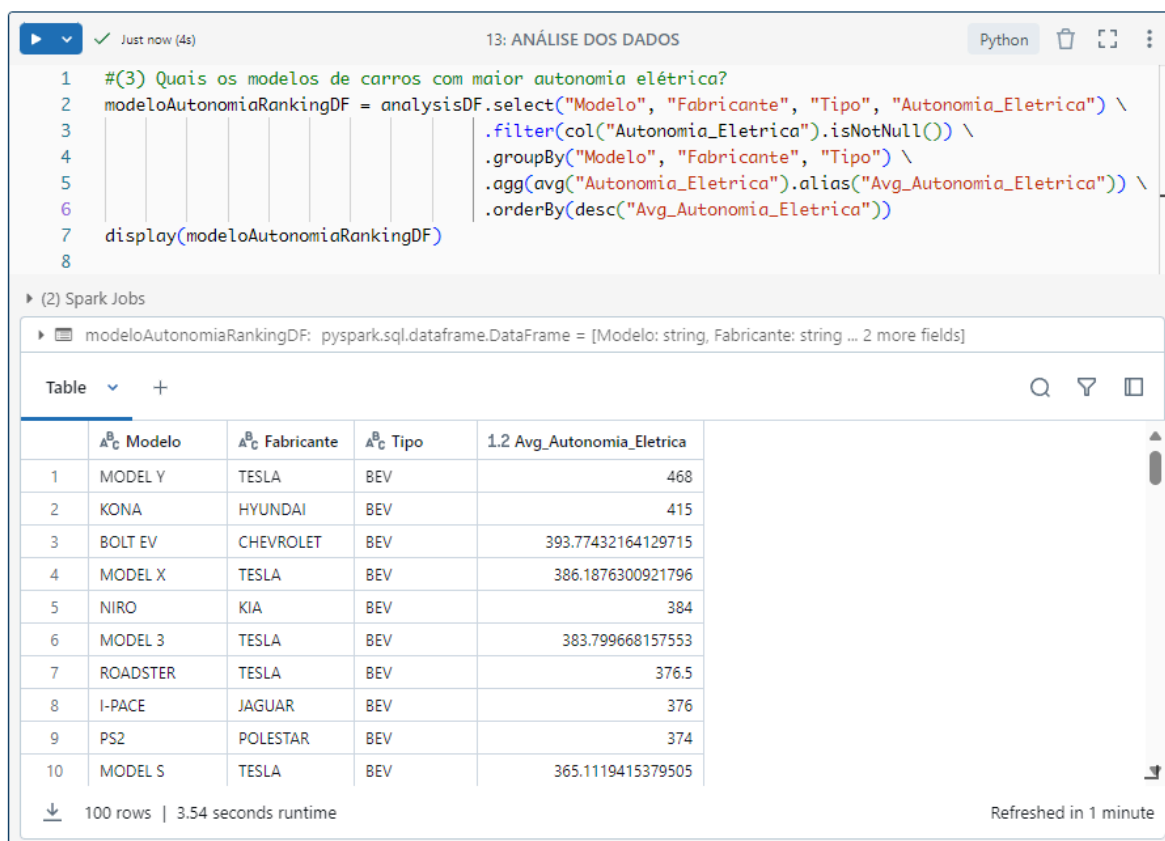
Pelos resultados do ranqueamento, observa-se que a fabricante Tesla apresenta uma distribuição mais homogênea de registros/vendas entre seus diferentes modelos de carro, com os dois primeiros modelos mais vendidos (MODEL Y e MODEL 3) registrando quantidades semelhantes de registros.

Além disso, é possível visualizar uma clara preferência do público quanto aos modelos da Tesla, pois os modelos MODEL Y e MODEL 3 apresentam uma liderança considerável de registros quando comparados aos modelos MODEL X e MODEL S

Por outro lado, a fabricante Nissan, apesar de ocupar o terceiro lugar em registros/vendas de carros elétricos no estado (14.239 registros), mostra-se competitiva no mercado apenas com o modelo Leaf (13.439 registros), que se destaca significativamente em termos de registros.

(3) Quais os modelos de carros com maior autonomia elétrica?

Este questionamento, em conjunto com o anterior, busca entender as escolhas dos consumidores em relação à eficiência dos veículos elétricos.



```

1 # (3) Quais os modelos de carros com maior autonomia elétrica?
2 modeloAutonomiaRankingDF = analysisDF.select("Modelo", "Fabricante", "Tipo", "Autonomia_Eletrica") \
3   .filter(col("Autonomia_Eletrica").isNotNull()) \
4   .groupBy("Modelo", "Fabricante", "Tipo") \
5   .agg(avg("Autonomia_Eletrica").alias("Avg_Autonomia_Eletrica")) \
6   .orderBy(desc("Avg_Autonomia_Eletrica"))
7 display(modeloAutonomiaRankingDF)
8

```

(2) Spark Jobs

modeloAutonomiaRankingDF: pyspark.sql.dataframe.DataFrame = [Modelo: string, Fabricante: string ... 2 more fields]

	Modelo	Fabricante	Tipo	1.2 Avg_Autonomia_Eletrica
1	MODEL Y	TESLA	BEV	468
2	KONA	HYUNDAI	BEV	415
3	BOLT EV	CHEVROLET	BEV	393.77432164129715
4	MODEL X	TESLA	BEV	386.1876300921796
5	NIRO	KIA	BEV	384
6	MODEL 3	TESLA	BEV	383.799668157553
7	ROADSTER	TESLA	BEV	376.5
8	I-PACE	JAGUAR	BEV	376
9	PS2	POLESTAR	BEV	374
10	MODEL S	TESLA	BEV	365.1119415379505

100 rows | 3.54 seconds runtime

Refreshed in 1 minute

Figura 29 - Ranking de carros com maior Autonomia Elétrica

Vale destacar que foi calculada a média dos valores de autonomia elétrica de cada modelo, proporcionando uma visão clara sobre quais modelos, em média, oferecem maior autonomia.

Como evidenciado nas Figuras 28 e 29, os quatro modelos mais populares da Tesla estão entre os dez carros elétricos com maior autonomia, o que pode ser um atrativo significativo para potenciais compradores de veículos elétricos. Um destaque notável vai para o modelo BOLT EV da Chevrolet, que não apenas se posiciona como o terceiro mais eficiente em termos de autonomia, mas também lidera em número de registros/vendas entre os modelos dessa fabricante.

Para entender mais profundamente as preferências dos consumidores em relação aos modelos de carros elétricos, seria necessário avaliar não apenas a eficiência, mas também as características técnicas oferecidas por cada modelo. No entanto, é interessante observar que com os dados disponíveis é possível identificar algumas tendências claras.

(4) Quais os tipos de carros mais com maior quantidade de registros?

Este questionamento visa analisar qual é a preferência dos consumidores em relação aos tipos de carros elétricos disponíveis.

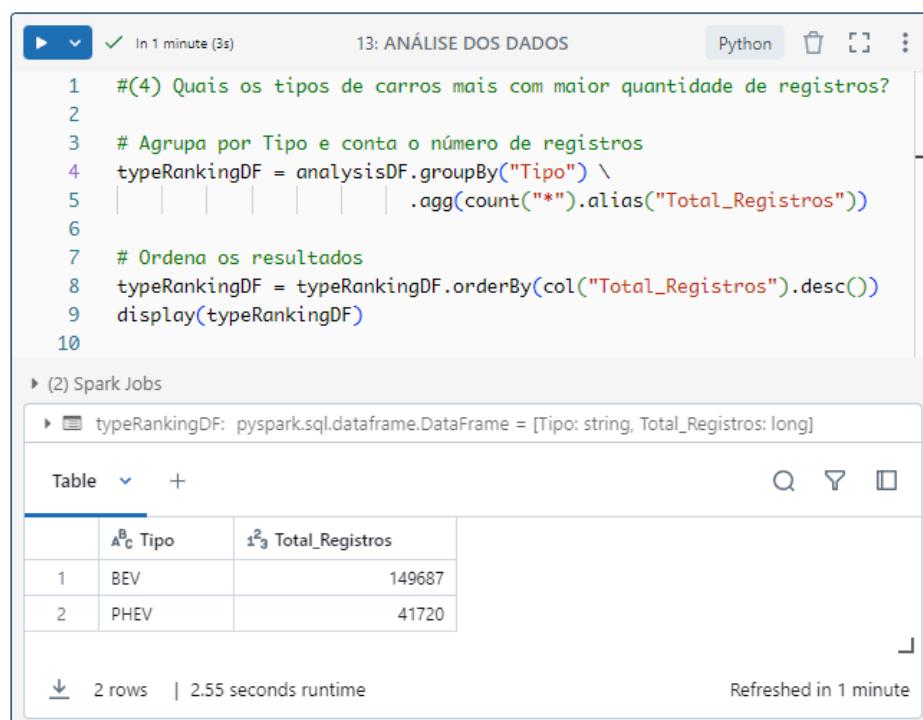


Figura 30 - Registros totais por Tipo de veículo elétrico

A partir dos resultados obtidos, observa-se que os modelos do tipo BEV (Battery Electric Vehicle) são significativamente mais procurados em comparação aos PHEV (Plug-in Hybrid Electric Vehicle). Isso pode indicar uma preferência geral por carros elétricos que operam exclusivamente com baterias, sem a necessidade de um motor a combustão auxiliar. Esta preferência pode refletir tanto preocupações ambientais quanto a busca por soluções de mobilidade sustentáveis e econômicas.

(5) Quais as cidades com maior número de carros registrados?

O objetivo ao analisar os condados e cidades com o maior número de carros registrados é compreender como os veículos elétricos estão distribuídos geograficamente no estado de Washington.

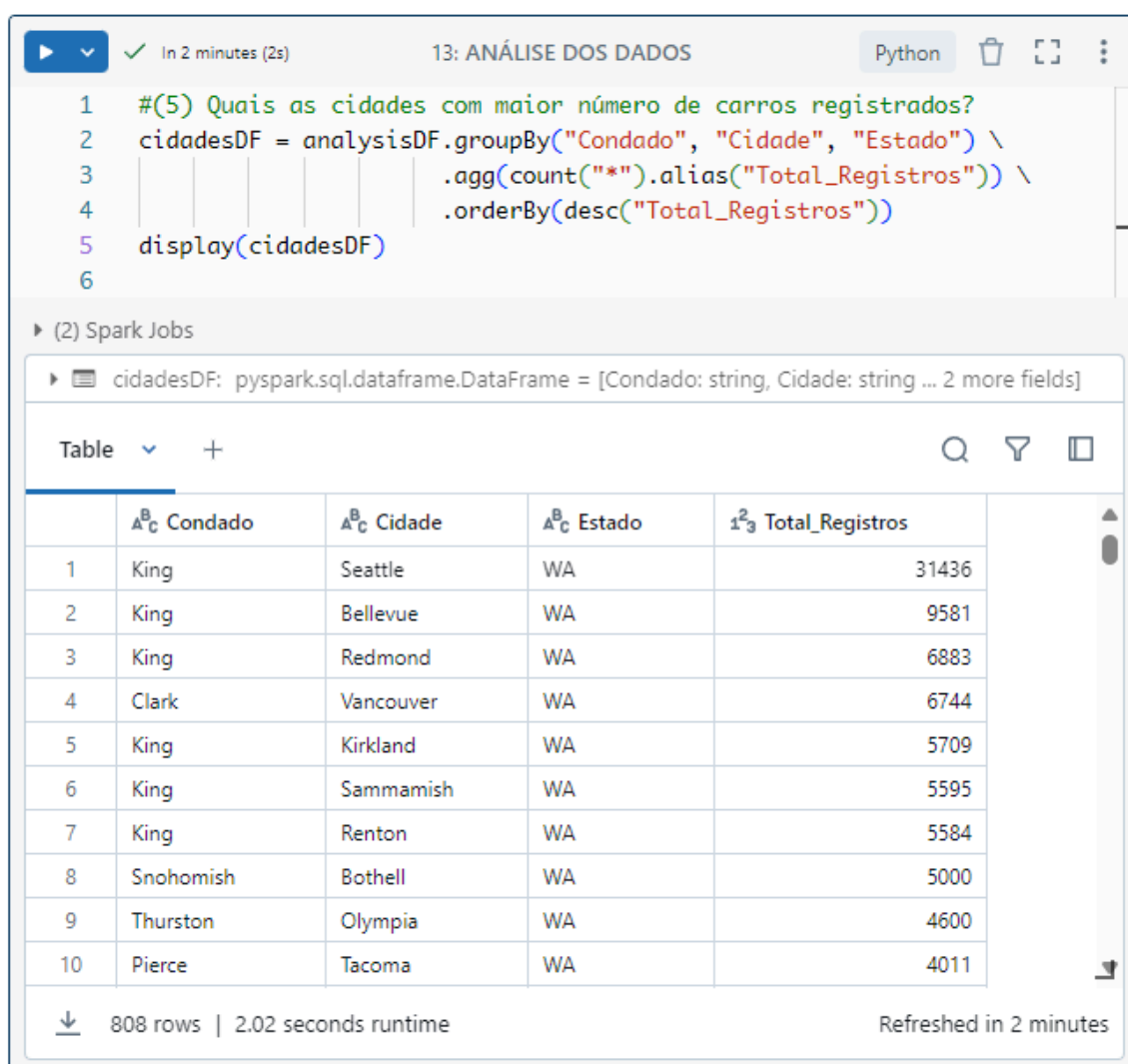


Figura 31 - Total de registros de veículos agrupados por Condado, Cidade e Estado

Conforme ilustrado na Figura 31, a cidade de Seattle se destacou com a maior quantidade de registros de veículos elétricos. Isso já era esperado visto que se trata da maior cidade do estado, no entanto, a justificativa da adoção desta tecnologia pode estar atrelada à presença, em maior quantidade, de infraestruturas de suporte a veículos elétricos e a preocupações ambientais mais acentuadas na região.

Observa-se que o condado de King, onde Seattle está localizada, possui o maior número de cidades listadas no topo da tabela. Isso sugere que a região, provavelmente, deve oferecer um suporte mais robusto para proprietários de veículos elétricos.

(6) Qual a porcentagem dos registros são de estados diferentes de Washington?

Neste item o propósito é avaliar se existe interesse por parte de outros estados em registrar ou comprar veículos no estado de Washington.



```
1  #(6) Qual a porcentagem dos registros são de estados diferentes de
2  Washington?
3  # Conta o total de registros na tabela
4  totalRows = analysisDF.count()
5
6  # Conta os registros que não são do estado de Washington
7  notWACount = analysisDF.filter(col("Estado") != "WA").count()
8
9  # Calcula a porcentagem
10 notWAPercentage = (notWACount / totalRows) * 100
11
12 print(f"Existem {notWACount} veículos em estados diferentes de
13 Washington, isso representa {notWAPercentage:.2f}% do total.")
```

► (6) Spark Jobs

Existem 414 veículos em estados diferentes de Washington, isso representa 0.22% do total.

Figura 32 - Porcentagem de proprietários de veículos de outros estados

Os resultados da revelam que apenas 0,22% dos proprietários de carros elétricos registrados em Washington residem em outros estados, indicando uma presença mínima de interesse externo.

É importante ressaltar que, com base nos dados limitados disponíveis, parece haver uma ausência significativa de interesse por parte de residentes de outros estados em registrar carros elétricos em Washington, sugerindo uma possível falta de incentivos ou políticas de atração para este mercado específico.

No entanto, é crucial reconhecer que essa conclusão é baseada em uma análise restrita dos registros disponíveis, e não é possível afirmar com certeza absoluta a inexistência de incentivos ou estratégias específicas adotadas pelo estado de Washington para promover a comercialização de veículos elétricos.

Considerações finais

A coluna "Id" não foi incorporada nas análises devido à sua falta de relevância direta para as questões de negócio abordadas neste trabalho. Apesar disso, optou-se por mantê-la no conjunto de dados, reconhecendo sua importância como uma identificação única de cada linha da tabela, facilitando assim a integridade e rastreabilidade dos dados.

Além disso, a coluna "Valor_de_mercado" não foi incluída nas análises devido à alta incidência de valores nulos, que poderiam distorcer os resultados. A presença significativa de valores ausentes neste atributo destacou a importância da qualidade dos dados na condução de análises precisas e confiáveis.

```
9  #(Qualidade) Contagem de valores de mercado nulos
10  numberOfRows = analysisDF.count()
11  nullMSRPValues = analysisDF.filter(col("Valor_de_mercado").isNull()).count()
12
13  print(f"Quantidade de linhas da tabela: {numberOfRows}")
14  print(f"Quantidade de Valores de mercado nulos: {nullMSRPValues}")
15
```

► (6) Spark Jobs

► analysisDF: pyspark.sql.dataframe.DataFrame = [Id: string, Modelo: string ... 9 more fields]

Quantidade de linhas da tabela: 191407
Quantidade de Valores de mercado nulos: 188069

Figura 33 - Contagem de valores de mercado nulos

Essas decisões garantiram a relevância e a precisão das conclusões apresentadas neste estudo. Contudo, é essencial reconhecer que análises futuras podem se beneficiar de um conjunto de dados mais completo e atualizado.

Autoavaliação

A princípio, todos os objetivos técnicos e pessoais propostos foram plenamente alcançados neste trabalho. O processo de aprendizado envolvendo ETL em plataformas de nuvem e a utilização de frameworks como PySpark para o processamento e análise de dados foi extremamente enriquecedor. Antes deste projeto, eu não possuía familiaridade com a maioria das ferramentas necessárias para sua realização, o que torna o progresso conquistado particularmente significativo.

Ao finalizar o trabalho, identificou-se uma oportunidade significativa de melhoria que, devido a limitações de tempo, não pôde ser implementada completamente. Observou-se a necessidade de desacoplar o pipeline de verificação de qualidade do pipeline de transformação dos dados. Essa separação poderia oferecer benefícios substanciais, como maior flexibilidade nos fluxos de trabalho. Além disso, possibilitaria que equipes distintas executem os processos de maneira paralela, promovendo uma eficiência operacional ainda maior.

Outro aspecto identificado para melhoria, porém não implementado devido a restrições de tempo, é a apresentação dos dados. Seria benéfico explorar outros frameworks Python, como Matplotlib, para a criação de gráficos e dashboards mais interativos e informativos. Reconhece-se que, em um contexto real, a visualização eficaz dos dados desempenha um papel crucial na comunicação de insights e na tomada de decisões informadas. Esta etapa não realizada representa uma oportunidade clara para futuras iterações do projeto, visando enriquecer a análise e o entendimento dos dados obtidos.

Por fim, gostaria de agradecer a todos os colegas e professores da disciplina de engenharia de dados pela experiência enriquecedora proporcionada ao longo deste projeto. Suas orientações, insights e colaboração foram fundamentais para o sucesso deste trabalho e para o meu desenvolvimento profissional. Agradeço também pelo apoio contínuo e pelo ambiente colaborativo que contribuíram significativamente para minha jornada de aprendizado nesta área.