

## Atividade Final - Nome Sobrenome (Python)

[Import notebook](#)

# Identificação

**Aluno:** Denilson Nunes do Lago

**Turma:** Data Science - 01/2025 Engenharia de Dados - Turma A

## Cenário

Uma empresa que vende produtos pela internet gostaria de conhecer melhor o perfil de seus clientes para impulsionar suas vendas. Além de fazer um acompanhamento do que está sendo vendido no site.

A área de marketing tem necessidade de explorar as informações sobre as vendas de produtos para sugerir promoções, descontos, vendas casadas e novos produtos aos seus clientes. E, também, realizar campanhas a públicos específicos.

Além disso, há gestores da empresa que necessitam monitorar as vendas em *near real-time*

Todo evento/ação do usuário ao comprar no site é publicado em um serviço de mensageria, onde se pode capturar informações de carrinho de compras e vendas realizadas.

Seu papel é criar um *pipeline* para extração e limpeza desses dados a fim de atender essas necessidades.

Utilizando como fonte de dados o arquivo *ecommerce.json*, faça as atividades.

## Atividade - Carga de Dados

- Criar e carregar uma tabela chamada `ecommerce_bronze`, com os dados *raw* do diretório `/Volumes/senac/default/ecommerce`
- Criar e carregar uma tabela chamada `ecommerce_silver`, com base na tabela da camada anterior, atendendo os seguintes requisitos:

;

## o Colunas:

`device, event_name, event_previous_timestamp, event_timestamp, city (ge`

- o As colunas do tipo *epoch* devem ser convertidas para *timestamp*
- o A coluna *items* deve ser *exploded*. Com há valores nulos, dê uma olhada em *explode\_outer*

- Criar e carregar uma tabela chamada *idades\_gold* com os dados distintos referentes a localização dos eventos (*geo*)
- Criar e carregar uma tabela chamada *produtos\_gold* com as informações distintas sobre os itens vendidos no *ecommerce*:

`identificador, nome, preço unitário`

- Criar e carregar uma tabela chamada *vendas\_gold* que contenha o


`user_id, coupon, item_id, event_date (derivado event_timestamp - formato '!`

, agregando a quantidade (*quantity*) e o valor total de venda do produto


- o Lembrando que o valor de venda (*total*) é o preço unitário (*price\_in\_usd*) x quantidade
- o Apenas as compras que foram efetivadas (`event_name = finalize`)

```
%sql
CREATE OR REPLACE TABLE ecommerce_bronze (value string);

INSERT INTO ecommerce_bronze (value)
select * from read_files('/Volumes/senac/default/ecommerce',
format => 'text');
```

►  \_sqldf: pyspark.sql.connect.dataframe.DataFrame = [num\_affected\_rows: long, num\_inserted\_rows: long]

Table


 This result is stored as `_sqldf` and can be used in other Python and SQL cells.

```
%sql
CREATE
OR REPLACE TEMP VIEW ecommerce_bronze_view AS (
  select
    from_json(
      value,
      'STRUCT<device: STRING, ecommerce: STRUCT<purchase_revenue_in_usd:
DOUBLE, total_item_quantity: BIGINT, unique_items: BIGINT>, event_name:
STRING, event_previous_timestamp: BIGINT, event_timestamp: BIGINT, geo:
STRUCT<city: STRING, state: STRING>, items: ARRAY<STRUCT<coupon: STRING,
item_id: STRING, item_name: STRING, item_revenue_in_usd: DOUBLE,
price_in_usd: DOUBLE, quantity: BIGINT>>, traffic_source: STRING,
user_first_touch_timestamp: BIGINT, user_id: STRING>'
    ) as json_result
  from
    ecommerce_bronze
);

SELECT * FROM ecommerce_bronze_view;
```


▶  `_sqldf: pyspark.sql.connect.dataframe.DataFrame = [json_result: struct]`

**Table**


 This result is stored as `_sqldf` and can be used in other Python and SQL cells.

```
%sql
CREATE
OR REPLACE TEMP VIEW ecommerce_bronze_formatted AS (
  SELECT
    json_result.device,
    json_result.event_name,
    CAST(json_result.event_previous_timestamp / 1e6 AS timestamp) as
event_previous_timestamp,
    CAST(json_result.event_timestamp / 1e6 AS timestamp) as
event_timestamp,
    json_result.geo.city as city,
    json_result.geo.state as state,
    json_result.traffic_source,
    json_result.user_id,
    explode_outer(json_result.items) as item
  FROM ecommerce_bronze_view
);

SELECT * FROM ecommerce_bronze_formatted
```

►  `_sqldf`: pyspark.sql.connect.dataframe.DataFrame = [device: string, event\_name: string ... 7 more fields]

Table


 This result is stored as `_sqldf` and can be used in other Python and SQL cells.

%sql


```
CREATE TABLE ecommerce_silver (  
  device STRING,  
  event_name STRING,  
  event_previous_timestamp TIMESTAMP,  
  event_timestamp TIMESTAMP,  
  city STRING,  
  state STRING,  
  traffic_source STRING,  
  user_id STRING,  
  coupon STRING,  
  item_id STRING,  
  item_name STRING,  
  item_revenue_in_usd DOUBLE,  
  price_in_usd DOUBLE,  
  quantity BIGINT  
);
```

```
%sql
INSERT INTO
  ecommerce_silver (
    device,
    event_name,
    event_previous_timestamp,
    event_timestamp,
    city,
    state,
    traffic_source,
    user_id,
    coupon,
    item_id,
    item_name,
    item_revenue_in_usd,
    price_in_usd,
    quantity
  )
SELECT
  device,
  event_name,
  event_previous_timestamp,
  event_timestamp,
  city,
  state,
  traffic_source,
  user_id,
  item.coupon,
  item.item_id,
  item.item_name,
  item.item_revenue_in_usd,
  item.price_in_usd,
  item.quantity
FROM
  ecommerce_bronze_formatted;

SELECT
  *
FROM
  ecommerce_silver
```

►  \_sqlidf: pyspark.sql.connect.dataframe.DataFrame = [device: string, event\_name: string ... 12 more fields]

Table

 This result is stored as `_sqldf` and can be used in other Python and SQL cells.

```
%sql
```


```
CREATE TABLE cidades_gold (  
  city STRING,  
  state STRING  
);
```

```
%sql
INSERT INTO
  cidades_gold (city, state)
SELECT
  DISTINCT city,
  state
FROM
  ecommerce_silver;

select * from cidades_gold
```

►  \_sqldf: pyspark.sql.connect.dataframe.DataFrame = [city: string, state: string]

**Table**

 This result is stored as `_sqldf` and can be used in other Python and SQL cells.




```
%sql
```


```
CREATE TABLE produtos_gold (  
  item_id STRING,  
  item_name STRING,  
  price_in_usd double  
);
```

```
%sql
```

```
INSERT INTO  
  produtos_gold (item_id, item_name, price_in_usd)  
SELECT  
  DISTINCT item_id,  
  item_name,  
  price_in_usd  
FROM  
  ecommerce_silver  
WHERE item_id IS NOT NULL;  
select  
  *  
from  
  produtos_gold;
```

►  \_sqlidf: pyspark.sql.connect.dataframe.DataFrame = [item\_id: string, item\_name: string ... 1 more field]

**Table**


 This result is stored as `_sqlidf` and can be used in other Python and SQL cells.

%sql


```
CREATE TABLE vendas_gold (  
  user_id STRING,  
  coupon string,  
  item_id string,  
  event_date timestamp,  
  city string,  
  state string,  
  quantity Bigint,  
  total Double  
);
```

%sql

```
INSERT INTO vendas_gold (user_id, coupon, item_id, event_date, city, state,  
  quantity, total)  
SELECT  
  user_id,  
  coupon,  
  item_id,  
  DATE(event_timestamp) AS event_date,  
  city,  
  state,  
  SUM(quantity) AS _quantity_,  
  SUM(quantity * price_in_usd) AS _total_  
FROM  
  ecommerce_silver  
WHERE  
  event_name = 'finalize'  
GROUP BY  
  user_id, coupon, item_id, event_date, city, state;
```

►  \_sqldf: pyspark.sql.connect.dataframe.DataFrame = [num\_affected\_rows: long, num\_inserted\_rows: long]

Table

 This result is stored as \_sqldf and can be used in other Python and SQL cells.

## Validação

Execute as células abaixo para verificar possíveis erros na carga dos dados

```
def check_table_results(table_name, num_rows, column_names):  
    assert spark.table(table_name), f"Tabela {table_name} inexistente"  
    assert set(spark.table(table_name).columns) == set(column_names), f"As  
colunas da tabela {table_name} nao foram criadas conforme orientacao"  
    assert spark.table(table_name).count() == num_rows, f"A tabela  
{table_name} deveria ter {num_rows} registros"
```

```

check_table_results("ecommerce_bronze", 500000, ['value'])
check_table_results("ecommerce_silver", 515101, ['device', 'event_name',
'event_previous_timestamp', 'event_timestamp', 'city', 'state',
'traffic_source', 'user_id', 'coupon' , 'item_id', 'item_name',
'item_revenue_in_usd', 'price_in_usd', 'quantity' ])
check_table_results("cidades_gold", 6549, ['city' , 'state'])
check_table_results("produtos_gold", 12, ['item_id' , 'item_name' ,
'price_in_usd'])
check_table_results("vendas_gold", 10719, ['user_id' , 'coupon' ,
'item_id', 'event_date', 'city' , 'state' , 'quantity', 'total'])
print("Tudo certo! Te vejo por aí, mas não se esqueça de responder as
próximas questões ;-)")

```

Tudo certo! Te vejo por aí, mas não se esqueça de responder as próximas questões ;-)

## Atividade - Consulta aos dados

Com base nas tabelas carregadas na camada *gold*, crie consultas que respondam os questionamentos abaixo, em células separadas.

1. Qual foi o valor total de vendas, onde foi utilizado algum cupom?
2. Faça um *ranking* dos 5 produtos mais vendidos, em termos de quantidade, em junho/2020. Projete o nome do produto, a quantidade vendida e o valor total.
3. Qual são os quatro nomes de cidades que mais aparecem em estados diferentes?
4. Monte uma consulta, mostrando seu resultado, onde cada `item_id` é transposto para uma nova coluna. Agrupe por cidade e estado. Cada coluna deve ser pivotada com base na quantidade e total da venda.

%sql

-- 1. Qual foi o valor total de vendas, onde foi utilizado algum cupom?


```

SELECT SUM(total) AS total_vendas_com_cupom
FROM vendas_gold
WHERE coupon IS NOT NULL


```

►  \_sqldf: pyspark.sql.connect.dataframe.DataFrame = [total\_vendas\_com\_cupom: double]

Table

 This result is stored as `_sqldf` and can be used in other Python and SQL cells.


```
%sql
```

 `> UsageError: %%sql is a cell magic, but the cell body is empty.`


```
%sql
```

```
-- 2 Faça um ranking dos 5 produtos mais vendidos, em termos de quantidade,
em junho/2020. Projete o nome do produto, a quantidade vendida e o valor
total.
```

```
SELECT
    item_id,
    SUM(quantity) AS quantity,
    SUM(total) AS value
FROM
    vendas_gold
WHERE
    event_date >= '2020-06-01' AND event_date <= '2020-06-30'
GROUP BY
    item_id
ORDER BY
    quantity DESC
LIMIT 5
```

►  `_sqldf`: pyspark.sql.connect.dataframe.DataFrame = [item\_id: string, quantity: long ... 1 more field]


**Table**

 This result is stored as `_sqldf` and can be used in other Python and SQL cells.


%sql



-- 3 Quais são os quatro nomes de cidades que mais aparecem em estados diferentes?

```
SELECT
  city,
  COUNT(DISTINCT state) AS qt_state
FROM
  cidades_gold
GROUP BY
  city
ORDER BY
  qt_state DESC
LIMIT
  4
```

►  `_sqldf: pyspark.sql.connect.dataframe.DataFrame = [city: string, qt_state: long]`

**Table**

 This result is stored as `_sqldf` and can be used in other Python and SQL cells.

- ▶  `df_vendas_gold: pyspark.sql.connect.dataframe.DataFrame = [user_id: string, coupon: string ... 6 more fields]`
- ▶  `pivot_df: pyspark.sql.connect.dataframe.DataFrame = [city: string, state: string ... 24 more fields]`

#### Table

