

✓ Import das Bibliotecas e Configuração do PySpark

```
# Instalar as dependências
!apt-get update -qq
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://archive.apache.org/dist/spark/spark-3.5.4/spark-3.5.4-bin-hadoop3.tgz
!tar xf spark-3.5.4-bin-hadoop3.tgz
!pip install -q findspark
!pip install pyspark==3.4.0

import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.5.4-bin-hadoop3"
import findspark
findspark.init()

import requests
import tarfile
import gzip
import shutil
import os
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .master('local[*]') \
    .appName("Iniciando com Spark") \
    .config('spark.ui.port', '4050') \
    .getOrCreate()
```

✓ ETL

```
def load_raw_data(url: str, nome_arquivo: str):
    """
    Baixa, descompacta e lê arquivos grandes (.json.gz, .csv.gz, .tar.gz) usando PySpark.

    Parâmetros:
        url (str): URL do arquivo
        nome_arquivo (str): Nome do arquivo a ser salvo localmente

    Retorna:
        pyspark.sql.DataFrame: DataFrame carregado com PySpark
    """
    # Baixar o arquivo
    response = requests.get(url)
    with open(nome_arquivo, "wb") as f:
        f.write(response.content)

    df_spark = None

    # .tar.gz: extrai e lê CSV
    if nome_arquivo.endswith(".tar.gz"):
        with tarfile.open(nome_arquivo, "r:gz") as tar:
            membros_csv = [m for m in tar.getmembers() if m.name.endswith(".csv") and not m.name.startswith(".")]
            if len(membros_csv) != 1:
                raise ValueError(f"Esperado 1 CSV no tar.gz, encontrado: {[m.name for m in tar.getmembers()]}")
            membro = membros_csv[0]
            tar.extract(membro)
            caminho_csv = membro.name
            df_spark = spark.read.csv(caminho_csv, header=True, inferSchema=True)
            # os.remove(caminho_csv)

    # .json.gz
    elif nome_arquivo.endswith(".json.gz"):
        # Spark lê gzip direto
        df_spark = spark.read.json(nome_arquivo, multiline=False)

    # .csv.gz
    elif nome_arquivo.endswith(".csv.gz"):
        df_spark = spark.read.csv(nome_arquivo, header=True, inferSchema=True)

    else:
        raise ValueError("Formato de arquivo não suportado. Use .json.gz, .csv.gz ou .tar.gz")

    # Limpar arquivo original
    # os.remove(nome_arquivo)
```

```
return df_spark
```

Dataset Pedidos

Contém dados de cerca de 3.6 milhões de pedidos realizados entre dez/18 e jan/19. Cada pedido possui um order_id e os seguintes atributos complementares:

- cpf (string): Cadastro de Pessoa Física do usuário que realizou o pedido
- customer_id (string): Identificador do usuário
- customer_name (string): Primeiro nome do usuário
- delivery_address_city (string): Cidade de entrega do pedido
- delivery_address_country (string): País da entrega
- delivery_address_district (string): Bairro da entrega
- delivery_address_external_id (string): Identificador do endereço de entrega
- delivery_address_latitude (float): Latitude do endereço de entrega
- delivery_address_longitude (float): Longitude do endereço de entrega
- delivery_address_state (string): Estado da entrega
- delivery_address_zip_code (string): CEP da entrega
- items (array[json]): Itens que compõem o pedido, bem como informações complementares como preço unitário, quantidade, etc.
- merchant_id (string): Identificador do restaurante
- merchant_latitude (float): Latitude do restaurante
- merchant_longitude (float): Longitude do restaurante
- merchant_timezone (string): Fuso horário em que o restaurante está localizado
- order_created_at (timestamp): Data e hora em que o pedido foi criado
- order_id (string): Identificador do pedido
- order_scheduled (bool): Flag indicando se o pedido foi agendado ou não (pedidos agendados são aqueles que o usuário escolheu uma data e hora para a entrega)
- order_total_amount (float): Valor total do pedido em Reais
- origin_platform (string): Sistema operacional do dispositivo do usuário
- order_scheduled_date (timestamp): Data e horário para entrega do pedido agendado

```
url = "https://data-architect-test-source.s3-sa-east-1.amazonaws.com/order.json.gz"
```

```
df_order = load_raw_data(url, "order.json.gz")
```

```
df_order.show(5)
```

```

+-----+-----+-----+-----+-----+-----+-----+
|      cpf|      customer_id|customer_name|delivery_address_city|delivery_address_country|delivery_address_district|delivery_ad
+-----+-----+-----+-----+-----+-----+-----+
|80532101763|7ba88a68bb2a3504c...|GUSTAVO|FRANCA|BR|JARDIM ESPRAIADO|
|43352103961|078acecdcf7fa89d3...|MICHELLE|SANTOS|BR|CAMPO GRANDE|
|38650991217|0e38a3237b5946e8a...|VICTOR|GUARULHOS|BR|JARDIM ROSSI|
|63579726866|cab1a004b7206d079...|ANNIE|SAO PAULO|BR|PARQUE SAO JORGE|
|90617788806|aa7edf5b166b8c843...|DANIEL|VITORIA|BR|JARDIM CAMBURI|
+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows

```

```
df_order.count()
```

```
3670826
```

```
from pyspark.sql.functions import count
```

```
# Verificando se há pedidos duplicados
```

```

# Agrupar por order_id e faz uma contagem()
df_duplicados = df_order.groupBy("order_id") \
    .agg(count("*").alias("qtd")) \
    .filter("qtd > 1")

```

```
# Mostra duplicados (se houver)
```

```
df_duplicados.show()
```

```

+-----+-----+
|      order_id|qtd|
+-----+-----+
|00017ff9feba98f3b...| 2|
|000debad1353b56e9...| 2|
|0019c69a93442a129...| 2|
|001c71aabf4112ea5...| 2|
|003ab5d27d4993420...| 2|
|004d108aa37ce0477...| 2|

```

```
|0058deaa9eee5587a...| 2|
|005d5d6fdbb669114...| 2|
|006457f74148ce726...| 2|
|0074b105df89a0011...| 2|
|00772bcaafb99834b...| 2|
|00773a80b79265e72...| 2|
|0084580cc420dda94...| 2|
|0090d695cbaf20e31...| 2|
|009e50c9eaf7f9452...| 2|
|00a65df0cff849d61...| 2|
|00aa39c9edc0b97c4...| 2|
|00afc228bc3a908fc...| 2|
|00be74bed4ad2e83e...| 2|
|00c6bb3ea61073042...| 2|
+-----+-----+
only showing top 20 rows
```

```
df_registros_duplicados = df_order.join(df_duplicados, on="order_id", how="inner").orderBy("order_id")
df_registros_duplicados.show()
```

```

+-----+-----+-----+-----+-----+-----+-----+-----+
|order_id|cpf|customer_id|customer_name|delivery_address_city|delivery_address_country|delivery_address|
+-----+-----+-----+-----+-----+-----+-----+-----+
|00000b67fac8e2e29...|11617322723|4a5eac97fcc0d8073...|LÍV|SAO JOSE DOS CAMPOS|BR|JARDIM
|00000b67fac8e2e29...|09998377521|4a5eac97fcc0d8073...|LÍV|SAO JOSE DOS CAMPOS|BR|JARDIM
|00000fa3ee5165ced...|48407845939|c1c7126ad0d6ca68e...|ANA|BELO HORIZONTE|BR|NC
|00000fa3ee5165ced...|00939442255|c1c7126ad0d6ca68e...|ANA|BELO HORIZONTE|BR|NC
|0000226b0983a454e...|77024981760|daa25fa7b34ed201f...|THAMIRE|SAO JOSE|BR|FORC
|0000226b0983a454e...|68600842117|daa25fa7b34ed201f...|THAMIRE|SAO JOSE|BR|FORC
|00002c7ba1ce44fed...|86688139192|adea82755658a9e87...|VICTÓRIA|FLORIANOPOLIS|BR|
|00002c7ba1ce44fed...|00480737631|adea82755658a9e87...|VICTÓRIA|FLORIANOPOLIS|BR|
|0000438bc9201fcb3...|95279027565|609d8e3978b79e482...|EDUARDO|SAO PAULO|BR|
|0000438bc9201fcb3...|95390637688|609d8e3978b79e482...|EDUARDO|SAO PAULO|BR|
|000052bf31bab2f3a...|54697292576|875bdfb8f3f234124...|JOZY|SANTO ANDRE|BR|PARQUE
|000052bf31bab2f3a...|37035660807|875bdfb8f3f234124...|JOZY|SANTO ANDRE|BR|PARQUE
|000061ade15ffd6a3...|80648531408|8fc2b1806f100998a...|GERSON|FORTALEZA|BR|
|000061ade15ffd6a3...|50254232901|8fc2b1806f100998a...|GERSON|FORTALEZA|BR|
|00009005e585d987a...|48760995030|7f7deb42616e2ad98...|VANESSA|RIO DE JANEIRO|BR|VJ
|00009005e585d987a...|89296213926|7f7deb42616e2ad98...|VANESSA|RIO DE JANEIRO|BR|VJ
|0000921a2f658253c...|61499065521|ea078911686f1b1fd...|MARIANA|GOIANIA|BR|SE
|0000921a2f658253c...|56636607728|ea078911686f1b1fd...|MARIANA|GOIANIA|BR|SE
|0000b0ad44e908b19...|30171764226|7bbf1b2e6d8159422...|ANA|SAO PAULO|BR|(
|0000b0ad44e908b19...|61241497540|7bbf1b2e6d8159422...|ANA|SAO PAULO|BR|(
+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 20 rows
```

```

from pyspark.sql.window import Window
from pyspark.sql.functions import row_number, col

# Há registros de pedidos duplicados, com CPF diferente e data do pedido diferente
# Vamos remover a coluna cpf (não necessária para a análise) e manter o registro mais recente de acordo com 'order_created_at'

# Remover a coluna 'cpf'
df_sem_cpf = df_order.drop("cpf")

# Criar janela particionando por colunas duplicadas (todas menos 'order_created_at')
# Obtemos o nome de todas as colunas (sem cpf e sem order_created_at)
chaves_particionamento = [c for c in df_sem_cpf.columns if c != "order_created_at"]

# Definir janela para manter o mais recente por 'order_created_at'
janela = Window.partitionBy(chaves_particionamento).orderBy(col("order_created_at").desc())

# Adicionar row_number e filtrar apenas o primeiro (mais recente)
df_order_deduplicado = df_sem_cpf.withColumn("row_num", row_number().over(janela)) \
    .filter(col("row_num") == 1) \
    .drop("row_num")

# Agrupar por customer_id e contar
df_duplicados = df_order_deduplicado.groupBy("order_id") \
    .agg(count("*").alias("qtd")) \
    .filter("qtd > 1")

# Mostrar duplicados (se houver)
df_duplicados.show()
```

```

+-----+-----+
|order_id|qtd|
+-----+-----+
+-----+-----+
```

```
df_order_deduplicado.show(5)
```

```

+-----+-----+-----+-----+-----+-----+
|customer_id|customer_name|delivery_address_city|delivery_address_country|delivery_address_district|delivery_address_external_id|de|
+-----+-----+-----+-----+-----+-----+
|      NULL|      ADOLFO|      CURITIBA|      BR|      HUGO LANGE|      3662952|
|      NULL|      ADRIANO|      BRASILIA|      BR|      ASA NORTE|      8265397|
|      NULL|      ADRIANO|      BRASILIA|      BR|      ASA NORTE|      8869308|
|      NULL|      ALEXANDRE|      JOAO PESSOA|      BR|      TAMBAU|      5181095|
|      NULL|      ALEXANDRE|      SAO PAULO|      BR|      BELA VISTA|      2251620|
+-----+-----+-----+-----+-----+-----+
only showing top 5 rows

```

```

# Verificando se há valores nulos em customer_id
df_order_deduplicado.filter(col("customer_id").isNull()).count()

```

```
5559
```

```

# Excluindo os valores nulos de customer_id (não será possível relacioná-los com as outras tabelas)
df_order_deduplicado = df_order_deduplicado.filter(col("customer_id").isNotNull())
df_order_deduplicado.count()

```

```
2427415
```

```

# Dataset Original -> 3.670.826
# Dataset Deduplicado -> 2.427.415

```

Dataset Usuários

Contém dados de cerca de 806k usuários do iFood. Cada usuário possui um customer_id e os seguintes atributos complementares:

- customer_id (string): Identificador do usuário
- language (string): Idioma do usuário
- created_at (timestamp): Data e hora em que o usuário foi criado
- active (bool): Flag indicando se o usuário está ativo ou não
- customer_name (string): Primeiro nome do usuário
- customer_phone_area (string): Código de área do telefone do usuário
- customer_phone_number (string): Número do telefone do usuário

```

url = "https://data-architect-test-source.s3-sa-east-1.amazonaws.com/consumer.csv.gz"
df_consumer = load_raw_data(url, "consumer.csv.gz")
df_consumer.show(5)

```

```

+-----+-----+-----+-----+-----+-----+-----+
|      customer_id|language|      created_at|active|customer_name|customer_phone_area|customer_phone_number|
+-----+-----+-----+-----+-----+-----+-----+
|e8cc60860e09c0bb1...|pt-br|2018-04-05 14:49:...|true|      NUNO|      46|      816135924|
|a2834a38a9876cf74...|pt-br|2018-01-14 21:40:...|true|      ADRIELLY|      59|      231330577|
|41e1051728eba1334...|pt-br|2018-01-07 03:47:...|true|      PAULA|      62|      347597883|
|8e7c1dcb64edf95c9...|pt-br|2018-01-10 22:17:...|true|      HELTON|      13|      719366842|
|7823d4cf4150c5dae...|pt-br|2018-04-06 00:16:...|true|      WENDER|      76|      543232158|
+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows

```

```

# Verificando duplicidade em customer_id
df_duplicados = df_consumer.groupBy("customer_id") \
    .agg(count("*").alias("qtd")) \
    .filter("qtd > 1")

```

```

# Mostrar duplicados (se houver)
df_duplicados.show()

```

```

+-----+-----+
|customer_id|qtd|
+-----+-----+
+-----+-----+

```


```

# Verificando nulos em customer_id
df_consumer.filter(col("customer_id").isNull()).count()

```

```
0
```

```
df_consumer.count()
```

 806156

Dataset Restaurantes


Contém dados de cerca de 7k restaurantes do iFood. Cada restaurante possui um id e os seguintes atributos complementares:

- id (string): Identificador do restaurante
- created_at (timestamp): Data e hora em que o restaurante foi criado
- enabled (bool): Flag indicando se o restaurante está ativo no iFood ou não
- price_range (int): Classificação de preço do restaurante
- average_ticket (float): Ticket médio dos pedidos no restaurante
- delivery_time (float): Tempo padrão de entrega para pedidos no restaurante
- minimum_order_value (float): Valor mínimo para pedidos no restaurante
- merchant_zip_code (string): CEP do restaurante
- merchant_city (string): Cidade do restaurante
- merchant_state (string): Estado do restaurante
- merchant_country (string): País do restaurante

```
url = "https://data-architect-test-source.s3-sa-east-1.amazonaws.com/restaurant.csv.gz"
```

```
df_merchants = load_raw_data(url, "restaurant.csv.gz")
```

```
df_merchants.show(5)
```



| id | created_at | enabled | price_range | average_ticket | takeout_time | delivery_time | minimum_order_value | merchar |
|----------------------|----------------------|---------|-------------|----------------|--------------|---------------|---------------------|---------|
| d19ff6fca6288939b... | 2017-01-23 12:52:... | false | 3 | 60.0 | 0 | 50 | 30.0 | |
| 631df0985fdbbaf27... | 2017-01-20 13:14:... | true | 3 | 60.0 | 0 | 0 | 30.0 | |
| 135c5c4ae4c1ec1fd... | 2017-01-23 12:46:... | true | 5 | 100.0 | 0 | 45 | 10.0 | |
| d26f84c470451f752... | 2017-01-20 13:15:... | true | 3 | 80.0 | 0 | 0 | 18.9 | |
| 97b9884600ea71923... | 2017-01-20 13:14:... | true | 3 | 60.0 | 0 | 0 | 25.0 | |


only showing top 5 rows

```
# Verificando duplicidade em id
```

```
df_duplicados = df_merchants.groupBy("id") \
    .agg(count("").alias("qtd")) \
    .filter("qtd > 1")
```

```
# Mostrar duplicados (se houver)
```

```
df_duplicados.show()
```



```

+---+---+
| id|qtd|
+---+---+
+---+---+

```

```
# Verificando nulos em id
```

```
df_merchants.filter(col("id").isNull()).count()
```

 0

Marcação dos usuários do Teste A/B


Contém uma marcação indicando se um usuário participou do teste A/B em questão. Assim como a base de usuários, cada usuário possui um customer_id. Os campos são:

- customer_id (string): Identificador do usuário
- is_target (string): Grupo ao qual o usuário pertence ('target' ou 'control').

```
url = "https://data-architect-test-source.s3-sa-east-1.amazonaws.com/ab_test_ref.tar.gz"
```

```
df_ab_test = load_raw_data(url, "ab_test_ref.tar.gz")
```

```
df_ab_test.show(5)
```



| customer_id | is_target |
|----------------------|-----------|
| 755e1fa18f25caec5... | target |
| b821aa8372b8e5b82... | control |
| d425d6ee4c9d4e211... | control |
| 6a7089eea0a5dc294... | target |
| dad6b7e222bab31c0... | control |

only showing top 5 rows

```
# Verificando duplicidade em customer_id
df_duplicados = df_ab_test.groupby("customer_id") \
    .agg(count("*").alias("qtd")) \
    .filter("qtd > 1")
```

```
# Mostrar duplicados (se houver)
df_duplicados.show()
```

```
↗ +-----+----+
  |customer_id|qtd|
  +-----+----+
  +-----+----+
```

```
# Verificando nulos em customer_id
df_ab_test.filter(col("customer_id").isNull()).count()
```

```
↗ 0
```

```
df_ab_test.count()
```

```
↗ 806467
```

✓ Unindo as Tabelas

```
# Como vamos analisar apenas o público que participou do teste A/B, podemos unir por inner join as tabelas df_ab_test df_consumer
df_consumer_ab_test = df_consumer.join(df_ab_test, on="customer_id", how="inner")
df_consumer_ab_test.show()
```

```
↗ +-----+-----+-----+-----+-----+-----+-----+-----+
  |customer_id|language|created_at|active|customer_name|customer_phone_area|customer_phone_number|is_target|
  +-----+-----+-----+-----+-----+-----+-----+-----+
  |000021924bf8192f6...|pt-br|2018-01-03 14:12:...|true|THIAGO|64|156381073|target|
  |00006f567cb362ba9...|pt-br|2018-04-06 03:20:...|true|AMANDA|72|980221683|target|
  |0000bb10fb47a1d6b...|pt-br|2018-01-04 22:01:...|true|JULIANA|64|235383327|control|
  |0000c21984ae00cef...|pt-br|2018-01-07 14:36:...|true|MARLOS|72|831139121|control|
  |0001226e517517758...|pt-br|2018-03-31 23:13:...|true|JULLYA|24|960187601|target|
  |0001274ea3bc24cee...|pt-br|2018-04-06 04:00:...|true|RITA|55|77599540|target|
  |00016cfd8c0af0a4...|pt-br|2018-04-06 04:00:...|true|MATHIAS|21|250909452|control|
  |000200d3759a5b4d0...|pt-br|2018-01-08 01:10:...|true|JOHANN|34|787427470|target|
  |00020951c8f263d74...|pt-br|2018-04-05 13:17:...|true|GABRIEL|87|662503035|target|
  |00021cd56b6d6c980...|pt-br|2018-01-04 15:16:...|true|SILVIA|87|781841665|target|
  |00021f6dc15d10418...|pt-br|2018-01-30 21:16:...|true|MARIANA|55|79649823|target|
  |0002287b123ac1afc...|pt-br|2018-02-04 23:07:...|true|CELSON|68|359555807|target|
  |00022b8c0c7af061f...|pt-br|2018-04-06 02:04:...|true|PRISCILA|83|798128500|control|
  |00024bc2f09ce5769...|pt-br|2018-01-14 21:30:...|true|Samantha|16|299190582|control|
  |00027035d16a4de43...|pt-br|2018-04-06 03:40:...|true|RAÚL|72|206142854|control|
  |000299d1aee7451f3...|pt-br|2018-03-20 18:36:...|true|PABLO|28|674243336|control|
  |00029b26fb2121119...|pt-br|2018-01-03 23:03:...|true|SIMONE|43|60750759|target|
  |0002cc7394d677fdf...|pt-br|2018-04-06 04:24:...|true|FRANCISCO|91|97850192|control|
  |0002d068e36949a0d...|pt-br|2018-01-09 21:28:...|true|GABRIELA|96|807574551|target|
  |0003030c4d06fe6a0...|pt-br|2018-01-06 08:49:...|true|LINDOMAR|70|793363614|control|
  +-----+-----+-----+-----+-----+-----+-----+-----+
```

only showing top 20 rows

```
# Agora vamos unir estas informações na tabela de pedidos, novamente queremos apenas os pedidos dos usuários que participaram do teste A/B
df_order_ab_test = df_order_deduplicado.join(df_consumer_ab_test, on="customer_id", how="inner")
df_order_ab_test.show()
```

```
↗ +-----+-----+-----+-----+-----+-----+
  |customer_id|customer_name|delivery_address_city|delivery_address_country|delivery_address_district|delivery_address_exterr|
  +-----+-----+-----+-----+-----+-----+
  |00021cd56b6d6c980...|SILVIA|CAMPINAS|BR|MANSOES SANTO ANT...|57|
  |00021cd56b6d6c980...|SILVIA|CAMPINAS|BR|MANSOES SANTO ANT...|57|
  |00021cd56b6d6c980...|SILVIA|CAMPINAS|BR|MANSOES SANTO ANT...|57|
  |0002cc7394d677fdf...|FRANCISCO|BRASILIA|BR|SETOR HABITACIONA...|96|
  |0006aba7fd94d9de3...|ANA|SAO JOSE|BR|CAMPINAS|91|
  |0006aba7fd94d9de3...|ANA|SAO JOSE|BR|CAMPINAS|91|
  |0009fce56b3f5a32a...|TANIA|SAO PAULO|BR|VILA IPOJUCA|96|
  |000ac4b0d62aaf489...|STEFANI|SAO PAULO|BR|VILA OLIMPIA|35|
  |001566671db5d822e...|SANDRA|RIO DE JANEIRO|BR|CENTRO|87|
  |001566671db5d822e...|SANDRA|RIO DE JANEIRO|BR|CENTRO|22|
  |0015f1e5b35a3d7b6...|THAYNÁ|RIO DE JANEIRO|BR|BANCARIOS|85|
  |00173c4d39960ee57...|LILIAN|LONDRINA|BR|LEONOR|77|
  |00173c4d39960ee57...|LILIAN|LONDRINA|BR|LEONOR|77|
  |00173c4d39960ee57...|LILIAN|LONDRINA|BR|LEONOR|77|
  |00173c4d39960ee57...|LILIAN|LONDRINA|BR|LEONOR|77|
  |00173c4d39960ee57...|LILIAN|LONDRINA|BR|LEONOR|77|
```

| | | | | | |
|----------------------|--------|----------|----|--------|----|
| 00173c4d39960ee57... | LILIAN | LONDRINA | BR | LEONOR | 77 |
| 00173c4d39960ee57... | LILIAN | LONDRINA | BR | LEONOR | 77 |
| 00173c4d39960ee57... | LILIAN | LONDRINA | BR | LEONOR | 77 |
| 00173c4d39960ee57... | LILIAN | LONDRINA | BR | LEONOR | 77 |

only showing top 20 rows

df_order_ab_test.count()

↗ 2426590

Por fim, vamos agregar as informações dos restaurantes na tabela de pedidos (aqui podemos unir com left join):

```
df_order_ab_test_merchant = df_order_ab_test.join(
    df_merchants.withColumnRenamed("created_at", "created_at_merchant"),
    df_order_ab_test.merchant_id == df_merchants.id,
    how="left"
)
```

df_order_ab_test_merchant.show()

| customer_id | customer_name | delivery_address_city | delivery_address_country | delivery_address_district | delivery_address_exterr |
|----------------------|---------------|-----------------------|--------------------------|---------------------------|-------------------------|
| 00021cd56b6d6c980... | SILVIA | CAMPINAS | BR | MANSOES SANTO ANT... | 57 |
| 00021cd56b6d6c980... | SILVIA | CAMPINAS | BR | MANSOES SANTO ANT... | 57 |
| 00021cd56b6d6c980... | SILVIA | CAMPINAS | BR | MANSOES SANTO ANT... | 57 |
| 0002c7394d677fdf... | FRANCISCO | BRASILIA | BR | SETOR HABITACIONA... | 96 |
| 0006aba7fd94d9de3... | ANA | SAO JOSE | BR | CAMPINAS | 91 |
| 0006aba7fd94d9de3... | ANA | SAO JOSE | BR | CAMPINAS | 91 |
| 0009fce56b3f5a32a... | TANIA | SAO PAULO | BR | VILA IPOJUCA | 96 |
| 000ac4b0d62aaf489... | STEFANI | SAO PAULO | BR | VILA OLIMPIA | 35 |
| 001566671db5d822e... | SANDRA | RIO DE JANEIRO | BR | CENTRO | 87 |
| 001566671db5d822e... | SANDRA | RIO DE JANEIRO | BR | CENTRO | 22 |
| 0015f1e5b35a3d7b6... | THAYNÁ | RIO DE JANEIRO | BR | BANCARIOS | 85 |
| 00173c4d39960ee57... | LILIAN | LONDRINA | BR | LEONOR | 77 |
| 00173c4d39960ee57... | LILIAN | LONDRINA | BR | LEONOR | 77 |
| 00173c4d39960ee57... | LILIAN | LONDRINA | BR | LEONOR | 77 |
| 00173c4d39960ee57... | LILIAN | LONDRINA | BR | LEONOR | 77 |
| 00173c4d39960ee57... | LILIAN | LONDRINA | BR | LEONOR | 77 |
| 00173c4d39960ee57... | LILIAN | LONDRINA | BR | LEONOR | 77 |
| 00173c4d39960ee57... | LILIAN | LONDRINA | BR | LEONOR | 77 |
| 00173c4d39960ee57... | LILIAN | LONDRINA | BR | LEONOR | 77 |
| 00173c4d39960ee57... | LILIAN | LONDRINA | BR | LEONOR | 77 |

only showing top 20 rows

```
for col_name in df_order_ab_test_merchant.columns:
    print(col_name)
```

```
↗ customer_id
customer_name
delivery_address_city
delivery_address_country
delivery_address_district
delivery_address_external_id
delivery_address_latitude
delivery_address_longitude
delivery_address_state
delivery_address_zip_code
items
merchant_id
merchant_latitude
merchant_longitude
merchant_timezone
order_created_at
order_id
order_scheduled
order_scheduled_date
order_total_amount
origin_platform
language
created_at
active
customer_name
customer_phone_area
customer_phone_number
is_target
id
created_at_merchant
enabled
price_range
average_ticket
```

```

takeout_time
delivery_time
minimum_order_value
merchant_zip_code
merchant_city
merchant_state
merchant_country

```

Vamos remover algumas colunas que não serão necessárias no contexto do teste A/B

```

colunas_para_remover = [
    "customer_name",
    "delivery_address_district",
    "delivery_address_external_id",
    "delivery_address_latitude",
    "delivery_address_longitude",
    "delivery_address_zip_code",
    "merchant_latitude",
    "merchant_longitude",
    "merchant_timezone",
    "customer_phone_area",
    "customer_phone_number",
    "id",
    "merchant_zip_code"
]

```

```

df_order_ab_test_merchant_clear_columns = df_order_ab_test_merchant.drop(*colunas_para_remover)
df_order_ab_test_merchant_clear_columns.show()

```

```

+-----+-----+-----+-----+-----+
| customer_id | delivery_address_city | delivery_address_country | delivery_address_state | items | merchant_id |
+-----+-----+-----+-----+-----+
| 00021cd56b6d6c980... | CAMPINAS | BR | SP | [{"name": "Execut... | b6e311babf9a86139. |
| 00021cd56b6d6c980... | CAMPINAS | BR | SP | [{"name": "Trio N... | 9a4310e54725f9e9b. |
| 00021cd56b6d6c980... | CAMPINAS | BR | SP | [{"name": "Abelha... | ba8b6425d0a9d3a88. |
| 0002cc7394d677fdf... | BRASILIA | BR | DF | [{"name": "Arroz ... | 49e6eea57aef6679f. |
| 000405bb6de6550fe... | RIO DE JANEIRO | BR | RJ | [{"name": "GRANDE... | 85d05d0d6c5a68a71. |
| 000405bb6de6550fe... | RIO DE JANEIRO | BR | RJ | [{"name": "PromHo... | c5750da4194fe1ee8. |
| 000405bb6de6550fe... | RIO DE JANEIRO | BR | RJ | [{"name": "N15 CH... | 7f729a71737503560. |
| 0006aba7fd94d9de3... | SAO JOSE | BR | SC | [{"name": "Filé M... | dcb6dcdcf1629fffab. |
| 0006aba7fd94d9de3... | SAO JOSE | BR | SC | [{"name": "Delíci... | 0d45d38236c105039. |
| 0009fce56b3f5a32a... | SAO PAULO | BR | SP | [{"name": "GALETO... | e9808433ad0812980. |
| 000ac4b0d62aaf489... | SAO PAULO | BR | SP | [{"name": "PORÇÃO... | 06f331f426e4a05a9. |
| 0010c10673b278b9f... | DIADEMA | BR | SP | [{"name": "MÉDIA ... | be134facabdcdbf2c. |
| 0010c10673b278b9f... | SAO PAULO | BR | SP | [{"name": "REFRIG... | 1d050b212a12b4caa. |
| 0010c10673b278b9f... | DIADEMA | BR | SP | [{"name": "BROTO"... | 4130cf4500b2c1d74. |
| 001566671db5d822e... | RIO DE JANEIRO | BR | RJ | [{"name": "HOT SP... | 55cca4d45b19e1799. |
| 001566671db5d822e... | RIO DE JANEIRO | BR | RJ | [{"name": "Salada... | 2624028d9c00e80ad. |
| 0015f1e5b35a3d7b6... | RIO DE JANEIRO | BR | RJ | [{"name": "GIGANT... | 059fa9f3ec70af529. |
| 00173c4d39960ee57... | LONDRINA | BR | PR | [{"name": "FRANGO... | c82bf24729962df79. |
| 00173c4d39960ee57... | LONDRINA | BR | PR | [{"name": "Dog Si... | beef912cfbf0fcb68. |
| 00173c4d39960ee57... | LONDRINA | BR | PR | [{"name": "Dog Fr... | beef912cfbf0fcb68. |
+-----+-----+-----+-----+-----+
only showing top 20 rows

```

```

# Salvando o dataframe df_order_ab_test_merchant_clear_columns em disco para evitar de rodar sempre a parte do ETL
df_order_ab_test_merchant_clear_columns.write \
    .mode("overwrite") \
    .option("compression", "gzip") \
    .parquet("df_order_ab_test_merchant_clear_columns")
shutil.make_archive("df_order_ab_test_merchant_clear_columns", 'zip', "df_order_ab_test_merchant_clear_columns")
# df_order_ab_test_merchant_clear_columns.columns

```

```

!ls /content/df_order_ab_test_merchant_clear_columns.zip

```

Análise de dados

Colunas do Dataset Final

- customer_id Identificador do usuário,
- delivery_address_city Cidade de entrega do pedido,
- delivery_address_country País da entrega,
- delivery_address_state Estado da entrega,
- items Itens que compõem o pedido, bem como informações complementares como preço unitário, quantidade, etc.,
- merchant_id Identificador do restaurante,
- order_created_at Data e hora em que o pedido foi criado,
- order_id Identificador do pedido,

- order_scheduled Flag indicando se o pedido foi agendado ou não (pedidos agendados são aqueles que o usuário escolheu uma data e hora para a entrega),
- order_scheduled_date Data e horário para entrega do
- pedido agendado,
- order_total_amount Valor total do pedido em Reais,
- origin_platform Sistema operacional do dispositivo do usuário,
- language Idioma do usuário,
- created_at Data e hora em que o usuário foi criado,
- active Flag indicando se o usuário está ativo ou não,
- is_target Grupo ao qual o usuário pertence ('target' ou 'control'),
- created_at_merchant Data e hora em que o restaurante foi criado,
- enabled Flag indicando se o restaurante está ativo no iFood ou não,
- price_range Classificação de preço do restaurante,
- average_ticket Ticket médio dos pedidos no restaurante,
- takeout_time é o timestamp em que o pedido foi embalado e ficou disponível para o entregador buscar. (obs. este campo não foi informado no case),
- delivery_time Tempo padrão de entrega para pedidos no restaurante,
- minimum_order_value Valor mínimo para pedidos no restaurante,
- merchant_city Cidade do restaurante,
- merchant_state Estado do restaurante,
- merchant_country País do restaurante

```
import zipfile
import os
```

```
# Ao executar pela primeira vez, é obrigatório a execução das células acima
# Aqui como opcional, pode-se carregar o ETL na máquina do colab, caso contrário será necessário executar novamente a parte do ETL (to
```

```
try:
    df_order_ab_test_merchant_clear_columns.show(5)

except:
    with zipfile.ZipFile('df_order_ab_test_merchant_clear_columns.zip', 'r') as zip_ref:
        zip_ref.extractall('df_order_ab_test_merchant_clear_columns')
    df_order_ab_test_merchant_clear_columns = spark.read.parquet('df_order_ab_test_merchant_clear_columns')
```

```
from pyspark.sql.functions import avg
```

```
# Valor médio dos pedidos por grupo.
df_ticket_medio = df_order_ab_test_merchant_clear_columns.groupBy("is_target") \
    .agg(avg("order_total_amount").alias("ticket_medio"))
```

```
df_ticket_medio.show()
```

```
↗ +-----+-----+
  |is_target|    ticket_medio|
  +-----+-----+
  |  control|47.917294188812164|
  |   target|47.809652936992656|
  +-----+-----+
```

```
# Volume médio de pedidos por usuário
df_pedidos_por_usuario = df_order_ab_test_merchant_clear_columns.groupBy("is_target", "customer_id") \
    .count() \
    .groupBy("is_target") \
    .agg(avg("count").alias("pedidos_medio_por_usuario"))
```

```
df_pedidos_por_usuario.show()
```

```
↗ +-----+-----+
  |is_target|pedidos_medio_por_usuario|
  +-----+-----+
  |  control|      2.803428289212653|
  |   target|      3.177162625100114|
  +-----+-----+
```

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

```
# % de usuários que agendam pedidos
df_agendamento = df_order_ab_test_merchant_clear_columns.groupBy("is_target") \
    .agg(mean(col("order_scheduled").cast("int")).alias("pct_agendado"))
```

```
df_agendamento.show()
```

```

+-----+-----+
|is_target|    pct_agendado|
+-----+-----+
|  control|1.613235259186533...|
|   target|1.299253352991601...|
+-----+-----+
```

```
from pyspark.sql.functions import countDistinct
```

```
# % de usuários que voltaram a fazer pedidos (múltiplos pedidos em datas diferentes)
df_reativacao = df_order_ab_test_merchant_clear_columns.groupBy("is_target", "customer_id") \
    .agg(countDistinct("order_created_at").alias("num_dias_com_pedido"))
```

```
# Considerando reativado quem fez pedido em mais de 1 dia
df_reativados = df_reativacao.withColumn("reativado", (col("num_dias_com_pedido") > 1).cast("int"))
```

```
df_reativados.groupBy("is_target") \
    .agg(avg("reativado").alias("pct_reincidentes")) \
    .show()
```

```

+-----+-----+
|is_target|  pct_reincidentes|
+-----+-----+
|  control|0.4762591804402172|
|   target| 0.576159356400437|
+-----+-----+
```

```
df_amostra = df_order_ab_test_merchant_clear_columns.select("is_target", "order_total_amount") \
    .filter(col("order_total_amount").isNotNull()) \
    .sample(False, 0.1, seed=42) # amostra de 10% (ajuste conforme o tamanho, o PySpark não possui módulos para calcular o teste t)
```

```
# Converte para pandas
df_pd = df_amostra.toPandas()
```

```
grupo_target = df_pd[df_pd["is_target"] == "target"]["order_total_amount"]
grupo_control = df_pd[df_pd["is_target"] == "control"]["order_total_amount"]
```

```
from scipy.stats import ttest_ind
```

```
t_stat, p_val = ttest_ind(grupo_target, grupo_control, equal_var=False)
```

```
print(f"T-statistic: {t_stat:.4f}")
print(f"P-value: {p_val:.4f}")
```

```
if p_val < 0.05:
    print("Diferença significativa no ticket médio entre os grupos.")
else:
    print("Não há diferença estatisticamente significativa no ticket médio.")
```

```

T-statistic: 0.4233
P-value: 0.6721
Não há diferença estatisticamente significativa no ticket médio.
```

```
# # Teste t para pedidos reativados
```

```
# from pyspark.sql.functions import countDistinct, col
```

```
# df_reativacao = df_order_ab_test_merchant_clear_columns.groupBy("is_target", "customer_id") \
#     .agg(countDistinct("order_created_at").alias("num_dias_com_pedido"))
```

```
# df_reativacao = df_reativacao.withColumn(
#     "reativado", (col("num_dias_com_pedido") > 1).cast("int")
# )
```

```
# df_reativacao_pd = df_reativacao.select("is_target", "reativado").toPandas()
```

```
# reat_target = df_reativacao_pd[df_reativacao_pd["is_target"] == "target"]["reativado"]
# reat_control = df_reativacao_pd[df_reativacao_pd["is_target"] == "control"]["reativado"]
```

```
# from scipy.stats import ttest_ind

# t_stat, p_val = ttest_ind(reat_target, reat_control, equal_var=False)

# print(f"T-statistic: {t_stat:.4f}")
# print(f"P-value: {p_val:.4f}")

# if p_val < 0.05:
#     print("Diferença significativa na reativação entre os grupos.")
# else:
#     print("Não há diferença estatisticamente significativa na reativação.")

# Teste z para pedidos reativados (variável binária)

from pyspark.sql.functions import countDistinct, col

df_reativacao = df_order_ab_test_merchant_clear_columns.groupBy("is_target", "customer_id") \
    .agg(countDistinct("order_created_at").alias("num_dias_com_pedido"))

df_reativacao = df_reativacao.withColumn(
    "reativado", (col("num_dias_com_pedido") > 1).cast("int")
)

df_reativacao_pd = df_reativacao.select("is_target", "reativado").toPandas()

# Contando o total e os reativados em cada grupo
controles = df_reativacao_pd[df_reativacao_pd["is_target"] == "control"]
targets = df_reativacao_pd[df_reativacao_pd["is_target"] == "target"]

n_control = len(controles)
x_control = controles["reativado"].sum()

n_target = len(targets)
x_target = targets["reativado"].sum()

from statsmodels.stats.proportion import proportions_ztest

# Dados para o teste
counts = [x_target, x_control]      # sucessos (reativações)
nobs = [n_target, n_control]       # total de usuários

# Teste bicaudal
z_stat, p_val = proportions_ztest(count=counts, nobs=nobs)

print(f"Z-statistic: {z_stat:.4f}")
print(f"P-value: {p_val:.4f}")

if p_val < 0.05:
    print("Diferença significativa na proporção de usuários reativados entre os grupos.")
else:
    print("Não há diferença estatisticamente significativa na proporção de reativação.")
```

```
↩ Z-statistic: 89.3702
P-value: 0.0000
Diferença significativa na proporção de usuários reativados entre os grupos.
```

```
from pyspark.sql.functions import countDistinct

# Contagem de usuários em cada grupo
df_order_ab_test_merchant_clear_columns.groupBy("is_target") \
    .agg(countDistinct("customer_id").alias("num_usuarios_unicos")) \
    .show()
```

```
↩ +-----+-----+
|is_target|num_usuarios_unicos|
+-----+-----+
| control|          360413|
| target|          445743|
+-----+-----+
```

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

# Contando o número de dias com pedido por usuário
df_repeticao = df_order_ab_test_merchant_clear_columns.groupBy("customer_id") \
    .agg(countDistinct("order_created_at").alias("dias_com_pedido"))
```

```
# Identificando usuários reativados (mais de 1 dia com pedido)
df_reativados = df_repeticao.filter(col("dias_com_pedido") > 1)

# Juntando com o DataFrame original para filtrar apenas os pedidos desses usuários
df_pedidos_reativados = df_order_ab_test_merchant_clear_columns.join(
    df_reativados.select("customer_id"), on="customer_id", how="inner"
)

# Contando pedidos por usuário reativado
df_media_pedidos = df_pedidos_reativados.groupBy("customer_id") \
    .count() \
    .agg({"count": "avg"}) \
    .withColumnRenamed("avg(count)", "media_pedidos_por_reativado")

df_media_pedidos.show()
```

```
↩ +-----+
  |media_pedidos_por_reativado|
  +-----+
  |          4.781916544720855|
  +-----+
```

```
from pyspark.sql.functions import avg, countDistinct, col

# Identificar usuários com pedidos em mais de 1 dia
df_dias = df_order_ab_test_merchant_clear_columns.groupBy("customer_id") \
    .agg(countDistinct("order_created_at").alias("dias_com_pedido"))

df_reativados = df_dias.filter(col("dias_com_pedido") > 1)

# Filtrar os pedidos dos usuários reativados
df_pedidos_reativados = df_order_ab_test_merchant_clear_columns.join(
    df_reativados.select("customer_id"), on="customer_id", how="inner"
)

# Calcular ticket médio
df_ticket_medio_reativados = df_pedidos_reativados.agg(
    avg("order_total_amount").alias("ticket_medio_reativados")
)

df_ticket_medio_reativados.show()
```

```
↩ +-----+
  |ticket_medio_reativados|
  +-----+
  |          47.86354821570643|
  +-----+
```

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

# Calculando ticket médio por plataforma e grupo (target ou control)
df_order_ab_test_merchant_clear_columns.groupBy("is_target", "origin_platform") \
    .agg(avg(col("order_total_amount")).alias("ticket_medio")) \
    .orderBy("is_target", "origin_platform") \
    .show()
```

```
↩ +-----+-----+-----+
  |is_target|origin_platform|    ticket_medio|
  +-----+-----+-----+
  | control|      NULL|          50.0|
  | control|    ANDROID| 43.44532666705258|
  | control|    DESKTOP| 46.30479106031281|
  | control|      IOS| 52.661987820068575|
  | control| WINDOWS_PHONE| 39.59971987051812|
  | target|    ANDROID| 43.02639192576987|
  | target|    DESKTOP| 46.36331193965382|
  | target|      IOS| 52.76396343255716|
  | target| WINDOWS_PHONE| 40.365826558265674|
  +-----+-----+-----+
```

```
from pyspark.sql.functions import col
from scipy.stats import ttest_ind

# Filtrando para plataforma WINDOWS_PHONE
df_windows = df_order_ab_test_merchant_clear_columns.filter(
    col("origin_platform") == "WINDOWS_PHONE"
).select("is_target", "order_total_amount")
```

```
# Convertendo para pandas
df_windows_pd = df_windows.toPandas()

# Separar os grupos
grupo_target = df_windows_pd[df_windows_pd["is_target"] == "target"]["order_total_amount"]
grupo_control = df_windows_pd[df_windows_pd["is_target"] == "control"]["order_total_amount"]

# Aplicar o teste t
t_stat, p_val = ttest_ind(grupo_target, grupo_control, equal_var=False)

print(f"T-statistic: {t_stat:.4f}")
print(f"P-value: {p_val:.4f}")

if p_val < 0.05:
    print("Diferença significativa no ticket médio entre os grupos para WINDOWS_PHONE.")
else:
    print("Não há diferença estatisticamente significativa no ticket médio para WINDOWS_PHONE.")
```

```
T-statistic: 2.1397
P-value: 0.0324
Diferença significativa no ticket médio entre os grupos para WINDOWS_PHONE.
```

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

# Obtendo a lista de estados únicos
estados = [row["delivery_address_state"] for row in df_order_ab_test_merchant_clear_columns.select("delivery_address_state").distinct()]

# Loop pelos estados e calcular o ticket médio por grupo (is_target)
for estado in estados:
    print(f"\n Estado: {estado}")

    df_estado = df_order_ab_test_merchant_clear_columns.filter(
        col("delivery_address_state") == estado
    )

    df_resultado = df_estado.groupBy("is_target") \
        .agg(avg("order_total_amount").alias("ticket_medio"))

    df_resultado.show()
```



```

|is_target|      ticket_medio|
+-----+-----+
| control|44.54602363543513|
| target|44.74075935957191|
+-----+-----+

```

📌 Estado: PR

```

+-----+-----+
|is_target|      ticket_medio|
+-----+-----+
| control|41.44672616231567|
| target|41.33195857700692|
+-----+-----+

```

```

from pyspark.sql.functions import col
from scipy.stats import ttest_ind

```

```

# 1. Filtrar para plataforma WINDOWS_PHONE
df_windows = df_order_ab_test_merchant_clear_columns.filter(
    col("delivery_address_state") == "AC"
).select("is_target", "order_total_amount")

```

```

# 2. Converter para pandas
df_windows_pd = df_windows.toPandas()

```

```

# 3. Separar os grupos
grupo_target = df_windows_pd[df_windows_pd["is_target"] == "target"]["order_total_amount"]
grupo_control = df_windows_pd[df_windows_pd["is_target"] == "control"]["order_total_amount"]

```

```

# 4. Aplicar o teste t
t_stat, p_val = ttest_ind(grupo_target, grupo_control, equal_var=False)

```

```

print(f"T-statistic: {t_stat:.4f}")
print(f"P-value: {p_val:.4f}")

```

```

if p_val < 0.05:
    print("Diferença significativa no ticket médio entre os grupos para Acre.")
else:
    print("Não há diferença estatisticamente significativa no ticket médio para Acre.")

```

➡ T-statistic: nan
P-value: nan
Não há diferença estatisticamente significativa no ticket médio para Acre.

```

from pyspark.sql.functions import col, datediff, floor

```

```

# Calcular a diferença em dias e converter para anos
df_com_anos = df_order_ab_test_merchant_clear_columns.withColumn(
    "anos_desde_cadastro",
    floor(datediff(col("order_created_at"), col("created_at")) / 365)
)

```

```

# Agrupar por is_target e anos desde o cadastro
df_ticket_medio_por_ano = df_com_anos.groupBy("is_target", "anos_desde_cadastro") \
    .agg(avg("order_total_amount").alias("ticket_medio")) \
    .orderBy("anos_desde_cadastro", "is_target")

```

```

# Exibindo resultado
df_ticket_medio_por_ano.show(truncate=False)

```

➡

```

+-----+-----+-----+
|is_target|anos_desde_cadastro|ticket_medio|
+-----+-----+-----+
|control  |0                  |46.914980084119684|
|target   |0                  |46.73190121983342|
|control  |1                  |49.05659743147708|
|target   |1                  |48.979091809876365|
+-----+-----+-----+

```

Comece a programar ou [gere código](#) com IA.