**Minerva Foods**

Documentação Técnica - Automação de Notebooks  
  
Documento de Implementação Técnica

# Objetivo do documento

Este documento tem como objetivo apresentar e documentar, de maneira técnica e padronizada, os scripts desenvolvidos em notebooks do projeto. A estrutura segue o modelo de documentação DIT, adotado pela Minerva Foods, garantindo organização, clareza e reprodutibilidade.

# Dim Empresa Ficticia Anonimizado Completo

## Tabela: refined.d\_empresas\_farol  
  
### Objetivo:  
Tabela da dimensão de empresas do indicador farol.  
  
### Fontes de Dados  
  
| Origem | Descrição |  
|------------------------------------|--------------------------------------------|  
|trusted.tb\_tab\_empresa\_farol | Tabela de empresas. |  
|raw.sharepoint\_dados\_filiais\_farol | Tabela do sharepoint com dados de filiais. |  
|refined.tb\_farol\_faturamento\_farol | Tabela com dados de faturamento. |  
  
  
### Histórico de alterações  
  
| Data | Desenvolvido por | Modificações |  
|------------|------------------|-----------------------|  
| 22/05/2025 | Michel Santana | Criação do notebook |

## Resumo do Código

O trecho de código em questão é responsável por importar e executar o notebook denominado `ingestion\_function`, que está localizado no diretório `../00\_config/`. Este processo é realizado através do comando `%run`, que tem a função de carregar todas as funções, variáveis e configurações definidas no notebook referenciado para o ambiente atual. A principal vantagem deste procedimento é a possibilidade de reutilizar lógica comum, como funções de ingestão de dados, sem a necessidade de duplicar o código. Isso contribui para a centralização de rotinas reutilizáveis e para a manutenção de notebooks modulares e organizados, otimizando o fluxo de trabalho e a eficiência do código.

## Código Fonte

# Importa e executa o notebook `ingestion\_function`, localizado em `../00\_config/`.  
#   
# O comando `%run` carrega todas as funções, variáveis e configurações definidas no notebook referenciado  
# para o ambiente atual. Isso permite reutilizar lógica comum, como funções de ingestão de dados, sem duplicação de código.  
#   
# Útil para centralizar rotinas reutilizáveis e manter notebooks modulares e organizados.

## Resumo do Código

O trecho de código em questão é responsável por executar um script Python armazenado em um diretório de configuração. Especificamente, o comando '%run' é uma diretiva do Jupyter Notebook que permite a execução de um arquivo Python externo como parte do notebook. O arquivo em questão, 'ingestion\_function', está localizado no diretório '00\_config'. Embora o conteúdo exato do script 'ingestion\_function' não seja fornecido, o nome sugere que ele contém funções relacionadas à ingestão de dados, possivelmente incluindo a leitura de dados de uma fonte externa, a realização de qualquer limpeza ou transformação de dados necessária e a preparação dos dados para análise posterior. A execução deste script como parte do notebook "Dim Empresa Ficticia Anonimizado Completo" provavelmente serve para garantir que os dados necessários para a análise estejam disponíveis e no formato correto.

## Código Fonte

%run ../00\_config/ingestion\_function

## Resumo do Código

O trecho de código em questão é responsável por definir e exibir informações sobre uma tabela de dimensão de empresas fictícias em um ambiente de Data Lake. Inicialmente, a variável 'debug' é definida como False, o que indica que o código está em um estado de produção e não em um estado de depuração. Em seguida, são definidas várias variáveis que especificam o local e o nome da tabela de destino no Data Lake. A variável 'container\_target' representa o contêiner de destino no Data Lake, 'directory' é o diretório fictício onde a tabela será armazenada, e 'table\_name' é o nome da tabela. 'delta\_table\_name' é o nome completo da tabela, incluindo o ambiente, o contêiner de destino e o nome da tabela. 'delta\_file' é o caminho completo do arquivo no Data Lake, incluindo o contêiner de destino, o nome do Data Lake fictício, o diretório e o nome da tabela. O comentário 'comment\_delta\_table' fornece uma descrição da tabela. Finalmente, o código imprime o nome da tabela e o caminho do arquivo para fins de verificação.

## Código Fonte

debug = False  
  
container\_target = 'camada\_destino'  
directory = 'diretorio\_ficticio'  
table\_name = 'dim\_empresa\_ficticia'  
delta\_table\_name = f'{environment}.{container\_target}.{table\_name}'  
delta\_file = f"abfss://{container\_target}@{nome\_datalake\_ficticio}.dfs.core.windows.net/{directory}/{table\_name}/"  
comment\_delta\_table = 'Tabela de dimensão de empresas fictícias.'  
  
print(f'delta\_table\_name = {table\_name}')  
print(f'delta\_file = {delta\_file}')

## Resumo do Código

O trecho de código em questão é responsável por criar um widget interativo denominado `reprocessar` que permite ao usuário controlar a carga da dimensão em um notebook. Este widget apresenta um menu dropdown com duas opções: "True" e "False". A variável `reprocessar` será definida como `True` se e somente se o usuário selecionar essa opção no notebook. O propósito principal deste código é fornecer ao usuário a capacidade de escolher, de maneira interativa, se a carga da dimensão será completa (indicada pela opção `overwrite`) ou incremental (indicada pela opção `merge`).

## Código Fonte

"""  
Cria um widget interativo chamado `reprocessar` para controle da carga da dimensão.  
  
- Exibe um dropdown com as opções "True" e "False".  
- A variável `reprocessar` será `True` apenas se o usuário selecionar essa opção no notebook.  
  
Objetivo: permitir que o usuário escolha, de forma interativa, se a carga será completa (`overwrite`) ou incremental (`merge`).  
"""  
  
dbutils.widgets.dropdown("reprocessar", "False", ["True", "False"], "Reprocessar dimensão?")  
reprocessar = dbutils.widgets.get("reprocessar") == "True"

## Resumo do Código

O trecho de código apresentado é responsável pela criação de uma tabela Delta Lake no Apache Spark, caso ela ainda não exista. A tabela, denominada conforme a variável "delta\_table\_name", é composta por uma série de campos que vão desde identificadores únicos (como "sk\_empresas" e "codigo\_ficticio") até informações detalhadas da empresa (como "nome\_razao\_social", "logradouro", "inscricao\_estadual", entre outros). A tabela é criada utilizando o formato Delta, que permite operações de leitura e gravação em escala de petabytes, e é localizada conforme o caminho especificado na variável "delta\_file". Além disso, um comentário é adicionado à tabela para fornecer informações adicionais sobre ela, conforme especificado na variável "comment\_delta\_table".

## Código Fonte

spark.sql(f"""  
CREATE TABLE IF NOT EXISTS {delta\_table\_name} (  
 sk\_empresas BIGINT GENERATED ALWAYS AS IDENTITY (START WITH 1 INCREMENT BY 1),  
 codigo\_ficticio INT,  
 doc\_empresa STRING,  
 nome\_razao\_social STRING,  
 nome\_fantasia STRING,  
 logradouro STRING,  
 inscricao\_estadual STRING,  
 inscricao\_municipal STRING,  
 complemento\_endereco STRING,  
 codigo\_postal STRING,  
 bairro STRING,  
 codigo\_cidade INT,  
 regiao STRING,  
 responsavel\_regional STRING,  
 categoria\_porte STRING,  
 data\_abertura\_atividade DATE,  
 data\_abertura\_empresa STRING,  
 dias\_atividade STRING,  
 horario\_atividade STRING,  
 segmento STRING,  
 tem\_gnv STRING,  
 tem\_etanol STRING,  
 tem\_supervisor STRING,  
 qtd\_funcionarios STRING,  
 tem\_diesel STRING,  
 tem\_troca\_oleo STRING,  
 codigo\_externo STRING,  
 marca\_parceira STRING,  
 email\_contato STRING,  
 email\_gerente STRING,  
 tamanho\_loja STRING,  
 situacao STRING,  
 data\_encerramento STRING,  
 data\_insercao TIMESTAMP,  
 data\_atualizacao TIMESTAMP  
)   
USING DELTA  
LOCATION '{delta\_file}'  
COMMENT '{comment\_delta\_table}';  
""")

## Resumo do Código

O trecho de código apresentado é uma consulta SQL executada em um ambiente Spark, cujo objetivo é selecionar um conjunto específico de colunas da tabela 'tb\_tab\_empresa\_farol' localizada no ambiente confiável. As colunas selecionadas incluem 'cod\_empresa', 'num\_cnpj', 'nom\_fantasia', 'nom\_razao\_social', 'logradouro', 'inscricao\_estadual', 'inscricao\_municipal', 'complemento\_endereco', 'codigo\_postal', 'bairro' e 'codigo\_cidade'. Após a execução da consulta, o resultado é armazenado em uma visualização temporária chamada 'tab\_empresa', substituindo qualquer visualização existente com o mesmo nome. A cláusula 'where 1=1' é uma condição sempre verdadeira, usada aqui para facilitar a adição de outras condições no futuro, se necessário.

## Código Fonte

spark.sql(f"""  
  
select  
 cod\_empresa  
 ,num\_cnpj  
 ,nom\_fantasia  
 ,nom\_razao\_social  
 ,logradouro  
 ,inscricao\_estadual  
 ,inscricao\_municipal  
 ,complemento\_endereco  
 ,codigo\_postal  
 ,bairro  
 ,codigo\_cidade  
from {environment}.trusted.tb\_tab\_empresa\_farol  
where 1=1  
""").createOrReplaceTempView('tab\_empresa')

## Resumo do Código

O trecho de código apresentado é uma consulta SQL executada no Apache Spark para selecionar uma série de colunas de uma tabela chamada 'sharepoint\_dados\_filiais' que está localizada em um ambiente especificado pela variável 'environment'. As colunas selecionadas incluem informações como código, região, categoria de porte, tamanho da loja, entre outros. A cláusula WHERE 1=1 é uma condição sempre verdadeira, usada para facilitar a adição de outras condições ao código, se necessário. Após a execução da consulta, o resultado é armazenado em uma visualização temporária chamada 'dados\_filiais'. O propósito deste código é preparar e estruturar os dados para análises subsequentes, permitindo que os usuários consultem e manipulem os dados da tabela 'sharepoint\_dados\_filiais' através da visualização temporária 'dados\_filiais'.

## Código Fonte

spark.sql(f"""  
  
select  
 codigo  
 ,regiao  
 ,categoria\_porte  
 ,tamanho\_loja  
 ,regiao  
 ,ga   
 ,categoria\_porte   
 ,data\_abertura  
 ,dias\_de\_func   
 ,horario\_de\_func   
 ,segmento  
 ,tem\_gnv  
 ,tem\_etanol  
 ,tem\_supervisor  
 ,qtd\_funcionarios  
 ,tem\_diesel  
 ,tem\_troca\_oleo  
 ,codigo\_externo  
 ,marca\_parceira  
 ,email\_contato  
 ,email\_gerente  
 ,tamanho\_loja  
 ,situacao  
 ,data\_encerramento  
from {environment}.raw.sharepoint\_dados\_filiais  
where 1=1;  
  
""").createOrReplaceTempView('dados\_filiais')

## Resumo do Código

O trecho de código apresentado é uma consulta SQL executada em um ambiente Spark, cujo objetivo é selecionar o código da empresa e a data de abertura da filial mais antiga, a partir da tabela 'tb\_farol\_faturamento' localizada na camada de destino do ambiente especificado. A consulta agrupa os resultados pelo código da empresa, garantindo que para cada empresa seja retornada a data mais antiga de abertura de filial. O resultado desta consulta é então armazenado em uma visualização temporária chamada 'tab\_faturamento', substituindo qualquer visualização existente com o mesmo nome. Esta visualização temporária pode ser usada para consultas subsequentes dentro da mesma sessão Spark.

## Código Fonte

spark.sql(f"""  
  
select  
 cod\_empresa  
 ,to\_date(min(data), 'yyyy-MM-dd') as data\_abertura\_filial  
from {environment}.camada\_destino.tb\_farol\_faturamento  
where 1=1  
group by  
 cod\_empresa;  
  
""").createOrReplaceTempView('tab\_faturamento')

## Resumo do Código

O trecho de código apresentado é uma consulta SQL executada em um ambiente Spark, cujo objetivo é criar um DataFrame chamado "source\_df". Esta consulta seleciona uma série de campos de três tabelas diferentes: "tab\_empresa", "dados\_filiais" e "tab\_faturamento". As tabelas são unidas através de operações de "left join" usando o campo "cod\_empresa" como chave comum. A consulta também inclui uma cláusula "where" com a condição "1=1", que é sempre verdadeira e, portanto, não tem efeito prático na filtragem dos dados. O resultado é ordenado pelo campo "nom\_fantasia" da tabela "tab\_empresa". Após a execução da consulta, o DataFrame resultante é registrado como uma tabela temporária chamada "source\_df" para uso posterior.

## Código Fonte

source\_df = spark.sql("""  
   
select   
 emp.cod\_empresa as codigo\_ficticio  
 ,emp.num\_cnpj as doc\_empresa  
 ,emp.nom\_razao\_social as nome\_razao\_social  
 ,emp.nom\_fantasia as nome\_fantasia  
 ,emp.logradouro  
 ,emp.inscricao\_estadual  
 ,emp.inscricao\_municipal  
 ,emp.complemento\_endereco  
 ,emp.codigo\_postal  
 ,emp.bairro  
 ,emp.codigo\_cidade  
 ,fil.regiao  
 ,fil.ga as responsavel\_regional  
 ,fil.categoria\_porte as categoria\_porte  
 ,fat.data\_abertura\_filial as data\_abertura\_atividade  
 ,fil.data\_abertura as data\_abertura\_empresa  
 ,fil.dias\_de\_func as dias\_atividade  
 ,fil.horario\_de\_func as horario\_atividade  
 ,fil.segmento  
 ,fil.tem\_gnv  
 ,fil.tem\_etanol  
 ,fil.tem\_supervisor  
 ,fil.qtd\_funcionarios  
 ,fil.tem\_diesel  
 ,fil.tem\_troca\_oleo  
 ,fil.codigo\_externo  
 ,fil.marca\_parceira  
 ,fil.email\_contato  
 ,fil.email\_gerente  
 ,fil.tamanho\_loja  
 ,fil.situacao  
 ,fil.data\_encerramento  
from tab\_empresa emp  
left join dados\_filiais fil  
 on emp.cod\_empresa = fil.codigo  
left join tab\_faturamento fat  
 on emp.cod\_empresa = fat.cod\_empresa  
where 1=1  
order by emp.nom\_fantasia  
""")  
  
source\_df.createOrReplaceTempView('source\_df')

## Resumo do Código

O trecho de código em questão é responsável por gerenciar a inserção de dados em uma tabela Delta. Se a variável "reprocessar" for verdadeira, o código sobrescreve os dados existentes na tabela Delta com os novos dados do DataFrame "source\_df", adicionando também um timestamp de inserção atual. Caso contrário, o código realiza uma operação de "merge", que compara os dados existentes na tabela Delta com os novos dados do DataFrame "source\_df" com base na coluna "codigo\_ficticio". Se um registro correspondente for encontrado, o código atualiza os valores desse registro com os novos dados. Se um registro correspondente não for encontrado, o código insere os novos dados como um novo registro na tabela Delta.

## Código Fonte

if reprocessar:  
 source\_df = source\_df.withColumn("data\_insercao", lit(current\_timestamp() ) )  
 source\_df.write.format("delta")\  
 .mode("overwrite")\  
 .saveAsTable(f"{delta\_table\_name}", path=f"{delta\_file}")  
 print(f"Carga overwrite realizada com sucesso! {delta\_table\_name}")  
else:  
 print("Realizando carga em Merge..")  
 spark.sql(f"""  
  
MERGE INTO {delta\_table\_name} AS target  
USING source\_df AS source  
 ON target.codigo\_ficticio = source.codigo\_ficticio  
WHEN MATCHED THEN  
 UPDATE SET  
 target.doc\_empresa = source.doc\_empresa,  
 target.nome\_razao\_social = source.nome\_razao\_social,  
 target.nome\_fantasia = source.nome\_fantasia,  
 target.logradouro = source.logradouro,  
 target.inscricao\_estadual = source.inscricao\_estadual,  
 target.inscricao\_municipal = source.inscricao\_municipal,  
 target.complemento\_endereco = source.complemento\_endereco,  
 target.codigo\_postal = source.codigo\_postal,  
 target.bairro = source.bairro,  
 target.codigo\_cidade = source.codigo\_cidade,  
 target.regiao = source.regiao,  
 target.responsavel\_regional = source.responsavel\_regional,  
 target.categoria\_porte = source.categoria\_porte,  
 target.data\_abertura\_atividade = source.data\_abertura\_atividade,  
 target.data\_abertura\_empresa = source.data\_abertura\_empresa,  
 target.dias\_atividade = source.dias\_atividade,  
 target.horario\_atividade = source.horario\_atividade,  
 target.segmento = source.segmento,  
 target.tem\_gnv = source.tem\_gnv,  
 target.tem\_etanol = source.tem\_etanol,  
 target.tem\_supervisor = source.tem\_supervisor,  
 target.qtd\_funcionarios = source.qtd\_funcionarios,  
 target.tem\_diesel = source.tem\_diesel,  
 target.tem\_troca\_oleo = source.tem\_troca\_oleo,  
 target.codigo\_externo = source.codigo\_externo,  
 target.marca\_parceira = source.marca\_parceira,  
 target.email\_contato = source.email\_contato,  
 target.email\_gerente = source.email\_gerente,  
 target.tamanho\_loja = source.tamanho\_loja,  
 target.situacao = source.situacao,  
 target.data\_encerramento = source.data\_encerramento,  
 target.data\_atualizacao = current\_timestamp()  
WHEN NOT MATCHED THEN  
 INSERT (  
 codigo\_ficticio,  
 doc\_empresa,  
 nome\_razao\_social,  
 nome\_fantasia,  
 logradouro,  
 inscricao\_estadual,  
 inscricao\_municipal,  
 complemento\_endereco,  
 codigo\_postal,  
 bairro,  
 codigo\_cidade,  
 regiao,  
 responsavel\_regional,  
 categoria\_porte,  
 data\_abertura\_atividade,  
 data\_abertura\_empresa,  
 dias\_atividade,  
 horario\_atividade,  
 segmento,  
 tem\_gnv,  
 tem\_etanol,  
 tem\_supervisor,  
 qtd\_funcionarios,  
 tem\_diesel,  
 tem\_troca\_oleo,  
 codigo\_externo,  
 marca\_parceira,  
 email\_contato,  
 email\_gerente,  
 tamanho\_loja,  
 situacao,  
 data\_encerramento,  
 data\_insercao  
 )  
 VALUES (  
 source.codigo\_ficticio,  
 source.doc\_empresa,  
 source.nome\_razao\_social,  
 source.nome\_fantasia,  
 source.logradouro,  
 source.inscricao\_estadual,  
 source.inscricao\_municipal,  
 source.complemento\_endereco,  
 source.codigo\_postal,  
 source.bairro,  
 source.codigo\_cidade,  
 source.regiao,  
 source.responsavel\_regional,  
 source.categoria\_porte,  
 source.data\_abertura\_atividade,  
 source.data\_abertura\_empresa,  
 source.dias\_atividade,  
 source.horario\_atividade,  
 source.segmento,  
 source.tem\_gnv,  
 source.tem\_etanol,  
 source.tem\_supervisor,  
 source.qtd\_funcionarios,  
 source.tem\_diesel,  
 source.tem\_troca\_oleo,  
 source.codigo\_externo,  
 source.marca\_parceira,  
 source.email\_contato,  
 source.email\_gerente,  
 source.tamanho\_loja,  
 source.situacao,  
 source.data\_encerramento,  
 current\_timestamp()  
 )  
   
 """)

Documento Interno - A divulgação sem autorização prévia viola as normas e diretrizes da organização.