

Super Market - Análise de Dados com Python, Google Colab e SQLite

Este projeto apresenta uma análise de dados de supermercado utilizando **Python em Google Colab**, com **persistência dos dados via SQLite** salvo diretamente no **Google Drive**.

Todos os notebooks e arquivos são versionados via **GitHub**, permitindo organização, colaboração e histórico de alterações.

Tecnologias

- Python (pandas, sqlite3, matplotlib, seaborn)
- Google Colab
- Google Drive (armazenamento do banco de dados)
- SQLite (banco local)
- GitHub (versionamento de código)

Estrutura do Projeto

1. **Google Drive:** Armazena o banco `super_market.db` de forma persistente.
2. **SQLite:** Gerencia os dados diretamente pelo Colab.
3. **GitHub:** Guarda o código-fonte e notebooks.

Como executar o notebook no Colab:

1 Executar tudo de uma vez:

Ideal ao abrir o notebook pela primeira vez.

Vá em **Ambiente de execução > Executar tudo** ou use o atalho `Ctrl+F9`.

2 Executar a partir de uma célula:

Útil se você já rodou parte do código e quer continuar.

Clique na célula desejada e vá em **Ambiente de execução > Executar a partir daqui**.

3 Executar manualmente (uma por uma):

Use `Shift+Enter` em cada célula para rodar individualmente.

Ótimo para revisar ou testar partes do código.



Iniciando: toda vez que abrir o notebook



1. Montar o Google Drive: `drive.mount('/content/drive')`



2. Definir o caminho do banco de dados no Drive



3. Conectar ao banco SQLite: `con = sqlite3.connect(caminho_banco)`



4. Importar as bibliotecas (pandas, sqlite3, os, etc.)



Verifique se o `con.close()` está comentado no final para evitar desconexão durante a execução completa



Etapa 1 - Montar o Google Drive (sempre ao abrir o notebook) :

Conecta seu Google Drive ao Colab para acessar e salvar arquivos diretamente, como o banco de dados SQLite.

```
from google.colab import drive # Importa a função para conectar o
    Google Drive no Colab
drive.mount('/content/drive') # Monta o Drive no diretório
    /content/drive para acesso aos arquivos
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call `drive.mount("/content/drive", force_remount=True)`.



Etapa 2 – Definir o caminho do banco de dados no Drive (sempre após montar o Drive):

Especificamos o caminho onde o arquivo `super_market.db` será salvo e garantimos que a pasta exista no Google Drive.

```
import os # Biblioteca para manipular caminhos e pastas no sistema
    operacional
caminho_banco = '/content/drive/MyDrive/Colab
    Notebooks/super_market/super_market.db' # Caminho completo
    do banco no Drive
os.makedirs(os.path.dirname(caminho_banco), exist_ok=True) # Cria a
    pasta se ainda não existir
```



Etapa 3 – Conectar ao banco SQLite (sempre após definir o

caminho): Estabelece a conexão com o banco de dados `super_market.db` no Drive. Se o arquivo não existir, será criado automaticamente.

```
import sqlite3 # Biblioteca para trabalhar com bancos SQLite em
    Python
con = sqlite3.connect(caminho_banco) # Conecta ao banco usando o
    caminho definido
cursor = con.cursor() # Cria um cursor para executar comandos SQL
```

Etapa 4 - Importando bibliotecas essenciais:

Estas bibliotecas são utilizadas ao longo do notebook para análise, visualização e manipulação de dados.

```
# Lista de bibliotecas python
import pandas as pd          # Manipulação de dados com
    DataFrames
import sqlite3              # Conexão e comandos SQL com banco
    SQLite
import csv                  # Leitura e escrita de arquivos CSV
import os                   # Operações com arquivos e
    diretórios
import matplotlib.pyplot as plt # Visualização de dados
import seaborn as sns       # Visualização de dados
```

Realizando alterações no código:

- Criar, consultar, inserir, atualizar e excluir dados
- Usar pandas, seaborn, matplotlib para análises e visualizações
- Sempre que fizer mudanças no banco, execute: con.commit()

O que você não precisa executar sempre:

- Criar tabelas já existentes
- Inserir dados duplicados

• Clonar GitHub se já foi feito nesta sessão

Criando a tabela Products:


Tabela usada para armazenar os produtos disponíveis no sistema.

Inclui dados como nome do item, categoria, estoque, tipo e preços de compra/venda.


A coluna OrderID funciona como identificador único para cada produto.

```
cursor.execute("""
    CREATE TABLE IF NOT EXISTS Products (
        OrderID INTEGER PRIMARY KEY AUTOINCREMENT,
        Item TEXT NOT NULL,
        Category TEXT,
        Stock INTEGER DEFAULT 0,
        Type TEXT,
        Purchase_Price REAL NOT NULL,
```

```

        sale_price REAL CHECK(sale_price >= 0)
    )
)
con.commit() #  salva a criação da tabela no banco

```

 Consultar tabelas existentes no banco: Usado para verificar quais tabelas já foram criadas no banco de dados SQLite.

```

result = cursor.execute("SELECT name FROM sqlite_master WHERE
                        type='table'")
result.fetchall()

[('Products',), ('sqlite_sequence',)]

```

Importar e Ler o CSV com pandas

Nesta etapa, usamos o pandas para ler o arquivo CSV com os dados dos produtos que serão inseridos no banco de dados.

```

# Lendo o CSV
df = pd.read_csv('/content/drive/MyDrive/Colab
                Notebooks/super_market/products_supermarket.csv')
# Visualiza as primeiras linhas
df.head(5)

```

	Order	Item	Category	Stock	Type	Purchase Price (USD)	Sale Price (USD)
0	1	Rice	Basic Foods	120	5kg pack	3.32	4.38
1	2	Beans	Basic Foods	95	1kg pack	1.49	2.11
2	3	Pasta	Basic Foods	150	500g pack	0.56	0.97
3	4	Soybean Oil	Basic Foods	80	900ml bottle	1.20	1.74
4	5	Sugar	Basic Foods	110	1kg pack	0.76	1.14

Renomear colunas para combinar com a tabela SQL

Renomeamos as colunas do DataFrame para garantir que os nomes sejam exatamente os mesmos da tabela criada no banco SQLite.

```

df.rename(columns={
    'Order': 'OrderID',

```

```
'Purchase Price (USD)': 'Purchase_Price',
'Sale Price (USD)': 'Sale_Price'
}, inplace=True)
con.commit()
```

🔍 Verificar se as colunas estão corretas

Antes de importar os dados, exibimos os nomes das colunas e as primeiras linhas do DataFrame para conferir se está tudo certo.

```
print("Colunas do DataFrame:", df.columns.tolist())
df.head()
```

Colunas do DataFrame: ['OrderID', 'Item', 'Category', 'Stock', 'Type', 'Purchase_Price', 'Sale_Price']

	OrderID	Item	Category	Stock	Type	Purchase_Price	Sale_Price
0	1	Rice	Basic Foods	120	5kg pack	3.32	4.38
1	2	Beans	Basic Foods	95	1kg pack	1.49	2.11
2	3	Pasta	Basic Foods	150	500g pack	0.56	0.97
3	4	Soybean Oil	Basic Foods	80	900ml bottle	1.20	1.74
4	5	Sugar	Basic Foods	110	1kg pack	0.76	1.14

✂ Inserir dados no banco de dados

Percorremos cada linha do DataFrame e inserimos os dados na tabela Products, usando `INSERT OR IGNORE INTO`


```
for _, row in df.iterrows():
    cursor.execute("""
        INSERT OR IGNORE INTO Products (OrderID, Item, Category,
        Stock, Type, Purchase_Price, Sale_Price)
        VALUES (?, ?, ?, ?, ?, ?, ?)
    """, tuple(row))
con.commit()
```

😊 Verificar dados salvos da tabela no banco

consulta na tabela para exibir os dados já inseridos e garantir que o processo de importação foi bem-sucedido.

```
df_produtos = pd.read_sql_query("SELECT * FROM Products", con)
df_produtos.head() # Mostra as 5 primeiras linhas
```

	OrderID	Item	Category	Stock	Type	Purchase_Price	Sale_Price
0	1	Rice	Basic Foods	120	5kg pack	3.32	4.38
1	2	Beans	Basic Foods	95	1kg pack	1.49	2.11
2	3	Pasta	Basic Foods	150	500g pack	0.56	0.97
3	4	Soybean Oil	Basic Foods	80	900ml bottle	1.20	1.74
4	5	Sugar	Basic Foods	110	1kg pack	0.76	1.14



 Total de Produtos inseridos: A quantidade de produtos, importados do '.csv' inseridos na tabela 'Products' do banco de dados 'super-market.db'

```
df_verificacao = pd.read_sql_query("SELECT * FROM Products", con)
print(f'Total de registros inseridos: {len(df_verificacao)}')
```

Total de registros inseridos: 100



Antes de fechar o notebook:

-  Execute con.commit() para salvar tudo
-  Execute con.close() para desconectar com segurança

-  **Confirme que o arquivo foi salvo no Google Drive**



Salvar alterações no banco de dados

Executamos con.commit() para confirmar as alterações e garantir que os dados fiquem salvos de forma permanente.

```
con.commit()
print("✅ Dados importados com sucesso para o banco SQLite!")
```




Dados importados com sucesso para o banco SQLite!

 **Executar esta célula SOMENTE ao finalizar todo o trabalho com o banco. retirar comentarios (#)**

 **Fechar a conexão com o banco**

Fechamos a conexão com o SQLite após terminar as operações, liberando recursos do sistema.

```
#  Fechar conexão com o banco de dados (só executar ao finalizar tudo)
# Use esta célula apenas quando terminar TODAS as edições e inserções no banco
# con.close()
# print("✅ Banco de dados desconectado com sucesso.")
```

5. Coisas opcionais:

- Fazer backup no GitHub
- Baixar o .db localmente
- Comentar o con.close() se quiser continuar testando

• Deixar este checklist visível e organizado no topo do notebook

 **Salvar no GitHub**

Após as alterações, podemos fazer backup do notebook e/ou do banco de dados, enviando-os para um repositório no GitHub como forma de controle de versão.

```
# @title
'''# ☁ Etapa: Backup automático no GitHub (com token oculto)

from getpass import getpass # Para ocultar a digitação do token
import shutil

#  Digitar token manualmente (não aparece na tela)
token = getpass("Digite seu GitHub Personal Access Token (oculto): ")

# Seus dados do repositório
usuario = "italomellors"
repositorio = "Data_Analysis_Projects"

#  Clonar o repositório (só uma vez por sessão)
```

```
!git clone https://{token}:\n      .[email protected]/{usuario}/{repositorio}.git\n\n# 📁 Mudar para a pasta do repositório clonado\n%cd {repositorio}\n\n# 📁 Copiar o banco do Drive para o repositório local\nshutil.copy('/content/drive/MyDrive/Colab\n      Notebooks/super_market/super_market.db', './super_market.db')\n\n# ✅ Git: Preparar, registrar e enviar os arquivos\n!git status\n!git add super_market.db\n!git commit -m "Backup automático do banco de dados SQLite"\n!git push''
```