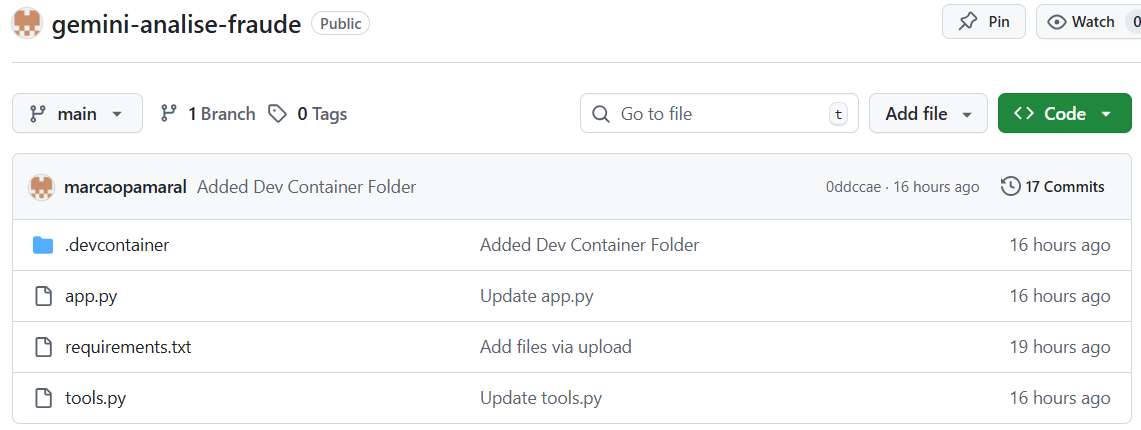
**Conteúdo dos Arquivos:**

****

1. **Requirements.txt**

streamlit

langchain

langchain-google-genai

langchain-core

pandas

matplotlib

google-generativeai

requests

1. **Tools.py**

import pandas as pd

import matplotlib.pyplot as plt

import os

import io

import sys

from io import BytesIO

import requests # Importação necessária para o carregamento via URL

# Variável global para a URL do arquivo grande (150MB)

# CORREÇÃO: O link do Dropbox foi alterado de 'dl=0' para 'dl=1' para forçar o download direto

PUBLIC\_CSV\_URL = "https://www.dropbox.com/scl/fi/ibuflwf3bvau3a624f3ep/creditcard.csv?rlkey=duuiekt9cskkoya6rf3opokht&st=n1b9m26x&dl=1"

def carregar\_dados\_ou\_demo():

"""Tenta carregar o creditcard.csv via URL pública ou cria um DataFrame de demonstração."""

# URL de Placeholder genérica para verificar se o usuário atualizou a chave

GENERIC\_PLACEHOLDER\_URL = "https://example.com/seu\_arquivo\_publico\_de\_150MB.csv"

# 1. Tenta carregar o arquivo da URL pública

# Verifica se a URL está configurada e não é o placeholder genérico

if PUBLIC\_CSV\_URL and PUBLIC\_CSV\_URL != GENERIC\_PLACEHOLDER\_URL:

try:

print(f"[INFO] Tentando carregar dados da URL: {PUBLIC\_CSV\_URL}")

# pd.read\_csv pode ler diretamente de uma URL

df\_retorno = pd.read\_csv(PUBLIC\_CSV\_URL)

print(f"[INFO] Dados carregados com sucesso via URL.")

return df\_retorno

except Exception as e:

print(f"[AVISO] Falha ao carregar dados da URL ({e}). Recorrendo ao carregamento local/demo.")

# 2. Tenta carregar o arquivo localmente (para desenvolvimento)

# Em um ambiente de nuvem, isso provavelmente falhará e levará ao demo

file\_path = 'data/creditcard.csv'

if os.path.exists(file\_path):

try:

df\_retorno = pd.read\_csv(file\_path)

print(f"[INFO] Dados carregados com sucesso de '{file\_path}' (Ambiente Local).")

return df\_retorno

except Exception as e:

print(f"[AVISO] Falha ao carregar arquivo local: {e}. Criando DataFrame de demonstração.")

# 3. Cria DataFrame de demonstração (Fallback final)

print(f"[AVISO] Arquivo não encontrado ou falha de URL. Criando DataFrame de demonstração.")

# DataFrame de demonstração com a estrutura exigida

data = {

'Time': range(100),

'Amount': [10 + i % 100 for i in range(100)],

'Class': [0] \* 95 + [1] \* 5 # 5% de fraude

}

# Adiciona as colunas V1 a V28 como zeros

for i in range(1, 29):

data[f'V{i}'] = [i \* 0.1 for i in range(100)]

colunas\_pca = [f'V{i}' for i in range(1, 29)]

colunas\_ordenadas = ['Time'] + colunas\_pca + ['Amount', 'Class']

df\_retorno = pd.DataFrame(data).reindex(columns=colunas\_ordenadas)

return df\_retorno

def consulta\_tool(df: pd.DataFrame, codigo\_python: str) -> str:

"""

Executa um trecho de código Python no DataFrame 'df' e retorna o resultado formatado.

Args:

df: O DataFrame de dados.

codigo\_python: O código Python (como string) para executar no DataFrame 'df'.

Returns:

O resultado da execução do código como uma string.

"""

if df is None:

return "Erro: O DataFrame não foi carregado corretamente."

# Usa um buffer de texto para capturar a saída padrão

stdout\_buffer = io.StringIO()

sys.stdout = stdout\_buffer

try:

# A função exec() executa o código; o DataFrame 'df' é passado no escopo local

exec\_locals = {'df': df}

exec(codigo\_python, {}, exec\_locals) # Usa escopo local para 'df'

output = stdout\_buffer.getvalue().strip()

if not output:

try:

# Tenta avaliar o código (eval) para obter o valor de retorno de expressões simples

result = eval(codigo\_python, {}, exec\_locals)

return str(result)

except Exception as e:

# Erro de eval ou código que não retorna valor

return f"Código Python executado. Retorno: OK (Use 'print()' para visualizar grandes outputs) | Erro Eval: {e}"

else:

return output

except Exception as e:

return f"Erro na execução do código Python: {e}"

finally:

sys.stdout = sys.\_\_stdout\_\_

def grafico\_tool(df: pd.DataFrame, tipo\_grafico: str, colunas: list, titulo: str) -> BytesIO | str:

"""

Gera um gráfico com base no tipo especificado e retorna o buffer de memória da imagem (BytesIO).

Args:

df: O DataFrame de dados.

tipo\_grafico: Tipo de gráfico ('hist', 'box', 'scatter', 'bar').

colunas: Lista de colunas a serem plotadas.

titulo: Título do gráfico.

Returns:

Um objeto BytesIO contendo o PNG do gráfico, ou uma string de erro.

"""

if df is None:

return "Erro: O DataFrame não foi carregado corretamente para gerar o gráfico."

try:

plt.figure(figsize=(10, 6))

# Lógica de plotagem (mantida)

if tipo\_grafico == 'hist' and len(colunas) == 1:

df[colunas[0]].hist(bins=50, edgecolor='black', alpha=0.7)

plt.title(f'Histograma de {colunas[0]}')

plt.xlabel(colunas[0])

plt.ylabel('Frequência')

elif tipo\_grafico == 'box' and len(colunas) == 1:

df.boxplot(column=colunas[0], by='Class', grid=False, figsize=(8, 6))

plt.suptitle('')

plt.title(f'Boxplot de {colunas[0]} por Classe (0=Normal, 1=Fraude)')

plt.xlabel('Classe')

plt.ylabel(colunas[0])

elif tipo\_grafico == 'scatter' and len(colunas) == 2:

col\_x, col\_y = colunas[0], colunas[1]

plt.scatter(df[col\_x], df[col\_y], c=df['Class'], cmap='coolwarm', alpha=0.6)

plt.title(f'Dispersão de {col\_x} vs {col\_y} (Cor por Fraude)')

plt.xlabel(col\_x)

plt.ylabel(col\_y)

plt.colorbar(label='Class (0=Normal, 1=Fraude)')

elif tipo\_grafico == 'bar' and colunas[0].lower() == 'class':

fraudes = df['Class'].value\_counts()

fraudes.plot(kind='bar', color=['skyblue', 'salmon'])

plt.title('Contagem de Transações por Classe')

plt.xlabel('Classe (0=Normal, 1=Fraude)')

plt.ylabel('Contagem')

plt.xticks(rotation=0)

else:

plt.close()

return f"Erro: Tipo de gráfico '{tipo\_grafico}' ou número de colunas inválido."

plt.suptitle(titulo, fontsize=16)

plt.tight\_layout()

# Salva o gráfico em um buffer de memória

buffer = BytesIO()

plt.savefig(buffer, format='png')

buffer.seek(0)

plt.close() # Libera a memória

return buffer

except KeyError as e:

plt.close()

return f"Erro: Coluna não encontrada: {e}. Colunas disponíveis: {df.columns.tolist()}"

except Exception as e:

plt.close()

return f"Erro inesperado ao gerar o gráfico: {e}"

1. **App.py**

import streamlit as st

import pandas as pd

import json

import time

import requests

from io import BytesIO

from tools import carregar\_dados\_ou\_demo, consulta\_tool, grafico\_tool # Importa as ferramentas e o carregador

# --- Configurações Iniciais ---

# URL da API do Gemini (usada para chamadas não-streaming)

API\_URL = "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.5-flash-preview-05-20:generateContent"

MODEL\_NAME = "gemini-2.5-flash-preview-05-20"

# Instrução do sistema para guiar o agente

SYSTEM\_INSTRUCTION = (

"Você é um Agente de Análise de Fraudes especializado em DataFrames pandas. "

"Sua função é responder a perguntas usando as ferramentas 'consulta\_tool' ou 'grafico\_tool'. "

"NÃO gere código Python diretamente na resposta; use as ferramentas."

"O DataFrame principal é chamado 'df' e contém colunas 'Time', 'V1' a 'V28', 'Amount' e 'Class'. "

"Sempre que o usuário pedir análise numérica ou estatística, use 'consulta\_tool'. "

"Sempre que o usuário pedir visualização (gráfico, histograma, boxplot), use 'grafico\_tool'."

"Responda de forma concisa e profissional, em português."

)

# --- Carregamento de Dados (Cache) ---

@st.cache\_data(show\_spinner="Carregando o DataFrame... (pode levar alguns minutos devido ao tamanho de 150MB)")

def load\_data():

"""Carrega o DataFrame (via URL) usando a função do tools.py."""

# Chama a função corrigida do tools.py que tenta carregar via URL pública

return carregar\_dados\_ou\_demo()

# Carrega o DataFrame no estado da aplicação

df = load\_data()

# --- Funções de Comunicação com a API ---

def call\_gemini\_api(history: list, tools: list | None = None) -> dict:

"""Função central para chamar a API do Gemini com backoff exponencial."""

# 1. Obtenção da Chave API

api\_key = st.secrets.get("GEMINI\_API\_KEY", "")

if not api\_key:

api\_key = st.session\_state.get("api\_key\_input", "")

if not api\_key:

st.error("Por favor, insira sua Chave de API Gemini na barra lateral.")

return {} # Retorna dicionário vazio para evitar crash

# 2. Construção do Payload

payload = {

"contents": history,

"systemInstruction": {"parts": [{"text": SYSTEM\_INSTRUCTION}]},

}

if tools:

payload["tools"] = tools

headers = {

'Content-Type': 'application/json'

}

# 3. Lógica de Backoff e Requisição

max\_retries = 5

for attempt in range(max\_retries):

try:

# Anexa a chave API diretamente na URL

response = requests.post(f"{API\_URL}?key={api\_key}", headers=headers, data=json.dumps(payload))

response.raise\_for\_status() # Lança exceção para códigos 4xx/5xx

# Se a resposta foi bem-sucedida, retorna o JSON

return response.json()

except requests.exceptions.HTTPError as http\_err:

# Captura o erro 400 que você está vendo

st.warning(f"Erro de comunicação com a API: {http\_err}. Resposta: {response.text}")

if attempt == max\_retries - 1:

st.error(f"Falha na comunicação com a API após {max\_retries} tentativas. Verifique sua chave ou o formato JSON.")

return {}

time.sleep(2 \*\* attempt)

except requests.exceptions.RequestException as req\_err:

# Captura outros erros de requisição (timeout, DNS, etc.)

st.warning(f"Erro de conexão: {req\_err}. Tentando novamente em {2\*\*attempt}s...")

if attempt == max\_retries - 1:

st.error(f"Falha na conexão com a API após {max\_retries} tentativas.")

return {}

time.sleep(2 \*\* attempt)

return {} # Retorno de segurança

def run\_conversation(prompt: str):

"""Gerencia o ciclo de conversa, incluindo a chamada de ferramentas."""

# 1. Adiciona a nova pergunta ao histórico de chat

st.session\_state.messages.append({"role": "user", "parts": [{"text": prompt}]})

# 2. Prepara a lista de ferramentas disponíveis

available\_tools = [

{

"functionDeclarations": [

{

"name": "consulta\_tool",

"description": "Executa código Python para consultar o DataFrame 'df' e retorna resultados como string. Use para obter estatísticas, valores, linhas específicas, etc.",

"parameters": {

"type": "OBJECT",

"properties": {

"codigo\_python": {"type": "STRING", "description": "O código Python a ser executado no DataFrame 'df'. Ex: df.shape[0]"}

},

"required": ["codigo\_python"]

}

},

{

"name": "grafico\_tool",

"description": "Gera um gráfico e retorna a imagem em buffer de memória. Use para histogramas, boxplots, dispersão (scatter) e gráficos de barra.",

"parameters": {

"type": "OBJECT",

"properties": {

"tipo\_grafico": {"type": "STRING", "description": "Tipo: 'hist', 'box', 'scatter' ou 'bar'."},

"colunas": {"type": "ARRAY", "items": {"type": "STRING"}, "description": "Lista de 1 ou 2 colunas para o gráfico. Ex: ['Amount']"},

"titulo": {"type": "STRING", "description": "Título descritivo para o gráfico."}

},

"required": ["tipo\_grafico", "colunas", "titulo"]

}

}

]

}

]

# 3. Primeira chamada: Envia a pergunta e o histórico para ver se o modelo usa a ferramenta

with st.spinner("🧠 Pensando... (Primeira Chamada)"):

response\_1 = call\_gemini\_api(st.session\_state.messages, tools=available\_tools)

# Se a primeira chamada falhou (retornou dicionário vazio)

if not response\_1:

st.session\_state.messages.pop() # Remove a última mensagem do usuário para tentar novamente

return

# 4. Processa a resposta (Texto ou Chamada de Função)

try:

candidate = response\_1["candidates"][0]

# 4.1. Se o modelo chamou uma função (Function Call)

if candidate["content"]["parts"] and "functionCall" in candidate["content"]["parts"][0]:

function\_call = candidate["content"]["parts"][0]["functionCall"]

func\_name = function\_call["name"]

func\_args = dict(function\_call["args"])

# Adiciona a chamada de função ao histórico

st.session\_state.messages.append(candidate["content"])

# Executa a função localmente

tool\_output = "Erro: Ferramenta não executada."

if func\_name == "consulta\_tool":

with st.spinner(f"🛠️ Executando consulta: `{func\_args.get('codigo\_python')}`"):

tool\_output = consulta\_tool(df, func\_args["codigo\_python"])

elif func\_name == "grafico\_tool":

with st.spinner(f"📊 Gerando gráfico: {func\_args.get('titulo')}"):

buffer\_ou\_erro = grafico\_tool(df, func\_args.get("tipo\_grafico"), func\_args.get("colunas"), func\_args.get("titulo"))

if isinstance(buffer\_ou\_erro, BytesIO):

# Se for BytesIO (gráfico), armazena no estado para exibição

st.session\_state.tool\_image = buffer\_ou\_erro

tool\_output = "Gráfico gerado com sucesso e salvo em buffer."

else:

# Se for string (erro)

tool\_output = buffer\_ou\_erro

# Adiciona o resultado da ferramenta ao histórico

tool\_result\_part = {

"functionResponse": {

"name": func\_name,

"response": {"output": tool\_output}

}

}

st.session\_state.messages.append({"role": "user", "parts": [tool\_result\_part]})

# Segunda chamada: Envia o resultado da ferramenta para o modelo gerar o texto final

with st.spinner("💬 Gerando resposta final... (Segunda Chamada)"):

response\_2 = call\_gemini\_api(st.session\_state.messages, tools=available\_tools)

if not response\_2:

st.session\_state.messages.pop() # Remove a mensagem de resultado da ferramenta

st.session\_state.messages.pop() # Remove a mensagem de chamada da ferramenta

st.session\_state.messages.pop() # Remove a mensagem original do usuário

return

# Extrai a resposta final do modelo

final\_text = response\_2["candidates"][0]["content"]["parts"][0]["text"]

# Adiciona a resposta final ao histórico e à interface

st.session\_state.messages.append({"role": "model", "parts": [{"text": final\_text}]})

st.rerun() # FORÇA O RERUN PARA EXIBIR A RESPOSTA IMEDIATAMENTE!

# 4.2. Se o modelo respondeu diretamente com texto

else:

final\_text = candidate["content"]["parts"][0]["text"]

st.session\_state.messages.append({"role": "model", "parts": [{"text": final\_text}]})

st.rerun() # FORÇA O RERUN PARA EXIBIR A RESPOSTA IMEDIATAMENTE!

except Exception as e:

st.error(f"Um erro ocorreu ao processar a resposta da API: {e}. Isso pode indicar um erro de parse do JSON da API.")

return

# --- Interface do Streamlit ---

st.set\_page\_config(page\_title="Agente de Análise de Fraudes (Gemini)", layout="wide")

st.title("FraudGuard: Agente de Análise de Fraudes 💳")

st.markdown("Use o poder do Gemini e pandas para analisar os dados de fraude de cartão de crédito (150MB).")

st.markdown("---")

# 1. Inicialização do Histórico e Imagem Temporária

if "messages" not in st.session\_state:

st.session\_state.messages = []

if "tool\_image" not in st.session\_state:

st.session\_state.tool\_image = None

if "api\_key\_input" not in st.session\_state:

st.session\_state.api\_key\_input = ""

# 2. Sidebar para API Key e Info

with st.sidebar:

st.header("Configuração")

st.info("Insira sua Chave API do Google Gemini. Se estiver no Streamlit Cloud, configure-a em 'Secrets'.")

# Campo para inserir a chave API manualmente (útil para desenvolvimento local)

api\_key\_input = st.text\_input("Sua Chave API Gemini:", type="password", help="A chave será armazenada apenas nesta sessão.")

st.session\_state.api\_key\_input = api\_key\_input

st.markdown("---")

st.header("Status dos Dados")

if df.shape[0] < 1000:

st.warning(f"Usando DataFrame de Demonstração (Linhas: {df.shape[0]}).")

st.write("Verifique se o link do Dropbox na 'tools.py' está acessível publicamente e se a URL termina em `dl=1`.")

else:

st.success(f"Dados Carregados com Sucesso! (Linhas: {df.shape[0]} | Colunas: {df.shape[1]})")

# 3. Exibição do Histórico de Chat

chat\_container = st.container()

with chat\_container:

# Itera sobre o histórico de mensagens para exibição

for message in st.session\_state.messages:

role = "assistant" if message["role"] == "model" else "user"

# Ignora as partes do histórico que são chamadas de função/resposta de ferramenta para o usuário final

if "functionCall" in message["parts"][0] or "functionResponse" in message["parts"][0]:

continue

# Exibe mensagens de texto

if "text" in message["parts"][0]:

with st.chat\_message(role):

st.markdown(message["parts"][0]["text"])

# Exibe o gráfico gerado pela ferramenta, se houver

if st.session\_state.tool\_image:

with st.chat\_message("assistant"):

st.image(st.session\_state.tool\_image, caption="Resultado da Visualização de Dados", use\_column\_width=True)

st.session\_state.tool\_image = None # Limpa a imagem após exibição

# 4. Input de Chat

if prompt := st.chat\_input("Pergunte sobre os dados (ex: 'Qual a média do Amount?')"):

# Limpa a imagem anterior antes de processar a nova pergunta

st.session\_state.tool\_image = None

with chat\_container:

with st.chat\_message("user"):

st.markdown(prompt)

# Inicia a conversa e o processamento de ferramentas

run\_conversation(prompt)

# 5. Adiciona o primeiro prompt de boas-vindas se o histórico estiver vazio

if not st.session\_state.messages:

st.session\_state.messages.append({"role": "model", "parts": [{"text": "Olá! Eu sou o FraudGuard. Tenho acesso ao seu DataFrame de fraudes. Como posso analisar seus dados hoje?"}]})

st.rerun() # Reinicia para mostrar a mensagem de boas-vindas