INSTITUT D'INTELLIGENCE ARTIFICIELLE APPLIQUÉE
AGENTE AUTÔNOMOS COM REDES GENERATIVAS

GRUPO 268 - Agentes Autônomos com IA Generativa

GUILHERME
LAION
JOSÉ
SAMILLA
LUCIANA
PEDRO
ANDRÉ
ISRAEL
FELIPE

AGENTES AUTÔNOMOS:
ANÁLISE DE CSV

https://github.com/guilhermegodoim/ia2a-grupo268/tree/main/Agente%20de%20Notas%20Fiscais (Link para os arquivos diretamente no github)
https://github.com/guilhermegodoim/ia2a-grupo268/blob/main/Agente%20CSV.rar (Link para a pasta .rar)

BRASIL
2025

1. Framework Escolhida

A solução foi desenvolvida com base na framework LangChain, uma ferramenta robusta para construção de agentes conversacionais e sistemas inteligentes que integram modelos de linguagem (LLMs) com dados externos, como arquivos, bancos de dados e APIs.

Além do LangChain, também foram utilizados:

- **Google Gemini API**: modelo de linguagem escolhido para inferência inteligente com suporte a raciocínio estruturado.
- **Pydantic**: framework para validação e estruturação rigorosa dos dados.
- **Pandas**: biblioteca essencial para manipulação de dados tabulares (CSV).
- **LangChain Experimental Toolkit**: permite a criação de agentes especializados que operam diretamente sobre DataFrames do Pandas.

2. Estrutura da Solução

A arquitetura da solução foi organizada em quatro componentes principais:

2.1 env – Variáveis de Ambiente

O arquivo .env é usado para armazenar informações sensíveis e configurações externas, como:

GOOGLE_API_KEY= Chave API

Essa abordagem:

- Protege credenciais de acesso a APIs (como a chave do Gemini)
- Permite reutilizar o código em diferentes ambientes sem alterações diretas
- Facilita o versionamento seguro (por exemplo, ignorando o .env no .gitignore)

2.2 agente.py – Núcleo da IA

Este código implementa um agente de IA especializado na análise de dados fiscais, usando a biblioteca LangChain, a IA Gemini do Google e validação de dados com Pydantic.

Ele lê dois arquivos .csv: um de notas fiscais e outro de produtos/serviços, padroniza e combina os dados e cria um agente de IA que responde perguntas sobre esses dados em português natural, sem gerar código.

2.2.1 . Bibliotecas Importadas

```
import os
import pandas as pd
from pydantic import BaseModel, ValidationError
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_experimental.agents import create_pandas_dataframe_agent
from typing import Optional
```

- os: manipulação de arquivos e variáveis de ambiente (.env).
- pandas: tratamento de tabelas e CSVs.
- pydantic: validação de dados estruturados.
- langchain_google_genai: usa o modelo **Gemini** do Google.
- create_pandas_dataframe_agent: cria o agente que entende perguntas sobre dados tabulares.
- Optional: usado nas classes para indicar campos que **podem ser nulos**.

2.2.2 Função padronizar_colunas

```python
# --------- Padronização das colunas ------------
def padronizar_colunas(df: pd.DataFrame) -> pd.DataFrame:
    df.columns = [
        col.strip()
           .upper()
           .replace(' ', '_')
           .replace('/', '_')
           .replace('Ç', 'C')
           .replace('Á', 'A')
           .replace('É', 'E')
           .replace('Í', 'I')
           .replace('Ó', 'O')
           .replace('Ú', 'U')
           .replace('Â', 'A')
           .replace('Ê', 'E')
           .replace('Ô', 'O')
        for col in df.columns
    ]
    return df
```

Essa função converte todos os nomes de colunas para maiúsculo e substitui espaços e caracteres especiais (como "ç", "á", "ê") para garantir que os nomes fiquem padronizados e compatíveis com a validação e uso posterior. Por exemplo:

"Razão Social Emitente" → "RAZAO_SOCIAL_EMITENTE"

2.2.3 Modelos de Dados com Pydantic

Define a estrutura esperada dos dados de cada nota fiscal. O Pydantic garante que:

- Os tipos estão corretos (ex: str, float)
- Campos obrigatórios estejam presentes
- Campos opcionais são tratados de forma segura
- A mesma lógica vale para a classe ProdutoNotaFiscal.

Obs: embora a validação esteja comentada no código atual, essa estrutura é importante para manter a integridade dos dados.

2.2.4 Função carregar_csvs_de_zip

```python
def carregar_csvs_de_zip(caminho_pasta: str) -> pd.DataFrame:
    arquivos_csv = [f for f in os.listdir(caminho_pasta) if f.endswith('.csv')]
    if len(arquivos_csv) < 2:
        raise ValueError("Esperado pelo menos 2 arquivos CSV na pasta para notas fiscais e produtos.")
```

Essa função:

- Lê os **dois arquivos .csv** esperados: um com dados de notas fiscais e outro com dados de produtos.
- Padroniza os nomes das colunas com a função padronizar_colunas. (Opcional)
- Valida os dados com os modelos NotaFiscal e ProdutoNotaFiscal (trecho comentado).
- Faz o merge entre os dois DataFrames, unindo pelas colunas em comum (ex: CHAVE_DE_ACESSO).
- Retorna o **DataFrame completo** com todos os dados combinados.

2.2.5. Função criar_agente

```python
# --------- Função para criar o agente LangChain com Gemini -----
def criar_agente(df: pd.DataFrame):
    api_key = os.getenv("GOOGLE_API_KEY")
    if not api_key:
        raise EnvironmentError("Variável de ambiente GOOGLE_API_KEY não definida. Configure sua chave API do Google.")

    llm = ChatGoogleGenerativeAI(
        model="gemini-1.5-flash",
        temperature=0,
        streaming=False
    )

    agente = create_pandas_dataframe_agent(
        llm=llm,
        df=df,
        verbose=True,
        agent_type="openai-tools",
        allow_dangerous_code=True,
        prefix = "Você é um especialista em análise de dados e deve responder em português de forma clara, objetiva e d
    )
    return agente
```

Essa é a função **principal da IA**. Ela:

1. Lê a variável de ambiente GOOGLE_API_KEY (salva no arquivo .env).
2. Cria uma instância do modelo **Gemini** com o LangChain:
3. Usa create_pandas_dataframe_agent para criar um agente que responde perguntas em linguagem natural com base no DataFrame.

Os parâmetros usados:

- llm=llm: IA utilizada.
- df=df: dados da empresa que serão analisados.
- verbose=True: imprime detalhes da execução no console (útil para debug).
- agent_type="openai-tools": agente robusto que usa ferramentas como análise tabular.
- allow_dangerous_code=True: permite o uso de código Python em segundo plano para responder perguntas (com segurança controlada).
- prefix: define o **comportamento do agente**, ou seja, uma instrução de sistema para que ele: “Responda em português, de forma clara, objetiva e direta, **sem escrever código**."

2.3 agente.py – Núcleo da IA

Este script cria uma **interface gráfica da aplicação** que roda na web usando o **Streamlit**. Ele permite que o usuário:

1. Veja os dados carregados.
2. Digite **perguntas em linguagem natural**.
3. Veja a resposta do **agente de IA** (baseado no Gemini + LangChain).

2.3.1 Importações e Configuração

```python
import streamlit as st
from dotenv import load_dotenv
from utils import descompactar_arquivos  # Sua função para extrair zip
from agent import carregar_csvs_de_zip, criar_agente
```

- streamlit: biblioteca que cria a **interface web interativa**.
- dotenv: carrega variáveis do arquivo .env (como chaves de API).
- utils: contém a função descompactar_arquivos() que descompacta o .zip com os CSVs.
- agent: importa as funções que montam os dados e o agente inteligente.

2.3.2 Inicialização da Interface

```python
load_dotenv()

st.title("📊 Agente Inteligente - Análise de Notas Fiscais")
```

- load_dotenv(): carrega as variáveis de ambiente do arquivo .env.
- st.title(...): exibe o título do aplicativo na interface web.

2.3.3 Upload e Extração dos Arquivos

```python
# Extrair arquivos zip para pasta temporária
pasta_csv = descompactar_arquivos()
```

- Esta função **lê o arquivo .zip** enviado pelo usuário (ou um arquivo padrão local) e **extrai os CSVs para uma pasta temporária**.
- Isso permite que os arquivos sejam lidos como `pandas.DataFrame` depois.

2.3.4 Carregamento e Exibição dos Dados

```
# Carregar e validar CSVs, retornar DataFrame completo
df = carregar_csvs_de_zip(pasta_csv)
```

- Lê os dois arquivos CSV extraídos (Notas Fiscais e Produtos).
- Padroniza os nomes das colunas.
- Combina os dados em um único DataFrame.
- Pode incluir validação com Pydantic (se habilitada).

```
st.write("✅ Dados carregados com sucesso:")
st.dataframe(df.head())
```

- Mostra uma tabela com as **primeiras linhas dos dados carregados** na tela do usuário.

```
if df.empty:
    st.error("❌ Erro: DataFrame está vazio. Verifique os arquivos CSV.")
```

- Exibe uma mensagem de erro se os dados estiverem vazios (por falha no upload ou leitura).

2.3.5 Criação do Agente Inteligente

```
# Criar agente com LLM
agente = criar_agente(df)
```

- Usa o Gemini (via LangChain) para criar um agente que **entende o conteúdo da tabela**.
- Esse agente está pronto para **responder perguntas** com base nos dados carregados.

2.3.6 Entrada da Pergunta do Usuário

```python
# Pergunta do usuário
pergunta = st.text_input("Digite sua pergunta sobre os dados:")
```

- Exibe um campo para o usuário escrever perguntas, como:
  - “Qual o fornecedor com maior valor total?”
  - “Quantas notas foram emitidas para o RJ?”

2.3.6 Processamento e Resposta

```python
if pergunta:
    with st.spinner("🧠 Processando a resposta..."):
        try:
            resposta = agente.invoke({"input": pergunta})
            st.success("Resposta:")
            st.write(resposta["output"] if isinstance(resposta, dict) else resposta)
        except StopIteration:
            st.error("Erro interno: modelo retornou resposta incompleta ou vazia.")
        except Exception as e:
            st.error(f"Erro ao processar a pergunta: {e}")
```

- Quando o usuário digita uma pergunta:
  - Mostra um **ícone de carregamento** ("🧠 Processando...")
  - Envia a pergunta para o agente com `invoke({"input": pergunta})`
  - Exibe a **resposta formatada** no painel da interface.

3. RESULTADOS

A imagem abaixo demonstra a interface gráfica onde o usuário poderá interagir com o agente inteligente.

Para fins de teste, foi feito 4 questionamentos para a IA responder:

1) Qual é o fornecedor que teve maior montante recebido?

R: A partir dos dados fornecidos, a FORTBRAS AUTOPECAS S.A. teve o maior montante recebido, com um total de R$ 1746,39.

2) Qual item teve maior volume entregue (em quantidade)?

R:O item com maior volume entregue foi "ARROZ BRANCO (TIPO 1) EMB. CONTENDO 5 KG", com 13 unidades.

3) Qual item teve menor volume entregue (em quantidade)? E em qual estado o produto foi entregue?

R:O item com menor volume entregue foi "PASTILHA NOVA RANGER 2013/ TROLLER T4 15//PN:PD1434", com apenas 1 unidade. Este produto foi entregue no Distrito Federal (DF).

4) Quais foram os produtos recebidos pela Secretaria de Gestão e Ensino em Segurança Pública?

R:Qual foi o produto recebido pela Secretaria de Gestão e Ensino em Segurança Pública?