Ai Agents para engenharia de dados vale a pena?

vale muito a pena aprender a criar AI agents se você quer se destacar na área de engenharia de dados, principalmente com o cenário atual de automação e uso de inteligência artificial. Aqui vão alguns motivos:

1. Automação de tarefas repetitivas

AI agents podem automatizar tarefas comuns de engenharia de dados como:

Monitoramento de pipelines

Validação de qualidade de dados

Correção automática de erros simples

Geração de documentação técnica

2. Integração com ferramentas low-code/no-code

Você pode usar AI agents com ferramentas como KNIME, Power BI, ou até com scripts em Python para automatizar decisões e gerar insights em tempo real.

3. Apoio em processos de ETL/ELT

Agentes podem:

Identificar gargalos no pipeline

Otimizar a execução com base em histórico

Tomar decisões sobre ordenação e particionamento de dados

4. Aplicações avançadas com IoT e dados geoespaciais

No seu caso, que tem interesse em IoT e mapas:

Um AI agent pode aprender padrões de sensores e sugerir ajustes em tempo real

Pode combinar dados geoespaciais com previsão do tempo, uso do solo, etc., para tomar decisões inteligentes

5. Diferencial no mercado

Poucos engenheiros de dados hoje têm experiência prática com agentes de IA — aprender isso pode te colocar à frente da concorrência, principalmente em empresas que trabalham com dados em larga escala ou em tempo real.

Se quiser, posso te sugerir um mini-roadmap de como começar a criar AI agents voltados para engenharia de dados. Quer?

Perfeito! Aqui vai um mini-roadmap prático para você começar a criar AI agents focados em engenharia de dados, com foco em ferramentas modernas, baixo código e integração com seu interesse em IoT e mapas:

---

1. Fundamentos de AI Agents

Objetivo: entender o que são agentes e como funcionam.

Conceitos básicos: agentes reativos vs. proativos, ambiente, sensores e atuadores.

Estude LLM-based agents (tipo LangChain, Semantic Kernel, CrewAI).

Ferramentas: LangChain, Autogen, CrewAI (Python).

---

2. Linguagem e Ambiente

Objetivo: montar o básico para criar scripts e fluxos automatizados.

Fortaleça seu Python (com foco em automação e APIs).

Aprenda a usar APIs RESTful (ex: para conectar com bases externas).

Domine bibliotecas como: pandas, requests, json, datetime.

---

3. Construindo Agentes Simples

Objetivo: criar agentes que executam tarefas específicas.

Exemplo 1: Agente que monitora um banco de dados e avisa sobre dados ausentes.

Exemplo 2: Agente que valida um dataset e envia alertas por e-mail.

Ferramentas: Python + LangChain ou Autogen + KNIME (se quiser no-code).

---

4. Integração com Engenharia de Dados

Objetivo: aplicar agentes nos seus pipelines e automações.

Conectar agentes com:

Pipelines ETL (usando Airflow, KNIME, Azure Data Factory…)

Dashboards (como Power BI, Looker ou Superset)

Automatizar:

Coleta e limpeza de dados

Monitoramento de sensores (IoT)

Enriquecimento com dados geoespaciais (via APIs do Mapbox, Google Maps ou OpenStreetMap)

---

5. Projetos Práticos

Objetivo: ter cases reais para mostrar.

Projeto 1: Agente que lê dados de sensores (simulados), detecta anomalias e avisa via e-mail ou Teams.

Projeto 2: Agente que coleta dados meteorológicos de uma API, cruza com dados geoespaciais, e faz alertas de risco para agricultura.

Projeto 3: Agente que interage com um banco de dados (PostgreSQL ou BigQuery), gera relatórios automáticos e envia por e-mail.

---

6. Aprimoramento

Objetivo: tornar os agentes mais inteligentes.

Aprender a usar RAG (Retrieval-Augmented Generation) para enriquecer a base de conhecimento dos agentes.

Usar vetores e embeddings para permitir busca semântica de dados.

Criar chains de agentes que tomam decisões em etapas (ex: análise + ação + relatório).

---

Se quiser, posso te montar um primeiro projeto simples com passo a passo (ex: "um agente que limpa dados e detecta valores ausentes"). Quer começar com esse?

Show! Vamos montar um projeto simples de AI Agent que limpa dados e detecta valores ausentes. Ideal pra começar e aplicar em qualquer pipeline de engenharia de dados. Ele vai:

1. Carregar um dataset (CSV)

2. Verificar colunas com valores ausentes

3. Gerar um pequeno relatório

4. Tomar uma ação (ex: preencher, dropar ou alertar)

5. Registrar tudo num log

---

Projeto: Data Cleaner Agent (versão básica)

Pré-requisitos:

Python instalado

VS Code ou Jupyter

Bibliotecas: pandas, langchain, openai (ou outro LLM se quiser)

---

Passo 1: Setup

Instale as dependências:

pip install pandas langchain openai

---

Passo 2: Script base

import pandas as pd

from langchain.chat\_models import ChatOpenAI

from langchain.agents import initialize\_agent, Tool

from langchain.agents.agent\_types import AgentType

# Carregar dados

df = pd.read\_csv("seu\_arquivo.csv")

# Função para detectar valores ausentes

def checar\_valores\_nulos():

nulls = df.isnull().sum()

return nulls[nulls > 0].to\_string()

# Função para preencher nulos (simplesmente como exemplo)

def preencher\_nulos():

return df.fillna("VALOR\_PADRÃO")

# Função de log

def gerar\_log():

return f"Linhas: {df.shape[0]}, Colunas: {df.shape[1]}\n\nNulos:\n{checar\_valores\_nulos()}"

# Ferramentas para o agente

tools = [

Tool(name="Checar Nulos", func=checar\_valores\_nulos, description="Verifica colunas com valores ausentes"),

Tool(name="Preencher Nulos", func=preencher\_nulos, description="Preenche os valores ausentes com padrão"),

Tool(name="Gerar Log", func=gerar\_log, description="Gera um relatório simples sobre os dados")

]

# Inicializa o agente

llm = ChatOpenAI(temperature=0, model="gpt-3.5-turbo") # Pode ser outro modelo local ou da Azure

agent = initialize\_agent(tools, llm, agent=AgentType.ZERO\_SHOT\_REACT\_DESCRIPTION)

# Roda o agente com uma pergunta

resposta = agent.run("Verifique os valores ausentes e gere um log.")

print(resposta)

---

Passo 3: Teste

Use um CSV com alguns dados faltantes e veja o agente analisar e tomar uma ação (ele pode ser ajustado pra decidir quando dropar ou preencher).

---

Ideias para evoluir depois:

Fazer o agente perguntar ao usuário o que fazer com nulos (dropar, preencher, alertar)

Enviar o log por e-mail

Integrar com KNIME ou Power BI

Incluir validação de tipos de dados

---

Se quiser, eu posso montar esse mesmo exemplo com KNIME + script Python ou transformar num pipeline com mais etapas. Quer ir pra esse próximo passo?

Beleza! Vamos transformar esse projeto em um pipeline de limpeza de dados no KNIME com integração de script Python — e manter a ideia do AI Agent como cérebro da operação.

---

Pipeline no KNIME: Data Cleaner Agent

Objetivo:

Carregar um CSV

Detectar valores nulos

Decidir o que fazer com eles (com ajuda de um mini agente em Python)

Executar a ação no KNIME (dropar, preencher, etc)

Gerar relatório final

---

1. Estrutura do pipeline KNIME

Você vai montar esse fluxo no KNIME:

1. CSV Reader – para carregar os dados

2. Missing Value Column Filter – para identificar colunas com nulos

3. Python Script (1=>1) – aqui entra o agente em Python

4. Rule Engine / Branch – toma decisões baseadas no output do script

5. Missing Value Node – para limpar conforme a escolha

6. Data Writer / Excel Writer – para salvar o resultado

7. Text Output / Report Node – para gerar o relatório

---

2. Script Python para o AI Agent (dentro do Python Script Node)

Esse script vai receber os dados e decidir o que fazer com os nulos:

import pandas as pd

from langchain.chat\_models import ChatOpenAI

from langchain.agents import initialize\_agent, Tool

from langchain.agents.agent\_types import AgentType

input\_table = input\_table.copy()

def checar\_valores\_nulos():

nulls = input\_table.isnull().sum()

return nulls[nulls > 0].to\_string()

def gerar\_log():

return f"Linhas: {input\_table.shape[0]}, Colunas: {input\_table.shape[1]}\n\nNulos:\n{checar\_valores\_nulos()}"

tools = [

Tool(name="Checar Nulos", func=checar\_valores\_nulos, description="Verifica colunas com valores ausentes"),

Tool(name="Gerar Log", func=gerar\_log, description="Gera relatório sobre os dados")

]

llm = ChatOpenAI(temperature=0, model="gpt-3.5-turbo")

agent = initialize\_agent(tools, llm, agent=AgentType.ZERO\_SHOT\_REACT\_DESCRIPTION)

decisao = agent.run("Verifique os nulos e diga se devemos dropar linhas, preencher ou só gerar um alerta.")

output\_table = pd.DataFrame({'DECISAO': [decisao], 'LOG': [gerar\_log()]})

> Isso gera uma pequena tabela de output com a decisão e um log.

---

3. No KNIME, após o script:

Use o Rule Engine para interpretar a DECISAO (ex: se contém "dropar", vá por um caminho; se "preencher", outro).

Aplique os nós correspondentes:

Row Filter + Missing Value para aplicar as ações

Data Writer para salvar os dados limpos

Text Output para salvar o relatório do log

---

4. Melhorias possíveis

Integrar com email (usando Email Sender Node) para enviar o log

Rodar em batch com múltiplos arquivos

Incluir classificação de tipo de dado e outliers

---

Se quiser, posso montar o fluxo visual com nomes de cada node e configurações sugeridas (pra você montar mais rápido no KNIME). Quer?

Perfeito! Aqui está o fluxo visual detalhado para montar o pipeline no KNIME, passo a passo com nomes dos nodes, conexões e sugestões de configuração:

---

Fluxo KNIME: Data Cleaner Agent

1. CSV Reader

Nome do Node: Ler Dados CSV

Configuração: Escolha seu arquivo .csv

Output: dados\_brutos

---

2. Python Script (1⇒1)

Nome do Node: Agente Python

Configuração:

Use o script que te passei antes com langchain

Certifique-se de que a execução do Python no KNIME esteja configurada

Output esperado: tabela com colunas DECISAO e LOG

---

3. Table Row to Variable

Nome do Node: Decisão para Variável

Conecta ao output do Python Script

Configuração: Converte as colunas DECISAO e LOG em variáveis

Objetivo: Para que o próximo node tome decisões com base nessas variáveis

---

4. Missing Value

Nome do Node: Tratamento de Nulos

Conecta ao output do CSV Reader

Configuração:

Se a variável DECISAO contiver "preencher", configure para preencher com média, valor padrão, etc.

Use a aba Flow Variables para condicionar essa configuração

---

5. Row Filter

Nome do Node: Dropar Linhas com Nulos

Conecta ao CSV Reader

Configuração:

Filtro baseado em valores ausentes

Use uma regra para remover linhas com missing baseado na variável DECISAO

Pode ser condicional (ativado só se DECISAO contiver "dropar")

---

6. CASE Switch (Flow Variable Value)

Nome do Node: Switch de Ação

Configuração:

Usa a variável DECISAO

Se "dropar": segue para Row Filter

Se "preencher": segue para Missing Value

Senão: segue direto pro Writer (sem limpeza)

---

7. Data Writer (ou Excel Writer)

Nome do Node: Salvar Dados Limpos

Configuração: Escolha formato e destino

---

8. Variable to Table Row

Nome do Node: Relatório em Tabela

Conecta à variável LOG

Transforma o log em uma linha de tabela para salvar

---

9. CSV Writer

Nome do Node: Salvar Relatório

Conecta ao node anterior

Salva o log da análise/ação

---

Fluxo final esperado:

CSV Reader → Python Script → Table Row to Variable → CASE Switch

↘ ↘

Relatório → CSV Writer Tratamento ou Dropar → Writer

---

Se quiser, posso te enviar um template .knwf (fluxo do KNIME) ou te guiar com prints de como montar. Quer um desses?