Challenge Técnico - Parte 1

Implementación de Agente Autónomo para Clusterización

Objetivo:
Construir un *agente automático* que reciba un conjunto de datos tabulares y realice una clusterización adecuada, siguiendo criterios de autonomía mínima esperada.

Entregables:

- Script principal: Código en Python que implemente el agente.
- Archivo README.md que explique (máximo 1 carilla):
    - Cómo correrlo.
    - Decisiones de diseño principales.
    - Supuestos asumidos.
- Visualizar clusters si el dataset tiene 2 o 3 dimensiones principales.

Requerimientos Técnicos:

- Entrada:
    1. Archivo `.csv` provisto con datos no etiquetados.
- Tareas mínimas del agente:
    1. Cargar los datos de entrada.
    2. Preprocesarlos automáticamente:
        - Imputación de valores nulos.

        - Normalización o estandarización según corresponda.

    3. Determinar automáticamente un número razonable de clusters (por ejemplo, usando Elbow Method, Silhouette Score u otra técnica).

    4. Ejecutar una técnica de clustering adecuada (K-Means, DBSCAN, Agglomerative, etc.).

    5. Devolver el dataset con la columna de cluster asignada.

    6. Guardar el resultado en un nuevo archivo `.csv`.

Challenge Técnico - Parte 2

Implementación de Agente de Clusterización usando LangChain

Objetivo:
Construir un agente utilizando la librería **LangChain** que permita clusterizar el dataset recibido en la Parte 1, respondiendo a instrucciones en lenguaje natural.

Requerimientos:

- El agente debe ser capaz de:
    - Recibir una orden textual (ejemplo: "clusterizá el dataset iris_data_challenge.csv").
    - Ejecutar internamente una función de clusterización.
    - Guardar el resultado con los clusters asignados en un nuevo `.csv`.
    - Responder con un resumen simple del proceso (cantidad de clusters, características generales, etc.).

Restricciones obligatorias:

- El flujo debe estar estructurado utilizando **LangChain**.

Cómo simular el LLM:
Utilizar una clase simple que herede de `BaseLanguageModel` y que interprete las instrucciones del usuario de manera local.

Nota: Se provee un ejemplo básico de FakeLLM para ilustrar la idea general. Se puede y debe ajustar la implementación para asegurar compatibilidad completa con el agente y las tools en LangChain actual.

```
from langchain_core.language_models import BaseLanguageModel

class FakeSimpleLLM(BaseLanguageModel):
    def _call(self, messages, **kwargs):
        user_message = messages[-1]['content'].lower()
        if "clusteriz" in user_message:
            return
'{"tool_calls":[{"name":"clusterizar_dataset","args":{"path_csv":"ir
is_data_challenge.csv"}}]}'
        else:
            return "No entiendo la instrucción."

    @property
    def _llm_type(self):
        return "fake-simple-llm"
```

Este `FakeSimpleLLM` debe integrarse al agente para tomar decisiones sobre qué herramienta (`Tool`) usar.

Framework permitido:

- `langchain` versión estable.
- Python 3.8 o superior.

- Librerías de apoyo como `pandas`, `scikit-learn`, `numpy` para operaciones de datos.

Entregables:

- Script Python funcional (`agent_cluster.py` o similar).
- Archivo README.md explicativo que contenga:
    - Cómo correr el agente localmente.
    - Qué supuestos tomaron.
    - Cómo interpretar la respuesta del agente.
    - Incluir un pequeño diagrama de flujo que represente cómo interactúan Usuario ➔ Agente ➔ Herramienta.