Deep Learning para Business Analytics

Día 13 · Tema 5 — Tu primer agente con Claude Code

Bucle de percepción, decisión, acción.

Eduardo C. Garrido-Merchán

Profesor Propio Adjunto, Métodos Cuantitativos, ICADE

ecgarrido@comillas.edu

Madrid, Día 13

ejemplos del día

Lo que vamos a cubrir hoy.

1. Qué es un agente

2. Claude Code en práctica

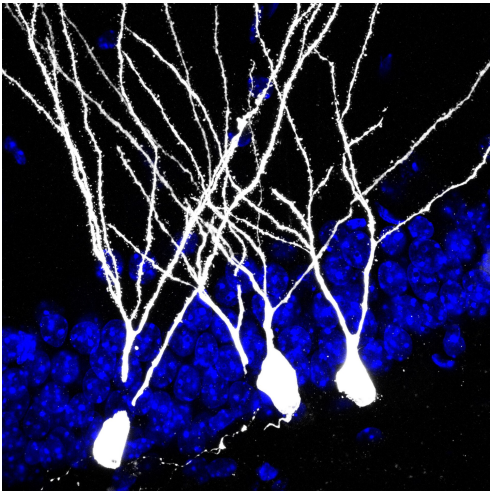
3. El CLAUDE.md: *system prompt* con jerarquía

4. Una sesión real, paso a paso

Al final del día: tienes Claude Code (o opencode + Ollama) operativo, sabes escribir un `CLAUDE.md` útil y lanzas tu primer agente sobre un repo real.

Parte 1 / 4

Qué es un agente



Tres niveles de automatización: script, chatbot, agente 🤖.

Tres niveles de automatización con IA.

Script

Tareas predefinidas y rígidas.
No decide nada.

Chatbot

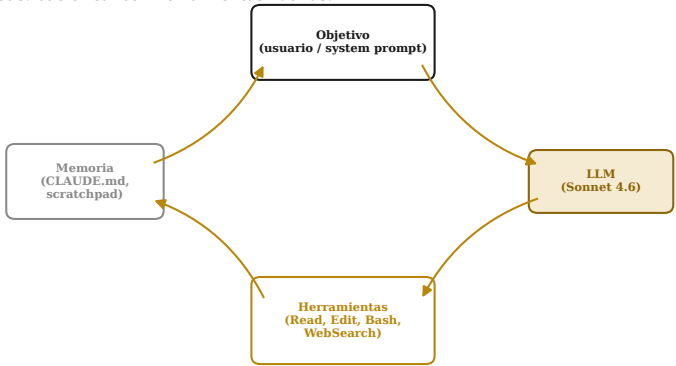
Conversación, sin acciones.
Decide qué decir, no qué hacer.

Agente

Decide qué herramientas usar
y ejecuta acciones reales.

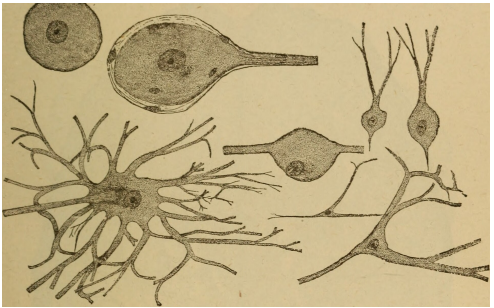
Bucle del agente 🤖 con herramientas 🔧 reales.

Claude Code: bucle real con herramientas nativas.



Parte 2 / 4

Claude Code en práctica



Instalación en cinco minutos.

Cloud (Claude Code). `npm install -g @anthropic-ai/claude-code`. Necesitas cuenta y crédito en Anthropic.

Libre (opencode + Ollama). `curl -fsSL https://opencode.ai/install | bash` y `opencode -model ollama/qwen2.5-coder:7b`. Cero coste por sesión.

Primer comando 💬**.** En el repo del proyecto, ejecuta `claude` (o `opencode`). El agente 🤖 se conecta, lee `CLAUDE.md` y queda a la espera.

Modos de ejecución. *Interactivo* (conversación), *no interactivo* (`claude -print ...` para scripts), *plan* (`/plan` antes de aplicar cambios).

Idea clave. Probadlo hoy con un comando: "lee este repo y dime qué hace en 4 frases".

Herramientas 🔧 nativas: lo que el agente 🤖 puede hacer por ti.

Herramientas que Claude Code expone al agente por defecto.

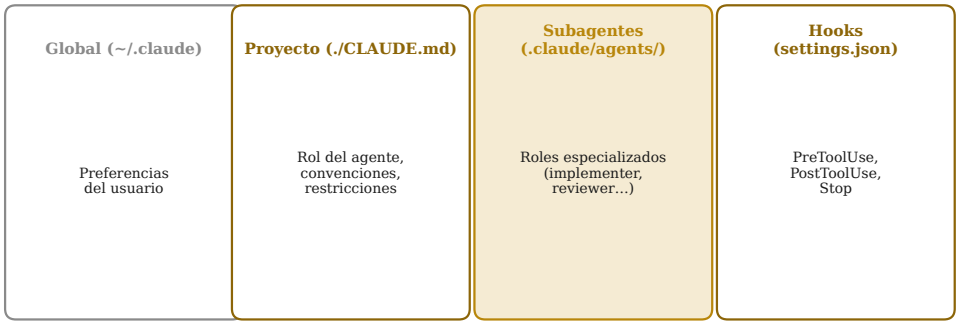
Read Leer archivos del repo	**Edit / Write** Modificar o crear archivos	**Bash** Ejecutar comandos del shell	**Glob / Grep** Buscar por patrón
WebSearch Buscar en internet	**Agent** Lanzar subagente	**TaskCreate** Crear sub-tareas	

Parte 3 / 4

El CLAUDE.md: *system prompt* con jerarquía

Cuatro niveles de instrucción ⚙️ para Claude Code.

Cuatro niveles de instrucción para Claude Code.



Cómo escribir un CLAUDE.md ⚙️ útil, en cinco bloques.

1. **Rol.** ."Eres el asistente de ingeniería de un proyecto de forecasting para Mercadona."
2. **Convenciones.** Código Python tipo-hint, tests con pytest, formateo black, sin docstrings largos."
3. **Restricciones.** "No instales librerías nuevas sin pedir permiso. No toques `docker-compose.yml`."
4. **Contexto** 💬**.** ."El dataset está en `data/online_retail.parquet`, el modelo base se entrena con `scripts/train.py`."
5. **Política de IA.** "Declara siempre en el commit cuando uses IA; conserva los prompts en `prompts/`."

Idea clave. Un CLAUDE.md de 200 líneas en el proyecto vale más que cualquier prompt ingenioso aislado.

Concepto → aplicación: qué hacer hoy con un agente 🤖.

Tarea	Caso	Entradas	KPI
Refactor automatizado	Migrar 200 archivos Py 2.7 a Py 3 (BBVA legacy).	Repo + objetivo 🎯.	Tests OK / día.
Documentación	Generar README de un módulo nuevo.	Código fuente.	Tiempo a entrega.
Revisión de PR	Revisar PR de un compañero antes de merge.	PR diff.	Issues detectados.
Análisis exploratorio	Cargar CSV y producir informe con gráficos.	CSV de negocio.	Tiempo a insight.
Triage de tickets	Clasificar y responder borradores en Zendesk.	Lista tickets.	% auto-resueltos.

Idea clave. Cualquier tarea repetitiva que combine código, datos y texto es candidata.

Parte 4 / 4

Una sesión real, paso a paso

Sesión real de Claude Code 🧠: 7 turnos para un refactor con retry.

Ejemplo de sesión a mano: refactor con retry, 7 turnos del agente.

Usuario	Encuentra todas las funciones de mi proyecto que llaman a la API de Anthropic y añádeles retry.
Agente -> Grep	Busca 'anthropic.messages.create' en src/
Resultado	3 archivos encontrados: utils/llm.py, agents/main.py, tools/summary.py
Agente -> Read	Lee los 3 archivos para entender el patrón actual
Agente -> Edit	Inserta un decorador @retry(stop=stop_after_attempt(3)) en cada función
Agente -> Bash	Lanza pytest -k retry
Agente	Reporte final: 3 funciones modificadas, 1 test añadido, pytest verde.

Buenas prácticas de uso desde día uno.

1. Pide plan antes de aplicar 🎯. `/plan` fuerza al agente a contar qué va a hacer antes de tocar archivos. Revisas y apruebas.

2. Commitea el CLAUDE.md ⚙️. Es parte del repo. Compañeros y agente comparten contexto.

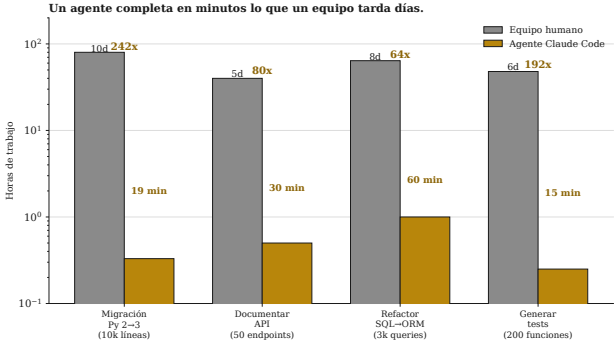
3. Hook PreToolUse para protecciones. Evita que Bash haga `rm -rf` sin confirmación. Configurable en `settings.json`.

4. Audita trazas. El agente deja registro de cada comando ejecutado. Cuando algo sale mal, lo reproduces.

5. Empieza pequeño. La primera semana, tareas de 30 minutos. Cuando confías, escalas a refactors completos.

Idea clave. Un agente es un junior nuevo: necesita instrucciones claras, revisión y barandillas.

Un agente refactoriza 10.000 lineas en 20 minutos 🚀.



Dato. Tareas de migracion de codigo que un equipo tarda dias, un agente las completa en minutos.

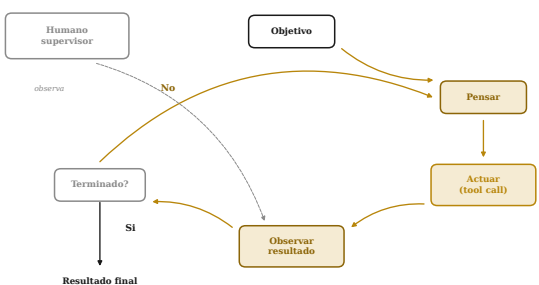
Clave 🤖**.** Los agentes no son chatbots. Ejecutan acciones: leen archivos, escriben codigo, corren tests, iteran. El humano supervisa.

Ejemplo real. Migracion Python 2.7 a 3 en un banco: 10.000 lineas, 20 minutos, 0 errores post-review.

IDEA CLAVE. Cualquier tarea repetitiva con codigo, datos y texto es candidata a agente.

Un agente es un becario muy rapido con acceso a Google 🧠.

El bucle de un agente: piensa, actua, observa, repite.



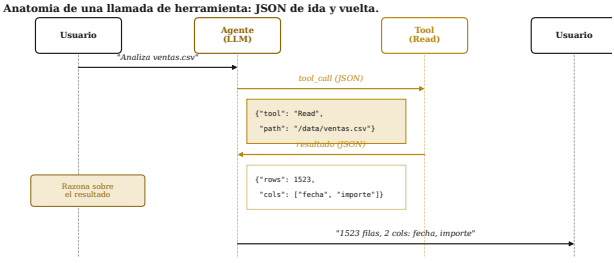
Bucle 🔧. El patron es simple: piensa, actua, observa, repite. La magia esta en que el LLM puede razonar sobre el resultado y cambiar de estrategia.

Supervisor. El humano observa el proceso y puede intervenir en cualquier momento. El agente no tiene autonomia total.

Terminacion. El agente decide cuando ha terminado o cuando necesita mas informacion del usuario.

Idea clave. Piensa – Actua – Observa – Repite: cuatro palabras para recordar el patron.

Anatomia de una llamada de herramienta: JSON de ida y vuelta 🔧.



Paso 1. El usuario pide analizar un CSV. El agente decide que herramienta necesita.

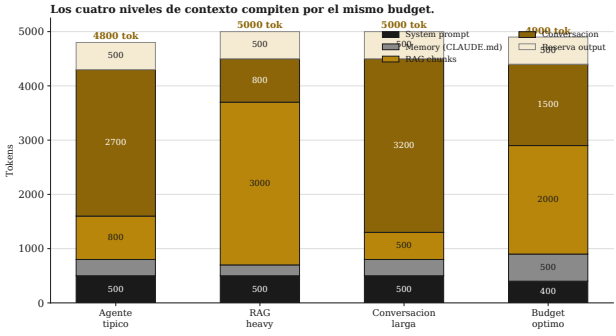
Paso 2 ⚙️. Emite un JSON estructurado: `{tool:Read, path:/data/ventas.csv}`

Paso 3. La herramienta devuelve el resultado en JSON. El agente lo lee y decide el siguiente paso.

Paso 4. El agente razona sobre el resultado: cuantas filas, que columnas, que patron seguir.

Idea clave. JSON estructurado convierte al agente en una pieza testeable, no en una caja negra.

Los cuatro niveles de contexto: system, memory, RAG, conversacion 📚.



Restriccion 💬**.** Todos los niveles compiten por el mismo budget de tokens:

$$\sum_i t_i \;\leq\; B$$

Trade-off. Mas chunks RAG = menos historial de conversacion. Mas system prompt = menos espacio para el usuario.

Budget tipico. Con un modelo de 200k tokens, un agente bien configurado usa 4.800 para contexto y reserva el resto para razonamiento largo.

Idea clave. Gestionar el contexto es la primera decision de diseno de cualquier agente.

Lo que el alumno se lleva a casa de este día.

Concepto 1. Un agente 🤖 cierra el bucle *objetivo* → *razonar* → *actuar* → *observar* con herramientas reales.

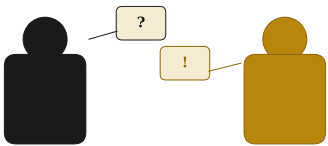
Concepto 2. Claude Code (cloud) y opencode + Ollama (libre) son equivalentes en capacidad; difieren en coste y calidad.

Concepto 3. El CLAUDE.md ⚙️ (rol + convenciones + restricciones + contexto + política IA) determina la calidad del agente.

Para la práctica. Crear un CLAUDE.md para un proyecto propio y pedir al agente que documente todo el repo en una sesión.

Próximo día. Día 14 (Tema 5.2): subagentes, MCP, orquestador-trabajador.

Pausa para debatir: ¿Dejarías a un agente tocar tu repo sin revisión?



La pregunta. Claude Code en modo no interactivo ejecuta cambios sin que tú apruebes paso a paso. ¿Lo permites en tu equipo?

Posición A. Sí, con tests verdes y rollback automático.

Posición B. No, sin revisión humana antes de cada `git push`.

Reglas. 5 minutos, dos turnos de palabra por bando, móviles boca abajo. Yo modero, no participo.

Hasta el Día 14.

Multi-agente y MCP.

Eduardo C. Garrido-Merchán

ecgarrido@comillas.edu