

# AI ACADEMY

## Applicare l'Intelligenza Artificiale nello sviluppo software

# AI ACADEMY

## Cursor / Copilot Productivity 30/06/2025

# INTRODUZIONE DELL'ISTRUTTORE

*Tamas Szakacs*

## *Formazione*

- Laureato come programmatore matematico
- MBA in management

## *Principali esperienze di lavoro*

- Amministratore di sistemi UNIX
- Oracle DBA
- Sviluppatore di Java, Python e di Oracle PL/SQL
- Architetto (solution, enterprise, security, data)
- Ricercatore tecnologico e interdisciplinare di IA

## Dedicato alla formazione continua

- Teorie, modelli, framework IA
- Ricerche IA
- Strategie aziendali
- Trasformazione digitale
- Formazione professionale

*email: [tamas.szakacs@proficegroup.it](mailto:tamas.szakacs@proficegroup.it)*

# MOTIVI E RIASSUNTO DEL CORSO

L'**Intelligenza Artificiale (AI)** è oggi il motore dell'innovazione in ogni settore, grazie alla sua capacità di analizzare dati, automatizzare processi e generare nuove soluzioni. Questo corso offre una panoramica completa e pratica sullo sviluppo di applicazioni AI moderne, guidando i partecipanti dall'ideazione al rilascio in produzione.

Attraverso una **combinazione di teoria chiara ed esercitazioni pratiche**, saranno affrontate le tecniche e gli strumenti più attuali: **machine learning, deep learning, reti neurali, Large Language Models (LLM), Transformers, Retrieval Augmented Generation (RAG)** e progettazione di agenti AI.

Le competenze acquisite saranno applicate in progetti concreti, dallo sviluppo di chatbot all'integrazione di modelli generativi, fino al deploy di soluzioni AI in ambienti reali e collaborativi.

Il percorso è pensato per chi vuole imparare a progettare, valutare e integrare sistemi AI di nuova generazione, con particolare attenzione alle best practice di programmazione, collaborazione in team, sicurezza, valutazione delle performance ed etica dell'AI.

**DURATA: 17 GIORNI**

# OBIETTIVI

Il percorso formativo è progettato per **giovani consulenti junior**, con una conoscenza base di programmazione, che stanno iniziando un percorso professionale nel settore AI.

**L'obiettivo centrale è fornire una panoramica pratica, completa e operativa sull'intelligenza artificiale moderna**, guidando ogni partecipante attraverso tutte le fasi fondamentali.



# OBIETTIVI

- Allineare conoscenze AI, ML, DL di tutti i partecipanti
- Saper usare e orchestrare modelli LLM (closed e open-weight)
- Costruire pipeline RAG complete (retrieval-augmented generation)
- Progettare agenti AI semplici con strumenti moderni (LangChain, tool calling)
- Capire principi di valutazione, robustezza e sicurezza dei sistemi GenA
- Migliorare la produttività come sviluppatori usando tool GenAI-driven
- Padroneggiare best practice di sviluppo, versioning e deploy AI
- Introdurre i fondamenti di Graph Data Science e Knowledge Graph
- Ottenere capacità di valutazione dei modelli e metriche
- Comprensione dell'etica e dei bias nei modelli di intelligenza artificiale
- Approfondire le normative di riferimento: AI Act, compliance e governance AI

Il corso è **estremamente pratico** (circa il 40% del tempo in esercitazioni hands-on, notebook, challenge e hackathon), con l'utilizzo di Google Colab, GitHub, e tutti gli strumenti necessari per lavorare su progetti reali e simulati.



# STRUTTURA DELLE GIORNATE – PROGRAMMA BREVE

**Tutte le giornate sono di 8 ore (9:00-17:00), con 1 ora di pausa suddivisa (mezz'ora pranzo, due pause da 15 min durante la mattina e il pomeriggio).**

La progettazione sintetica delle giornate:

Giorno	Tema	Breve descrizione
1	Git & Python clean-code	Collaborazione su progetti reali, versionamento, codice pulito e testato
2	Machine Learning Supervised	Modelli supervisionati per predizione e classificazione
3	Machine Learning Unsupervised	Clustering, riduzione dimensionale, scoperta di pattern
4	Prompt Engineering avanzato	Scrivere e valutare prompt efficaci per modelli generativi
5	LLM via API (multi-vendor)	Uso pratico di modelli LLM via API, autenticazione, deployment
6	Come costruire un RAG	Pipeline end-to-end per Retrieval-Augmented Generation
7	Tool-calling & Agent design	Progettare agenti AI che usano strumenti esterni
8	Hackathon: Agentic RAG	Challenge pratica: chatbot agentic RAG in team

# STRUTTURA DELLE GIORNATE – PROGRAMMA BREVE

**Tutte le giornate sono di 8 ore (9:00-17:00), con 1 ora di pausa suddivisa (mezz'ora pranzo, due pause da 15 min durante la mattina e il pomeriggio).**

La progettazione sintetica delle giornate:

Giorno	Tema	Breve descrizione
9	Hackathon: Rapid Prototyping	Da prototipo a web-app con Streamlit e GitHub
10	AI Productivity Tools	Workflow con IDE AI-powered, automazione e refactoring assistito
11	Docker & HF Spaces Deploy	Deployment di app GenAI containerizzate o su HuggingFace Spaces
12	AI Act & ISO 42001 Compliance	Fondamenti di compliance e governance AI
13	Knowledge Base & Graph Data Science	Introduzione a Knowledge Graph e query con Neo4j
14	Model evaluation & osservabilità	Metriche avanzate, explainability, strumenti di valutazione
15	AI bias, fairness ed etica applicata	Analisi dei rischi, metriche e mitigazione dei bias
16-17	Project Work & Challenge finale	Lavoro a gruppi, POC/POD, presentazione e votazione progetti



# METODOLOGIA DEL CORSO

## 1. Approccio introduttivo ma avanzato

Il corso è introduttivo nei concetti base dell'AI applicata allo sviluppo, ma affronta anche tecnologie, modelli e soluzioni avanzate per garantire un apprendimento completo.

## 2. Linguaggio adattato

Il linguaggio utilizzato è chiaro e adattato agli studenti, con spiegazioni dettagliate dei termini tecnici per favorirne la comprensione e l'apprendimento graduale.

## 3. Esercizi pratici

Gli esercizi pratici sono interamente svolti online tramite piattaforme come Google Colab o notebook Python, eliminando la necessità di installare software sul proprio computer.

## 4. Supporto interattivo

È possibile porre domande in qualsiasi momento durante le lezioni o successivamente via email per garantire una piena comprensione del materiale trattato.

# NOTA

Il corso segue un **approccio laboratoriale**: ogni giornata combina sessioni teoriche chiare e concrete con molte attività pratiche supervisionate, per sviluppare *competenze reali* immediatamente applicabili.

I partecipanti lavoreranno spesso in gruppo, useranno notebook in Colab e versioneranno codice su GitHub, vivendo una vera simulazione del lavoro in azienda AI.

**Nessun prerequisito avanzato richiesto**: si partirà dagli strumenti e flussi fondamentali, con una crescita graduale verso le tecniche più attuali e richieste dal mercato.

# ORARIO TIPICO DELLE GIORNATE

Orario	Attività	Dettaglio
09:00 – 09:30	Teoria introduttiva	Concetti chiave, schema della giornata
09:30 – 10:30	Live coding + esercizio guidato	Esempio pratico, notebook Colab
10:30 – 10:45	<i>Pausa breve</i>	
10:45 – 11:30	Approfondimento teorico	Tecniche, best practice
11:30 – 12:30	Esercizio hands-on individuale	Sviluppo o completamento di codice
12:30 – 13:00	Discussione soluzioni + Q&A	Condivisione e correzione
13:00 – 14:00	<i>Pausa pranzo</i>	
13:30 – 14:15	Teoria avanzata / nuovi tools	Nuovi strumenti, pattern, demo
14:15 – 15:30	Esercizio a gruppi / challenge	Lavoro di squadra su task reale
15:30 – 15:45	<i>Pausa breve</i>	
15:45 – 16:30	Sommario teorico e pratico	
16:30 – 17:00	Discussioni, feedback	Riepilogo, best practice, domande aperte

# DOMANDE?

## Cominciamo!

# OBIETTIVI DELLA GIORNATA

## Obiettivi della giornata

- Comprendere i vantaggi degli IDE AI-powered nella produttività quotidiana del programmatore.
- Saper distinguere tra le diverse modalità di AI Assistant: completamento automatico, chat, refactoring, test generation.
- Imparare a utilizzare Cursor e GitHub Copilot per aumentare efficienza e qualità del codice.
- Applicare tecniche di pair-programming virtuale e code review AI-based.
- Automatizzare attività ripetitive: docstring, unit test, refactor, regex rewrite.
- Analizzare metriche di produttività (KPI) per valutare l'impatto reale degli strumenti AI.
- Integrare l'IDE con sistemi di controllo versione (Git) per il lavoro collaborativo.

# SCOPO DEL PROGETTO

**Creare il chatbot con agentic RAG usando rapid prototyping con Streamlit**

## **Capacità o caratteristiche richieste**

- ☐ Anonimizzazione dei dati sensibili (min. nomi e IBAN) usando NER e regex
- ☐ Gestione di documenti con tecniche RAG nei prompt
- ☐ Uso di embedding, chunking, vettorizzazione dei documenti e prompt (similarity search su documenti)
- ☐ Autonomia / agente per gestire files e risposte (con creazione di files, per esempio risposta mail in un file)
- ☐ Chat e gestione di una singola sessione con la sequenza dei messaggi con Langchain

## **Visualizzazioni richieste con Streamlit**

- ☐ Chat input e output in formato history (tipo ChatGPT)
- ☐ Bottone per caricare documenti (da gestire con autonomia dal modello)
- ☐ Elenco dei file caricati
- ☐ (Opzionale) Gestione di diverse sessioni
- ☐ (Opzionale) Gestione di memoria, a disposizione di ogni sessione



# DELIVERABLES E DOCUMENTI MINIMI PER IL PROGETTO Prof/ce

Per consegnare i lavori bisogna preparare questi deliverable:

- **Risposta progettuale**  
Breve descrizione di come il team ha risolto il problema proposto, con riferimento alle scelte principali.
- **Schema architetturale**  
Schema essenziale (testuale o diagramma semplice) dell'architettura della soluzione e dei principali componenti (modello, agenti, pipeline, API, ecc.).
- **Codice sorgente**  
Tutto il codice sviluppato, con commenti chiari e autoesplicativi (documentazione del codice generata direttamente dai commenti).
- **Dataset di test**  
File di dati usati per le prove e le demo, rappresentativi dei casi d'uso.
- **Metodo di test**  
Breve descrizione del metodo di test applicato: quali casi, quali dati, come è stato valutato il funzionamento.
- **(Opzionale – da aggiungere dopo lezione su etica/EU AI Act)**  
Eventuali note su conformità etica, privacy e regole AI.

# IDE AI-POWERED

- Un IDE AI-powered è un ambiente di sviluppo integrato che utilizza l'intelligenza artificiale per assistere il programmatore.
- Esempi: **Cursor, GitHub Copilot, Visual Studio Code (con plugin AI), JetBrains AI.**
- L'AI analizza il codice mentre scrivi e offre suggerimenti, completamenti, refactoring e automazione di task ripetitivi.
- L'obiettivo: velocizzare il lavoro, ridurre errori e aumentare la produttività.

## Perché usare un IDE AI-powered?

- Migliora la velocità di scrittura del codice (autocompletamento intelligente).
- Aiuta a scoprire errori o incongruenze in tempo reale.
- Genera documentazione e test in automatico.
- Semplifica il refactoring di codice esistente.
- Favorisce l'apprendimento: suggerisce best practice e soluzioni che potresti non conoscere.

# IDE AI-POWERED

## Funzionalità principali

- **Code Completion:** suggerimenti e completamento automatico di funzioni, variabili, interi blocchi.
- **AI Chat:** dialogo con un assistente AI integrato per domande sul codice, spiegazioni, debug.
- **Refactoring Assistito:** modifica strutturale del codice (rinomina, estrai funzione, semplifica).
- **Test Generation:** generazione automatica di unit test o stub.
- **Documentazione Automatica:** creazione di docstring e commenti su richiesta.

## Dove funzionano e come si avviano

Gli IDE AI-powered sono disponibili come:

- Estensioni/plugin per editor noti (VS Code, PyCharm, JetBrains, ecc.)
- Applicazioni desktop (es. Cursor).
- Integrazioni cloud (GitHub Codespaces).

**Avvio:** basta installare il plugin/estensione e autenticarsi (spesso con account GitHub o Microsoft).

**Nota:** alcuni strumenti richiedono abbonamento o trial, altri hanno funzionalità gratuite limitate.

# PRINCIPALI PLUGIN AI PER VS CODE

Nome Plugin	Funzione principale	Note
<b>GitHub Copilot</b>	Suggerimenti di codice, completamento, chat AI	Richiede abbonamento
<b>GitHub Copilot Chat</b>	Chat conversazionale dentro VS Code	Incluso con Copilot
<b>Tabnine</b>	Autocompletamento predittivo con AI	Gratuito base, versione PRO
<b>Amazon CodeWhisperer</b>	Suggerimenti di codice (multi-linguaggio)	Gratuito, login AWS
<b>CodeGPT</b>	Chatbot e completamento (usa OpenAI, Azure, ecc.)	Richiede API key
<b>Continue</b>	Assistente AI open source, suggerimenti e refactoring	Gratuito, vari modelli supportati
<b>Google AI Studio (Preview)</b>	Chat, completamento, spiegazioni	In beta/test, login Google
<b>Kite</b>	Autocompletamento AI (supporto limitato dal 2023)	Non più aggiornato

# CURSOR

**Cursor** è un IDE AI-powered di nuova generazione, costruito a partire da Visual Studio Code ma con un'**integrazione nativa di AI** ancora più avanzata.

Sfrutta i modelli linguistici di OpenAI (GPT-4, GPT-4o, Claude, ecc.) per aiutare nello sviluppo software con:

- **Suggerimenti di codice** mentre scrivi (come Copilot, ma ancora più avanzato).
- **Chat AI integrata**: puoi chiedere chiarimenti, spiegazioni, refactoring, generazione di test, docstring, ecc.
- **Refactoring automatico**: AI può riscrivere o migliorare intere parti di codice su richiesta.
- **Ricerca semantica nei progetti**: trova rapidamente funzioni, variabili, commenti in base al significato.
- **Gestione automatica di branch, merge, e Git**.

Funziona su **Windows, macOS e Linux**.

Richiede una **registrazione** e un account per usare le funzioni AI complete.

## Scopo:

Aumentare la produttività, ridurre errori, e rendere più facile lavorare anche su progetti di grandi dimensioni grazie al supporto intelligente dell'AI.

# CURSOR

## Data Sharing

Help improve Cursor for everyone

By default, Cursor learns from code to help you get the best experience possible.

- You're in control.** Turn off anytime in Settings → Privacy.
- Data sharing is off the first day.** After one day of use, Cursor stores and learns from your prompts, codebase, edit history, and other usage data to improve the product.


[More on Privacy Policy and Security](#)

☐

I'm fine with Cursor learning from my code or I'll turn it off in Settings


Continue

## Review Settings

 **Language for AI**


Set language for AI responses

English ▾

 **Open from Terminal**


``cursor`` command ▾

Install

 Do you want to install the recommended 'Python' extension from ms-python for the Python language?

Show Recommendations

Install

 Want to launch Cursor from the command line? You can add the ``code`` or ``cursor`` commands now, or later with `Ctrl+Shift+P` and typing `'install code command'`.

Never show again

Install 'cursor'

Install 'code'



# CURSOR VS. COPILOT

Funzione	Cursor	GitHub Copilot (VS Code)
Completamento codice	Sì (molto avanzato, GPT-4/GPT-4o)	Sì (GPT-3.5/GPT-4 Turbo)
Chat AI integrata	Sì (nativa e contestuale)	Sì (con estensione Copilot Chat)
Refactoring automatico	Sì (AI può riscrivere parti di codice)	Limitato, non nativo
Spiegazione codice (“Explain”)	Sì (AI, anche legacy code)	Sì
Generazione docstring	Sì	Sì
Generazione test unitari	Sì (stub test auto)	Sì (meno avanzato)
Ricerca semantica	Sì (full project, AI-powered)	Limitata, ricerca classica
Gestione Git avanzata	Sì (branch, merge, AI assistiti)	No (solo funzioni standard di VSCode)
Modelli AI supportati	GPT-4, GPT-4o, Claude, custom	Solo OpenAI (Copilot)
Interfaccia	Basata su VS Code, ma dedicata	VS Code con estensioni Copilot
Requisiti	Iscrizione Cursor, abbonamento AI	Abbonamento Copilot
Sistema operativo	Win, Mac, Linux	Win, Mac, Linux

# PAIR-PROGRAMMING VIRTUALE

Il pair-programming virtuale è una pratica in cui due sviluppatori collaborano sullo stesso codice in tempo reale, anche se lavorano da remoto.

Obiettivi principali:

- Migliorare la qualità del codice e ridurre errori.
- Favorire la condivisione della conoscenza all'interno del team.
- Accelerare il problem-solving, alternando i ruoli di “driver” (scrive il codice) e “navigator” (dà indicazioni, revisiona, suggerisce strategie).

**Strumenti tipici:**

- Editor collaborativi (VS Code Live Share, Cursor sharing, Google Colab).
- Videochiamata e schermo condiviso (Zoom, Teams, Google Meet).
- Chat AI come supporto aggiuntivo (Copilot Chat, Cursor AI).

# TECNICHE E BUONE PRATICHE

- **Driver & Navigator:** alternarsi nei ruoli per mantenere attenzione e scambio attivo.
- **Time-boxing:** lavorare a sessioni brevi e intervallate (es. 25-30 minuti), con brevi pause.
- **Codice condiviso in tempo reale:** utilizzare strumenti che permettono di vedere e modificare il codice simultaneamente.
- **Uso di AI come terzo “pair”:** coinvolgere l’assistente AI per suggerimenti, refactoring, risoluzione di bug o spiegazioni.
- **Chiarezza nella comunicazione:** discutere prima l’obiettivo, usare commenti e TODO per orientarsi, annotare le decisioni prese.
- **Debrief finale:** riassumere insieme cosa è stato fatto, individuare problemi, fissare i prossimi step.

# MODALITÀ COMPLETION VS CHAT

## Modalità Completion vs Chat

### Completion (“Completamento”)

- L’AI propone porzioni di codice automaticamente mentre scrivi.
- Ideale per suggerire una riga, una funzione, un commento o continuare un blocco.
- Non richiede domande esplicite: l’AI si basa solo sul contesto immediato.
- Utilizzo tipico: velocizzare la scrittura, completare rapidamente codice ripetitivo o boilerplate.

**Esempio:** mentre digiti `def calcola_totale()`, il modello propone l’intera funzione.

### Chat

- L’AI risponde a domande esplicite poste dallo sviluppatore.
- Supporta dialoghi multi-turno: puoi chiedere spiegazioni, refactoring, generazione di codice, debug, suggerimenti su librerie, ecc.
- Il contesto può essere più ampio (file, progetto intero, codice selezionato).
- Utilizzo tipico: approfondire, risolvere dubbi, ricevere consigli mirati.

**Esempio:** “Spiega come funziona questa funzione”, “Genera i test per questa classe”, “Trova bug in questo codice”.

# REFACTOR-PREVIEW & ACCEPT-ALL

## Refactor-preview

- Gli strumenti AI-powered (come Cursor o Copilot) permettono di eseguire il refactoring automatico del codice: migliorano nomi di variabili, suddividono funzioni troppo lunghe, eliminano codice duplicato, ecc.
- Prima di applicare i cambiamenti, viene mostrata un'anteprima dettagliata: puoi vedere tutte le modifiche suggerite (spesso in stile "diff" o "compare").
- Questo consente di valutare e selezionare le parti da accettare o modificare prima di aggiornare il codice reale.

## Accept-all

- Con un clic, puoi **accettare tutte le modifiche suggerite** dall'AI e applicarle direttamente al progetto.
- Utile quando le modifiche sono molte o il refactoring è standard.
- Attenzione: meglio sempre controllare che i cambiamenti siano corretti e coerenti col progetto, specialmente in team.

## Vantaggi:

- Risparmio di tempo su modifiche ripetitive o estese.
- Controllo granulare grazie all'anteprima.

# GENERAZIONE DOCSTRING AUTOMATICA

Gli strumenti AI-powered integrati negli IDE (come Cursor, Copilot, VS Code AI) possono **generare automaticamente le docstring** per funzioni, classi e moduli Python.

Basta selezionare una funzione o classe e richiedere la docstring: l'AI analizza nome, parametri e logica per produrre una descrizione chiara e dettagliata.

Le docstring generate includono spesso:

- Descrizione del comportamento della funzione
- Elenco dei parametri e del tipo
- Tipo e descrizione del valore di ritorno
- Eventuali eccezioni sollevate

## Vantaggi:

- Risparmio di tempo, soprattutto su progetti estesi.
- Favorisce la leggibilità e la manutenibilità del codice.
- Promuove buone pratiche di documentazione nel team.



# STUB DI TEST UNITARI AUTOMATICI

- Gli strumenti AI negli IDE (es. Copilot, Cursor) possono **generare automaticamente** gli “**stub**” di test **unitari** per funzioni e classi.
- Uno **stub** di test è una struttura di base di test, pronta da completare con i casi specifici.
- L'AI analizza il codice da testare e crea le funzioni di test seguendo lo stile della libreria scelta (unittest, pytest, ecc.).

## Vantaggi:

- Accelerazione della scrittura dei test, anche in progetti grandi o legacy.
- Standardizzazione del formato dei test.
- Incoraggia a scrivere test per ogni funzione.

## Esempio, dato questo codice:

```
def somma(a, b):  
    return a + b
```

## Lo stub generato automaticamente:

```
import unittest  
  
class TestSomma(unittest.TestCase):  
    def test_somma(self):  
        # TODO: aggiungi casi di test  
        result = somma(2, 3)  
        self.assertEqual(result, 5)  
  
if __name__ == '__main__':  
    unittest.main()
```

# “EXPLAIN THIS CODE” SU LEGACY CODE

Gli strumenti AI-powered negli IDE (come Cursor, Copilot, Codeium) possono **analizzare porzioni di codice esistenti (anche legacy o poco documentate)** e generare spiegazioni dettagliate in linguaggio naturale. È sufficiente selezionare il codice da chiarire e scegliere l'opzione “Explain this code”: l'AI descrive cosa fa ogni funzione, ciclo o struttura.

## Molto utile per:

- Onboarding di nuovi membri del team su codice datato
- Refactoring e manutenzione di codice poco chiaro
- Preparare la documentazione o la formazione interna

## Vantaggi:

- Riduce il tempo necessario a comprendere codice complesso o scritto da altri.
- Aiuta a individuare errori, bug o debiti tecnici nascosti.
- Facilita la collaborazione tra sviluppatori di diversa esperienza.

## Esempio:

Codice legacy  
senza commenti:

```
for u in utenti:  
    if u['attivo'] and u['ruolo'] ==  
    'admin':  
        invia_notifica(u['email'])
```

## Output AI:

Per ogni utente attivo con ruolo 'admin', il programma invia una notifica all'email dell'utente.

# SYNC BIDIREZIONALE CON GIT

Gli IDE AI-powered (come Cursor, VS Code + Copilot) **integrano funzioni di sincronizzazione bidirezionale con Git.**

## **Permettono di:**

- Lavorare in locale e spingere/recuperare modifiche dal repository remoto con un click.
- Visualizzare in tempo reale i cambiamenti suggeriti dall'AI e committarli direttamente.
- Ricevere suggerimenti di commit message generati automaticamente.

## **Vantaggi:**

- Evita conflitti tra versioni locali e remote.
- Facilita il versioning continuo durante l'uso degli assistenti AI.
- Supporta workflow collaborativi e sicuri.

# REGEX-REWRITE BULK

Con l'AI integrata, gli IDE permettono di **reformulare o correggere molte espressioni regolari (regex) contemporaneamente** su grandi codebase.

## Funzionalità tipiche:

- Suggerimento di regex ottimizzate e più leggibili.
- Applicazione massiva (*bulk*) di modifiche a tutte le occorrenze di una regex nel progetto.
- Preview delle sostituzioni prima dell'applicazione definitiva.

## Vantaggi:

- Riduce il rischio di errori nelle sostituzioni ripetitive.
- Fa risparmiare tempo rispetto alla modifica manuale.
- Migliora la qualità e la coerenza delle regex nei progetti complessi.

# KPI: LINES WRITTEN VS ACCEPTED

I moderni IDE AI-powered offrono statistiche sulle **linee di codice suggerite dall'AI ("written")** e su quelle effettivamente **accettate dagli sviluppatori ("accepted")**.

Questo KPI (Key Performance Indicator) misura:

- **Tasso di accettazione:** quante righe scritte dall'AI vengono ritenute utili e integrate nel progetto.
- **Qualità dei suggerimenti:** un basso tasso può indicare suggerimenti poco rilevanti o non precisi.
- **Efficienza:** permette di confrontare la produttività con e senza AI.

## Analisi impatto AI:

- Un alto rapporto accepted/written segnala **buona collaborazione uomo-macchina**.
- Aiuta a individuare aree in cui serve più training o personalizzazione dell'AI.
- Permette di ottimizzare il workflow, ridurre il tempo di revisione e accelerare il rilascio.

## Esempio pratico

In una settimana:

- Lines written (AI): 3000
- Lines accepted (umano): 1200
- **Acceptance rate: 40%**

# DOMANDE?

## Lavoro in gruppi



# TIPI DI PROGRAMMI PYTHON DA ESEGUIRE DALL'IDE

- **Esecuzione script principale**  
Avviare l'applicazione o il modulo principale del progetto.
- **Avvio di test automatici/unitari**  
Lanciare suite di test (pytest, unittest, ecc.) per verificare la correttezza del codice.
- **Code coverage**  
Analizzare quali parti del codice sono coperte dai test (strumenti: coverage.py).
- **Linting e formattazione**  
Eseguire tool come flake8, black, isort per controllare e correggere lo stile e la qualità del codice.
- **Deploy sul server**  
Script per caricare il programma o aggiornarlo su server di produzione o test.
- **Configurazione ambienti**  
Script per impostare variabili d'ambiente, generare file di configurazione o .env specifici.
- **Sincronizzazione dati**  
Script per importare/esportare dati tra ambienti di sviluppo, test, produzione.

# TIPI DI PROGRAMMI PYTHON DA ESEGUIRE DALL'IDE

- **Trasferimento file/code**  
Caricare/scaricare file di codice o dati tramite SCP, FTP, S3 ecc.
- **Refactoring e migrazioni**  
Eseguire comandi di refactoring strutturale o migrazioni di database (es. Alembic, Django manage.py migrate).
- **Monitoraggio e logging**  
Avviare script per tracciare log, performance, errori in tempo reale.
- **Analisi statica/dinamica**  
Lanciare tool per analisi approfondite (type checking con mypy, profiling, ecc.).
- **Correzioni automatiche di errori**  
Script che individuano e correggono problemi comuni o suggeriti da tool AI.
- **Creazione/aggiornamento documentazione**  
Generare automaticamente documentazione tecnica dal codice sorgente (sphinx, pdoc).
- **Build e packaging**  
Preparare pacchetti Python da distribuire (setup.py, poetry build).

# STRUTTURA BASE DI UN PROGETTO PYTHON

## Directory principale del progetto

- /src – Codice sorgente principale (moduli, package)
- /data – File di dati (test, dataset, embedding, ecc.)
- /tests – Test automatici/unitari
- /notebooks – Notebook Jupyter (per demo, sperimentazione)
- /docs – Documentazione tecnica aggiuntiva

## File fondamentali

- main.py o app.py – Punto di ingresso dell'applicazione
- requirements.txt o pyproject.toml – Dipendenze Python
- README.md – Descrizione progetto, uso, requisiti
- CHANGELOG.md – Cronologia delle modifiche/testuale e sintetica

# DOCUMENTAZIONE E FILE DI SUPPORTO

## README.md

- Presentazione del progetto
- Istruzioni rapide per avvio
- Esempi d'uso/minimo tutorial
- Contatti, riferimenti utili

## CHANGELOG.md

- Versionamento cronologico (data, versione, autore)
- Breve descrizione delle modifiche rilevanti (feature, bugfix, refactor)
- Esempio:  
## [v1.2.0] - 2025-06-20
  - Aggiunto supporto a RAG con FAISS
  - Fix errori di compatibilità Python 3.12

# STRUTTURE E MODULI TECNICI

## Esempio di Chatbot RAG Agentico

### Modularità del codice

Moduli separati per:

- Interfaccia utente/chatbot (es. interface.py)
- Gestione RAG e embedding (es. rag\_module.py)
- Integrazione agenti (es. agent.py)
- Utility comuni (es. utils.py)

Separazione netta tra logica applicativa, dati e configurazione, funzioni e visualizzazione

### Configurazione

- File .env o config.yaml per parametri sensibili, API key, path, ecc.
- Uso di dotenv o simili per caricare la configurazione

# TESTING, CI/CD E GESTIONE VERSIONI

## Cartella /tests

- Test automatici/unitari per le principali funzioni/feature
- Uso di pytest o unittest

## Notebook demo

- Esempi di uso reale e di casi limite in notebook (es. notebooks/demo.ipynb)

## CI/CD (*Continuous Integration/Continuous Deployment*)

- File .github/workflows/ci.yml per GitHub Actions (opzionale)
- Script/test automatici ad ogni push/pull request

## Gestione delle versioni

- Aggiornare sempre CHANGELOG.md
- Usare tag/versioni git per le release

# BEST PRACTICE E CHECKLIST FINALE

## Documentazione interna

- Commenti chiari, docstring nelle funzioni/classi
- Eventuale uso di generatori documentazione (sphinx, pdoc, ecc.)

## Codice pulito

- Seguire PEP8 (linting, formattazione)
- Tipizzazione (type hints) dove utile

## File di esempio

- Dataset di test piccolo e sintetico per demo
- Esempi di input/output nel README

## Pronto per la collaborazione

- Struttura ordinata e leggibile
- Nomi di file/folder e variabili chiari
- Nessun dato sensibile nel repo (usare .gitignore)



DOMANDE?

PAUSA

**GRAZIE PER L'ATTENZIONE**