Tabnine

GitHub Copilot

Codeium

Cursor

Windsurf

Copilot Chat

ChatGPT

Claude.ai

Jupyter AI

Ollama

LM Studio

Claude Code

Codex CLI

OpenHands

Devin

AutoGluon

VOTRE OUTIL
MANQUANT ?
AJOUTEZ-LE ICI

VOTRE OUTIL
MANQUANT ?
AJOUTEZ-LE ICI

Architecture

Générer la structure d'un projet, d'une classe, d'un module Python

Nombre de personnes :

Implémentation de fonctionnalités

Écrire une nouvelle fonctionnalité métier à partir d'une spécification en langage naturel

Nombre de personnes :

Conversion

Traduire du code Python 2 vers Python 3

Nombre de personnes :

Tests unitaires

Générer des cas de test pytest pour une fonction existante

Nombre de personnes :

Tests d'intégration

Écrire des tests qui couvrent plusieurs composants ensemble

Nombre de personnes :

Génération de données de test

Créer des fixtures ou des factories avec Faker/Hypothesis pour couvrir différents scénarios

Nombre de personnes :

Refactoring

Restructurer du code fonctionnel mais difficilement lisible

Nombre de personnes :

Revue de code

Analyser une PR pour détecter des bugs, des anti-patterns, des problèmes de lisibilité ou des cas non couverts

Nombre de personnes :

Docstrings et commentaires

Générer des docstrings et commentaires en ligne à partir du code Python existant

Nombre de personnes :

Typage

Générer des type hints à partir du code Python existant

Nombre de personnes :

Documentation technique

Rédiger un README ou un guide d'utilisation à partir du code source

Nombre de personnes :

Débogage d'erreurs connues

Coller une stacktrace et obtenir une explication

Nombre de personnes :

Bugs logiques complexes

Diagnostiquer un comportement inattendu sans message d'erreur

Nombre de personnes :

Conception de modules

Proposer une structure de projet ou des choix de design patterns adaptés au besoin métier

Nombre de personnes :

Pipelines CI-CD

Générer ou compléter des fichiers GitHub Actions ou GitLab CI pour un projet Python

Nombre de personnes :

Audit de vulnérabilités

Analyser du code Python pour détecter des injections SQL, des dépendances vulnérables ou des failles OWASP courantes

Nombre de personnes :

Implémentation sécurisée

Écrire de l'authentification ou du chiffrement en suivant les bonnes pratiques cybersécurité

Nombre de personnes :

UNE ACTIVITÉ
MANQUANTE ?
AJOUTEZ-LA ICI

Exploration initiale (EDA)

Générer du code pandas pour explorer un nouveau dataset : distributions, valeurs manquantes, corrélations, aperçu des types de colonnes.

Nombre de personnes :

Visualisation de données

Générer des graphiques matplotlib, seaborn ou plotly à partir d'une description : histogrammes, heatmaps, scatter plots, time series.

Nombre de personnes :

Formulation d'hypothèses

Demander à l'IA d'interpréter des résultats statistiques ou de suggérer des hypothèses à tester à partir d'une description du jeu de données.

Nombre de personnes :

Création de features

Suggérer et coder de nouvelles variables à partir des colonnes existantes : encodage, agrégats, ratios, lags, features temporelles.

Nombre de personnes :

Nettoyage & imputation

Générer du code pour traiter les valeurs manquantes, les doublons, les outliers ou normaliser des colonnes texte incohérentes.

Nombre de personnes :

Choix & comparaison de modèles

Demander conseil sur le choix d'algorithmes selon le problème (classification, régression, clustering), la taille des données et les contraintes.

Nombre de personnes :

Tuning d'hyperparamèt

Générer du code GridSearchCV, Optuna ou Ray Tune. Interpréter les courbes d'apprentissage et suggérer des plages à explorer.

Nombre de personnes :

Construction de pipelines ML

Écrire un pipeline sklearn complet : preprocessing, encodage, modèle, cross-validation. Reproductible et prêt pour la mise en production.

Nombre de personnes :

Choix de métriques

Conseiller sur les métriques adaptées au problème (AUC, F1, RMSE, MAPE…) et expliquer leurs compromis selon le contexte métier.

Nombre de personnes :

Interprétabilité (SHAP, LIME)

Générer du code SHAP ou LIME, interpréter les feature importances et formuler une explication métier des prédictions du modèle.

Nombre de personnes :

Mise en production du modèle

Générer une API FastAPI ou Flask autour d'un modèle sklearn, avec sérialisation, validation des entrées et gestion des erreurs.

Nombre de personnes :

Monitoring & détection de drift

Écrire des scripts de surveillance des performances en production, détecter le data drift et alerter en cas de dégradation du modèle.

Nombre de personnes :

Prompt engineering

Concevoir, itérer et évaluer des prompts pour des tâches de classification, extraction, résumé ou génération avec un LLM via API.

Nombre de personnes :

Construction d'un pipeline RAG

Mettre en place un système Retrieval-Augmented Generation : chunking, embeddings, vector store, retrieval et intégration d'un LLM.

Nombre de personnes :

Rédaction de rapports d'analyse

Transformer des résultats bruts (métriques, graphiques, tables) en un rapport structuré compréhensible pour des parties prenantes non-techniques.

Nombre de personnes :

Détection de biais & équité

Analyser un modèle pour détecter des biais sur des groupes sensibles, mesurer la fairness et proposer des stratégies de mitigation.

Nombre de personnes :

UNE ACTIVITÉ MANQUANTE ? AJOUTEZ-LA ICI

UNE ACTIVITÉ MANQUANTE ? AJOUTEZ-LA ICI

Rédaction d'un plan de cours

Générer la structure d'un module de formation Python : objectifs pédagogiques, progression, découpage en séances, prérequis.

Nombre de personnes :

Création de supports (slides, notebooks)

Produire des slides de présentation, des notebooks Jupyter pédagogiques avec cellules de code et zones d'exercices pré-remplies.

Nombre de personnes :

Adaptation à un niveau donné

Reformuler une explication technique pour un public débutant, intermédiaire ou expert. Trouver la bonne analogie, le bon niveau d'abstraction.

Nombre de personnes :

Génération d'exercices Python

Créer des exercices de difficulté progressive sur un concept donné (fonctions, classes, décorateurs…) avec consignes et jeux de tests.

Nombre de personnes :

Création de QCM & quiz

Générer des questions à choix multiples avec distracteurs plausibles pour tester la compréhension d'un concept Python ou d'une notion algorithmique.

Nombre de personnes :

Génération de corrigés commentés

Produire des solutions d'exercices avec explications pas à pas, variantes possibles et points de vigilance pour l'apprenant.

Nombre de personnes :

Correction automatique de code

Évaluer le code rendu par un apprenant selon une grille de critères : exactitude, lisibilité, respect des conventions, gestion des cas limites.

Nombre de personnes :

Feedback individualisé

Générer un retour personnalisé sur le code ou le raisonnement d'un apprenant : ce qui fonctionne, ce qui peut être amélioré, pistes de progression.

Nombre de personnes :

Déblocage guidé (sans donner la réponse)

Aider un apprenant bloqué en posant des questions socratiques plutôt qu'en donnant la solution. Maintenir l'effort cognitif.

Nombre de personnes :

Explication de code existant

Décomposer un bout de code Python inconnu ligne par ligne pour aider un apprenant à comprendre ce qu'il lit ou ce qu'il a copié.

Nombre de personnes :

Parcours d'apprentissage personnalisé

Construire un programme sur mesure à partir du niveau, des objectifs et du temps disponible d'un apprenant. Ressources, jalons, exercices ciblés.

Nombre de personnes :

Veille & mise à jour des connaissances

Demander un résumé des nouveautés Python (3.12, 3.13…), des évolutions d'une lib, ou des pratiques qui ont changé depuis qu'on a appris.

Nombre de personnes :

Montée en compétence sur un concept

Utiliser l'IA comme tuteur pour apprendre un concept nouveau : générateurs, async/await, métaclasses… avec exemples progressifs et exercices.

Nombre de personnes :

Prise en main d'une codebase

Faire expliquer par l'IA l'architecture d'un projet inconnu, ses conventions, ses patterns, pour réduire le temps d'intégration d'un nouveau développeur.

Nombre de personnes :

Vérification de la compréhension réelle

S'assurer qu'un apprenant a vraiment compris et n'a pas juste copié la réponse de l'IA. Évaluer l'apprentissage profond vs la surface.

Nombre de personnes :

Développer l'autonomie de l'apprenant

Apprendre à un apprenant à se passer progressivement de l'IA, à trouver les ressources par lui-même, à développer son sens critique.

Nombre de personnes :

UNE ACTIVITÉ MANQUANTE ? AJOUTEZ-LA ICI

UNE ACTIVITÉ MANQUANTE ? AJOUTEZ-LA ICI

Itérations infinies

Beaucoup d'itérations nécessaires pour affiner le code généré, au point de dépasser le temps qu'on aurait mis à coder soi-même.

Nombre de personnes :

Changement d'outils

Pression de changer d'outil IA dès qu'un nouveau modèle sort, par peur de rater le plus performant du moment, au détriment de la maîtrise d'un outil.

Nombre de personnes :

Perte de compétences

En déléguant trop à l'IA, on perd progressivement la capacité à lire, déboguer et architecturer du code Python de manière autonome.

Nombre de personnes :

Coûts des tokens

Dépassement des limites de tokens ou du budget alloué, surtout avec de grands fichiers ou des contextes complexes envoyés en prompt.

Nombre de personnes :

Code non fiable

Le code généré fonctionne en surface mais contient des bugs subtils, des anti-patterns ou ignore les bonnes pratiques Python.

Nombre de personnes :

Fuites de données

Risque d'envoyer du code propriétaire, des clés API ou des données sensibles au modèle, avec des implications de confidentialité non négligeables.

Nombre de personnes :

Perte de contexte

L'IA génère du code sans connaître l'architecture existante, produisant des solutions inadaptées à la codebase, aux conventions ou aux contraintes métier.

Nombre de personnes :

Code boîte noire

Difficulté à maintenir du code généré qu'on ne comprend pas vraiment, créant une dépendance à l'IA pour toute future modification.

Nombre de personnes :

Prompt structuré

Définir en amont le contexte, les contraintes et le format de sortie attendu dans le prompt pour réduire le nombre d'itérations correctrices.

Nombre de personnes :

Choisir et rester

Se fixer sur un outil pendant au moins un mois avant d'évaluer. La maîtrise d'un outil surpasse les gains marginaux d'un changement.

Nombre de personnes :

Coder avant de générer

Esquisser soi-même la logique ou le squelette du code avant de demander à l'IA de compléter. Cela maintient la compréhension et l'autonomie technique.

Nombre de personnes :

Modèle local avec Ollama

Utiliser un modèle local pour les tâches répétitives ou volumineuses.

Nombre de personnes :

Revue + tests obligatoires

Traiter le code généré comme une PR externe : relecture ligne à ligne, linting automatique et exiger des tests unitaires dans le prompt lui-même.

Nombre de personnes :

IA locale pour le code sensible

Pour tout code contenant des secrets ou des données métier, utiliser un modèle local afin de ne rien envoyer à l'extérieur.

Nombre de personnes :

Fichier de règles projet

Maintenir un fichier AGENTS.md décrivant l'architecture, les conventions et les contraintes. L'IA s'y réfère automatiquement à chaque génération.

Nombre de personnes :

Exiger une explication

Systématiquement demander à l'IA d'expliquer chaque bloc généré ligne par ligne. Ne valider le code que lorsqu'on est capable de le réécrire sans aide.

Nombre de personnes :

Code de nettoyage aveugle

L'IA génère des pipelines de nettoyage génériques qui masquent des problèmes métier spécifiques aux données (valeurs aberrantes, encodages, biais de collecte...).

Nombre de personnes :

Expériences non reproductibles

Le code généré omet le versionnage des données ou des modèles rendant les expériences ML impossibles à reproduire fidèlement

Nombre de personnes :

Perte du raisonnement statistique

En déléguant l'analyse à l'IA, on perd la capacité à choisir les bons tests statistiques, détecter les biais ou interpréter correctement les résultats.

Nombre de personnes :

Explosion des coûts compute

Le code généré ignore les optimisations mémoire entraînant des coûts disproportionnés sur de grands datasets.

Nombre de personnes :

Data leakage silencieux

L'IA génère des pipelines ML avec des fuites de données qui gonflent artificiellement les métriques sans déclencher d'erreur.

Nombre de personnes :

Données sensibles exposées

Envoi d'échantillons de données sensibles dans le prompt pour déboguer avec des risques RGPD et de confidentialité importants.

Nombre de personnes :

Métriques inadaptées

L'IA propose des métriques standard (accuracy, MSE) sans connaître le contexte métier optimisant pour le mauvais objectif sur des données déséquilibrées.

Nombre de personnes :

Notebooks ingérables

Accumulation de notebooks générés cellule par cellule, sans structure ni modularité, impossibles à maintenir, tester ou faire évoluer sans l'aide de l'IA.

Nombre de personnes :

Prompt avec schema et statistiques

Fournir à l'IA les types, describe() et un échantillon anonymisé du DataFrame avant de demander le pipeline. Le contexte données produit un code de nettoyage adapté.

Nombre de personnes :

Template d'expérience MLflow

Maintenir un template de script avec MLflow pré-configuré. Demander à l'IA de compléter ce squelette plutôt que de partir de zéro.

Nombre de personnes :

Analyser avant de coder

Formuler soi-même l'hypothèse, choisir le modèle adapté puis demander à l'IA uniquement l'implémentation. Préserve le raisonnement analytique.

Nombre de personnes :

Modèle local avec Ollama

Utiliser un modèle local pour générer du code data sur de vrais datasets.

Nombre de personnes :

Checklist anti-leakage

Demander systématiquement à l'IA de vérifier : split avant preprocessing, absence de features temporelles futures, validation croisée stratifiée.

Nombre de personnes :

Données synthétiques pour déboguer

Générer un jeu de données synthétique reproduisant la structure des vraies données. Envoyer uniquement ce proxy à l'IA pour déboguer sans exposer de données réelles.

Nombre de personnes :

Fiche métier dans le prompt

Inclure dans chaque prompt : la définition métier de la cible, le coût d'une fausse alarme / d'un faux négatif et la distribution des classes. L'IA choisit alors la bonne métrique.

Nombre de personnes :

Convertir en scripts modulaires

Demander à l'IA de refactoriser chaque notebook en modules Python réutilisables.

Nombre de personnes :

Copier sans comprendre

L'apprenant colle le code généré dans son exercice sans chercher à comprendre la logique.

Nombre de personnes :

Fausse impression de maîtrise

Le code tourne, l'exercice est rendu mais l'apprenant ne saurait pas le réécrire seul. L'IA crée un écart croissant entre la performance perçue et les compétences réelles.

Nombre de personnes :

Court-circuit du débogage

Dès la première erreur, l'apprenant demande à l'IA de corriger. Il ne développe jamais la capacité à raisonner sur l'état du programme.

Nombre de personnes :

Progression artificielle

L'apprenant avance dans le cursus grâce à l'IA, abordant des notions avancées sans avoir consolidé les bases créant des lacunes invisibles mais profondes.

Nombre de personnes :

Hallucinations prises pour argent comptant

L'IA génère du code avec des fonctions inexistantes ou des comportements incorrects. L'apprenant sans références solides n'a pas les outils pour détecter ces erreurs.

Nombre de personnes :

Démotivation par facilité

Résoudre des problèmes trop facilement grâce à l'IA supprime la satisfaction de l'effort et de la découverte, ressorts essentiels pour persévérer dans l'apprentissage.

Nombre de personnes :

Style de code incohérent

L'IA produit du code dont le niveau varie : parfois trop avancé, parfois trop verbeux et inadapté au stade d'apprentissage de l'apprenant.

Nombre de personnes :

Dépendance à l'autocomplétion

L'apprenant ne sait plus écrire du code sans suggestions en temps réel. En examen ou en entretien technique, sans IA, il se retrouve bloqué sur des tâches basiques.

Nombre de personnes :

IA comme examinateur

Demander à l'IA d'expliquer le code qu'on vient d'écrire soi-même, puis de poser des questions dessus. L'IA devient un outil de vérification de compréhension, pas de production.

Nombre de personnes :

Règle du brouillon d'abord

Imposer d'écrire une solution, même imparfaite, avant tout recours à l'IA. L'IA ne corrige qu'après une tentative réelle, ce qui ancre l'effort cognitif nécessaire à l'apprentissage.

Nombre de personnes :

Déboguer sans IA d'abord

S'imposer 15 minutes de débogage autonome avant de consulter l'IA. Développe le muscle du raisonnement sur les erreurs.

Nombre de personnes :

Exercices hors-ligne réguliers

Pratiquer sur des environnements sans IA pour mesurer ses compétences réelles et détecter les lacunes masquées par l'assistance.

Nombre de personnes :

Vérifier avec la documentation

Systématiquement croiser le code généré avec la documentation de la librairie. Développe le réflexe de lecture de documentation et l'esprit critique face aux hallucinations.

Nombre de personnes :

Demander des indices, pas la solution

Reformuler ses demandes : "donne-moi un indice" ou "explique-moi le concept dont j'ai besoin" plutôt que "écris le code". L'IA devient un tuteur socratique, pas un distributeur de solutions.

Nombre de personnes :

Adapter le niveau dans le prompt

Préciser son niveau dans chaque prompt : "je débute, utilise uniquement des boucles for et des fonctions simples". L'IA produit du code pédagogiquement cohérent avec le stade d'apprentissage.

Nombre de personnes :

Modèle local pour s'entraîner

Utiliser un modèle local léger pour pratiquer sans connexion. Réponses plus limitées = moins de béquilles, plus d'effort cognitif autonome.

Nombre de personnes :