Stage de formation data et IA

Introduction

Bienvenue dans cette formation intensive destinée à initier et encadrer un profil junior dans le domaine de la Data Science et de l'Intelligence Artificielle, avec Python.

Cette formation est conçue pour un stagiaire ayant des bases en programmation (niveau Licence 1 en informatique), et vise à lui transmettre les compétences essentielles pour comprendre, manipuler, modéliser et mettre en production des solutions basées sur les données et les modèles d'IA.

Elle s'articule autour de projets concrets et progressifs, intégrant dès le départ les bonnes pratiques de développement, la documentation, la structuration en notebooks, l'usage de GitHub pour le versionning, et l'intégration de l'intelligence artificielle générative (textuelle et vocale) dans des cas d'usage réalistes.

La pédagogie repose sur la pratique, avec une alternance chaque semaine entre l'apprentissage de concepts clés, la mise en œuvre dans des notebooks Python, des projets guidés et des interactions avec des modèles IA. Le stagiaire manipule de vraies données, construit des modèles, génère du texte, converse avec l'ordinateur par la voix, et documente ses travaux sur GitHub.

À l'issue de ces quatre semaines, il aura acquis des bases solides pour évoluer en tant qu'assistant développeur IA, avec un premier portfolio de projets techniques à valoriser.

Objectifs pédagogiques

- Acquérir une maîtrise solide de Python pour l'analyse de données
- Comprendre et mettre en œuvre des modèles de Machine Learning supervisés
- Travailler avec les données vocales (Speech-to-Text & Text-to-Speech)
- Concevoir et exposer des modèles via des API Web
- S'initier aux concepts fondamentaux de l'IA Générative
- Développer des mini-projets intégrés avec des livrables propres et utilisables

Méthodologie

La formation repose sur :

- Un rythme hebdomadaire par bloc de compétences
- Des notebooks interactifs pour expérimenter le code
- Des projets intégrés (Titanic, assistant IA, moteur vocal, chatbot…)
- Un suivi hebdomadaire dans Notion (avec feedbacks & objectifs)
- Une structure de projet versionnée sur GitHub (avec README et code propre)

À la fin de cette formation, le stagiaire aura produit plusieurs projets complets, utilisables et déployables, tout en ayant acquis les fondations techniques nécessaires pour poursuivre dans le domaine de la Data Science et de l'IA.

Sommaire

Outils & librairies utilisés tout au long de la formation :
→ scikit-learn, pandas, seaborn, matplotlib
→ speechrecognition, whisper, pyttsx3, gTTS
→ fastapi, streamlit, openai, sentence-transformers
→ docker, joblib (bonus)

Semaine 1 – Introduction à Python, Numpy, Pandas & Exploration de données

Objectifs :

- Poser les bases de la programmation scientifique avec Python
- Manipuler des tableaux avec NumPy
- Analyser des données tabulaires avec Pandas
- Réaliser une première exploration de données (EDA)
- Préparer un dataset pour une tâche de Machine Learning

Contenu technique

Python scientifique :

- Types de données, boucles, conditions
- Fonctions et modules
- Utilisation de Jupyter Notebook

NumPy :

- Création et manipulation de tableaux (`ndarray`)
- Indexation, slicing, opérations mathématiques
- Notion de vectorisation

Pandas :

- Lecture de fichiers CSV
- Séries et DataFrame
- Filtrage, tri, statistiques descriptives
- GroupBy et agrégation
- Gestion des valeurs manquantes

Visualisation :

- Utilisation de `matplotlib` et `seaborn`
- Graphiques simples : histogrammes, scatter plots, heatmaps
- Visualisation des relations entre variables (Titanic)

Préparation de données (prétraitement de base) :

- Nettoyage de colonnes
- Transformation de variables catégorielles
- Création de nouvelles features
- Encodage (`LabelEncoder`, `OneHotEncoder`)
- Détection de corrélations

Projet intégré (Titanic EDA)

Objectif : Mener une analyse exploratoire du jeu de données Titanic

1. Lecture et visualisation du dataset (avec seaborn)
2. Nettoyage de base (NaN, types, catégories)
3. Analyse des survivants selon : sexe, classe, âge
4. Préparation d'un jeu propre pour modélisation (en semaine 2)

Librairies utilisées

- Python (>=3.8)
- numpy, pandas
- matplotlib, seaborn
- jupyterlab

Livrables :

- Notebook titanic_eda.ipynb
- README : étapes de nettoyage et EDA, conclusions
- Fichier titanic_clean.csv prêt pour la modélisation

Semaine 2 – Modélisation ML supervisée + Voix (STT & TTS)

Objectifs :

- Comprendre la logique d'un modèle de Machine Learning supervisé
- Construire un pipeline simple de classification
- Évaluer un modèle
- **Ajouter une interface vocale** : parler au modèle, écouter les résultats

Contenu technique

Modélisation (Scikit-learn) :

- Split des données (train/test)
- Classification binaire (KNN, DecisionTree, RandomForest)
- Évaluation : `accuracy`, `f1_score`, `confusion_matrix`
- Sauvegarde du modèle avec `joblib` ou `pickle`

STT – Speech-to-Text :

- Introduction à **Whisper (openai-whisper ou faster-whisper)**
- Alternatives plus simples : `SpeechRecognition` + Google STT
- Prendre la voix du stagiaire en entrée, convertir en texte (ex : nom, âge, sexe)
- Extraction des features depuis la voix

TTS – Text-to-Speech :

- Introduction à **pyttsx3** (local)
- ou **gTTS** (via Google)
- Génération audio des résultats (ex : "Vous êtes prédit comme survivant avec 83% de probabilité.")

Projet intégré (Titanic Vocal)

Interaction vocale :

1. L'utilisateur parle (ex : "Marie, 25 ans, femme, 1ère classe")
2. Le système convertit → texte → features → prédiction
3. L'IA prédit (modèle RandomForest)
4. Résultat retourné à l'oral avec TTS

Librairies utilisées

- `scikit-learn`, `pandas`, `joblib`
- `speechrecognition`, `pyaudio` ou `whisper`
- `pyttsx3` ou `gtts` pour la synthèse vocale

Livrables :

- Notebook `titanic_model_audio.ipynb`
- Enregistrements vocaux de test
- Fichier audio généré via TTS
- README : comment ça marche, flux d’entrée-sortie

Bonus (si temps dispo) :

- Interface Streamlit avec bouton "parler" et lecture audio
- Mini-API avec FastAPI pour STT + prédiction + TTS

Semaine 3 – API, Interfaces Web & Déploiement

Objectifs :

- Comprendre le fonctionnement d'une API
- Construire une API avec FastAPI pour exposer un modèle ML
- Créer une interface Web simple (formulaire)
- Permettre à un utilisateur de faire une prédiction via le Web
- Initier le stagiaire au déploiement local avec Docker

Contenu technique

API avec FastAPI :

- Démarrage rapide avec FastAPI
- Création de routes GET/POST
- Passage de paramètres (body, query)
- Retour de JSON contenant des prédictions
- Documentation automatique avec Swagger UI (/docs)

Connexion modèle ML :

- Chargement du modèle entraîné (RandomForest du Titanic)
- Endpoint /predict pour soumettre des données utilisateur
- Structure d'input avec Pydantic
- Logique de transformation des données utilisateur (features)

Interface Web (formulaire HTML/CSS ou Streamlit) :

- Création d'un formulaire : nom, âge, sexe, classe, etc.
- Soumission au backend API
- Affichage de la prédiction dans l'interface utilisateur

→ Option 1 : Streamlit simple (si pas de frontend HTML)
→ Option 2 : HTML/CSS + JS fetch POST vers API

Déploiement local :

- Introduction à Docker
- Écriture d'un Dockerfile simple
- Construction d'une image et exécution d'un conteneur localement
- Accès à l'API via localhost:8000/docs ou à l'interface Web

Projet intégré : Titanic Web App

Objectif : accéder au modèle ML via une interface utilisateur

1. L'utilisateur entre ses données via formulaire web
2. Envoie des données à l'API FastAPI

3. API prédit avec le modèle et renvoie un JSON
4. L'interface affiche la probabilité de survie

Librairies utilisées

- fastapi, uvicorn
- pydantic
- joblib, scikit-learn
- streamlit ou HTML/CSS (form)
- docker (installation requise)

Livrables :

- API FastAPI opérationnelle : fichier main.py
- Dockerfile fonctionnel
- Interface utilisateur Web ou Streamlit
- Test local de bout en bout
- README expliquant l'architecture et comment exécuter

Bonus (si temps dispo) :

- Envoi vocal via STT + requête à l'API
- Version cloud : déploiement sur Render ou Railway
- Ajout de logging ou monitoring dans l'API

Semaine 4 – Introduction à l'IA Générative et discriminative

Objectifs :

- Comprendre les bases de l'intelligence artificielle générative
- Découvrir les modèles de langage : BERT, BART, T4.
- Manipuler des embeddings de texte (vectorisation)
- Réaliser un premier projet de génération de texte simple

Contenu technique

Fondamentaux de l'IA Générative :

- Différence entre IA classique et IA générative
- Cas d'usage : génération de texte, images, code, voix…
- Modèles célèbres : GPT, Claude, Mistral, LLaMA, Deepseek

Prompt Engineering :

- Qu'est-ce qu'un prompt ?
- Prompt efficace (clarité, rôles, contraintes, exemples)
- Prompt chaining : chaîner plusieurs appels

Utilisation d'un LLM (via API OpenAI ou modèle local) :

- Structure d'un appel d'API GPT (chat ou completions)
- Réglages du comportement (temperature, top_p, stop, etc.)
- Génération de texte, résumé, classification

Embeddings (vecteurs de texte) :

- Représenter un texte comme un vecteur
- Utilisation d'OpenAI embeddings ou HuggingFace Transformers
- Calcul de similarité de textes (cosine similarity)

→ Application : moteur de recherche de documents + réponse contextuelle

Projet intégré : Chatbot simple + moteur sémantique

Deux volets possibles :

1. Un assistant IA qui répond à des questions de l'utilisateur (via GPT-3.5/4 ou modèle local)
2. Un moteur de recherche basé sur la similarité de textes dans des documents locaux (vectorisation avec embeddings)

Librairies utilisées

- openai ou llama-cpp-python (modèle local)
- langchain (optionnel pour pipeline)
- sentence-transformers (pour les embeddings)
- tiktoken (tokenisation)
- pandas, numpy

Livrables :

- Notebook d'exploration `week4_ia_generative_intro.ipynb`
- Script d'appel API ou chatbot local
- Données vectorisées et moteur de similarité
- README avec instructions et explication des prompts

Bonus (si temps dispo) :

- Intégration voix : poser sa question à l'oral (STT) et recevoir une réponse audio (TTS)