**FORMATION IA Engineer**

**PROJET II : Requetes des services IA**

**I.1. Livrable**

- Un Jupyter Notebook contenant les réponses aux requêtes API et la visualisation des résultats

- Un support de présentation complété sur la base du template fourni.

**I.2. Présentation du livrable (bilan ou soutenance)**

C’est un bilan qui sera effectué avec le Mentor sur mes compétences :

* Discussion sur ma fiche d'autoévaluation et de mes commentaires dans la colonne "Notes".
* Explication sur les difficultés rencontrées et ce qui a été plus difficile.
* Présentation de mes points forts, ce que j’ai apprécié accomplir et pourquoi ces tâches paraissaient plus faciles.
* Identification des actions à mener par la suite : quel cours à revoir, quels sont les éléments à approfondir et sur lesquels qu’il faille rester vigilant.

**I.3. Contenu du Projet**

1. **Initiez-vous à Python pour l'analyse de données**

***a.1. Installez python et Anaconda***

pas de problème pour l’installation. Anaconda Navigator version 2.6.6 installée à la place de la version 2.6.3. Installe Python 3.12.7

lancement de Jupyter Notebook

vu les classes, les fonctions, les modules, les packages

créer un module %%writefile module.py

créer un package

* import os
* os.makedirs(r"C:\Users\Lina\mon\_projet\mesmodules\monpackage", exist\_ok=True)
* %%writefile C:/Users/Lina/mon\_projet/mesmodules/monpackage/\_\_init\_\_.py

Random.choices(liste, k=nombre) permet de sélectionner un échantillage avec remise

Random.sample(liste,k=nombre) permet de sélectionner un échantillon sans remise

Pour saisir une valeur

while True:

try:

montant = int(input("Veuillez entrer un montant entier : "))

break

except ValueError:

print("Erreur : veuillez saisir un nombre entier valide.")

1. **Découvrez les librairies Python pour la Data Science**

**a partire du 23 juin**

Numpy : librairie principale pour les calculs scientifiques et manipulation des tableaux avec indice (seulement le même type de données dans un tableau Array), matrice..

Panda : la librairie la plus importante pour la manipulation des données avec des dataframes (gèrent plusieurs types de données comme sur Excel)

Pour les graphiques :

* les **courbes, diagrammes en ligne et parfois les diagrammes à barres** pour représenter une évolution d’une caractéristique (comme la démographie d’un pays) dans le temps ;
* **les graphiques circulaires** (appelés plus vulgairement des graphiques camemberts) et **surtout les diagrammes à barres** (histogramme) pour représenter une répartition ou une comparaison ;
* ou encore **des cartes** lorsqu’on souhaite afficher une information géographique.
* Enfin **les nuages de points** lorsque l’on veut représenter plusieurs variables mais à éviter car illisible lorsqu’il y a plus de 2 variables.

Les principales librairies de python

| **Librairie** | **Utilisation principale** |
| --- | --- |
| pandas | Tableaux et bases de données |
| numpy | Calcul numérique/matrices |
| matplotlib | Graphiques |
| seaborn | Statistiques et graphiques |
| scikit-learn | Machine Learning |
| tensorflow | Deep Learning |
| PIL/Pillow | Images |
| requests | Web, API |
| os/sys | Système, fichiers |
| jupyter | Notebooks interactifs |

J ai créé un compte dans :

* GIT : compte google
* Kaggle : compte google

**EXO 1 : Gerez votre environnement avec Poetry**

Python : 3.13.5 version

Poetry 2.1.4 : gestion de version des packages et mis en place d’environnement virtuel isolé

py -V permet de connaitre la version de Python si on nr connait pas le repertoire d’installation de python

Installation faite :

* Anaconda Navigator et ensuite lancement de Jupiter pour créer des modules et fonctions en python a tester
* Python 3.13.5
* Pip install pipx : Cette commande **installe le package pipx** en utilisant l’outil pip version 2.5
* python -m pipx ensurepath : prépare ton système pour utiliser facilement tous les outils installés via pipx (comme poetry, jupyter, etc.) dans le terminal.
* Repertoire d’installation de Python C:\Users\laeti\AppData\Local\Programs\Python\Python313\python.exe
* Pour activer l’environnement poetry pour python, ecrire sur la ligne de commande :
  + poetry env info –path
  + <path>\Scripts\activate.bat

**EXO 1 : Requetes des services IA**

**Developpement Projet FashionTrend**

1. Création du répertoire FashionTrend de mon projet avec les sous répertoires suivants :
   1. Data:\ raw – processed - external
   2. Src:\ data – features – models -vizualisation
   3. Models
   4. Notebooks
   5. Scripts
   6. Api
   7. Docs
   8. Outputs
   9. Tests
2. Installation de l’environnement de test Poetry 2.1.4

Je lance l’invite de commande windows. Je me positionne dans le répertoire de mon projet. Je ance ensuite la commande *Poetry init*

1. Je crée mon notebook
   * Installer et enregistrer le kernel de ton env Poetry

*Poetry add -D ipykernel*

*poetry run python -m ipykernel install --user --name=FashionTrend\_env --display-name "Python (FashionTrend)"*

* + Lancer jupyterLab 4.4.6 depuis l’environnement Poetry de mon application
    - Generer le fichier de configuration ***jupyter\_lab\_config.py***

*jupyter lab --generate-config*

* + - modifier ce fichier avec Notepad en écrivant ceci

c.ServerApp.root\_dir = 'D:/IA/Appli/FashionTrend/notebooks'

* + - ouvrir jupiter lab via anaconda
  + Créer mon notebook en choisissant l’environnement *Python* (*FashionTrend)*

1. Mettre dans “Notes” la version exacte de Python et des bibliothèques,

ça rassure sur la reproductibilité.

Mon fichier d’environnement est *requirements.txt qu’il faut créer (pip freeze > requirements.txt)*

Creation de la première cellule de mon notebook avec un code qui note les librairies et génère automatiquement le fichier requirement.txt

1. Vérifier que mon .env est bien ignoré par Git et que rien est exposé dans mon notebook/ scripts.
   1. Verifie que git est installé dans l’ordinateur ou faire l’Installation de Git 2.51.0 pour gérer les versions de mon code
   2. Initier mon projet avec Git

Avec l’invite de commande, se mettre dans la racine de mon projet et ensuite exécuter *git init*

* 1. Créer mon fichier *.gitignore* dans la racine de FashionTrend

On crée le fichier **.gitignore** avec NotePad (pour ignorer .env, les notebooks checkpoints, etc.) :

* 1. Ajouter et committer le fichier .gitignore

*git add .gitignore*

*git commit -m "Ajout du fichier .gitignore pour ignorer .env et fichiers temporaires"*

* 1. Même si .env est ignoré, il faut vérifier que l’on a pas **imprimé** ou **commit accidentellement** une valeur sensible. Pour cela, il faut un **script Python prêt à l’emploi** pour vérifier que rien de sensible (ex: .env, clés API, tokens) n’a fuité dans **tes notebooks, scripts et – optionnellement – l’historique Git**. Ainsi, on crée le script python scan\_secrets.py dans la racine du FashionTrend. On exécute sur le prompt la commande

*python scan\_secrets.py --skip-outputs --min-entropy 4.9 --out secrets\_report.md*

*python scan\_secrets.py --history --skip-outputs --min-entropy 4.9*

*python scan\_secrets.py --summary-only --out secrets\_report.md*

* 1. Effacer les sorties notebooks avant commit :

Jupyter : Kernel > Restart & Clear Output + enregistrer

* 1. Configurer git avec mon email noreply avec l’invite de commande

*git config --global user.name "linalaetis"*

*git config --global user.email "205674545+linalaetis@users.noreply.github.com"*

* 1. Après je fais mon premier commit

*git add .gitattributes*

*git add --renormalize .*

*git commit -m "Renormalisation des fins de ligne"*

*avec cette procédure mon depot est bien normalisé et portable. Les commandeq suivantes verifient que : git status –ignored (mon env est bien ignoré par Git)*

1. Pousser mon dépôt sur gitHub pour le partager sur internet

*git push -u origin main*

1. connecter ton dépôt local à GitHub

* on cree le depot sur GitHub en s’identifiant: https://github.com/linalaetis/FashionTrend.git
* on le connecte avec l’invite de cde :

git remote add origin <https://github.com/linalaetis/FashionTrend.git>

* on verifie l’enregistrement : *git remote -v*

on cree son token si onn l’a pas :

pour l’appli c’est : ghp\_xFL60ON2hwumfjx7Je4PbHJGydDoGh1pjS8H

**token essentiel pour git push/pull**

* on pousse la branche locale vers GitHub : *git push -u origin main*

1. mettre a jour un depot connecté a git

*git add .*

*git commit -m "Message clair décrivant les modifications" (commit local)*

*git push (envoyer les changements du PC vers GitHub)*

*git pull origin main (pour envoyer les changements de GitHub vers le PC)*

1. ajouter sur GitHub
   1. fichier README.MD et LICENCE et choisir la licence MT
   2. description et topics
   3. protéger la branche main
   4. stocker les variables et les secrets (comme cles clés API) , jamais dans .env :

Onglet **Settings → Secrets and variables → Actions**

* 1. kk

1. **rendre fonctionnel et propre un depot ou mon projet**
   * Créer et Copier .pre-commit-config.yaml à la racine du projet dans FashionTrend.
   * Créer et Copie tout le dossier .github/workflows/ci.yml dans mon dépôt
   * **Ajouter les deps de dev** (qualité + tests) avec Poetry (sur powershell)

*poetry add --group dev ruff black pre-commit nbstripout pytest pytest-cov (une seule fois)*

* + Activer pre-commit et nettoyer une première fois

*poetry run pre-commit install (une seule fois)*

*poetry run pre-commit run --all-files*

* + **Pousser la CI GitHub Actions :** (pour déclencher la CI GitHub)

*git add .pre-commit-config.yaml .github/workflows/ci.yml pyproject.toml*

*git commit -m "Quality automation: pre-commit + CI (ruff/black/pytest/secret-scan)"*

*git push*

1. Faire des texte

écrire quelques tests dans /tests (même simples, comme un test\_smoke.py qui vérifie que src/fashiontrend s’importe bien), puis vérifier que tout passe dans la CI après un git push.

1. Utilisation les datasets de Kaggle et tester certains de ses algorithmes
2. Utilisation les datasets de Kaggle et tester certains de ses algorithmes
   1. ajouter Kaggle dans mes dépendances avec l’invite de commande :

*Poetry add kaggle joblib ipykernel*

* 1. créer ma clé API kaggle (clé privé a ne pas partager) nommé kaggle.json que je place dans C:\utilisaeurs\laeti\kaggle

1. crée ma clé API kaggle nommé kaggle.json que j’ai placé dans C:\utilisaeurs\laeti\kaggle

**FICHE MEMO GIT AU QUOTIDIEN**

l’URL du dépôt GitHub lié est https://github.com/linalaetis/FashionTrend.git

1. Vérifier l’état du dépôt : git status
2. Ajouter tous les fichiers modifiés : git add .
3. Ajoute un fichier précis : git add README.md
4. Enregistrer un commit : git commit -m "Message clair sur la modification"

Envoyer sur GitHub : git push -u origin main

1. Met à jour ton dépôt local avec la dernière version distante de GitHub : git pull origin main
2. Voir l’historique des commits : git log --oneline --graph –decorate
3. Annuler un fichier ajouté par erreur : git reset README.md
4. Restaurer un fichier modifié : git restore README.md

👉 Affiche l’URL du dépôt GitHub lié (ex: origin <https://github.com/linalaetis/FashionTrend.git>).

Veux-tu que je t’écrive un **petit tableau récapitulatif** des commandes que tu dois lancer au quotidien (dev local vs CI/CD) ?