Projet d’examen — Mini pipeline de bout en bout

Objectif

Réaliser un projet complet de data science, de bout en bout, en partant d’un jeu de données public et en terminant avec un **modèle entraîné et évalué** ainsi qu’un **dépôt GitHub reproductible**.

Ceci est un examen : votre note est basée **principalement sur le contenu de votre dépôt GitHub** (code, analyses, commits, README, reproductibilité).

Choix du jeu de données (en choisir 1)

Choisissez **un seul** jeu de données dans la liste ci-dessous (liens Kaggle). Choisissez un dataset de taille petite/moyenne, suffisamment simple pour être exploré et modélisé dans le temps imparti.

> Vous pouvez proposer un autre dataset Kaggle, mais il doit être approuvé à l’avance et avoir une difficulté comparable.

10 datasets Kaggle adaptés aux débutants

1. Titanic - Machine Learning from Disaster (classification)
    - https://www.kaggle.com/competitions/titanic
2. House Prices: Advanced Regression Techniques (regression)
    - https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques
3. Bike Sharing Demand (regression / time features)
    - https://www.kaggle.com/competitions/bike-sharing-demand
4. Pima Indians Diabetes Database (classification)
    - https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database
5. Heart Disease UCI (classification)
    - https://www.kaggle.com/datasets/redwankarimsony/heart-disease-data
6. Adult Census Income (classification)

    - https://www.kaggle.com/datasets/uciml/adult-census-income

7. Loan Prediction Problem Dataset (classification)
    - https://www.kaggle.com/datasets/altruistdelhite04/loan-prediction-problem-dataset
8. Palmer Penguins (classification)
    - https://www.kaggle.com/datasets/parulpandey/palmer-archipelago-antarctica-penguin-data
9. Netflix Movies and TV Shows (EDA approfondie ; modélisation optionnelle via NLP/genres)
    - https://www.kaggle.com/datasets/shivamb/netflix-shows
10. Students Performance in Exams (régression/classification/EDA)

- https://www.kaggle.com/datasets/spscientist/students-performance-in-exams

---

Livrables requis (ce que vous devez remettre)

Rendez un **lien vers un dépôt GitHub** contenant tout ce qui est listé ci-dessous.

A. Structure du dépôt (recommandée)

Vous pouvez adapter, mais votre dépôt doit être propre, clair et facile à naviguer.

- `README.md` (TRÈS détaillé ; voir section ci-dessous)
- `data/`
    - `raw/` (optionnel : ne pas committer de très gros fichiers ; petits datasets OK)
    - `processed/` (optionnel)
- `notebooks/` (notebooks d'EDA / analyse exploratoire)
- `src/` (code Python réutilisable : prétraitement, entraînement, évaluation)
- `reports/` ou `figures/` (graphiques/images/tableaux exportés)
- `models/` (artefacts du modèle entraîné ; uniquement si taille raisonnable)
- `requirements.txt` ou `pyproject.toml`
- UI / déploiement optionnels :
    - `app.py` (Streamlit)

B. Le projet doit inclure TOUTES les étapes principales de data science

Votre dépôt doit montrer clairement chaque étape, idéalement via des notebooks/scripts séparés, et le tout doit être documenté dans le README.

1. **Acquisition des données**

    - Expliquer d'où vient le dataset (lien Kaggle)
    - Décrire comment il est téléchargé/chargé
    - Si vous ne commitez pas les données brutes, fournir une méthode reproductible pour les obtenir

2. **Compréhension des données**

    - Dimensions du dataset, signification des colonnes, définition de la cible (target)
    - Types de données, aperçu des valeurs manquantes

3. **Nettoyage & prétraitement**

    - Gérer les valeurs manquantes (justifier les choix)
    - Gérer les valeurs aberrantes (si pertinent)
    - Encoder les variables catégorielles
    - Mettre à l'échelle (scaling) si nécessaire
    - Éviter les fuites de données (data leakage), notamment autour du split train/test

4. **Analyse exploratoire des données (EDA complète attendue)** Vous devez aller au-delà de quelques graphiques.

    - Analyse univariée (distributions, asymétrie, outliers)
    - Analyse bivariée/multivariée (relations, interactions)
    - Analyse de la cible (lien entre les features et la target)
    - Corrélation/association (selon le type de variables)
    - Visualisations avec interprétation (pas seulement des figures)
    - Section « insights » claire : ce que vous avez appris et comment cela a influencé la modélisation

5. **Modélisation (plusieurs modèles requis)** Entraîner et évaluer **au moins 3 modèles différents** adaptés à votre problème. Exemples :

    - Classification : régression logistique, Random Forest, Gradient Boosting, SVM, XGBoost/LightGBM (optionnel), MLP, etc.
    - Régression : Linear/Ridge/Lasso, Random Forest Regressor, Gradient Boosting Regressor, etc.

6. **Explication des modèles (obligatoire)** Fournir des explications sur :

- Ce qu'est chaque modèle, avec vos propres mots
- Pourquoi vous l'avez choisi
- Les hypothèses (si pertinent)
- Les variables les plus importantes (importance des features / coefficients)
- L'analyse d'erreurs (où le modèle échoue, quels groupes/régions/segments)

7. **Optimisation des hyperparamètres (obligatoire)**

    - Utiliser au moins une méthode systématique : `GridSearchCV`, `RandomizedSearchCV`, Optuna (optionnel)
    - Décrire clairement l'espace de recherche (search space) et le justifier
    - Utiliser une stratégie de validation croisée (CV) correcte
    - Comparer et rapporter les résultats « baseline » vs « après tuning »

8. **Évaluation & sélection du modèle**

    - Utiliser des métriques adaptées (ex. accuracy/F1/AUC ; RMSE/MAE/R2)
    - Validation croisée (recommandée) et un jeu de test final (recommandé)
    - Comparer les modèles dans un tableau clair
    - Justifier le choix final

9. **Reproductibilité**

    - Fixer les seeds (aléatoire)
    - Setup d'environnement clair
    - Une commande (ou une séquence de notebooks) permettant de reproduire les résultats

---

Optionnel (bonus) : Déploiement + UI

Vous pouvez, en option, construire une interface simple permettant de faire des prédictions sur de nouvelles entrées.

Approche suggérée : **Streamlit**

- Créer `app.py` qui :
    - Charge votre pipeline/modèle entraîné
    - Propose des champs (widgets) pour les variables d'entrée
    - Affiche la prédiction + un indicateur de confiance/incertitude (si pertinent)
    - Affiche quelques graphiques clés d'EDA ou un résumé du modèle
- Inclure dans le README :
    - Comment lancer l'app en local
    - Captures d'écran / GIF
    - Optionnel : lien hébergé (si vous déployez)

> Restez simple : l'UI n'est pas notée sur le design, mais sur la justesse, la clarté et la reproductibilité.

Exigences pour le README (TRÈS détaillé)

Votre `README.md` doit se lire comme une documentation « professionnelle » de passation de projet.

Inclure au minimum :

- Titre du projet + description courte
- Lien du dataset + ce que vous prédisez / analysez
- Type de problème (classification/régression/etc.)
- Installation et environnement
    - Version de Python
    - Instructions d'installation
- Reproduire les résultats
    - Ordre exact d'exécution des notebooks/scripts
- Résumé EDA (puces + figures clés)
- Résumé de modélisation
    - Modèles entraînés
    - Tableau des métriques
    - Modèle final choisi + justification
- Résumé du tuning d'hyperparamètres
    - Ce que vous avez tuné, pourquoi, et quels gains
- Section analyse d'erreurs
- Explication de la structure du projet
- Limites / pistes d'amélioration
- Références

Exigences d'utilisation de GitHub (la note dépend de cela)

- Votre travail doit être dans un **dépôt GitHub**.
- **De nombreux commits sont attendus** tout au long du projet.
    - Ne faites **pas** un seul gros commit à la fin.
    - Les messages de commit doivent être explicites (ex. « Stratégie valeurs manquantes », « Modèle baseline + métriques », « Tuning RandomForest (randomized search) »).
- La preuve d'un développement itératif compte : un suivi d'expériences (même simple) est encouragé.

Barème indicatif (basé sur le contenu du dépôt GitHub)

Votre note est basée sur ce qui est vérifiable dans le dépôt.

1) Qualité du dépôt & reproductibilité (20%)

- Structure claire, code propre, nommage cohérent
- Reproductibilité de l'environnement (`requirements.txt` / `pyproject.toml`)
- Instructions d'exécution claires
- Pas de fuite de données (data leakage)

2) Qualité & complétude du README (15%)

- Détail, lisibilité, et caractère actionnable
- Contient insights, méthodologie et résultats

3) Profondeur de l'EDA (20%)

- EDA complète avec interprétation statistique/visuelle correcte
- Les insights influencent clairement les choix de modélisation

4) Diversité & rigueur de la modélisation (20%)

- Au moins 3 modèles entraînés et comparés équitablement
- Split train/test correct et/ou validation croisée
- Choix de métriques pertinents

5) Tuning d'hyperparamètres & validation (15%)

- Méthode de tuning systématique
- Stratégie de CV correcte
- Comparaison claire baseline vs tuned

6) Explication & analyse (10%)

- Explications des modèles avec vos propres mots
- Importance des features / coefficients
- Analyse d'erreurs et limites

Bonus (jusqu'à +5%) : UI/déploiement Streamlit propre et bien documenté.

Règles et remarques

- Vous pouvez utiliser des outils d'IA pour vous aider, mais votre dépôt final doit démontrer votre compréhension.
- Le plagiat est interdit : rédigez vos explications vous-même et citez vos sources.
- Restez raisonnables en calcul : pas besoin de gros réseaux de neurones pour cet examen.

Remise

Remettre :

- URL du dépôt GitHub
- (Optionnel) URL de l’application déployée

Assurez-vous que le dépôt est public (ou accessible à l’enseignant) et qu’il contient tous les livrables requis.