📄 Project Charter : "ManaMetrics - The Hybrid Asset Valuator"

Rôle simulé : Lead Data Scientist & ML Engineer **Domaine :** NLP, Regression, MLOps **Stack :** Python, PyTorch, Scikit-Learn, FastAPI, MLflow, Docker

1. Executive Summary

L'objectif est de développer un système d'IA capable de prédire le **prix de marché (EUR/USD)** d'une carte Magic: The Gathering en se basant sur ses caractéristiques intrinsèques (statistiques de jeu) et sémantiques (texte de règles). Le projet démontre la capacité à fusionner des données structurées et non structurées via une architecture **Multi-Modale** (Tabular + NLP), tout en assurant un cycle de vie MLOps complet. Stratégie hybride (ETL Spark -> ML Python)

2. Architecture Technique Globale

Le système est divisé en 4 blocs logiques :

1. **Data Ingestion (ETL) :** Collecte, nettoyage et Feature Engineering.
2. **Model Registry (Training) :** Entraînement des baselines (ML) et des modèles avancés (DL).
3. **Serving Layer :** API pour l'inférence.
4. **Monitoring :** Suivi de la dérive des données (Drift) et performances.

3. Roadmap Détaillée & Implémentation

🔷 Phase 1 : Data Engineering & ETL (Fondations)

Objectif : Constituer un dataset propre et riche ("Gold Layer"). Utiliser PySpark meme si c'est overkill

Tâches :

- **Source de données :** Utiliser l'API Scryfall (plus robuste que magicthegathering.io pour les prix) ou ton API actuelle.
- **Pipeline de nettoyage :**
  - Gestion des NULLs (ex: les cartes non-creatures n'ont pas de "Force/Endurance").
  - Conversion des symboles de mana ({2}{R}{R}) en features numériques (cmc=4, red_devotion=2).
- **Feature Engineering avancé :**
  - *One-Hot Encoding* pour la rareté et les sets.
  - *Date Delta :* Calculer l'âge de la carte (Date actuelle - Date de sortie).
- **Split :** Création stricte de Train / Validation / Test sets (attention à la fuite de données temporelle !).

Livrable : Un fichier processed_data.parquet et un script etl.py.

🔷 Phase 2 : La Baseline "Traditional ML" (Scikit-Learn)

Objectif : Avoir un point de comparaison solide et interprétable. Repasser sur du Python classique.
Tâches :

- Utiliser uniquement les données tabulaires (Coût, Force, Rareté, Année).
- Entraîner un **XGBoost Regressor** ou **Random Forest**.
- Optimisation des hyperparamètres avec GridSearchCV ou Optuna.
- **Interprétabilité :** Générer un graphique **SHAP Values** pour montrer aux "métiers" quelles features influencent le prix (ex: "La rareté Mythic multiplie le prix par 10").

Livrable : Un notebook 01_baseline_ml.ipynb avec les scores R^2 et RMSE.

🔷 Phase 3 : Deep Learning & NLP Fine-Tuning (PyTorch)

Objectif : Capturer la valeur cachée dans le texte des règles ("La sémantique").
C'est ici que tu montres ton expertise **Deep Learning**.
Architecture Hybride (Multi-Input Model) :

1. **Branche A (Texte) :** Un modèle Transformer (ex: DistilBERT ou RoBERTa).
   - *Action :* **Fine-Tuning**. Tu réentraînes les dernières couches du BERT sur le corpus de texte Magic pour qu'il comprenne que "Destroy all creatures" est un effet puissant.
2. **Branche B (Tabulaire) :** Un MLP (Multi-Layer Perceptron) classique pour les stats numériques.
3. **Fusion (Concatenation) :** Les vecteurs de sortie de A et B sont fusionnés.
4. **Tête de prédiction :** Une couche linéaire finale pour prédire le prix (output scalaire).

Formule de Loss (Custom Loss) : Tu peux utiliser une MSE classique, ou une **MSLE (Mean Squared Logarithmic Error)** pour pénaliser moins les erreurs sur les cartes très chères (Black Lotus) et se concentrer sur les cartes standard.
L=N1 i=1∑N (log(yi +1)−log(y^ i +1))2
Livrable : Un script train_hybrid_model.py utilisant PyTorch Lightning (pour la propreté du code).

🔷 Phase 4 : MLOps & Industrialisation

Objectif : Sortir du Notebook ("It works on my machine" n'est pas acceptable).
Tâches :

- **Tracking :** Utiliser **MLflow** pour logger chaque expérimentation (paramètres, courbes de loss, version du dataset).
- **API :** Créer une API **FastAPI** avec un endpoint /predict.
  - *Input :* JSON de la carte.
  - *Output :* Prix estimé + Intervalle de confiance.
- **Docker :** Conteneuriser l'API.

4. Indicateurs de Performance (KPIs)

Pour un expert, il faut distinguer les métriques techniques des métriques "Business".

Type	Indicateur	Objectif / Seuil de succès
Régression	**RMSE** (Root Mean Square Error)	< 2.5€ (Sur les cartes standards)
Régression	**MAPE** (Mean Absolute Percentage Error)	< 15% (L'erreur relative est souvent plus parlante)
Classement	**Top-k Accuracy**	Si on trie les cartes par prix prédit, est-ce que le top 10 réel est dans le top 10 prédit ?
Engineering	**Inference Latency**	< 100ms par requête API

5. Structure du Repo GitHub (Best Practices)

Voici la structure de dossier que je te recommande pour faire pro :
Plaintext

```
ManaMetrics/
├── .github/workflows/   # CI/CD (Tests automatiques)
├── data/
│   ├── raw/             # Données brutes API
│   └── processed/       # Données nettoyées (.parquet)
├── models/              # Binaires des modèles
(sauvegardés)
├── notebooks/           # Exploration & Preuves de concept
├── src/                 # Code source modulaire
│   ├── data/            # Scripts ETL
│   ├── features/        # Transformation des features
│   ├── models/          # Architectures PyTorch & Sklearn
│   └── serving/         # Code de l'API FastAPI
├── tests/               # Unit tests (Pytest)
├── Dockerfile
├── pyproject.toml       # Gestion des dépendances (Poetry
ou pip)
├── MLproject            # Config MLflow
└── README.md            # La vitrine de ton projet
```