PDF 4 : Mise en place du Pipeline PMU en temps réel

1. Objectif du Pipeline

Mettre en place un flux automatisé pour collecter les données des courses PMU en temps réel (cotes, météo, non-partants), les stocker dans une base de données locale (SQLite) et calculer automatiquement les mises optimales (Kelly) en fonction des cotes actualisées.

2. Environnement technique

- **OS** : Windows 11
- **Python** : 3.12.3
- **Éditeur** : VS Code
- **Bibliothèques** :
  - `requests`
  - `beautifulsoup4`
  - `pandas`
  - `matplotlib`
- **Base de données** : SQLite (`donnees_courses.db`)

3. Étapes détaillées

Étape 1 : Création de l'environnement virtuel

```
cd %HOMEPATH%\Desktop\Pmu-pipeline
python -m venv venv
.\venv\Scripts\activate.bat
pip install --upgrade pip
pip install requests beautifulsoup4 pandas matplotlib
```

Étape 2 : Script de collecte en temps réel (`live_collector.py`)

- Collecte les cotes de Tuyaux-Turf, météo OpenWeatherMap, et non-partants Zone-Turf
- Stockage dans SQLite toutes les 30 secondes

Structure base SQLite :

```
CREATE TABLE courses (
    ts TEXT, num_cheval TEXT, nom_cheval TEXT, cote REAL,
    non_partant INTEGER, meteo_desc TEXT, temp REAL, vent REAL,
humidite INTEGER
)
```

Commande lancement :

```
python live_collector.py
```

Étape 3 : Validation du Pipeline

- Vérification données enregistrées : extraction CSV

```
import sqlite3, pandas as pd
df = pd.read_sql("SELECT * FROM courses ORDER BY ts DESC",
sqlite3.connect("donnees_courses.db"))
df.to_csv("historique_cotes.csv", index=False)
```

- Visualisation rapide des cotes (matplotlib) :

```
import sqlite3, pandas as pd, matplotlib.pyplot as plt
CHEVAL = "3"
q = f"""SELECT ts, cote FROM courses WHERE num_cheval='{CHEVAL}'
ORDER BY ts"""
df = pd.read_sql(q, sqlite3.connect("donnees_courses.db"),
parse_dates=["ts"])
plt.plot(df["ts"], df["cote"])
plt.xlabel("Temps")
plt.ylabel("Cote")
plt.title(f"Cote du cheval {CHEVAL}")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Étape 4 : Calcul automatisé de la mise optimale (Kelly)

Script `ingest_latest.py` (dans dossier Nova_v5)

```
import sqlite3, pandas as pd
from math import floor

DB = r"..\donnees_courses.db"
CHEVAL = "3"
BANKROLL = 100.0
EDGE = 0.05

def latest_snapshot(num: str):
    df = pd.read_sql(
        f"""SELECT ts, cote FROM courses WHERE num_cheval='{num}'
AND ts=(SELECT MAX(ts) FROM courses)""",
        sqlite3.connect(DB)
    )
    return None if df.empty else df.iloc[0]

snap = latest_snapshot(CHEVAL)
if snap is None:
    print(f"Aucune cote pour le cheval {CHEVAL}")
else:
    ts, cote = snap
    b = cote - 1
    p = 1 / cote
    p_edge = p + EDGE
    kelly = max(0, (b * p_edge - (1 - p_edge)) / b)
    stake = floor(BANKROLL * kelly)

    print(f"[{ts}] Cheval {CHEVAL} – cote {cote}")
    print(f"Kelly {kelly:.3f} → mise conseillée ≈ {stake} €")
```

4. Fonctionnement en pratique

- Terminal 1 : collecte des données

```
python live_collector.py
```

- Terminal 2 : calcul de la mise optimale juste avant la course

```
python ingest_latest.py
```

5. Ajustements pratiques

- Si l'opérateur PMU n'autorise que les mises par incréments d'1€ :

```
BET_STEP = 1.0
stake = floor((BANKROLL * kelly) / BET_STEP) * BET_STEP
```

Toujours arrondir vers le bas pour respecter la sécurité Kelly.

6. Résumé et prochaines étapes

Le Pipeline PMU ainsi créé permet une gestion optimale et rationnelle des mises en temps réel. Il constitue un socle solide pour des améliorations futures : intégration d'un modèle prédictif plus avancé, gestion multi-courses, et création d'un tableau de bord interactif.