TieBreaker IA — Guide du projet pour le 24 octobre

1) En bref

TieBreaker est une IA Python qui prédit l'issue des matchs ATP (WTA à venir) à partir d'un historique 1968→aujourd'hui. L'accent est mis sur un pipeline propre, reproductible et sans fuite : parser CLI, dataset A/B, features de forme récente, exécutable POSIX.

2) Ce qui est déjà prêt ✓

• CLI : interroger classements et matches en détail depuis la ligne de commande (rank/match).
• Exécutable POSIX : `./TieBreaker` à la racine (généré par `main.py`).
• Builder A/B : A = mieux classé à la date du match, y = 1 si A gagne ; features de base (ranks/points/âges, surface/round, best_of).
• Features "forme récente" (rolling N=20) : win rate, 1re balle, points gagnés 1re/2e, aces par jeu de service, variantes par surface.

3) Installation & prérequis

• Python 3.10+ recommandé.
• Dépendances principales : pandas, numpy (XGBoost sera ajouté à l'étape suivante).
• Installer depuis `requirements.txt`t :
pip install -r requirements.txt

4) Démarrage □

4.1 Générer l'exécutable
python src/main.py build
./TieBreaker rank --player "Novak Djokovic"

-> Novak Djokovic — Rang ATP 7 (3910 pts) au 2024-12-30

4.2 Retrouver un match précis
./TieBreaker match --p1 "Carlos Alcaraz" --p2 "Novak Djokovic" --year 2023 --tournament Wimbledon --round F

-> 2023-07-03 — Wimbledon (Grass) | Best-of-5 | Carlos Alcaraz def. Novak Djokovic 1-6 7-6(6) 6-1 3-6 6-4 (283 min)

4.3 Construire le dataset A/B (exemples)
python -m src.build_dataset --data-root data --years 2023 2024 --add-recent-form --out data/processed/dataset_outcome.parquet
python -m src.build_dataset --data-root data --all-years --min-year 1990 --limit 100000

5) Commandes & options essentielles
5.1 `./TieBreaker` (CLI)
• `rank` :
./TieBreaker rank --player "<Nom>" [--date YYYY-MM-DD] [--data-root <chemin>]
• `match` :
./TieBreaker match --p1 "<A>" --p2 "<B>" [--year YYYY] [--tournament STR] [--round F|SF|QF|R16|R32|R64|R128] [--surface Hard|Clay|Grass|Carpet] [--date YYYY-MM-DD] [--all-years] [--data-root <chemin>]

5.2 `src/build_dataset.py`

--data-root, --years, --all-years, --min-year, --max-year, --limit, --out
--add-recent-form, --lookback-matches (20 par défaut), --min-recent-matches (10 par défaut)

5.3 `main.py` (builder POSIX)
python main.py build # crée ./TieBreaker à la racine
python main.py clean # supprime ./TieBreaker
[option] --project-root <chemin> pour forcer la racine du projet

6) Notions clés

6.1 Dataset A vs B (vulgarisé)

• Comment : on range les joueurs dans un ordre neutre (A = mieux classé ≤ date du match), puis on apprend y = 1 si A gagne.
• Avantages : pas de fuite, une seule ligne par match, features “diff A−B” faciles à interpréter (`rank_diff`, etc.).

6.2 Features de forme récente (N=20)

• Calculées par joueur sur ses N derniers matchs strictement avant la date cible (anti-fuite par `shift`).
• Exemples : `win_rate_20`, `first_in_pct_20`, `first_won_pct_20`, `second_won_pct_20`, `aces_per_SvGm_20`, `df_per_SvGm_20`, + versions par surface.
• Intégration : seules les **différences A−B** sont ajoutées au dataset final, + flags `recent_form_missing_A/B`.

7) Prochaine étape : XGBoost

• Objectif : entraîner un modèle proba `P(A gagne)` avec split temporel strict (train ≤ 2021, val 2022–2023, test 2024+).
• Modèle : XGBoost.
• Métriques : AUC, LogLoss, Brier ; sauvegarde `models/outcome_model.pkl` + `reports/train_metrics.json`.
• Étape d’après : commande `predict` qui charge le modèle et renvoie la proba pour un duel (p1, p2, contexte).