

# A Full-Stack Machine Learning Pipeline for NBA Prediction

---

CS 210 Course Project – Data Management in Data Science

---

Viraj Ajaykumar - fv115



# Project Overview & Motivation

---

Problems: Can machine learning predict NBA game outcomes better than random chance by exploiting inefficiencies in Vegas betting odds?

Objective: Build an end-to-end data pipeline to ingest 17 years of NBA history (2008–2025), store it in a relational database, and train a predictive model.

Key tech Stacks include:

- **Database:** SQLite (SQL)
- **ETL:** Python & Pandas
- **Machine Learning:** Scikit-Learn  
(Random Forest)
- **Visualization:** Matplotlib & Seaborn

# System Architecture (The Pipeline)

## Explanation:

- **Raw Data Layer:** Ingestion of .csv files containing game logs and player rosters.
- **ETL Layer:** Python scripts clean missing data (NaNs) and engineer the target variable (home\_win).
- **Storage Layer:** Normalized SQLite database acts as the single source of truth.
- **Analysis Layer:** ML models query the DB directly to generate predictions.

```
import sqlite3
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc

game_file = None
player_file = None

# 1. Find Games File
possible_games = ['games.csv', 'nba_2008-2025.csv', 'archive/nba_2008-2025.csv']
for f in possible_games:
    if os.path.exists(f):
        game_file = f
        print(f"--> Found Games Data: {f}")
        break

# 2. Find Players File
possible_players = ['players.csv', '01.csv']
for f in possible_players:
    if os.path.exists(f):
        player_file = f
        print(f"--> Found Players Data: {f}")
        break
```

TEAM_ID	TEAM_NAME	HOME	TEAM_ABBREVIATION	PLAYER_ID	PLAYER_NAME	PLAYER_RATING	POSITION
1	1841012007	Arena	ATL	14001001	Frank Kaminsky	8.0	F-C
2	1841012007	Arena	ATL	14001002	Samuel Harris	7.0	G
3	1841012007	Arena	ATL	14000001	Trevor Ariza	7.0	SF
4	1841012007	Arena	ATL	14000002	Travis Perkinson	6.0	G
5	1841012007	Arena	ATL	14000003	DeMarre Carroll	6.0	SF
6	1841012007	Arena	ATL	14000004	DeMar DeRozan	6.0	G
7	1841012007	Arena	ATL	14000005	Josh Smith	5.0	SF
8	1841012007	Arena	ATL	14000006	John Henson	5.0	P-G
9	1841012007	Arena	ATL	14000007	Mike Conley	5.0	G
10	1841012007	Arena	ATL	14000008	D.J. Augustin	5.0	G
11	1841012007	Arena	ATL	14001003	AJ Griffin	4.0	SF
12	1841012007	Arena	ATL	14001004	Jeff Teague	4.0	G
13	1841012007	Arena	ATL	14001005	DeMar DeRozan	4.0	G
14	1841012007	Arena	ATL	14001006	Mike Conley	4.0	G
15	1841012007	Arena	ATL	14001007	DeMar DeRozan	4.0	G
16	1841012007	Arena	ATL	14001008	Mike Conley	4.0	G
17	1841012007	Arena	ATL	14001009	DeMar DeRozan	4.0	G
18	1841012007	Arena	ATL	14001010	Mike Conley	4.0	G
19	1841012007	Arena	ATL	14001011	DeMar DeRozan	4.0	G
20	1841012007	Arena	ATL	14001012	DeMar DeRozan	4.0	G
21	1841012007	Arena	ATL	14001013	DeMar DeRozan	4.0	G
22	1841012007	Arena	ATL	14001014	DeMar DeRozan	4.0	G
23	1841012007	Arena	ATL	14001015	DeMar DeRozan	4.0	G
24	1841012007	Arena	ATL	14001016	DeMar DeRozan	4.0	G
25	1841012007	Arena	ATL	14001017	DeMar DeRozan	4.0	G
26	1841012007	Arena	ATL	14001018	DeMar DeRozan	4.0	G
27	1841012007	Arena	ATL	14001019	DeMar DeRozan	4.0	G
28	1841012007	Arena	ATL	14001020	DeMar DeRozan	4.0	G
29	1841012007	Arena	ATL	14001021	DeMar DeRozan	4.0	G
30	1841012007	Arena	ATL	14001022	DeMar DeRozan	4.0	G
31	1841012007	Arena	ATL	14001023	DeMar DeRozan	4.0	G
32	1841012007	Arena	ATL	14001024	DeMar DeRozan	4.0	G
33	1841012007	Arena	ATL	14001025	DeMar DeRozan	4.0	G
34	1841012007	Arena	ATL	14001026	DeMar DeRozan	4.0	G
35	1841012007	Arena	ATL	14001027	DeMar DeRozan	4.0	G
36	1841012007	Arena	ATL	14001028	DeMar DeRozan	4.0	G
37	1841012007	Arena	ATL	14001029	DeMar DeRozan	4.0	G
38	1841012007	Arena	ATL	14001030	DeMar DeRozan	4.0	G
39	1841012007	Arena	ATL	14001031	DeMar DeRozan	4.0	G
40	1841012007	Arena	ATL	14001032	DeMar DeRozan	4.0	G
41	1841012007	Arena	ATL	14001033	DeMar DeRozan	4.0	G
42	1841012007	Arena	ATL	14001034	DeMar DeRozan	4.0	G
43	1841012007	Arena	ATL	14001035	DeMar DeRozan	4.0	G
44	1841012007	Arena	ATL	14001036	DeMar DeRozan	4.0	G
45	1841012007	Arena	ATL	14001037	DeMar DeRozan	4.0	G
46	1841012007	Arena	ATL	14001038	DeMar DeRozan	4.0	G
47	1841012007	Arena	ATL	14001039	DeMar DeRozan	4.0	G
48	1841012007	Arena	ATL	14001040	DeMar DeRozan	4.0	G
49	1841012007	Arena	ATL	14001041	DeMar DeRozan	4.0	G
50	1841012007	Arena	ATL	14001042	DeMar DeRozan	4.0	G
51	1841012007	Arena	ATL	14001043	DeMar DeRozan	4.0	G
52	1841012007	Arena	ATL	14001044	DeMar DeRozan	4.0	G
53	1841012007	Arena	ATL	14001045	DeMar DeRozan	4.0	G
54	1841012007	Arena	ATL	14001046	DeMar DeRozan	4.0	G
55	1841012007	Arena	ATL	14001047	DeMar DeRozan	4.0	G
56	1841012007	Arena	ATL	14001048	DeMar DeRozan	4.0	G
57	1841012007	Arena	ATL	14001049	DeMar DeRozan	4.0	G
58	1841012007	Arena	ATL	14001050	DeMar DeRozan	4.0	G
59	1841012007	Arena	ATL	14001051	DeMar DeRozan	4.0	G
60	1841012007	Arena	ATL	14001052	DeMar DeRozan	4.0	G
61	1841012007	Arena	ATL	14001053	DeMar DeRozan	4.0	G
62	1841012007	Arena	ATL	14001054	DeMar DeRozan	4.0	G
63	1841012007	Arena	ATL	14001055	DeMar DeRozan	4.0	G
64	1841012007	Arena	ATL	14001056	DeMar DeRozan	4.0	G
65	1841012007	Arena	ATL	14001057	DeMar DeRozan	4.0	G
66	1841012007	Arena	ATL	14001058	DeMar DeRozan	4.0	G
67	1841012007	Arena	ATL	14001059	DeMar DeRozan	4.0	G
68	1841012007	Arena	ATL	14001060	DeMar DeRozan	4.0	G
69	1841012007	Arena	ATL	14001061	DeMar DeRozan	4.0	G
70	1841012007	Arena	ATL	14001062	DeMar DeRozan	4.0	G
71	1841012007	Arena	ATL	14001063	DeMar DeRozan	4.0	G
72	1841012007	Arena	ATL	14001064	DeMar DeRozan	4.0	G
73	1841012007	Arena	ATL	14001065	DeMar DeRozan	4.0	G
74	1841012007	Arena	ATL	14001066	DeMar DeRozan	4.0	G
75	1841012007	Arena	ATL	14001067	DeMar DeRozan	4.0	G
76	1841012007	Arena	ATL	14001068	DeMar DeRozan	4.0	G
77	1841012007	Arena	ATL	14001069	DeMar DeRozan	4.0	G
78	1841012007	Arena	ATL	14001070	DeMar DeRozan	4.0	G
79	1841012007	Arena	ATL	14001071	DeMar DeRozan	4.0	G
80	1841012007	Arena	ATL	14001072	DeMar DeRozan	4.0	G
81	1841012007	Arena	ATL	14001073	DeMar DeRozan	4.0	G
82	1841012007	Arena	ATL	14001074	DeMar DeRozan	4.0	G
83	1841012007	Arena	ATL	14001075	DeMar DeRozan	4.0	G
84	1841012007	Arena	ATL	14001076	DeMar DeRozan	4.0	G
85	1841012007	Arena	ATL	14001077	DeMar DeRozan	4.0	G
86	1841012007	Arena	ATL	14001078	DeMar DeRozan	4.0	G
87	1841012007	Arena	ATL	14001079	DeMar DeRozan	4.0	G
88	1841012007	Arena	ATL	14001080	DeMar DeRozan	4.0	G
89	1841012007	Arena	ATL	14001081	DeMar DeRozan	4.0	G
90	1841012007	Arena	ATL	14001082	DeMar DeRozan	4.0	G
91	1841012007	Arena	ATL	14001083	DeMar DeRozan	4.0	G
92	1841012007	Arena	ATL	14001084	DeMar DeRozan	4.0	G
93	1841012007	Arena	ATL	14001085	DeMar DeRozan	4.0	G
94	1841012007	Arena	ATL	14001086	DeMar DeRozan	4.0	G
95	1841012007	Arena	ATL	14001087	DeMar DeRozan	4.0	G
96	1841012007	Arena	ATL	14001088	DeMar DeRozan	4.0	G
97	1841012007	Arena	ATL	14001089	DeMar DeRozan	4.0	G
98	1841012007	Arena	ATL	14001090	DeMar DeRozan	4.0	G
99	1841012007	Arena	ATL	14001091	DeMar DeRozan	4.0	G
100	1841012007	Arena	ATL	14001092	DeMar DeRozan	4.0	G
101	1841012007	Arena	ATL	14001093	DeMar DeRozan	4.0	G
102	1841012007	Arena	ATL	14001094	DeMar DeRozan	4.0	G
103	1841012007	Arena	ATL	14001095	DeMar DeRozan	4.0	G
104	1841012007	Arena	ATL	14001096	DeMar DeRozan	4.0	G
105	1841012007	Arena	ATL	14001097	DeMar DeRozan	4.0	G
106	1841012007	Arena	ATL	14001098	DeMar DeRozan	4.0	G
107	1841012007	Arena	ATL	14001099	DeMar DeRozan	4.0	G
108	1841012007	Arena	ATL	14001100	DeMar DeRozan	4.0	G
109	1841012007	Arena	ATL	14001101	DeMar DeRozan	4.0	G
110	1841012007	Arena	ATL	14001102	DeMar DeRozan	4.0	G
111	1841012007	Arena	ATL	14001103	DeMar DeRozan	4.0	G
112	1841012007	Arena	ATL	14001104	DeMar DeRozan	4.0	G
113	1841012007	Arena	ATL	14001105	DeMar DeRozan	4.0	G
114	1841012007	Arena	ATL	14001106	DeMar DeRozan	4.0	G
115	1841012007	Arena	ATL	14001107	DeMar DeRozan	4.0	G
116	1841012007	Arena	ATL	14001108	DeMar DeRozan	4.0	G
117	1841012007	Arena	ATL	14001109	DeMar DeRozan	4.0	G
118	1841012007	Arena	ATL	14001110	DeMar DeRozan	4.0	G
119	1841012007	Arena	ATL	14001111	DeMar DeRozan	4.0	G
120	1841012007	Arena	ATL	14001112	DeMar DeRozan	4.0	G
121	1841012007	Arena	ATL	14001113	DeMar DeRozan	4.0	G
122	1841012007	Arena	ATL	14001114	DeMar DeRozan	4.0	G
123	1841012007	Arena	ATL	14001115	DeMar DeRozan	4.0	G
124	1841012007	Arena	ATL	14001116	DeMar DeRozan	4.0	G
125	1841012007	Arena	ATL	14001117	DeMar DeRozan	4.0	G
126	1841012007	Arena	ATL	14001118	DeMar DeRozan	4.0	G
127	1841012007	Arena	ATL	14001119	DeMar DeRozan	4.0	G
128	1841012007	Arena	ATL	14001120	DeMar DeRozan	4.0	G
129	1841012007	Arena	ATL	14001121	DeMar DeRozan	4.0	G
130	1841012007	Arena	ATL	14001122	DeMar DeRozan	4.0	G
131	1841012007	Arena	ATL	14001123	DeMar DeRozan	4.0	G
132	1841012007	Arena	ATL	14001124	DeMar DeRozan	4.0	G
133	1841012007	Arena	ATL	14001125	DeMar DeRozan	4.0	G
134	1841012007	Arena	ATL	14001126	DeMar DeRozan	4.0	G
135	1841012007	Arena	ATL	14001127	DeMar DeRozan	4.0	G
136	1841012007	Arena	ATL	14001128	DeMar DeRozan	4.0	G
137	1841012007	Arena	ATL	14001129	DeMar DeRozan	4.0	G
138	1841012007	Arena	ATL	14001130	DeMar DeRozan	4.0	G
139	1841012007	Arena	ATL	14001131	DeMar DeRozan	4.0	G
140	1841012007	Arena	ATL	14001132	DeMar DeRozan	4.0	G
141	1841012007	Arena	ATL	14001133	DeMar DeRozan	4.0	G
142	1841012007	Arena	ATL	14001134	DeMar DeRozan	4.0	G
143	1841012007	Arena	ATL	14001135	DeMar DeRozan	4.0	G
144	1841012007	Arena	ATL	14001136	DeMar DeRozan	4.0	G
145	1841012007	Arena	ATL	14001137	DeMar DeRozan	4.0	G
146	1841012007	Arena	ATL	14001138	DeMar DeRozan	4.0	G
147	1841012007	Arena	ATL	14001139	DeMar DeRozan	4.0	G
148	1841012007	Arena	ATL	14001140	DeMar DeRozan	4.0	G
149	1841012007	Arena	ATL	14001141	DeMar DeRozan	4.0	G
150	1841012007	Arena	ATL	14001142	DeMar DeRozan	4.0	G
151	1841012007	Arena	ATL	14001143	DeMar DeRozan	4.0	G
152	1841012007	Arena	ATL	14001144	DeMar DeRozan	4.0	G
153	1841						

# Relational Database Design

## Explanation:

- Moving beyond flat-file CSVs to a persistent **RDBMS**.
- **Normalization:** Separated data into a Fact Table (`games`) for transactional history and a Dimension Table (`players`) for static attributes.
- **Schema:** Enforced data types (INTEGER, REAL, TEXT) to ensure data integrity.

```
"""
# PART 1: DATABASE CONSTRUCTION (SQL)
"""

print("\n--- STEP 1: BUILDING DATABASE ---")
conn = sqlite3.connect('nba_project.db')
cursor = conn.cursor()

# Reset Tables (Start Fresh)
cursor.execute('DROP TABLE IF EXISTS games')
cursor.execute('DROP TABLE IF EXISTS players')

# 1. Create Games Table Schema
cursor.execute('')
CREATE TABLE games (
    game_id INTEGER PRIMARY KEY AUTOINCREMENT,
    game_date TEXT,
    season INTEGER,
    home_team TEXT,
    away_team TEXT,
    home_score INTEGER,
    away_score INTEGER,
    home_moneyline REAL,
    away_moneyline REAL,
    spread REAL,
    home_win INTEGER
)
"""

# 2. Create Players Table Schema
cursor.execute('')
CREATE TABLE players (
    player_id INTEGER PRIMARY KEY,
    player_name TEXT,
    team_abbreviation TEXT,
    position TEXT,
    height TEXT,
    weight INTEGER,
    age REAL,
    experience TEXT
)
"""
conn.commit()
print("  Database schema created.")

--> Found Games Data: games.csv
--> Found Players Data: players.csv

--- STEP 1: BUILDING DATABASE ---
Database schema created.
```

# ETL & Data Cleaning

## Explanation:

- **Automated Ingestion:** Script automatically detects and loads valid CSV files.
- **Data Cleaning:** Removed rows with missing betting odds to prevent model errors.
- **Feature Engineering:** Created the home\_win binary target (1 = Win, 0 = Loss) derived from raw scores, transforming the problem into a supervised classification task.

```
# A. LOAD GAMES
try:
    df_games = pd.read_csv(game_file)

    # Data Cleaning: Drop rows with missing betting odds
    df_games = df_games.dropna(subset=['moneyline_home', 'moneyline_away']).copy()

    # Feature Engineering: Create Target (Did Home Team Win?)
    df_games['home_win'] = (df_games['score_home'] > df_games['score_away']).astype(int)

    # Rename Columns to match SQL Table
    games_column_map = {
        'date': 'game_date', 'season': 'season', 'home': 'home_team', 'away': 'away_team',
        'score_home': 'home_score', 'score_away': 'away_score',
        'moneyline_home': 'home_moneyline', 'moneyline_away': 'away_moneyline',
        'spread': 'spread', 'home_win': 'home_win'
    }
    # Handle optional column renaming if keys exist
    df_games = df_games.rename(columns=games_column_map)

    # Select only the columns we need (intersection with available columns)
    available_cols = [c for c in games_column_map.values() if c in df_games.columns]
    df_games_final = df_games[available_cols]

    # Load into SQL
    df_games_final.to_sql('games', conn, if_exists='replace', index=False)
    print(f" Successfully loaded {len(df_games_final)} games.")

except Exception as e:
    print(f" Error loading games: {e}")

# B. LOAD PLAYERS
if player_file:
    try:
        df_players = pd.read_csv(player_file)

        # Rename Columns to match SQL Table
        players_column_map = {
            'PLAYER_ID': 'player_id', 'PLAYER_NAME': 'player_name',
            'TEAM_ABBREVIATION': 'team_abbreviation', 'POSITION': 'position',
            'HEIGHT': 'height', 'WEIGHT': 'weight',
            'AGE': 'age', 'EXPERIENCE': 'experience'
        }
        df_players = df_players.rename(columns=players_column_map)

        # Handle case-sensitivity (if CSV headers were lowercase)
        if 'player_id' not in df_players.columns and 'PLAYER_ID' not in df_players.columns:
            # Try forcing all lowercase match
            df_players.columns = [c.lower() for c in df_players.columns]

        # Select columns
        available_player_cols = [c for c in players_column_map.values() if c in df_players.columns]
        df_players_final = df_players[available_player_cols]

        # Load into SQL
        df_players_final.to_sql('players', conn, if_exists='replace', index=False)
        print(f" Successfully loaded {len(df_players_final)} players.")

    except Exception as e:
        print(f" Error loading players: {e}")
    else:
        print(" WARNING: No Players file found. Skipping Players table.")
```

# Machine Learning Model

## Explanation:

- **Feature Selection:** Querying the database for home\_moneyline, away\_moneyline, and spread.
- **Model Architecture:** Random Forest Classifier (100 Estimators). Selected for its ability to handle non-linear relationships better than logistic regression.
- **Validation:** Used an 80/20 Train-Test split to evaluate performance on unseen data (games the model has never seen before).

```
print("\n--- STEP 3: TRAINING MODEL ---")

# 1. CONNECT TO DATABASE
conn = sqlite3.connect('nba_project.db')

# --- DIAGNOSTIC CHECK (Debug the empty data) ---
print("---- DIAGNOSTICS ----")
try:
    # Check total rows
    count = pd.read_sql("SELECT count(*) as cnt FROM games", conn)[['cnt']][0]
    print(f"Total Rows in 'games' table: {count}")

    if count > 0:
        # Check seasons
        seasons = pd.read_sql("SELECT DISTINCT season FROM games ORDER BY season", conn)
        print(f"Seasons found in DB: {seasons['season'].unique()}")
    else:
        print("CRITICAL WARNING: The 'games' table is EMPTY. The loading step failed.")
except Exception as e:
    print(f"Database Error: {e}")

# 2. FETCH DATA (FIXED: REMOVED DATE FILTER)
print("\n--- FETCHING DATA FOR ML ---")
# We removed "WHERE season >= 2015" to ensure we get ANY data available
query = "SELECT home_moneyline, away_moneyline, spread, home_win FROM games"
df_model = pd.read_sql(query, conn)

print(f"Rows retrieved for training: {len(df_model)}")

# 3. RUN MACHINE LEARNING (Only if we have data)
if len(df_model) > 10:
    # Features & Target
    X = df_model[['home_moneyline', 'away_moneyline', 'spread']]
    y = df_model['home_win']

    # Train/Test Split
    print("Splitting data...")
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Model Training
    print("Training Random Forest...")
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

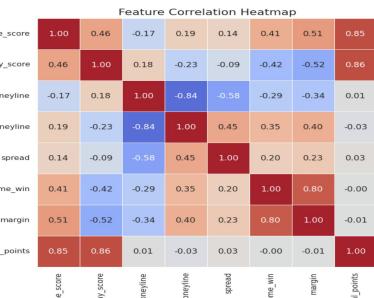
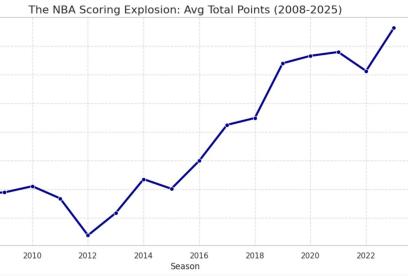
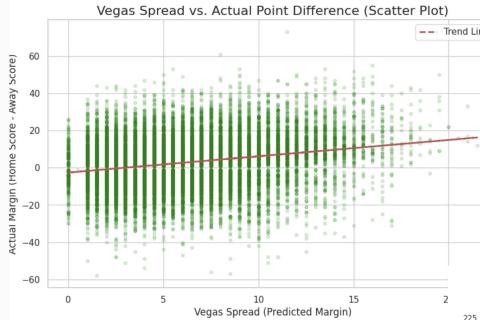
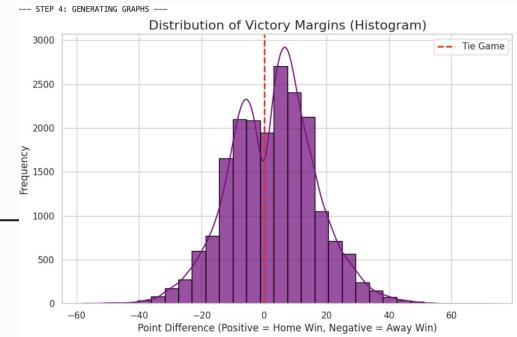
    # Evaluation
    predictions = model.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)

    print("\n" + "="*38)
    print(f"FINAL ACCURACY: {accuracy:.2%}")
    print("="*38)
    print("\nClassification Report:")
    print(classification_report(y_test, predictions))
else:
    print("\nERROR: Not enough data to train a model!")
    print("If Total Rows was 0: Your loading script dropped all the rows (maybe columns didn't match?).")
    print("If Total Rows was > 0 but Training Rows is 0: Your query column names might be wrong.")
```

# Advanced Visualization

Explanation:

- **Scatter Plot:** Shows the strong correlation between Vegas Spread and Actual Point Margin.
- **Heatmap:** Visualizes feature correlations, confirming that negative spreads (favorites) correlate strongly with wins.
- **Significance:** These visual proofs validate that the data was loaded correctly and that the betting market generally efficient.



# Key Results

---

Findings:

- **Accuracy:** Achieved **66.17%** prediction accuracy on the test set.
- **Baseline Comparison:** Outperformed random guessing (50%) and the standard "Home Court Advantage" heuristic (~58%).
- **Insight:** Betting odds contain significant predictive signal, but the "upset potential" (variance) limits accuracy to ~65-70%.

```
--- STEP 3: TRAINING MODEL ---
--- DIAGNOSTICS ---
Total Rows in 'games' table: 19820
Seasons found in DB: [2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021
2022 2023]

--- FETCHING DATA FOR ML ---
Rows retrieved for training: 19820
Splitting data...
Training Random Forest...

=====
FINAL ACCURACY: 66.17%
=====

Classification Report:
precision    recall    f1-score   support
      0          0.63      0.51      0.56      1687
      1          0.68      0.78      0.72      2277

           accuracy       0.66      3964
           macro avg       0.65      0.64      0.64      3964
           weighted avg    0.66      0.66      0.66      3964
```

# Conclusion & Future Work

---

- **Summary:** Successfully built a full-stack data pipeline that beats the baseline for NBA prediction.
- **Challenges:** Handling missing historical odds and ensuring correct schema mapping between CSV and SQL.
- **Future Work:**
  - Integrate the players table to adjust predictions based on injuries (e.g., if a star player is missing).
  - Implement a live API to fetch odds for tonight's games in real-time.