

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
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
TEAM_ID	1610612737	1610612738	1610612739	1610612740	1610612741	1610612742	1610612743	1610612744	1610612745	1610612746	1610612747	1610612748	1610612749	1610612750	1610612751	1610612752	1610612753
TEAM_NAME	Atlanta	Boston	Brooklyn	Charlotte	Chicago	Cleveland	Dallas	Denver	Detroit	Golden State	Houston	Indiana	Lakers	Memphis	Miami	Minnesota	New York
TEAM_ABBREVIATION	ATL	BOS	BKN	CHA	CHI	CLE	DAL	DEN	DET	GSW	HOU	IND	LAL	MEM	MIA	MIN	NYK
PLAYER_ID	1610612737	1610612738	1610612739	1610612740	1610612741	1610612742	1610612743	1610612744	1610612745	1610612746	1610612747	1610612748	1610612749	1610612750	1610612751	1610612752	1610612753
PLAYER_NAME	Paul George	Alvin Robertson	John Collins	Frank Kaminsky	Devin Harris	Jeff Teague	Devin Booker	Michael Carter-Williams	Andre Drummond	Stephen Curry	James Harden	Victor Oladipo	Anthony Davis	Ja Morant	Jimmy Butler	Karl-Anthony Towns	Joel Embiid
PLAYER_POINTS	18.1	18.1	18.1	18.1	18.1	18.1	18.1	18.1	18.1	18.1	18.1	18.1	18.1	18.1	18.1	18.1	18.1
POSITION	PG	PG	PG	PG	PG	PG	PG	PG	PG	PG	PG	PG	PG	PG	PG	PG	PG

	season	date	regular	playoffs	away	home	all
1	2008	2007-10-30	True	Falco	yes	no	no
2	2008	2007-10-30	True	Falco	abs	no	no
3	2008	2007-10-30	True	Falco	abs	no	no
4	2008	2007-10-31	True	Falco	ph	no	no
5	2008	2007-10-31	True	Falco	ph	ind	no
6	2008	2007-10-31	True	Falco	no	no	no
7	2008	2007-10-31	True	Falco	ch	ba	no
8	2008	2007-10-31	True	Falco	de	de	no
9	2008	2007-10-31	True	Falco	ba	man	no
10	2008	2007-10-31	True	Falco	no	ba	no
11	2008	2007-10-31	True	Falco	oc	ba	no
12	2008	2007-11-01	True	Falco	de	ma	no
13	2008	2007-11-01	True	Falco	no	ma	no
14	2008	2007-11-01	True	Falco	ph	ba	no
15	2008	2007-11-02	True	Falco	no	ch	no
16	2008	2007-11-02	True	Falco	de	ut	no

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 to 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)

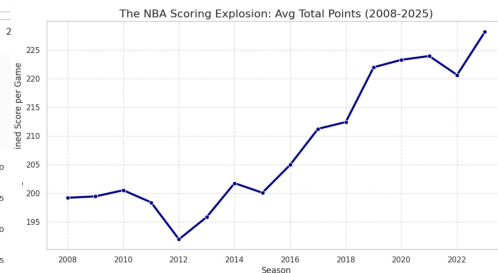
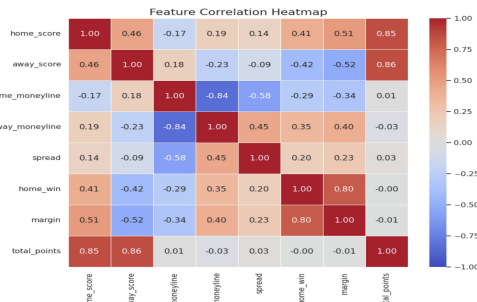
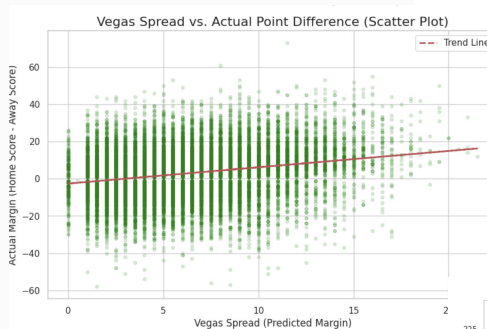
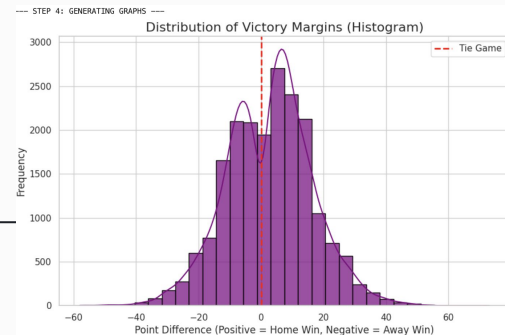
    # Evaluation
    predictions = model.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)

    print("\n" + "="*30)
    print(f"FINAL ACCURACY: {accuracy:.2%}")
    print("="*30)
    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.