Final Project: Final Project

Brandon Trinkle

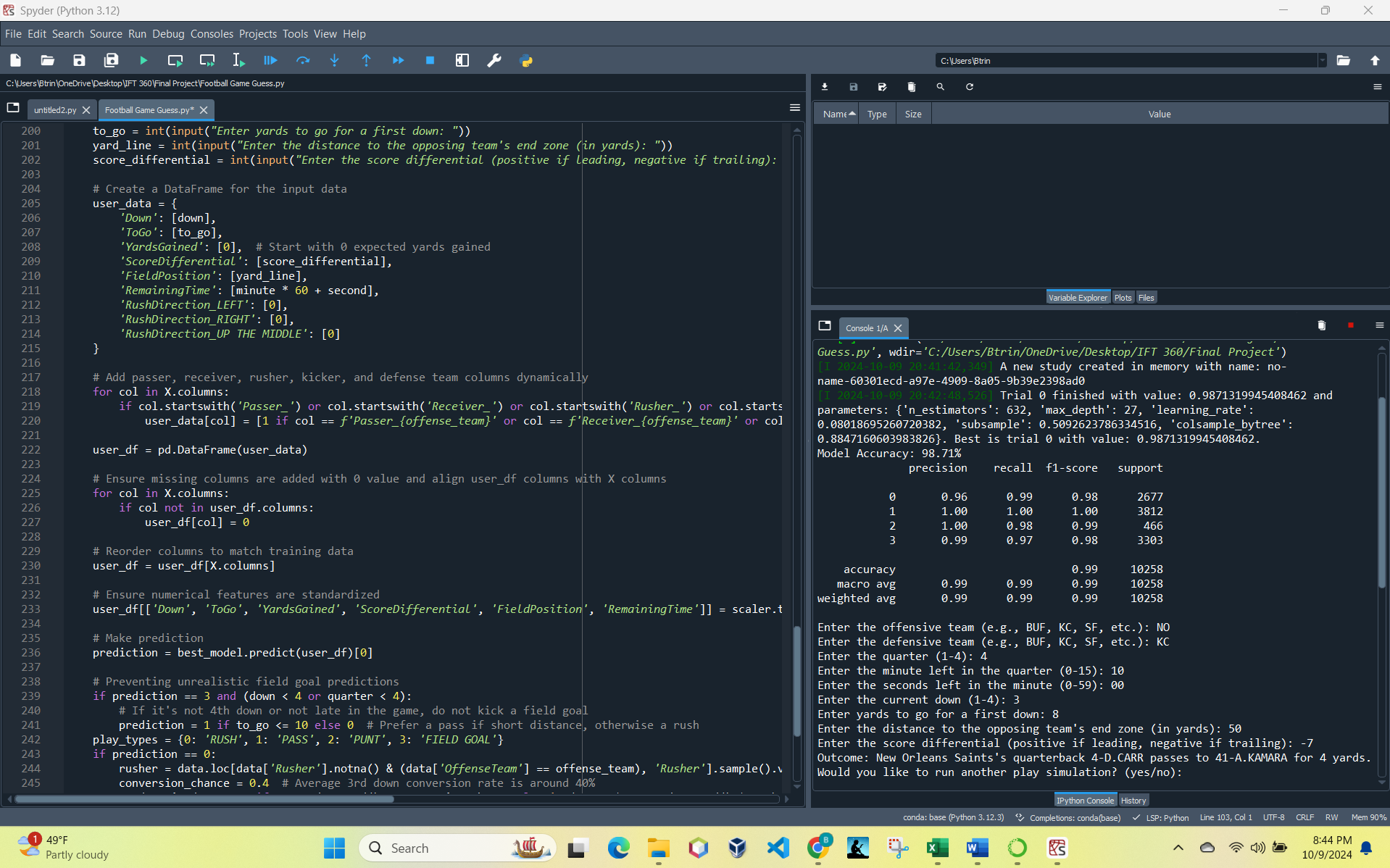
Arizona State University

Course Number: IFT 360

Professor Durgesh Sharma

10/6/24

**Football Game Strategy AI (Coach's Assistant)**



**Data set:**

pbp-2023.csv (NFL Savant, 2024)

**Zoom Recording Link:**

https://asu.zoom.us/rec/share/LO\_PGhYjhhJJiN2qRd09-49XR\_a03MqwIW6F7yY31cHSqCB1u-FpHJQwf0WdQCrT.WWz9sgFCXQCmEVZC?startTime=1728522849000

Passcode: YSK0vz$8

*I could not get audio to work, so I provided a summary of the model along with a description of the functions.*

**Summary:**

This project is a football play prediction model that is used to forecast play types such as rush, pass, punt, or field goal. The core of this project involves applying machine learning to historical football data to predict future plays based on game conditions. The model uses XGBoost, a powerful boosting algorithm known for its efficiency and accuracy in handling complex datasets. I train this model using a dataset containing all the plays from 2023 season, and I use feature extraction techniques to derive key elements like passer, receiver, yards gained, and situational context such as score differential and field position. These features are crucial in understanding the strategic decisions that teams make. To enhance the model's predictive power, we use Optuna, which performs hyperparameter optimization. This process can be thought of as a guided search—similar to an optimization problem—where Optuna explores various parameter combinations to maximize the model's performance. This is not a traditional search like depth-first or breadth-first but rather an optimization search, balancing between exploration and exploitation to find the best settings for the model.

Once trained, the model uses a greedy-like approach to predict the play type that has the highest probability given the current game situation. By incorporating context, such as down, distance, and score, the model makes informed predictions, aiming to mirror the decision-making process of a coach. Additionally, we add logic to prevent unrealistic predictions, such as avoiding a field goal on third down unless it's a critical scenario.

This project showcases how AI can be applied in sports analytics, using data-driven insights to predict strategic decisions and help us better understand the dynamics of the game.

**Functions:**

extract\_play\_features(description):

Extracts detailed features from the Description column of each play. Identifies whether a pass was completed, who the passer and receiver were, the direction of a rush, yards gained, if a turnover (fumble or interception) occurred, and identifies the kicker. Uses regular expressions to parse the play description and return various features.

objective(trial):

A function used by Optuna for hyperparameter optimization. Defines a parameter space (n\_estimators, max\_depth, learning\_rate, subsample, and colsample\_bytree) and trains an XGBoost model. Evaluates the model’s accuracy on a validation set and returns the score to guide further parameter tuning.

predict\_play\_scenario():

Allows user interaction to simulate a game scenario. Takes input from the user, including offensive and defensive teams, quarter, down, yards to go, etc. Constructs a feature set based on user inputs, transforms the data to align with the training features, and uses the trained model to predict the play type. Contains logic to prevent unrealistic outcomes, such as avoiding a field goal on non-critical downs. Provides a detailed description of the predicted play, incorporating player names and yardage outcomes.

**Code:**

# -\*- coding: utf-8 -\*-

"""

Created on Mon Oct 7 15:43:25 2024

Author: Brandon Trinkle

Class: IFT 360

Professor Durgesh Sharma

"""

import pandas as pd

import re

import numpy as np

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from xgboost import XGBClassifier

from sklearn.metrics import accuracy\_score, classification\_report

import optuna

# Load the dataset

file\_path = 'C://Users//Btrin//OneDrive//Desktop//IFT 360//Final Project//pbp-2023.csv'

data = pd.read\_csv(file\_path)

# Define team abbreviations map

team\_abbreviations = {

'ARI': 'Arizona Cardinals',

'ATL': 'Atlanta Falcons',

'BAL': 'Baltimore Ravens',

'BUF': 'Buffalo Bills',

'CAR': 'Carolina Panthers',

'CHI': 'Chicago Bears',

'CIN': 'Cincinnati Bengals',

'CLE': 'Cleveland Browns',

'DAL': 'Dallas Cowboys',

'DEN': 'Denver Broncos',

'DET': 'Detroit Lions',

'GB': 'Green Bay Packers',

'HOU': 'Houston Texans',

'IND': 'Indianapolis Colts',

'JAX': 'Jacksonville Jaguars',

'KC': 'Kansas City Chiefs',

'LV': 'Las Vegas Raiders',

'LAC': 'Los Angeles Chargers',

'LAR': 'Los Angeles Rams',

'MIA': 'Miami Dolphins',

'MIN': 'Minnesota Vikings',

'NE': 'New England Patriots',

'NO': 'New Orleans Saints',

'NYG': 'New York Giants',

'NYJ': 'New York Jets',

'PHI': 'Philadelphia Eagles',

'PIT': 'Pittsburgh Steelers',

'SF': 'San Francisco 49ers',

'SEA': 'Seattle Seahawks',

'TB': 'Tampa Bay Buccaneers',

'TEN': 'Tennessee Titans',

'WAS': 'Washington Commanders'

}

# Extract new features from the 'Description' column

def extract\_play\_features(description):

is\_pass\_completed = 0

rush\_direction = None

yards\_gained = 0

passer = None

receiver = None

rusher = None

is\_intercepted = 0

is\_fumble = 0

kicker = None

# Check if the pass was completed and extract passer and receiver

pass\_match = re.search(r"(\d{1,2}-[A-Z]\.[A-Za-z]+) PASS .\*? TO (\d{1,2}-[A-Z]\.[A-Za-z]+)", description)

if pass\_match:

passer = pass\_match.group(1)

receiver = pass\_match.group(2)

if "COMPLETE" in description:

is\_pass\_completed = 1

if "INTERCEPTED" in description:

is\_intercepted = 1

# Extract rusher if it's a rush play

rush\_match = re.search(r"(\d{1,2}-[A-Z]\.[A-Za-z]+) .\*?(UP THE MIDDLE|LEFT|RIGHT)", description)

if rush\_match:

rusher = rush\_match.group(1)

rush\_direction = rush\_match.group(2)

# Extract yards gained

yards\_match = re.search(r"(\d+)\s+YARD", description)

if yards\_match:

yards\_gained = int(yards\_match.group(1))

# Extract kicker if it's a field goal

kicker\_match = re.search(r"(\d{1,2}-[A-Z]\.[A-Za-z]+) .\*?FIELD GOAL", description)

if kicker\_match:

kicker = kicker\_match.group(1)

# Extract fumble information

if "FUMBLE" in description:

is\_fumble = 1

return is\_pass\_completed, rush\_direction, yards\_gained, passer, receiver, rusher, is\_intercepted, is\_fumble, kicker

# Apply the function

data[['IsPassCompleted', 'RushDirection', 'YardsGained', 'Passer', 'Receiver', 'Rusher', 'IsIntercepted', 'IsFumble', 'Kicker']] = data['Description'].apply(lambda x: pd.Series(extract\_play\_features(str(x))))

# Score differential as a synthetic feature

if 'ScoreDifferential' not in data.columns:

data['ScoreDifferential'] = data.get('OffenseScore', 0) - data.get('DefenseScore', 0) # Placeholder feature, calculate difference if scores available # Placeholder feature

# Field position as a synthetic feature representing the distance to the end zone

if 'FieldPosition' not in data.columns:

if 'YardLine' in data.columns:

data['FieldPosition'] = data['YardLine']

else:

data['FieldPosition'] = 50 # Default value for field position if not available

# Adding remaining time as a synthetic feature calculated from minute and second columns

if 'RemainingTime' not in data.columns:

if 'Minute' in data.columns and 'Second' in data.columns:

data['RemainingTime'] = data['Minute'] \* 60 + data['Second'] # Calculate remaining time in seconds

else:

data['RemainingTime'] = 0 # Default value for remaining time if not available # Calculate remaining time in seconds

# Prepare features and target variable

rush\_direction\_dummies = pd.get\_dummies(data['RushDirection'], prefix='RushDirection')

passer\_dummies = pd.get\_dummies(data['Passer'], prefix='Passer')

receiver\_dummies = pd.get\_dummies(data['Receiver'], prefix='Receiver')

rusher\_dummies = pd.get\_dummies(data['Rusher'], prefix='Rusher')

kicker\_dummies = pd.get\_dummies(data['Kicker'], prefix='Kicker')

# Add score differential

data = pd.concat([data, rush\_direction\_dummies, passer\_dummies, receiver\_dummies, rusher\_dummies, kicker\_dummies], axis=1)

# Drop unnecessary columns for model training

X = data.drop(columns=['Description', 'RushDirection', 'PlayType', 'GameId', 'GameDate', 'Passer', 'Receiver', 'Rusher', 'Kicker', 'Minute', 'Second'])

y = data['PlayType'].apply(lambda x: 0 if x == 'RUSH' else (1 if x == 'PASS' else (2 if x == 'PUNT' else 3)))

# Convert boolean columns to integers and handle non-numeric values

X = X.apply(pd.to\_numeric, errors='coerce').fillna(0)

# Ensure target variable has no NaN values

y = y.fillna(0).astype(int)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize the numerical features

scaler = StandardScaler()

X\_train[['Down', 'ToGo', 'YardsGained', 'ScoreDifferential', 'FieldPosition', 'RemainingTime']] = scaler.fit\_transform(X\_train[['Down', 'ToGo', 'YardsGained', 'ScoreDifferential', 'FieldPosition', 'RemainingTime']])

X\_test[['Down', 'ToGo', 'YardsGained', 'ScoreDifferential', 'FieldPosition', 'RemainingTime']] = scaler.transform(X\_test[['Down', 'ToGo', 'YardsGained', 'ScoreDifferential', 'FieldPosition', 'RemainingTime']])

# Hyperparameter Tuning for XGBoost Model

def objective(trial):

param = {

'objective': 'multi:softmax',

'num\_class': 5,

'n\_estimators': trial.suggest\_int('n\_estimators', 100, 1500),

'max\_depth': trial.suggest\_int('max\_depth', 5, 30),

'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.1, log=True),

'subsample': trial.suggest\_float('subsample', 0.5, 1.0),

'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.5, 1.0)

}

model = XGBClassifier(\*\*param)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

return accuracy\_score(y\_test, y\_pred)

study = optuna.create\_study(direction='maximize')

study.optimize(objective, n\_trials=1)

best\_params = study.best\_params

best\_model = XGBClassifier(\*\*best\_params)

best\_model.fit(X\_train, y\_train)

# Test the model's accuracy with the best parameters

y\_pred = best\_model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Model Accuracy: {accuracy \* 100:.2f}%")

# Display classification report

classification\_rep = classification\_report(y\_test, y\_pred)

print(classification\_rep)

# Function to make predictions based on user input

def predict\_play\_scenario():

while True:

offense\_team = input("Enter the offensive team (e.g., BUF, KC, SF, etc.): ").upper()

if offense\_team in team\_abbreviations:

break

else:

print("Invalid team abbreviation. Please try again.")

while True:

defense\_team = input("Enter the defensive team (e.g., BUF, KC, SF, etc.): ").upper()

if defense\_team in team\_abbreviations:

break

else:

print("Invalid team abbreviation. Please try again.")

quarter = int(input("Enter the quarter (1-4): "))

minute = int(input("Enter the minute left in the quarter (0-15): "))

second = int(input("Enter the seconds left in the minute (0-59): "))

down = int(input("Enter the current down (1-4): "))

to\_go = int(input("Enter yards to go for a first down: "))

yard\_line = int(input("Enter the distance to the opposing team's end zone (in yards): "))

score\_differential = int(input("Enter the score differential (positive if leading, negative if trailing): "))

# Create a DataFrame for the input data

user\_data = {

'Down': [down],

'ToGo': [to\_go],

'YardsGained': [0], # Start with 0 expected yards gained

'ScoreDifferential': [score\_differential],

'FieldPosition': [yard\_line],

'RemainingTime': [minute \* 60 + second],

'RushDirection\_LEFT': [0],

'RushDirection\_RIGHT': [0],

'RushDirection\_UP THE MIDDLE': [0]

}

# Add passer, receiver, rusher, kicker, and defense team columns dynamically

for col in X.columns:

if col.startswith('Passer\_') or col.startswith('Receiver\_') or col.startswith('Rusher\_') or col.startswith('Kicker\_') or col.startswith('DefenseTeam\_'):

user\_data[col] = [1 if col == f'Passer\_{offense\_team}' or col == f'Receiver\_{offense\_team}' or col == f'Rusher\_{offense\_team}' or col == f'Kicker\_{offense\_team}' or col == f'DefenseTeam\_{defense\_team}' else 0]

user\_df = pd.DataFrame(user\_data)

# Ensure missing columns are added with 0 value and align user\_df columns with X columns

for col in X.columns:

if col not in user\_df.columns:

user\_df[col] = 0

# Reorder columns to match training data

user\_df = user\_df[X.columns]

# Ensure numerical features are standardized

user\_df[['Down', 'ToGo', 'YardsGained', 'ScoreDifferential', 'FieldPosition', 'RemainingTime']] = scaler.transform(user\_df[['Down', 'ToGo', 'YardsGained', 'ScoreDifferential', 'FieldPosition', 'RemainingTime']])

# Make prediction

prediction = best\_model.predict(user\_df)[0]

# Preventing unrealistic field goal predictions

if prediction == 3 and (down < 4 or quarter < 4):

# If it's not 4th down or not late in the game, do not kick a field goal

prediction = 1 if to\_go <= 10 else 0 # Prefer a pass if short distance, otherwise a rush

play\_types = {0: 'RUSH', 1: 'PASS', 2: 'PUNT', 3: 'FIELD GOAL'}

if prediction == 0:

rusher = data.loc[data['Rusher'].notna() & (data['OffenseTeam'] == offense\_team), 'Rusher'].sample().values[0] if offense\_team in data['OffenseTeam'].values else 'Unknown Rusher'

conversion\_chance = 0.4 # Average 3rd down conversion rate is around 40%

yards\_gained = to\_go if np.random.rand() < conversion\_chance else int(to\_go \* np.random.rand() \* 0.5) # 40% chance to get the first down, otherwise gain between 0-50% of to\_go

outcome = f"{team\_abbreviations[offense\_team]}'s running back {rusher} runs for {yards\_gained} yards."

elif prediction == 1:

passer = data.loc[data['Passer'].notna() & (data['OffenseTeam'] == offense\_team), 'Passer'].sample().values[0] if offense\_team in data['OffenseTeam'].values else 'Unknown Quarterback'

receiver = data.loc[data['Receiver'].notna() & (data['OffenseTeam'] == offense\_team), 'Receiver'].sample().values[0] if offense\_team in data['OffenseTeam'].values else 'Unknown Receiver'

yards\_gained = int(max(1, to\_go \* 0.5 + (to\_go \* 0.5 \* np.random.rand()))) # Randomly predict between 50% to 100% of yards to go

outcome = f"{team\_abbreviations[offense\_team]}'s quarterback {passer} passes to {receiver} for {yards\_gained} yards."

else:

outcome = f"Predicted play type: {play\_types[prediction]}"

print(f"Outcome: {outcome}")

while True:

predict\_play\_scenario()

repeat = input("Would you like to run another play simulation? (yes/no): ").strip().lower()

if repeat != 'yes':

break