# AMERICAN FOOTBALL BETTING

Iman Janoo, Travis Roth, Drew Blik, Evelyn Ochoa

# AGENDA

01 EXECUTIVE SUMMARY

02 DATA DESCRIPTION

03 DATA PREPARATION

04 DATA MINING SOLUTION

05 RESULTS & RECOMMENDATIONS

# 01

# EXECUTIVE SUMMARY

# PROBLEM VS. SOLUTION

## PROBLEM

How do we increase the
NFL better's profitability?

## SOLUTION

Predict the winner!
(accurately)

# OUR GOAL

1. Predict the Winner
2. Beat the Baseline
3. Be Profitable

02

# DATA DESCRIPTION

Response variables, predictors, sample dataset, EDA

# DATA DESCRIPTION

- Response Variables + Predictors
- Sample Dataset
- Exploratory Data Analysis

# RESPONSE VARIABLE AND FEATURES DESCRIPTIONS

| | | | | | |
|---|---|---|---|---|---|
| Response | home_win | Refers to a binary outcome where 1 indicates home team won, and 0 indicates that the home team lost or tied | | | |
| 1 | schedule_date | Date of the scheduled event | 11 | weather_detail | Detailed information about weather conditions during event (temperature, precipitation, etc.). |
| 2 | schedule_season | Sports season in which game takes place | 12 | stadium_name | Name of the stadium hosting the event. |
| 3 | schedule_week | Specific week within the sports season | 13 | stadium_type | Classification of the stadium (e.g., indoor, outdoor, retractable roof). |
| 4 | schedule_playoff | Binary indicator (1 or 0) denoting if the event is part of playoffs or postseason. | 14 | stadium_capacity | Maximum seating capacity of the stadium. |
| 5 | score_home | Points earned by the home team in the event. | 15 | stadium_latitude | Geographical latitude coordinates of the stadium's location. |
| 6 | score_away | Points earned by the home team in the event. | 16 | stadium_longitude | Geographical longitude coordinates of the stadium's location. |
| 7 | team_favorite_id | Unique identifier or code for the favored team in the event. | 17 | stadium_azimuthangle | Angle representing the stadium's orientation relative to a specific direction. |
| 8 | spread_favorite | Point spread favoring the favored team in the event. | 18 | stadium_elevation | Height or elevation above sea level of the stadium's location. |
| 9 | over_under_line | Betting line indicating the total expected score from both teams in the event. | 19 | team_home | Home team participating in the scheduled event. |
| 10 | stadium | Name or identifier of the stadium where the event is held | 20 | team_away | Visiting team participating in the scheduled event. |

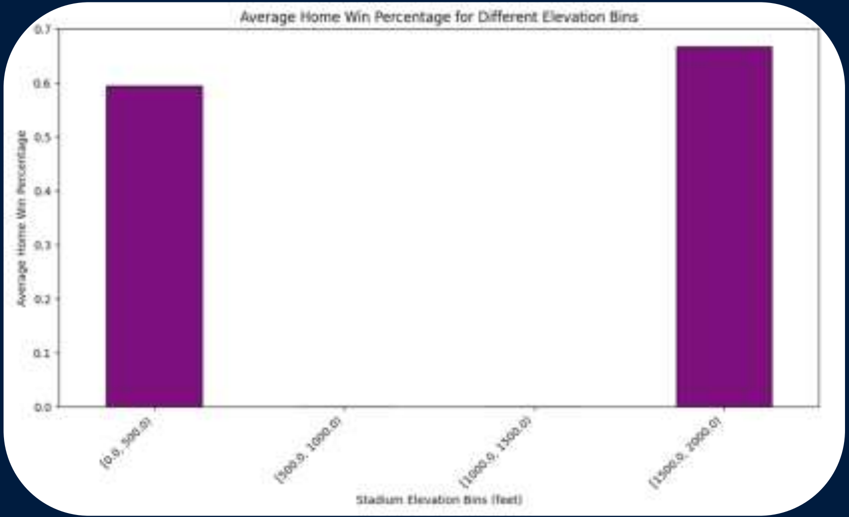
# SAMPLE DATASET

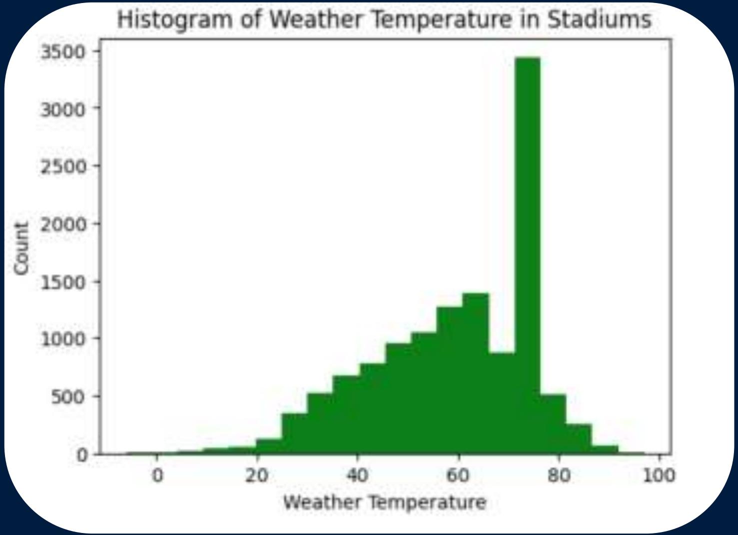
| | schedule_date | schedule_season | schedule_week | schedule_playoff | score_home | score_away | team_favorite_id | spread_favorite | over_under_line | stadium |
|---|---|---|---|---|---|---|---|---|---|---|
| **727** | 1/11/1970 | 1969 | Superbowl | TRUE | 23.0 | 7.0 | MIN | -12.0 | 39 | Tulane Stadium |
| **1105** | 1/16/1972 | 1971 | Superbowl | TRUE | 24.0 | 3.0 | DAL | 6.0 | 34 | Tulane Stadium |
| **1483** | 1/13/1974 | 1973 | Superbowl | TRUE | 24.0 | 7.0 | MIA | 6.5 | 33 | Rice Stadium |
| **1672** | 1/12/1975 | 1974 | Superbowl | TRUE | 6.0 | 16.0 | PIT | -3.0 | 33 | Tulane Stadium |
| **1861** | 1/18/1976 | 1975 | Superbowl | TRUE | 17.0 | 21.0 | PIT | -7.0 | 36 | Orange Bowl |

| | weather_detail | stadium_name | stadium_type | stadium_capacity | tadium_latitud | stadium_longitude | stadium_azimuthangle | stadium_elevation | team_home | team_away |
|---|---|---|---|---|---|---|---|---|---|---|
| **727** | indoor | Caesars Superdome | indoor | 76468.0 | NaN | NaN | NaN | NaN | NO | ATL |
| **1105** | NaN | MetLife Stadium | NaN | 82500.0 | 40.813528 | -74.074361 | 345.5 | 2.1 | NYG | PHI |
| **1483** | NaN | Levi's Stadium | NaN | 685000.0 | 37.40300 | -121.97000 | 330 | 2.4 | SF | LAR |
| **1672** | NaN | Nissan Stadium | NaN | 69143.0 | 36.166389 | -86.771389 | 334.5 | 182.9 | TEN | JAX |
| **1861** | NaN | FedEx Field | NaN | 79000.0 | 38.907778 | -76.864444 | 295 | 15.2 | WAS | DAL |

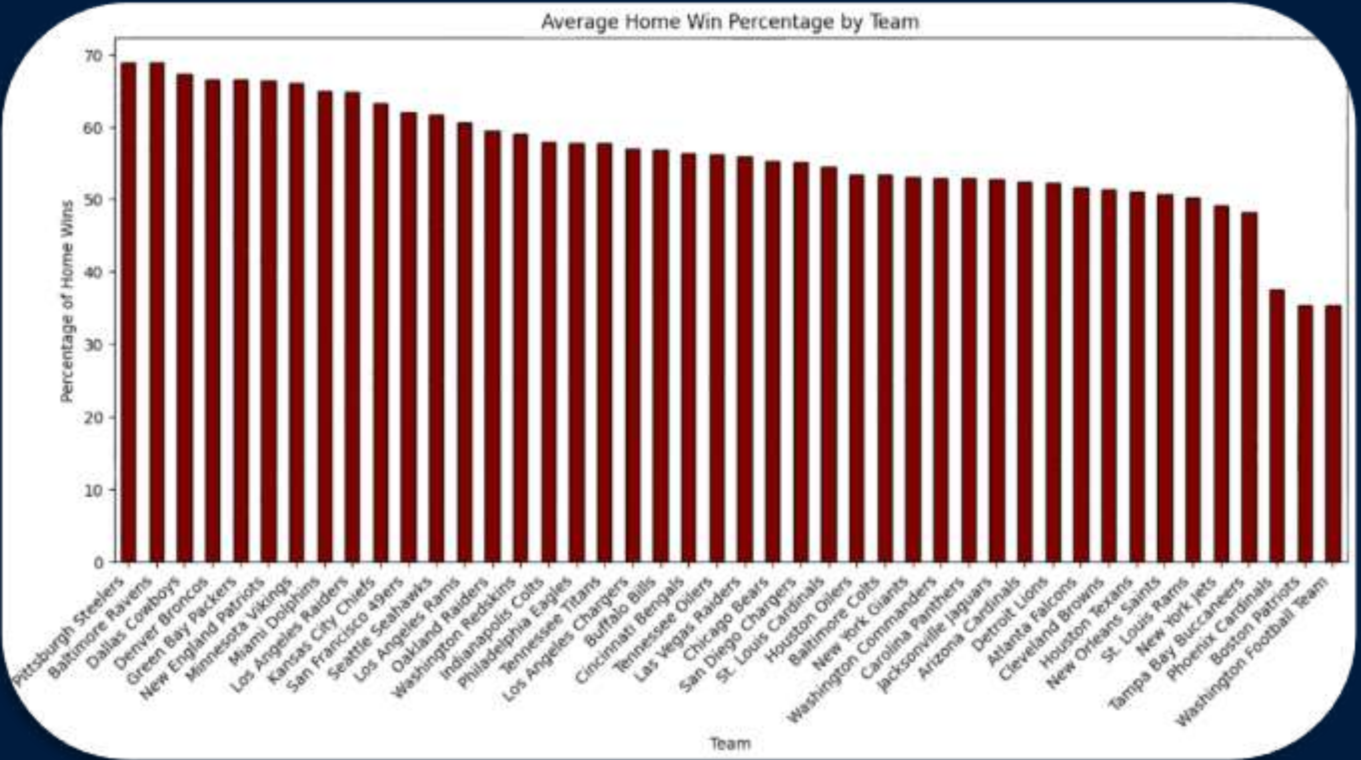
5 rows × 24 columns

# EXPLORATORY DATA ANALYSIS



# EXPLORATORY DATA ANALYSIS



# 03

# DATA PREPARATION

Constructing, Cleaning, & Feature Engineering

# DATA CONSTRUCTION & CLEANING

- MERGING DATA SOURCES
- HANDLING NON-NUMERIC & CATEGORICAL VARIABLES
- HANDLING MISSING VALUES

# MERGING DATA SOURCES

```
# Drop unnecessary columns from 'teams'
teams.drop(columns=['Conference', 'Division', 'ID'], inplace=True)

# Map team names to abbreviations for 'team_home'
df = pd.merge(df, teams, left_on='team_home', right_on='Name', how='left')
df.rename(columns={'Abbreviation': 'team_home_abbrev'}, inplace=True)
df.drop('Name', axis=1, inplace=True)

# Map team names to abbreviations for 'team_away'
df = pd.merge(df, teams, left_on='team_away', right_on='Name', how='left')
df.rename(columns={'Abbreviation': 'team_away_abbrev'}, inplace=True)
df.drop('Name', axis=1, inplace=True)

# Drop original team name columns ('team_home' and 'team_away')
df.drop(columns=['team_home', 'team_away'], inplace=True)

# Rename abbreviation columns to 'team_home' and 'team_away'
df.rename(columns={'team_home_abbrev': 'team_home', 'team_away_abbrev': 'team_away'}, inplace=True)

df.dropna(subset=['team_home', 'team_away'], how='any', inplace=True)
# Verify DataFrame after the operations
print(df.head())
```

# MERGING DATA SOURCES

```
# need to add spread column relative to home team (not relative to favorite team)
def adjust_point_spread(row):
  if row['team_home'] != row['team_favorite_id']:
    return row['spread_favorite']
  else:
    return -row['spread_favorite']
df['spread_favorite'] = df.apply(adjust_point_spread, axis=1)
```

# HANDLING NON-NUMERIC & CATEGORICAL VARIABLES

```
# Create dummy variables
#set default value in weather_detail to prevent NANs
df['weather_detail'].fillna('normal', inplace=True)

# Get dummies for categorical columns
categorical_columns = ['team_home','team_away','team_favorite_id','weather_detail','stadium_type']
dummies = pd.get_dummies(df[categorical_columns], prefix=categorical_columns)

# Concatenate the dummies with the original DataFrame
df = pd.concat([df, dummies], axis=1)

# Drop the original categorical columns if needed
df.drop(categorical_columns, axis=1, inplace=True)

df = df.dropna(subset=['over_under_line', 'weather_temperature', 'weather_wind_mph', 'weather_humidity'])
```

# HANDLING MISSING VALUES

```
#drop teams that no longer exist, since they had no abbreviation value in the key dictionary
df.dropna(subset=['team_home', 'team_away'], how='any', inplace=True)
```

```
def predict(dataframe, season_choice, model_choice, drop_features, target_variable):
  import pandas as pd
  from sklearn.ensemble import RandomForestClassifier
  # evaluate a model to predict game winner based on features
  # Split the data into training and testing based on the schedule_date

  dataframe.dropna(axis = 0, inplace = True)
```

```
#interpolate missing values
columns_to_interpolate = ['weather_humidity', 'stadium_elevation', 'stadium_azimuthangle']
df[columns_to_interpolate] = df[columns_to_interpolate].interpolate(method='linear')
```

# FEATURE ENGINEERING

- ADJUSTING FOR SPREAD
- TIME SERIES COLUMNS
- ROLLING SEASON RECORD

# FEATURE ENGINEERING: ADJUSTING FOR SPREAD

```
#add home_win
df['score_diff_spread_adj'] = df['score_home'] - df['score_away'] - df['spread_favorite']
def home_win(difference):
  if difference < 0:
    return False
  else:
    return True
df['home_win_spread_adj'] = df['score_diff_spread_adj'].apply(home_win)
```

```
#add home_win
df['score_diff'] = df['score_home'] - df['score_away']
def home_win(difference):
  if difference < 0:
    return False
  else:
    return True
df['home_win'] = df['score_diff'].apply(home_win)
```

# FEATURE ENGINEERING: TIME SERIES COLUMN

```
#clean date and sort rows by date
from datetime import datetime

df['schedule_date']= pd.to_datetime(df['schedule_date'])
df.sort_values('schedule_date', inplace=True)

# Filter and keep rows where 'schedule_date' is earlier or equal to today's date
today_date = datetime.today()
data = df[df['schedule_date'] <= today_date]

# add year, month columns
df['year'] = df['schedule_date'].dt.year
df['month'] = df['schedule_date'].dt.month
```

# FEATURE ENGINEERING: ROLLING SEASON RECORD

```
window_size = 1

# Calculate cumulative wins for home team
df['team_home_rolling_wins'] = df.groupby(['team_home', 'schedule_season'])['home_win'].cumsum() - df['home_win']

# Calculate cumulative wins for away team
df['team_away_rolling_wins'] = df.groupby(['team_away', 'schedule_season'])['home_win'].apply(lambda x: x[::-1].cumsum()[::-1]) - df['home_win']

# Replace NaN values (resulting from the first game of each team) with 0
df['team_home_rolling_wins'].fillna(0, inplace=True)
df['team_away_rolling_wins'].fillna(0, inplace=True)

#drop teams that no longer exist, since they had no abbreviation value in the key dictionary
df.dropna(subset=['team_home', 'team_away'], how='any', inplace=True)
```

# TRAIN TEST SPLIT :
# BACKTESTING A STRATEGY

| START SEASON | TRAINING SET | TESTING SET |
|---|---|---|
| 1980 | 1979 | 1980 |
| 1990 | 1979-1989 | 1990 |
| 2022 | 1979-2021 | 2022 |

# TRAIN TEST SPLIT :
# BACKTESTING A STRATEGY

```
def predict(dataframe, season_choice, model_choice, drop_features, target_variable):
  import pandas as pd
  from sklearn.ensemble import RandomForestClassifier
  # evaluate a model to predict game winner based on features
  # Split the data into training and testing based on the schedule_date

  dataframe.dropna(axis = 0, inplace = True)

  train_data = dataframe[dataframe['schedule_season'] < season_choice]
  test_data = dataframe[dataframe['schedule_season'] == season_choice]


  # Feature selection
  X_train = train_data.drop(drop_features, axis=1) # drop target variable and others
  y_train = train_data[target_variable]
  X_test = test_data.drop(drop_features, axis=1) # drop target variable and others
  y_test = test_data[target_variable]
  # creating the model
  model = model_choice.fit(X_train, y_train)
  X_train['y_pred'] = model.predict(X_train)
  X_test['y_pred'] = model.predict(X_test)
```

# TRAIN TEST SPLIT :
# BACKTESTING A STRATEGY

```
[ ]  def backtest(data,model,drop_features,target,start_season= 1980,step=1):
       all_seasons = {}

       for i in range(start_season,2023,step):
         if data[data['schedule_season']==i].shape[0] != 0:
           season = predict(data,i,model,drop_features,target)
           all_seasons.update({i: season})

       as_df = pd.DataFrame(all_seasons)
       as_df = as_df.transpose()
       as_df.columns = ['season_choice','rows_train','rows_test','cm_train','cm

       #calculate cumulatives
       as_df['base_profit_cum'] = as_df['baseline_profit_test'].cumsum()
       as_df['model_profit_cum'] = as_df['test_profit'].cumsum()

       return as_df
```

# TRAIN TEST SPLIT :
# BACKTESTING A STRATEGY

| | season_choice | rows_train | rows_test |
|---|---|---|---|
| 2016 | 2016 | 3106 | 203 |
| 2017 | 2017 | 3309 | 232 |
| 2018 | 2018 | 3541 | 233 |
| 2019 | 2019 | 3774 | 67 |
| 2020 | 2020 | 3841 | 98 |
| 2021 | 2021 | 3939 | 133 |
| 2022 | 2022 | 4072 | 98 |

# 04

# DATA MINING SOLUTION

# HOW BETTING WORKS

## ODDS DETERMINE PAYOUT

Score:
Home: 15 vs Away: 10
Spread: +6 (favoring home)

## NOT ADJUSTED

Home wins!
15 > 10

## SPREAD ADJUSTED

Away wins!
15 < 10+6

# RANDOM FORESTS CLASSIFIER

## BASE MODEL

### MINIMUM SAMPLES

Reduced min_sample_leaf to 10 due to small dataset

### CLASS WEIGHTS

Tried favoring false negatives over  false positives

# LOGISTIC REGRESSIONN

## MAXIMUM ITERATIONS

Increased max_iter=1000 to get closer to convergence

## SOLVER

Set solver='newton-cholesky' to handle many categorical features

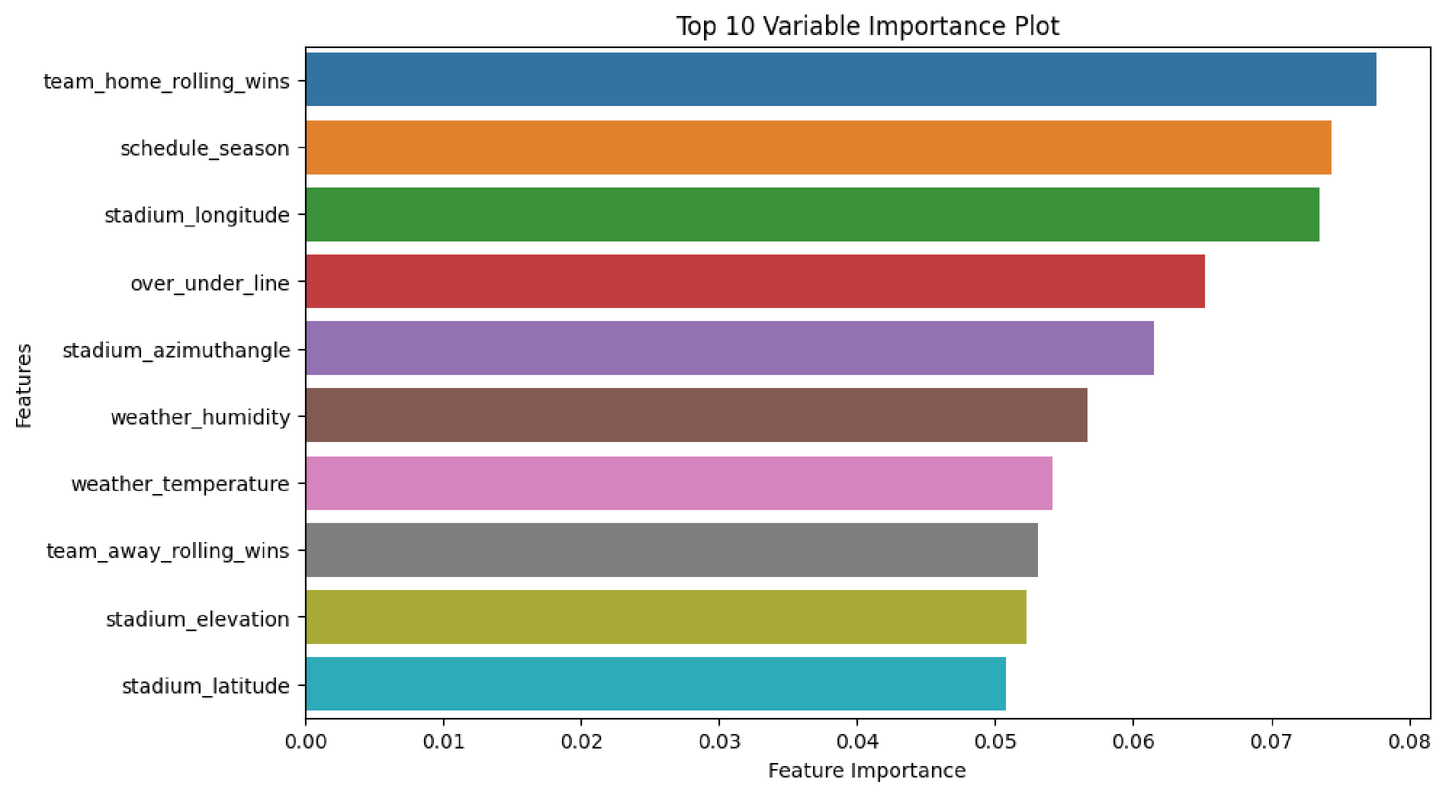
# 05

# RESULTS & RECOMMENDATIONS

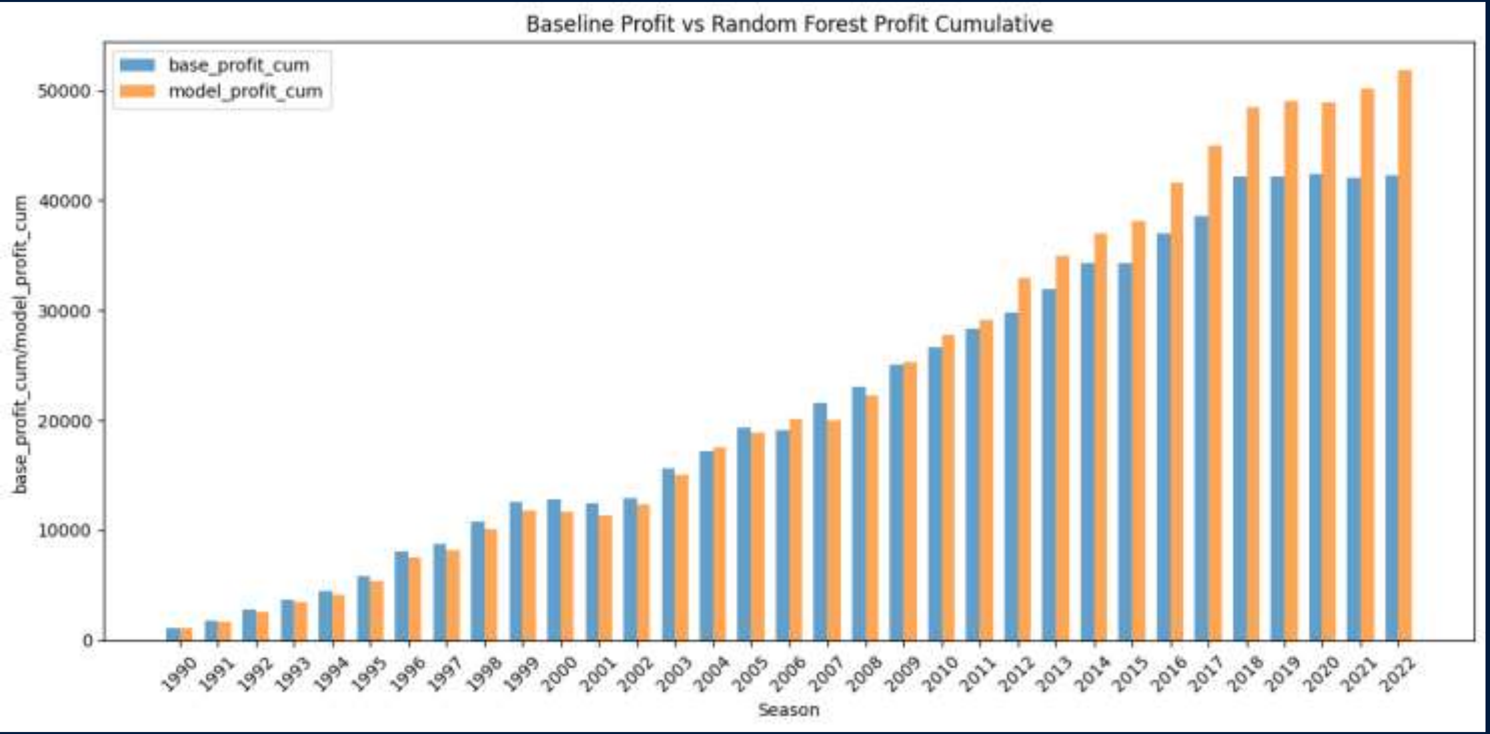
# MODEL PERFORMANCE - NORMAL

| METRIC | BASELINE | RANDOM FOREST | LOGISTIC REGRESSION |
| --- | --- | --- | --- |
| ACCURACY (%) | 60 | 61 | 58 |
| YEARLY PROFIT ($) | 1,282 | 1,530 | 1,081 |
| CUMULATIVE PROFIT ($) | 42,330 | 50,500 | 35,680 |

# FEATURE IMPORTANCE: RANDOM FORESTS



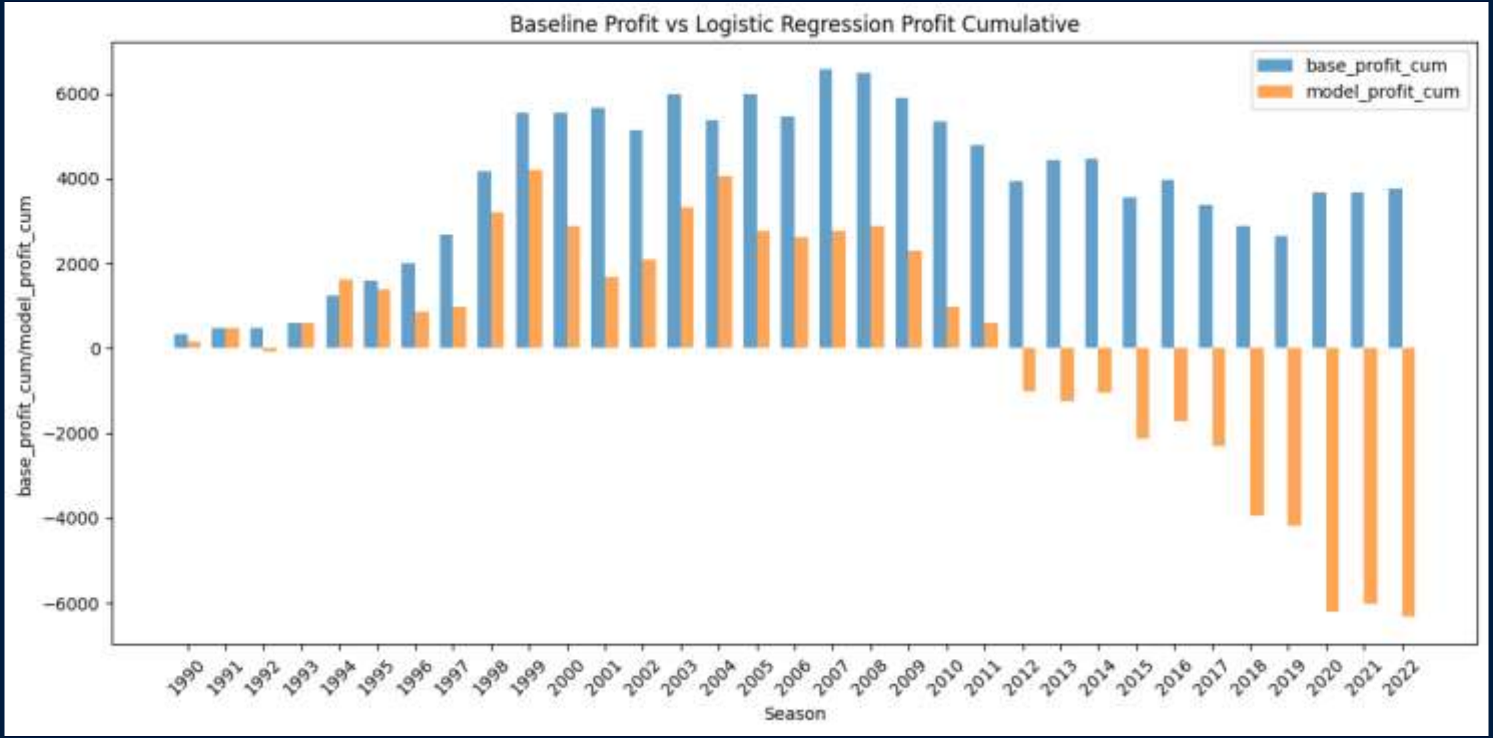
# MODEL PERFORMANCE - NORMAL



# MODEL PERFORMANCE– SPREAD ADJ.

| METRIC | BASELINE | RANDOM FOREST | LOGISTIC REGRESSION |
|---|---|---|---|
| ACCURACY (%) | 54 | 52 | 53 |
| YEARLY PROFIT ($) | 113 | -202 | -191 |
| CUMULATIVE PROFIT ($) | 3,760 | -6,690 | -6,310 |

# MODEL PERFORMANCE– SPREAD ADJ.



# WHAT WE LEARNED

Historic betting odds are extremely valuable

Bookies have advanced models to generate odds

Nuanced team performance and composition data could improve model

# THANK YOU!