Table of Contents

DTSC691: Applied Data Science Capstone

Michael Hart

Machine Learning Project Overview: NFL Win Probability

This project develops a real-time win probability model for NFL games using machine learning and public data, replicating sportsbook probabilities with near-perfect accuracy and demonstrating advantages in select scenarios.

Introduction

Objective

My objective was to build a machine learning model that predicts real-time NFL win probabilities. Unlike sportsbooks, which adjust odds to balance risk and encourage wagers, this approach aims to estimate *unbiased probabilities* that reflect true game outcomes without market distortion.

Why It Matters

Sportsbook lines reflect both game state and wagering behavior. An unbiased model can generalize more effectively, supporting fair decision making and evaluation without inherited market biases.

Scope

My focus was on pre-snap win probabilities for both regular and postseason games, using only publicly available data. To keep the analysis targeted, player-level predictions, in-play statistics (e.g., Expected Points Added, Win Probability Added), and external factors such as weather were excluded.

By narrowing the scope in this way, the project remained centered on its core question: Can machine learning applied to public data approximate or even *improve* upon sportsbook probability models?

Data and Preprocessing

Data Source

The dataset was sourced from nflfastR. It includes every NFL play since 1999 with nearly 400 features, making it an excellent foundation for feature engineering and machine learning.

Initial Exploration

I began with exploratory data analysis, examining distributions and correlations of key variables to identify relationships with the target.

Feature Engineering

I engineered a `time_weight` feature to give extra emphasis to later-game plays, as these have a stronger influence on win probability. Additional engineered features relative to the home team include `home_pos`, `home_score_differential`, `home_spread_line`, and `yardline_100_home`.

Cleaning

I cleaned the data by removing plays with missing or invalid values. To ensure only valid decision points were modeled, I excluded ties (14 games) and non-play events such as timeouts, penalties, and period changes. Scoring discrepancies were identified in 2001-2002 Jacksonville Jaguars home games, leading me to restrict the dataset to seasons 2003 and later. Additionally, error analysis revealed unreliable target values in overtime plays, which were also excluded.

Transformation and Scaling

I scaled numeric features using StandardScaler and one-hot encoded the categorical feature `down`, wrapping all preprocessing into a pipeline for consistency and automation.

I clipped and logit-transformed the target (`vegas_home_wp`) to stabilize regression and probability outputs, especially for extreme probabilities. These transformations were implemented as utility functions in *utils.py*, ensuring consistency throughout the workflow.

Modeling

Algorithms Evaluated

I evaluated Ridge Regression, Lasso, Random Forest, Neural Network (Multilayer Perceptron), and XGBoost models.

Hyperparameter Tuning

I used RandomizedSearchCV for initial hyperparameter tuning (100 iterations) on both the linear models (Ridge, Lasso) and the tree-based models (Random Forest, XGBoost). For the Neural Network, I used KerasTuner to explore architectures ranging from 16 to 1,024 neurons across 2 to 5 layers. All models were evaluated with 3-fold cross-validation. I then used Optuna for final model refinement, running 500 optimization trials.

Evaluation Metrics

Models were evaluated using:

- MSE: Mean squared error for numeric accuracy.
- R^2: Variance explained.
- Log Loss: For probability calibration.
- Brier Score: Measures how close predicted probabilities are to actual outcomes.

While MSE and R^2 were used during early model comparisons, Log Loss and Brier Scores were applied primarily to the final XGBoost model to assess calibration based on game result.

Final Model Selection

XGBoost outperformed all the other models in both MSE and R^2. It balanced bias and variance well and avoided overfitting, making it the best candidate for deployment.

Results and Interpretation

Performance Summary

XGBoost achieved:

- **MSE**: **0.00034**. Indicates our model can replicate sportsbook probabilities to within ~0.02 percentage points on average (RMSE = 0.018).
- **R^2**: **0.997**. The model explains nearly all variation in sportsbook probabilities.
- **Log loss**: **0.44**. Competitive with sportsbook-grade models.
- **Brier Score**: **0.146**. Well-calibrated predictions. Close to 0 is better.

Visual Diagnostics

Diagnostic plots (scatterplots, distribution plots, and calibration curves) demonstrate the model's accuracy and reliability. Scatterplots show tight clustering against sportsbook probabilities, distribution plots reflect adaptation to reduced home-field advantage in the 2024 holdout season, and calibration curves highlight minor optimistic bias below 0.4 and conservative bias above 0.4.

Interpretation

Ridge and Lasso underperformed, likely due to their inability to model complex nonlinear interactions between features and probability. Random Forest and Neural Network models performed much better, but ultimately XGBoost showed the strongest results and was selected for deployment.

On the holdout set, the XGBoost model's low MSE and near-perfect R^2 demonstrate its ability to closely replicate sportsbook probabilities. Its Log Loss of 0.44 and Brier Score of 0.146 demonstrate strong calibration and probabilistic accuracy. Performance analysis by point spread showed our model generalized better than the sportsbook for 3-point home favorites, which represent nearly 1 in 10 games. While performance is excellent, results may partially reflect anchoring to market consensus (sportsbook lines). Future refinements could include training separate models by point spread category.

User Interface Integration

Design Goals

The interface was designed for intuitive use by both technical and non-technical users.

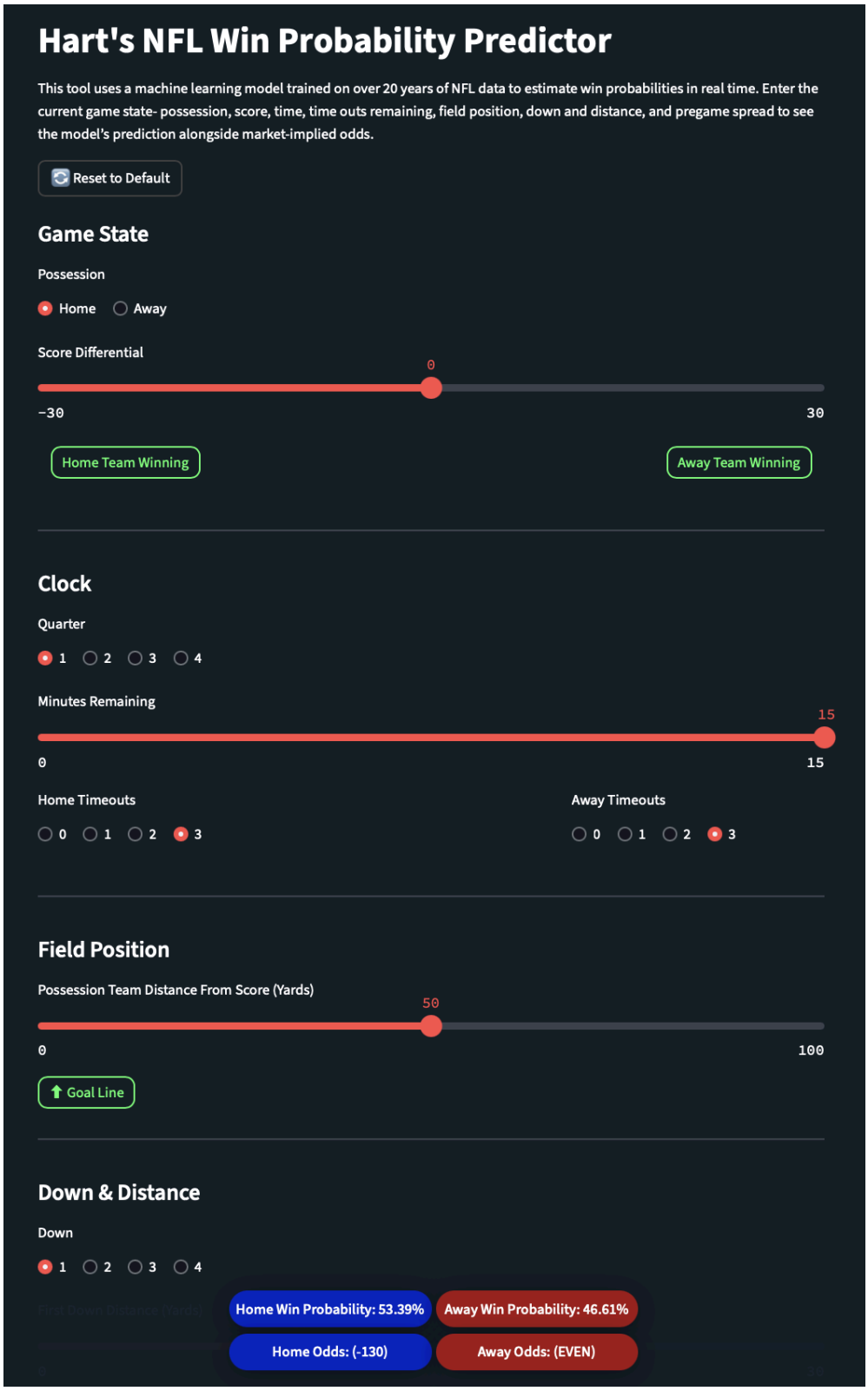
Streamlit Implementation

- **Input**: Score, possession, time remaining, timeouts remaining, field position, down and distance, and pregame point spread.
- **Output**: Predicted win probability and implied market odds.

Usability testing with non-technical NFL fans confirmed the UI's accessibility and real-time responsiveness. Its minimal input requirements and instant probability display enable effective, live use during games.

Cloud-based deployment: https://nfl-prob-predictor.streamlit.app/Model_UI

UI Example:


Hart's NFL Win Probability Predictor
This tool uses a machine learning model trained on over 20 years of NFL data to estimate win probabilities in real time. Enter the current game state- possession, score, time, time outs remaining, field position, down and distance, and pregame spread to see the model's prediction alongside market-implied odds.
Reset to Default
Game State
Possession
Home
Away
Score Differential
0
-30
30
Home Team Winning
Away Team Winning
Clock
Quarter
1
2
3
4
Minutes Remaining
15
0
15
Home Timeouts
0
1
2
3
Away Timeouts
0
1
2
3
Field Position
Possession Team Distance From Score (Yards)
50
0
100
Goal Line
Down & Distance
Down
1
2
3
4
Home Win Probability: 53.39%
Away Win Probability: 46.61%
Home Odds: (-130)
Away Odds: (EVEN)

Reflections and Future Work

While the final deliverable meets the project goals, the process revealed multiple opportunities for future refinement, particularly in preprocessing, hyperparameter optimization, and model specialization.

One key example arose during iterative error analysis, which identified invalid target values in overtime games (<1% of plays). Excluding these improved calibration and underscored the value of using prediction discrepancies as a diagnostic tool.

Early in the project, I often duplicated preprocessing steps across multiple notebooks. While this allowed efficient iteration, it also created unnecessary redundancy. Consolidating these steps into a shared utility module (as I did with the clipping + logit/expit functions) would ensure consistency across notebooks and simplify iterative modeling.

Hyperparameter optimization also revealed opportunities for improvement. Although RandomizedSearchCV and GridSearchCV provided valuable baselines, they were computationally expensive. Switching to Optuna in place of GridSearchCV proved far more efficient for exploring hyperparameters. In hindsight, starting with Optuna initially would have saved significant time and computational resources.

Looking ahead, training separate models for specific point spreads could further improve calibration and better capture game context (e.g., heavy favorites vs. close matchups).

Overall, this project illustrates a complete machine learning workflow from data acquisition through evaluation and deployment. The final model is both accurate and practical for real-world applications, exemplifying the value of applied data science.

Notebook 1: Data Loading and Initial Cleaning

In [2]:
```
import pandas as pd
import numpy as np
from nfl_data_py import import_pbp_data
import seaborn as sns
import matplotlib.pyplot as plt

year_list = []

for i in range(1999, 2025):
    year_list.append(i)

# df = import_pbp_data(year_list)
df = pd.read_csv('nfl_data_1999_2024.csv', low_memory = False)

pd.set_option('display.max_colwidth', None)
```

Data Overview

- Source: `nfl_data_py` play-by-play dataset (1999-2024)
- Granularity: One row per play (~1.2 million rows, 393 columns)
- Key fields: `game_id`, `posteam`, `yardline_100`, `qtr`, `down`, `ydstogo`, `game_seconds_remaining`, `score_differential`, `vegas_home_wp`

The dataset's granularity enables real-time win probability modeling, as each row represents a decision point before the ball is snapped.

Note: Games prior to 2003 were excluded due to invalid score data. This filtering ensures training labels are accurate and consistent. Overtime plays were also excluded due to invalid target values.

Initial Cleaning

In [3]:
```
# View snapshot of top rows.
df.head()
```

Out[3]:

	Unnamed: 0	play_id	game_id	old_game_id	home_team	away_team
0	0	35.0	1999_01_ARI_PHI	1.999091e+09	PHI	ARI
1	1	60.0	1999_01_ARI_PHI	1.999091e+09	PHI	ARI
2	2	82.0	1999_01_ARI_PHI	1.999091e+09	PHI	ARI
3	3	103.0	1999_01_ARI_PHI	1.999091e+09	PHI	ARI
4	4	126.0	1999_01_ARI_PHI	1.999091e+09	PHI	ARI

5 rows × 393 columns

In [4]:
```
# Drop redundant index column.
df = df.drop('Unnamed: 0', axis = 1)
```

In [5]:
```
# View shape.
df.shape
```

Out[5]:
```
(1230855, 392)
```

In [ ]:
```
# View info and data types.
df.info(verbose=True)
```

In [ ]:
```
# Create alphabetical list of columns for easy reference.
col_list = list(df.columns)
col_list.sort()
col_list
```

In [5]:
```
# View number of games.
len(df['game_id'].unique())
```

Out[5]:
```
6988
```

Feature Engineering

We reframe key features relative to the home team to provide a consistent baseline across games.

Without this step, possession-based features such as `score_differential` would create inconsistencies for modeling.

In [6]:
```
# Create home possession and home score differential.
df['home_score_differential'] = np.where(df['home_pos'] == 1,
df['score_differential'], 0-df['score_differential'])
```

In [7]:
```
# Confirm home score differential
df[df['game_id'] == '2003_02_CAR_TB'][['score_differential',
'home_score_differential', 'desc']].iloc[-50:-40]
```

Out[7]:

	score_differential	home_score_differential	desc
187082	6.0	-6.0	(2:18) 48-S.Davis left tackle to CAR 48 for 4 yards (99-W.Sapp, 55-D.Brooks).
187083	NaN	NaN	Timeout #3 by TB at 02:11.
187084	6.0	-6.0	(2:11) 48-S.Davis left end to 50 for 2 yards (20-R.Barber).
187085	6.0	-6.0	(2:00) 10-T.Sauerbrun punts 40 yards to TB 10, Center-56-J.Kyle. 86-K.Williams ran ob at TB 18 for 8 yards (88-K.Hankton).
187086	-6.0	-6.0	(1:49) 14-B.Johnson pass incomplete to 19-K.Johnson (27-D.Grant).
187087	-6.0	-6.0	(1:43) 14-B.Johnson pass to 86-K.Williams ran ob at CAR 39 for 43 yards (30-M.Minter).
187088	-6.0	-6.0	(1:35) 14-B.Johnson pass incomplete to 32-M.Pittman (23-R.Howard).
187089	-6.0	-6.0	(1:28) 14-B.Johnson pass incomplete to 87-K.McCardell. RECEIVER RULED OUT OF BOUNDS
187090	-6.0	-6.0	(1:22) 14-B.Johnson pass to 32-M.Pittman pushed ob at CAR 28 for 11 yards (54-W.Witherspoon).
187091	-6.0	-6.0	(1:15) 14-B.Johnson pass to 32-M.Pittman to CAR 18 for 10 yards (54-W.Witherspoon).

In [8]:
```
# Create winner and home_win labels.
df['winner'] = np.where(df['result'] > 0, df['home_team'],
```

```
df['away_team'])
df['home_win'] = (df['home_team'] == df['winner']).astype(int)
```

We also create a `home_win` label for supervised evaluation.

While not used in model training, it allows cross-checking model predictions against actual outcomes.

In [9]:
```
# Standardize yardline to home perspective.
df['yardline_100_home'] = np.where(df['home_pos'] == 1,
df['yardline_100'], 100-df['yardline_100'])
```

We standardize field position so it always represents distance to the opponent's end zone for the home team. This avoids possession-based inconsistencies.

In [10]:
```
# Create time_weight feature.
df['time_weight'] = (1-df['game_seconds_remaining']/3600)
```

Time_weight essentially reverses and scales the game clock. The beginning of the game has a time_weight of 0, and the end has a time_weight of 1.

In [11]:
```
# Set overtime time_weight to 1.
df['time_weight'] = np.where(df['qtr'] > 4, 1, df['time_weight'])
```

Overtime is indicated as quarter 5 (6 if double OT). Although NFL overtime rules are not technically sudden death (in which the first team to score wins), it's reasonable to suggest that any play in overtime is just as important as the end of the 4th quarter and should therefore be weighted as such. For this reason, we set any overtime period to a fixed time_weight of 1.

In [12]:
```
# View and count ties.
print(df[df['result'] == 0]['game_id'].unique())
print("Count: ", len(df[df['result'] == 0]['game_id'].unique()))
```

```
['2002_10_ATL_PIT' '2008_11_PHI_CIN' '2012_10_STL_SF' '2013_12_MIN_GB'
 '2014_06_CAR_CIN' '2016_07_SEA_ARI' '2016_08_WAS_CIN'
'2018_01_PIT_CLE'
 '2018_02_MIN_GB' '2019_01_DET_ARI' '2020_03_CIN_PHI' '2021_10_DET_PIT'
 '2022_01_IND_HOU' '2022_13_WAS_NYG']
Count:  14
```

NFL tie games are incredibly rare, with only 14 occurring in the last 22 seasons. We exclude them.

Fun fact: The Bengals have four of the fourteen ties, two of which are against the Eagles.

In [12]:
```
# Exclude ties.
df = df[df['result'] != 0].copy()
```

In [13]:
```
# Remove non-play rows.
df = df[df['play_type_nfl'] != 'COMMENT']
df = df[df['play_type_nfl'] != 'END_QUARTER']
df = df[df['play_type_nfl'] != 'END_GAME']
df = df[df['play_type_nfl'] != 'TIMEOUT']
df = df[df['play_type_nfl'] != 'PENALTY']
df = df[df['play_type_nfl'] != 'UNSPECIFIED']
```

These rows are not valid game plays and are irrelevant to our model's training.

In [14]:
```
# Create cleaned dataframe.
df_cleaned = df.copy()

# Clear original large dataframe to save memory.
# del df
```

In [16]:
```
# Glimpse of cleaned dataframe.
df_cleaned.head()
```

Out[16]:

	play_id	game_id	old_game_id	home_team	away_team	season_typ
0	35.0	1999_01_ARI_PHI	1.999091e+09	PHI	ARI	RE
1	60.0	1999_01_ARI_PHI	1.999091e+09	PHI	ARI	RE
2	82.0	1999_01_ARI_PHI	1.999091e+09	PHI	ARI	RE
3	103.0	1999_01_ARI_PHI	1.999091e+09	PHI	ARI	RE
4	126.0	1999_01_ARI_PHI	1.999091e+09	PHI	ARI	RE

5 rows × 398 columns

In [15]:
```
# Export cleaned dataframe to csv.
df_cleaned.to_csv('df_cleaned.csv', index = True)
```

Notebook 2: EDA

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import  mean_squared_error, r2_score
import joblib

import math

# Set max column width to None.
pd.set_option('display.max_colwidth', None)
```

Exploratory Data Analysis

Goal: Explore relationships between play-level pre-snap features and the Vegas win probability (`vegas_home_wp`). Identify features that appear correlated, redundant, or invalid, and drop or transform accordingly. Features based on events that occur during the play will be excluded.

In [2]:
```
# Import cleaned dataset.
df = pd.read_csv('df_cleaned.csv', index_col = 0, low_memory = False)
```

In [7]:
```
# View shape.
df.shape
```

Out[7]:
```
(1085358, 398)
```

In [19]:
```
# View scoring features.
df_scores = df[['game_id', 'desc', 'total_home_score',
'total_away_score', 'home_score', 'away_score',
'home_score_differential', 'result']].copy()
df_scores.iloc[-13:]
```

Out[19]:

	game_id	desc	total_home_score	total_away_score
1230839	2024_22_KC_PHI	(3:43) (Shotgun) 15-P.Mahomes scrambles left end to PHI 7 for 7 yards (55-B.Graham).	40.0	6.0

1230840	2024_22_KC_PHI	(2:58) (Shotgun) 15-P.Mahomes pass short left to 8-D.Hopkins for 7 yards, TOUCHDOWN.	40.0	12.0
1230842	2024_22_KC_PHI	54-L.Chenal kicks onside 14 yards from KC 40 to PHI 46. 29-A.Maddox (didn't try to advance) to PHI 46 for no gain.	40.0	14.0
1230843	2024_22_KC_PHI	(2:52) 7-K.Pickett in at QB. (Shotgun) 14-K.Gainwell left guard to 50 for 4 yards (23-D.Tranquill).	40.0	14.0
1230845	2024_22_KC_PHI	(2:47) (Shotgun) 14-K.Gainwell up the middle to KC 49 for 1 yard (90-C.Omenihu).	40.0	14.0
1230846	2024_22_KC_PHI	(2:05) (Shotgun) 14-K.Gainwell left tackle to 50 for -1 yards (32-N.Bolton, 98-T.Wharton).	40.0	14.0
1230847	2024_22_KC_PHI	(1:59) (Shotgun) 7-K.Pickett pass incomplete short left to 83-J.Dotson.	40.0	14.0
1230848	2024_22_KC_PHI	(1:56) (Shotgun) 15-P.Mahomes pass deep	40.0	20.0

		middle to 1-X.Worthy for 50 yards, TOUCHDOWN.		
1230849	2024_22_KC_PHI	TWO-POINT CONVERSION ATTEMPT. 15-P.Mahomes pass to 8-D.Hopkins is complete. ATTEMPT SUCCEEDS.	40.0	22.0
1230850	2024_22_KC_PHI	7-H.Butker kicks onside 9 yards from KC 35 to KC 44. 81-G.Calcaterra (didn't try to advance) to KC 44 for no gain.	40.0	22.0
1230851	2024_22_KC_PHI	(1:47) 7-K.Pickett kneels to KC 46 for -2 yards.	40.0	22.0
1230852	2024_22_KC_PHI	(1:05) 7-K.Pickett kneels to KC 47 for -1 yards.	40.0	22.0
1230853	2024_22_KC_PHI	(:27) 7-K.Pickett kneels to KC 48 for -1 yards.	40.0	22.0

The two score features are counterintuitive. Total_home_score is the running score, while home_score is the final score.

In [20]:
```
# Confirm validity of score_differential.
df_scores['home_score_delta'] = df_scores['total_home_score'] -
df_scores['home_score']
df_scores_subset = df_scores.groupby('game_id').last().copy()
df_scores_subset.sort_values(by = 'home_score_delta').head(15)
```

Out[20]:

	desc	total_home_score	total_away_score	home_sc

game_id			
2001_09_CIN_JAX	(:19) 12-J.Quinn to JAX 19 for -1 yards.	4.0	39.0
2002_04_NYJ_JAX	(:23) 9-D.Garrard to JAX 18 for -1 yards.	4.0	27.0
2001_16_KC_JAX	(:06) (Shotgun) 8-M.Brunell pass incomplete to 82-J.Smith (24-W.Bartee).	6.0	50.0
2001_11_BAL_JAX	(:04) (Shotgun) 8-M.Brunell sacked at JAX 35 for -1 yards (96-A.Thomas).	3.0	42.0
2002_05_PHI_JAX	2-D.Akers extra point is GOOD, Center-88-M.Bartrum, Holder-10-K.Detmer.	10.0	43.0
2002_13_PIT_JAX	(:33) 10-K.Stewart to PIT 49 for -1 yards.	5.0	43.0
2001_01_PIT_JAX	(:14) 8-M.Brunell to PIT 17 for -1 yards.	3.0	21.0
2002_01_IND_JAX	(:01) (Shotgun) 8-M.Brunell pass incomplete to 86-M.Ross (28-I.Bashir).	7.0	46.0

2002_08_HOU_JAX	(:02) (Shotgun) 8-M.Brunell pass incomplete to 81-B.Shaw (42-M.Coleman).	5.0	35.0
2001_03_CLE_JAX	(:23) (Shotgun) 12-J.Quinn FUMBLES (Aborted) at JAX 37, recovered by JAX-63-B.Meester at JAX 35. 63-B.Meester to CLE 42 for 23 yards (52-B.Boyer). Backward Pass Attempting to Pass to Mack In the Grasp of CB95	2.0	35.0
2002_14_CLE_JAX	4-P.Dawson extra point is GOOD, Center-97-R.Kuehl, Holder-17-C.Gardocki.	8.0	33.0
2002_10_WAS_JAX	(:25) 8-M.Brunell to JAX 9 for -1 yards.	14.0	19.0
2001_12_GB_JAX	(:41) 4-B.Favre to JAX 40 for -1 yards.	9.0	40.0
2001_02_TEN_JAX	(:29) 8-M.Brunell to TEN 43 for -1 yards.	7.0	12.0
	(:18) 11-		

2001_06_BUF_JAX	R.Johnson to JAX 49 for -1 yards.	4.0	19.0

There are a handful of games from 2001 and 2002 that contain scoring inconsistencies.

Interestingly, every one of these discrepancies happens to be a Jaguars home game. It's also noteworthy that the total score is still correct in each of these games, even though the score itself is not.

In [21]:
```
# View discrepancy detail.
game = '2001_09_CIN_JAX'
df_scores[df_scores['game_id'] == game].tail()
```

Out[21]:

	game_id	desc	total_home_score	total_away_sc
111863	2001_09_CIN_JAX	(:45) 3-J.Kitna pass to 83-D.Farmer pushed ob at JAX 15 for 9 yards (21-A.Beasley).	4.0	3
111864	2001_09_CIN_JAX	(:41) 3-J.Kitna pass to 89-M.Battaglia to JAX 6 for 9 yards (20-D.Darius).	4.0	3
111866	2001_09_CIN_JAX	(:33) 3-J.Kitna pass incomplete to 82-T.McGee (20-D.Darius).	4.0	3
111867	2001_09_CIN_JAX	(:27) 3-J.Kitna pass intended for 84-T.Houshmandzadeh INTERCEPTED by 21-A.Beasley at JAX 0. Touchback.	4.0	3
111868	2001_09_CIN_JAX	(:19) 12-J.Quinn to JAX 19 for -1 yards.	4.0	3

The final score of this game was 30-13 and is correctly reflected in the home_score and away_score columns. Yet our play-by-play columns indicate the final score as 4-39, and score_differential is consistent with the latter.

Because these scoring inconsistencies appear across multiple columns and games, correcting them individually is not feasible.

To ensure target labels remain accurate, we exclude seasons prior to 2003 from the dataset.

In [3]:
```
# Filter to 2003 and later.
df = df[df['season'] >= 2003].copy()
```

In [10]:
```
# View new shape.
df.shape
```

Out[10]:
```
(912182, 398)
```

Filtered dataset still has 900,000+ rows.

In [12]:
```
# View game spreads.
df.groupby('game_id')[['home_team','spread_line']].first().tail()
```

Out[12]:

game_id	home_team	spread_line
2024_20_LA_PHI	PHI	7.0
2024_20_WAS_DET	DET	8.5
2024_21_BUF_KC	KC	1.5
2024_21_WAS_PHI	PHI	6.0
2024_22_KC_PHI	PHI	-1.5

Point spreads indicate the number of points by which a team is favored to win.

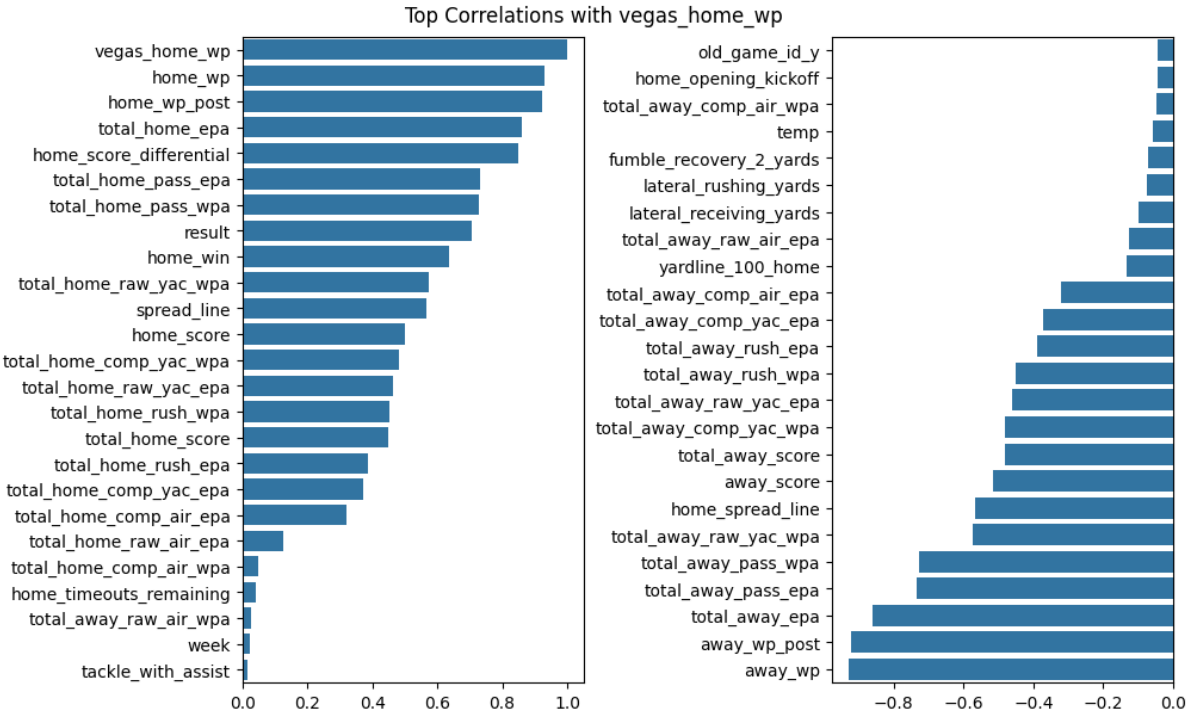
Teams that are favored to win are reflected by a negative number. Although this may seem counterintuitive, it makes sense to think about it from a score adjustment standpoint: if a team is favored -7, you would then subtract 7 from their score and re-evaluate. If they're still winning based on your result, they've "covered the spread." This feature essentially evens the playing field between two mismatched teams.

Game_id is in the format of season_week_away_home. Given this, it's apparent that the indicated spread is from the point of view of the away team based on Bleacher Report's 2024 NFC Championship preview (Washington @ Philadelphia), where their reported point spread is PHI -6. Our data, on the other hand, indicates +6 as seen above.

We reframe spreads from the home team's perspective so that negative indicates a home favorite. This improves interpretability and avoids confusion when training models.

In [4]:
```
# Create home spread.
df['home_spread_line'] = 0-df['spread_line']
```

In [8]:
```
# View correlations.
target = 'vegas_home_wp'
corr = df.corr(numeric_only = True)[target].sort_values(ascending =
False)

fig, ax = plt.subplots(1, 2, figsize = (10,6), constrained_layout =
True)
top_corr = corr[:25]
bottom_corr = corr[-30:-6]


sns.barplot(x = top_corr.values, y = top_corr.index,  ax = ax[0])
sns.barplot(x = bottom_corr.values, y = bottom_corr.index, ax =
ax[1])

fig.align_xlabels()

ax[0].set_ylabel('')
ax[1].set_ylabel('')
plt.suptitle('Top Correlations with vegas_home_wp')

plt.show()
```



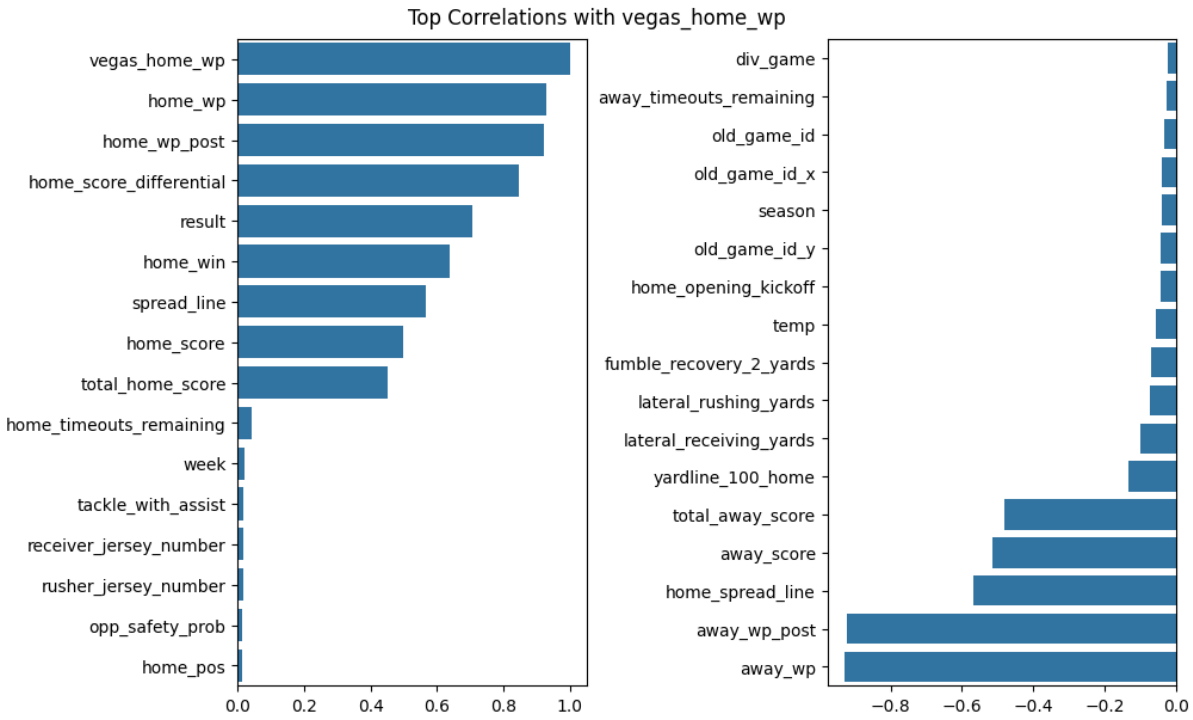
Although `epa` (expected points added) and `wpa` (win probability added) show strong correlations with the target, we exclude them because they are derived, post-play statistics that incorporate information not available before the snap.

Our objective is to model win probability *before the play occurs*.

In [9]:
```
# Exclude columns containing 'epa' or 'wpa'.
exclude_substrings = ['epa', 'wpa']

cols_to_keep = []

for col in df.columns:
    if 'epa' in col:
        continue
    if 'wpa' in col:
        continue
    else:
        cols_to_keep.append(col)
len(cols_to_keep)
```

Out[9]: 359

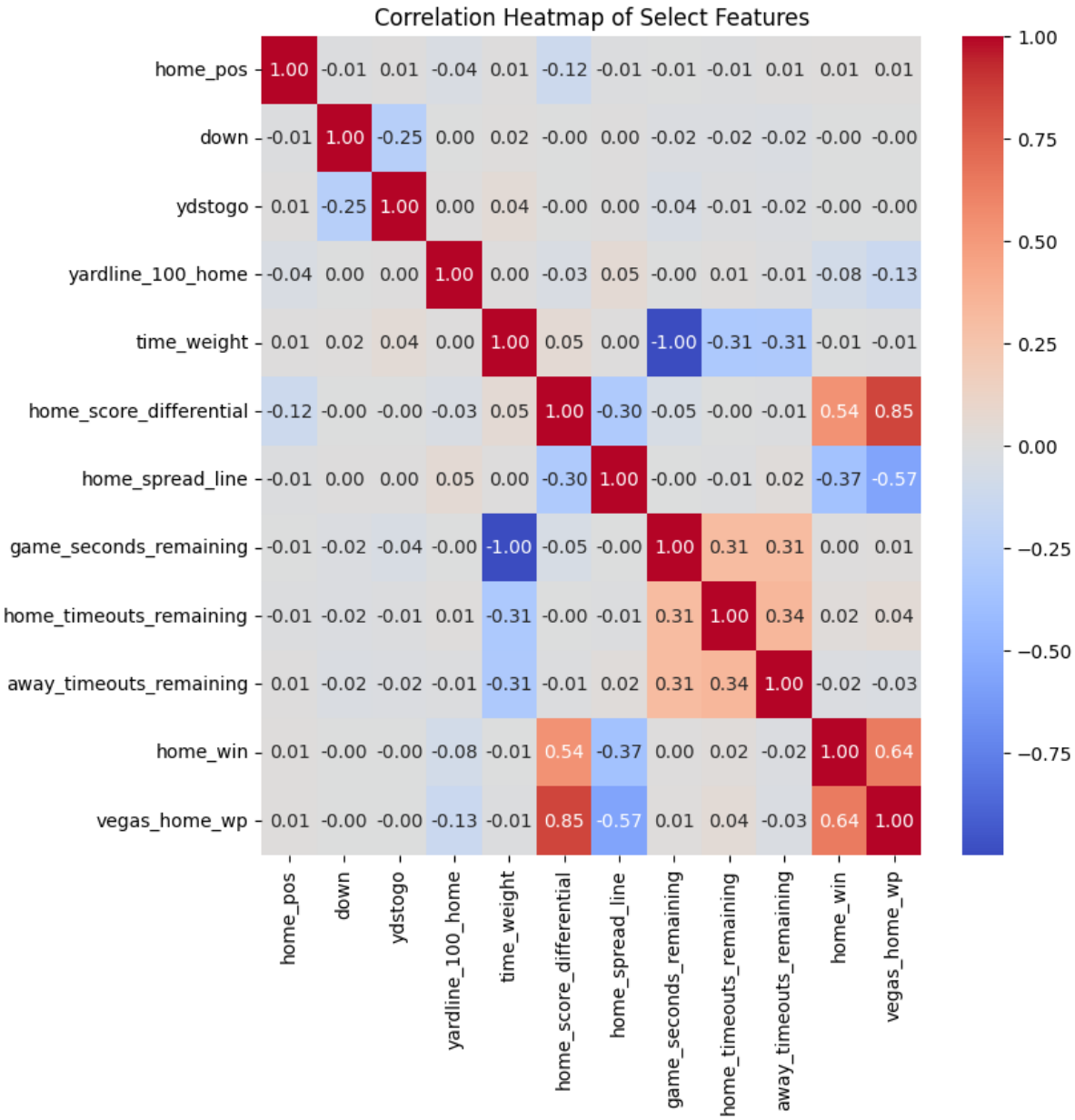
In [10]:
```
# Plot correlations.
filtered_corr = df[cols_to_keep].corr(numeric_only = True)
[target].sort_values(ascending = False)

top_corr = filtered_corr[:16]
bottom_corr = filtered_corr[-23:-6]

fig, ax = plt.subplots(1, 2, figsize = (10,6), constrained_layout =
True)

sns.barplot(x = top_corr.values, y = top_corr.index,  ax = ax[0])
sns.barplot(x = bottom_corr.values, y = bottom_corr.index, ax =
ax[1])

fig.align_xlabels()

ax[0].set_ylabel('')
ax[1].set_ylabel('')

plt.suptitle('Top Correlations with vegas_home_wp')
plt.show()
```



Based on these correlations, we subset our desired pre-play game state features for modeling.

In [11]:
```
# Create preliminary list of features.
features = ['game_id',
            'home_pos',
            'down',
            'ydstogo',
            'yardline_100_home',
            'time_weight',
            'home_score_differential',
            'desc',
            'play_type_nfl',
            'home_spread_line',
            'game_seconds_remaining',
            'home_timeouts_remaining',
            'away_timeouts_remaining',
            'home_win',
            'vegas_home_wp'
           ]
```

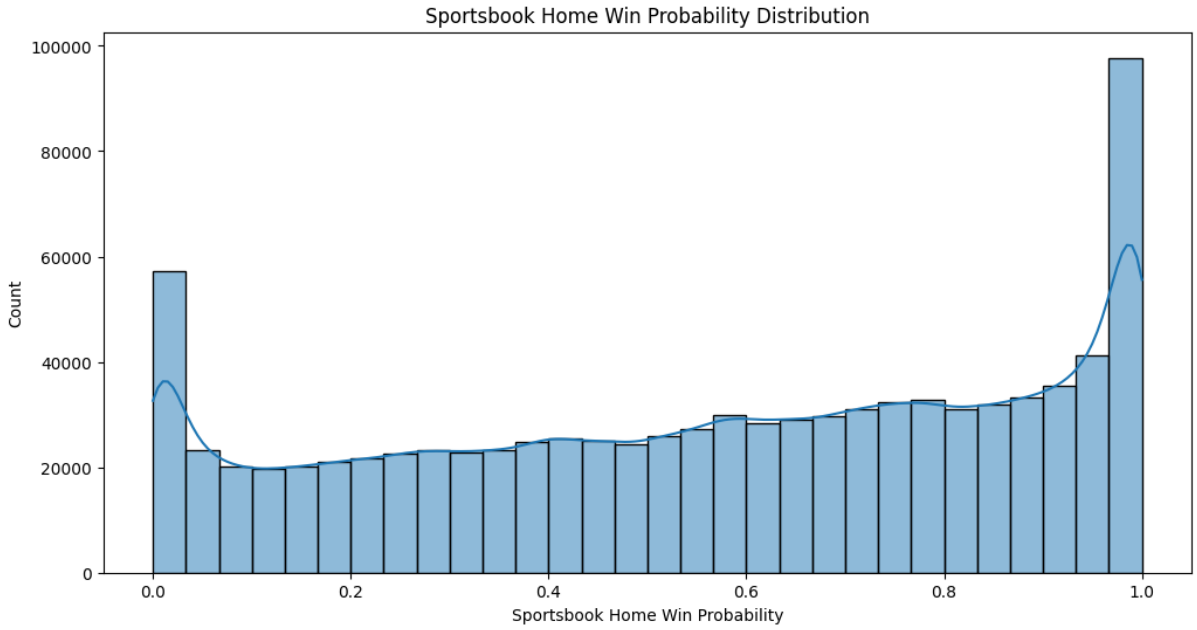
In [12]:
```
# View heatmap of correlations.
plt.figure(figsize=(8, 8))
sns.heatmap(df[features].corr(numeric_only=True), annot=True,
fmt=".2f", cmap="coolwarm")
plt.title("Correlation Heatmap of Select Features")
plt.show()
```



Our target is most correlated with score differential and the opening point spread, as well as home_win.

We are including home_win here as a reference point, and it will not be included in modeling.

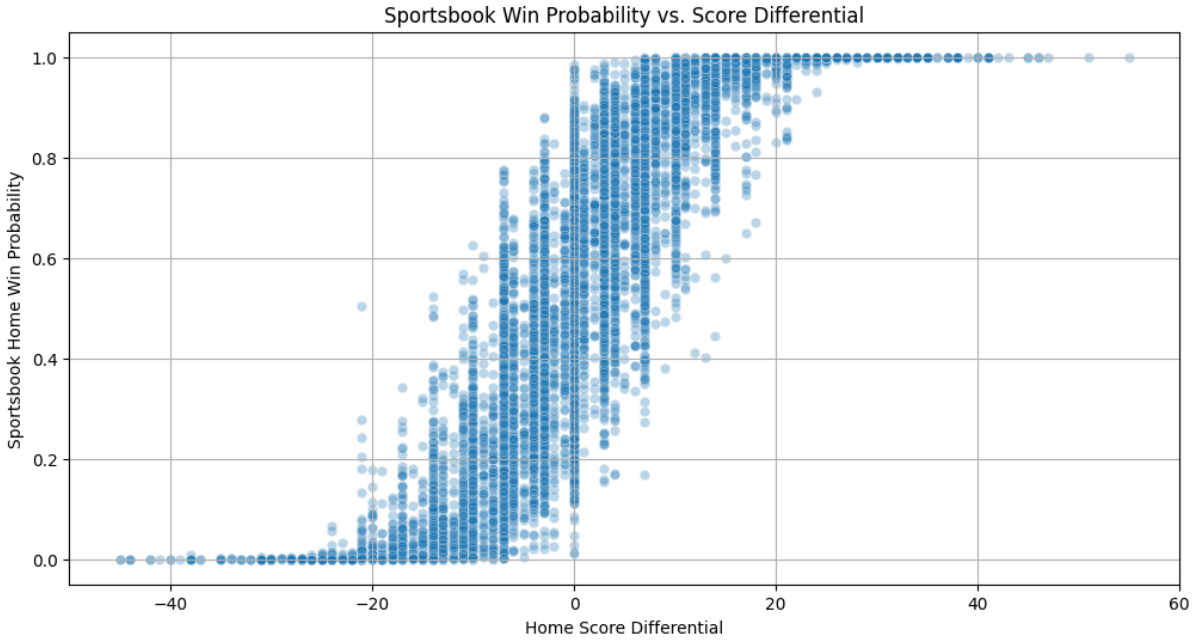
In [13]:
```
# View histogram of target.
plt.figure(figsize=(12, 6))
sns.histplot(df['vegas_home_wp'], bins=30, kde=True)
plt.title("Sportsbook Home Win Probability Distribution")
plt.xlabel("Sportsbook Home Win Probability")
plt.show()
```



Our target is bimodal, with peaks hear 0 and 1.

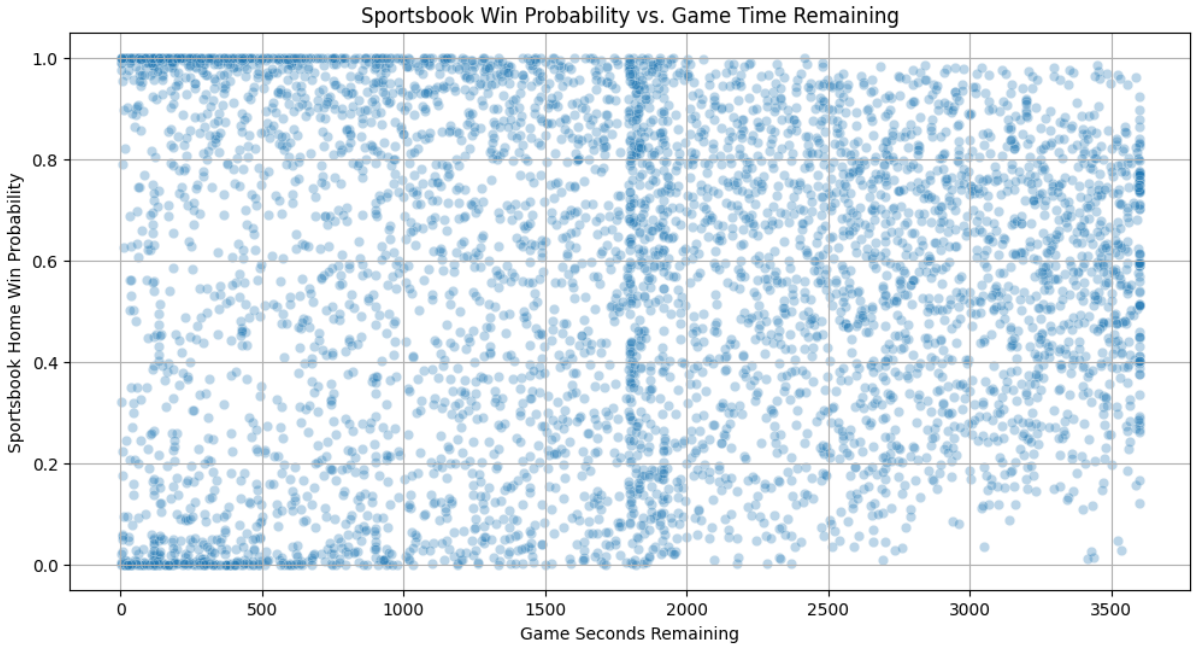
There is enough data spread out between the peaks that our model can learn from both close games and blowouts.

In [14]:
```
# View scatterplot of probability vs. score differential.
plt.figure(figsize=(12, 6))
sns.scatterplot(data=df.sample(10000, random_state = 42), # Random
sample.
                x='home_score_differential',
                y='vegas_home_wp',
                alpha=0.3)
plt.title("Sportsbook Win Probability vs. Score Differential")
plt.xlabel("Home Score Differential")
plt.ylabel("Sportsbook Home Win Probability")
plt.grid(True)
plt.show()
```



Little movement at probabilities near 0 and 1. Our model will likely perform better in between these extremes.

In [15]:
```
# View scatterplot of win probability vs. time remaining.
plt.figure(figsize=(12, 6))
sns.scatterplot(
    data=df.sample(5000, random_state = 42),
    x='game_seconds_remaining',
    y='vegas_home_wp',
    alpha=0.3
)
plt.title("Sportsbook Win Probability vs. Game Time Remaining")
plt.xlabel("Game Seconds Remaining")
plt.ylabel("Sportsbook Home Win Probability")
plt.grid(True)
plt.show()
```



Unsurprisingly, probabilities are centered toward the middle at the beginning of games and cluster around 0 and 1 toward the end.

In [16]:
```
# View null counts.
df[features].isna().sum()
# df = df[pre_features].copy()
```

Out[16]:
```
game_id                       0
home_pos                      0
down                      93053
ydstogo                       0
yardline_100_home          5934
time_weight                 179
home_score_differential    5934
desc                          0
play_type_nfl                 0
home_spread_line              0
game_seconds_remaining      179
home_timeouts_remaining       0
away_timeouts_remaining       0
home_win                      0
vegas_home_wp                 0
dtype: int64
```

In [17]:
```
# View description of down nulls.
df[df['down'].isna()]['desc']
```

Out[17]:
```
183951
GAME
183952
7-B.Gramatica kicks 59 yards from ARI 30 to DET 11. 18-E.Drummond to
DET 30 for 19 yards (34-D.Jackson).
183976
7-B.Gramatica kicks 60 yards from ARI 30 to DET 10. 18-E.Drummond to
DET 42 for 32 yards (82-K.Kasper).
183986
4-J.Hanson kicks 70 yards from DET 30 to ARI 0. 82-K.Kasper to ARI 21
for 21 yards (28-Bra.Walker).
184001     4-J.Hanson kicks 67 yards from DET 30 to ARI 3. 82-
K.Kasper to ARI 34 for 31 yards (28-Bra.Walker). PENALTY on DET-82-
C.Fitzsimmons, Offensive Offside, 5 yards, enforced at DET 30 - No
Play.

...
1230817
4-J.Elliott kicks 65 yards from PHI 35 to end zone, Touchback to the
KC 30.
1230826
4-J.Elliott kicks 69 yards from PHI 35 to KC -4. 81-N.Remigio to KC
25 for 29 yards (54-J.Trotter).
1230842
54-L.Chenal kicks onside 14 yards from KC 40 to PHI 46. 29-A.Maddox
(didn't try to advance) to PHI 46 for no gain.
1230849
TWO-POINT CONVERSION ATTEMPT. 15-P.Mahomes pass to 8-D.Hopkins is
complete. ATTEMPT SUCCEEDS.
1230850
7-H.Butker kicks onside 9 yards from KC 35 to KC 44. 81-G.Calcaterra
(didn't try to advance) to KC 44 for no gain.
Name: desc, Length: 93053, dtype: object
```

In [18]:
```
# View field position nulls.
df[df['yardline_100_home'].isna()]['desc']
```

Out[18]:
```
183951     GAME
184133     GAME
184300     GAME
184488     GAME
184673     GAME
           ...
1229932    GAME
1230113    GAME
1230303    GAME
1230480    GAME
1230679    GAME
Name: desc, Length: 5934, dtype: object
```

These null values appear to be a combination of timeouts, kickoffs, period changes, penalties, and conversions. We drop them.

In [19]:
```
# Drop nulls from desired columns.
df.dropna(subset = ['yardline_100'], inplace = True)
df.dropna(subset = ['down'], inplace = True)
```

In [20]:
```
# View remaining nulls
df[features].isna().sum()
```

Out[20]:
```
game_id                    0
home_pos                   0
down                       0
ydstogo                    0
yardline_100_home          0
time_weight                0
home_score_differential    0
desc                       0
play_type_nfl              0
home_spread_line           0
game_seconds_remaining     0
home_timeouts_remaining    0
away_timeouts_remaining    0
home_win                   0
vegas_home_wp              0
dtype: int64
```

No remaining nulls.

In [21]:
```
df_filtered = df[features].copy()
df_filtered.to_csv('df_filtered.csv')
```

EDA Summary

Key findings:

- Dropped pre-2003 due to scoring inconsistencies.
- Reframed spreads relative to home team for clarity.
- Excluded post-play features (EPA, WPA) to prevent data leakage.
- Dropped rows with null values.

Notebook 3: Model Selection and Training

In [1]:
```
import pandas as pd
import numpy as np

from sklearn.preprocessing import OneHotEncoder, StandardScaler,
FunctionTransformer
from sklearn.model_selection import train_test_split,
cross_val_score, RandomizedSearchCV
from sklearn.linear_model import Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer,
TransformedTargetRegressor
from sklearn.metrics import  mean_squared_error, r2_score
import joblib

import math

from xgboost import XGBRegressor

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense, Flatten

from scikeras.wrappers import KerasRegressor
import keras_tuner

from scipy.special import logit, expit
from utils import logit_func, expit_func

import optuna
import random
from datetime import datetime

import os

# Set max column width to None.
pd.set_option('display.max_colwidth', None)

# Import cleaned dataset.
df = pd.read_csv('df_filtered.csv', index_col = 0)
```

In this notebook, we evaluate several regression models for predicting pre-snap win probability, comparing linear, tree-based, and neural approaches.

We clip the range of target values to avoid undefined errors at 0 and 1. We then logit-transform our target to stabilize regression and avoid compression at the extremes of 0 and 1. The subsequent inverse transform allows predictions to be returned as probabilities.

To keep preprocessing reproducible and portable across experiments and deployment, this transformation is encapsulated in logit_func and expit_func within utils.py, and then wrapped in TransformedTargetRegressor. These ensure consistent handling of edge cases [0, 1] during both training and prediction.

Preprocessing

In [3]:
```
features_to_encode = [
    'down'
]

features_to_scale = [
    'yardline_100_home',
    'ydstogo',
    'home_score_differential',
    'home_spread_line'
]

passthrough_features = [
    'home_pos',
    'time_weight',
    'home_timeouts_remaining',
    'away_timeouts_remaining']

target = ['vegas_home_wp']

preprocessing = ColumnTransformer(
    transformers=[
        ('encoder', OneHotEncoder(drop='first', sparse_output=False),
features_to_encode),
        ('scaler', StandardScaler(), features_to_scale),
        ('noop', FunctionTransformer(validate=False),
passthrough_features)
    ],
    verbose_feature_names_out=False
).set_output(transform='pandas')


X = df.drop('vegas_home_wp', axis = 1)
y = df['vegas_home_wp']
```

Model Selection

Ridge/Lasso: These are linear models; regularization tests whether a simple linear approximation can capture win probability dynamics.

In [ ]:
```
# Ridge.


ridge_params = {
    'model__alpha': np.logspace(-3, 3, 20),
    'model__fit_intercept': [True, False],
    'model__max_iter': [1000, 5000, 10000],
    'model__tol': [1e-4, 1e-3, 1e-2],
    'model__solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sag',
'saga'],
    # 'model__solver': ['auto', 'svd', 'cholesky', 'sag', 'saga'],
    # 'model__positive': [True, False],
    'model__copy_X': [True, False]
}


ridge_model = Ridge(random_state = 42)

ridge_pipeline = Pipeline([
    ('preprocessing', preprocessing),
    ('model', ridge_model)
])

ridge_search = RandomizedSearchCV(
        ridge_pipeline,
        ridge_params,
        n_iter=100,
        cv=3,
        scoring = 'neg_mean_squared_error',
        verbose = 3,
        random_state = 42
    )

# Transformed target regressor with logit/expit transformation.
ridge_model = TransformedTargetRegressor(
    regressor = ridge_search,
    func = logit_func,
    inverse_func = expit_func
)


ridge_model.fit(X, y)
print(f'Ridge best score: {-ridge_model.regressor_.best_score_}')
print(f'Ridge best params: {ridge_model.regressor_.best_params_}')
```

In [ ]:
```
# Lasso.

lasso_params = {
```

```
    'model__alpha': np.logspace(-4, 1, 20),
    'model__max_iter': [1000, 5000, 10000],
    'model__fit_intercept': [True, False],
    'model__tol': [1e-4, 1e-3, 1e-2],
    'model__selection': ['cyclic', 'random'],
    'model__positive': [True, False],
    'model__warm_start': [True, False]
}


lasso_model = Lasso(random_state = 42)

lasso_pipeline = Pipeline([
    ('preprocessing', preprocessing),
    ('model', lasso_model)
])


lasso_search = RandomizedSearchCV(
        lasso_pipeline,
        lasso_params,
        n_iter=100,
        cv=3,
        scoring = 'neg_mean_squared_error',
        verbose = 3,
        random_state = 42
    )

lasso_model = TransformedTargetRegressor(
    regressor = lasso_search,
    func = logit_func,
    inverse_func = expit_func
)


lasso_model.fit(X, y)
print(f'Lasso best score: {-lasso_model.regressor_.best_score_}')
print(f'Lasso best params: {lasso_model.regressor_.best_params_}')
```

Random Forest: Tree-based ensemble that captures nonlinearities without heavy tuning, but less efficient than boosting methods.

In [ ]:
```
# Random Forest.

rf_params = {
            'model__n_estimators': [50, 100, 200, 500],
            'model__max_depth': [None, 10, 20, 30, 50],
            'model__min_samples_split': [2, 5, 10],
            'model__min_samples_leaf': [1, 2, 4],
            'model__max_features': ['sqrt', 'log2'],
            'model__bootstrap': [True, False]
        }
```

```
rf_model = RandomForestRegressor(random_state = 42)

rf_pipeline = Pipeline([
    ('preprocessing', preprocessing),
    ('model', rf_model)
])

rf_search = RandomizedSearchCV(
        rf_pipeline,
        rf_params,
        n_iter=100,
        cv=3,
        scoring = 'neg_mean_squared_error',
        verbose = 3,
        random_state = 42
    )

rf_model = TransformedTargetRegressor(
    regressor = rf_search,
    func = logit_func,
    inverse_func = expit_func
)


rf_model.fit(X, y)
print(f'Random forest best score: {-
rf_model.regressor_.best_score_}')
print(f'Random forest best params:
{rf_model.regressor_.best_params_}')
```

XGBoost: Gradient boosting, strong candidate for tabular data, capable of modeling interactions while controlling overfitting.

In [ ]:
```
# XGBoost.

xgb_params = {
    'model__n_estimators': [100, 300, 500],
    'model__learning_rate': [.01, .05, .1, .3],
    'model__subsample': [.5, .7, 1],
    'model__colsample_bytree': [.5, .7, 1],
    'model__reg_alpha': np.logspace(-4, 4, 20),
    'model__reg_lambda': np.logspace(-4, 4, 20),
    'model__gamma': [0, .1, .3, 1],
    'model__max_depth': [3, 5, 7, 10]
}

xgb_model = XGBRegressor(random_state = 42)

xgb_pipeline = Pipeline([
    ('preprocessing', preprocessing),
    ('model', xgb_model)
```

```
])

xgb_search = RandomizedSearchCV(
        xgb_pipeline,
        xgb_params,
        n_iter=100,
        cv=3,
        scoring = 'neg_mean_squared_error',
        verbose = 3,
        random_state = 42
    )

xgb_model = TransformedTargetRegressor(
    regressor = xgb_search,
    func = logit_func,
    inverse_func = expit_func
)


xgb_model.fit(X, y)
print(f'XGBoost best score: {-xgb_model.regressor_.best_score_}')
print(f'XGBoost best params: {xgb_model.regressor_.best_params_}')
```

Neural network: Explores deep learning's ability to model complex relationships, through it is less interpretable.

GridSearchCV and RandomizedSearchCV do not allow testing with different numbers of units among varying numbers of layers. Keras_tuner.RandomSearch will test 2, 3, 4, and 5 layers, with varying numbers of units in each layer.

Keras_tuner is not compatible with our pipeline as it does not employ the .fit() method. Therefore we transform X_train and X_test manually, and feed the transformed results into the tuner.

We perform 100 runs with 10 epochs each, and then evaluate the top 5 models. We also repeat this process several times to ensure consistent results.

In [ ]:
```
# Subset X and y.
X = df.drop('vegas_home_wp', axis = 1)
y = df['vegas_home_wp']

# Explicitly define logit function.
def logit_func(y):
    tol = 1e-5
    return logit(np.clip(y, tol, 1-tol))
def expit_func(y):
    return expit(y)

# Logit-transform target.
y = logit_func(y)
```

```
# Train-test split.
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = .2,
                                                    random_state =
42)

# Transform data.
X_train_transformed = preprocessing.fit_transform(X_train)
X_test_transformed = preprocessing.transform(X_test)

tf.get_logger().setLevel("ERROR")

# Create function to build keras models.
def build_model(hp):
    keras_model = Sequential()

    num_layers = hp.Int('num_layers',
                        min_value = 2,
                        max_value = 5)

    for i in range(num_layers):
        keras_model.add(Dense(hp.Choice(f'units_{i}', [16, 32, 64,
128, 256, 512, 1024]),
                        activation = 'relu'))
    keras_model.add(Dense(1, activation = 'linear'))

    keras_model.compile(optimizer = 'adam',
                  loss = 'mse',
                  metrics = ['mse'])
    return keras_model

# Instantiate tuner with build function.
tuner = keras_tuner.RandomSearch(
    build_model,
    objective = 'mse',
    max_trials = 100,
    overwrite = True
)

# Use tuner to search.
tuner.search(X_train_transformed,
             y_train,
             epochs = 10,
             validation_data = (X_test_transformed, y_test))

# Top models.
top_models = tuner.get_best_models(num_models=5)

# View summary of top models.
for i, model in enumerate(top_models):
```

```
    model.build(input_shape = (None, X_train_transformed.shape[1]))
    loss, acc = model.evaluate(X_test_transformed, y_test, verbose =
0)

    print(f"\nModel {i+1} Summary: (Accuracy: {acc:.4f})")
    print(f'test: {loss}')
    model.summary()
```

After several iterations, the top performing sequential neural network configuration was consistently 128/512/128/512/1.

We train a neural network with the optimal hyperparameters over 100 epochs.

In [ ]:
```
# Instantiate model.
model = Sequential()

# Set layers.
model.add(Dense(128, activation = 'relu'))
model.add(Dense(512, activation = 'relu'))
model.add(Dense(128, activation = 'relu'))
model.add(Dense(512, activation = 'relu'))
model.add(Dense(1, activation = 'linear'))

model.compile(optimizer = 'adam', loss = 'mse', metrics = ['mse'])

# Train model.
model.fit(X_train_transformed,
          y_train,
          batch_size = 32,
          epochs = 100,
          validation_data=(X_test_transformed, y_test))

# View summary
model.summary()
```

Best MSEs:

Ridge: 1.085
Lasso: 1.085
Random Forest: .047
XGBoost: .021
Neural Network: .039

The linear models (Ridge, Lasso) performed poorly, suggesting the relationship between features and win probability is nonlinear. Random forest and neural network improved fit substantially, but XGBoost provided the lowest MSE while remaining efficient and stable across folds. As a result, we select XGBoost as our production model.

We select Optuna for refinement because it explores parameter space more efficiently than grid or random search. This reduced training time while improving performance over baseline RandomizedSearchCV results.

XGBoost best hyperparameters from randomized search:

- 'model__colsample_bytree': 0.7,
- 'model__gamma': 0,
- 'model__learning_rate': 0.1,
- 'model__max_depth': 10,
- 'model__n_estimators': 300,
- 'model__reg_alpha': 1.623776739188721,
- 'model__reg_lambda': 11.288378916846883,
- 'model__subsample': 0.5

In [ ]:
```
# Set random seed for reproducibility.
np.random.seed(42)
random.seed(42)

# Subset X and y.
X = df.drop('vegas_home_wp', axis = 1)
y = df['vegas_home_wp']

# Train-test split.
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = .2,
                                                    random_state =
42)
# Define objective.
def objective(trial):
    # Define hyperparameter space.
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 100, 500),
        'max_depth': trial.suggest_int('max_depth', 3, 15),
        'learning_rate': trial.suggest_float('learning_rate', .01,
.3),
        'subsample': trial.suggest_float('subsample', 0.5, 1.0),
        'colsample_bytree': trial.suggest_float('colsample_bytree',
0.5, 1.0),
        'colsample_bylevel': trial.suggest_float('colsample_bylevel',
0.5, 1.0),
        'colsample_bynode': trial.suggest_float('colsample_bynode',
0.5, 1.0),
        'reg_alpha': trial.suggest_float('reg_alpha', 0.0, 20.0),
        'reg_lambda': trial.suggest_float('reg_lambda', 0.0, 20.0),
        'min_child_weight':trial.suggest_int('min_child_weight', 1,
20),
        'gamma': trial.suggest_int('gamma', 0, 5),
```

```
        'random_state': 42

    }

    # Instantiate model.
    xgb_model = XGBRegressor(random_state = 42)

    # Wrap model in pipeline.
    model_pipe = Pipeline([
        ('preprocessing', preprocessing),
        ('model', xgb_model)
        ])


    # Wrap pipeline in transformed target regressor.
    final_model = TransformedTargetRegressor(
        regressor = model_pipe,
        func = logit_func,
        inverse_func = expit_func
        )

    # Extract parameters from dictionary.
    final_model.set_params(
        **{f'regressor__model__{key}':value for key, value in
params.items()})

    # Set scoring to cross validation with negative MSE.
    score = cross_val_score(
        model_pipe,
        X_train,
        y_train,
        cv=3,
        scoring='neg_mean_squared_error',
        n_jobs=1
    )

    return float(score.mean())

# Set save path.
cwd = os.getcwd()
db_path = f"sqlite:///{cwd}/xgb_tuning_03.db"

# Random seed in sampler for reproducibility.
sampler = optuna.samplers.TPESampler(seed=42)

# Instantiate study with sampler.
study = optuna.create_study(
    study_name="xgb_tuning_03",
    direction="maximize",
    storage=db_path,
    load_if_exists=True,
    sampler = sampler
```

```
    # skip_if_exists=True
)

# Optimize study, 500 iterations.
study.optimize(objective, n_trials = 500)

# View best MSE.
study.best_value
```

In [ ]:
```
# View best hyperparameters.
best_params = study.best_params
best_params
```

In [ ]:
```
# Explicitly define best hyperparameters.
best_params = {
    'n_estimators': 387,
    'max_depth': 15,
    'learning_rate': 0.06537652719126918,
    'subsample': 0.9286366815201457,
    'colsample_bytree': 0.9793042383096315,
    'colsample_bylevel': 0.9450262365116351,
    'colsample_bynode': 0.9847881824040057,
    'reg_alpha': 0.003750294655665686,
    'reg_lambda': 16.297997163154285,
    'min_child_weight': 7,
    'gamma': 0}

# Print list copy-paste friendly list of hyperparameters.
for k, v in best_params.items():
    print(f'{k} = {v},')
```

Optuna gave us these hyperparameters:

- n_estimators = 387,
- max_depth = 15,
- learning_rate = 0.06537652719126918,
- subsample = 0.9286366815201457,
- colsample_bytree = 0.9793042383096315,
- colsample_bylevel = 0.9450262365116351,
- colsample_bynode = 0.9847881824040057,
- reg_alpha = 0.003750294655665686,
- reg_lambda = 16.297997163154285,
- min_child_weight = 7,
- gamma = 0,

In [48]:
```
X = df.drop('vegas_home_wp', axis = 1)
y = df['vegas_home_wp']
```

```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = .2,
                                                    random_state =
42)

# Instantiate XGBoost model with optimal hyperparameters.
xgb_refined_model = XGBRegressor(
    n_estimators = 387,
    max_depth = 15,
    learning_rate = 0.06537652719126918,
    subsample = 0.9286366815201457,
    colsample_bytree = 0.9793042383096315,
    colsample_bylevel = 0.9450262365116351,
    colsample_bynode = 0.9847881824040057,
    reg_alpha = 0.003750294655665686,
    reg_lambda = 16.297997163154285,
    min_child_weight = 7,
    gamma = 0,
    random_state = 42
)

# Model wrapped in pipeline.
model_pipe = Pipeline([
    ('preprocessing', preprocessing),
    ('model', xgb_refined_model)
])


# Wrap pipeline in transformed target regressor.
final_model = TransformedTargetRegressor(
    regressor = model_pipe,
    func = logit_func,
    inverse_func = expit_func
)


# Fit model.
final_model.fit(X_train, y_train)

# Predict.
y_pred = final_model.predict(X_test)

# View results.
print(f'Model R²: {r2_score(y_test, y_pred)}')
print(f'Model MSE: {mean_squared_error(y_test, y_pred)}')
```

```
Model R²: 0.9971068400060967
Model MSE: 0.00028795634696128486
```

Our final model resulted in an R^2 of .997, meaning the model explains over 99% of the variation in sportsbook win probability, along with an inverse-transformed

MSE of .0003.

In [42]:
```
# Export model and combined dataframe.
joblib.dump(final_model, 'final_model.pkl', compress=3)

df_preout = pd.concat([X, y], axis = 1)

y_pred_full = final_model.predict(X)

df_out = pd.concat([df_preout, pd.Series(y_pred_full, index =
df_preout.index, name = 'y_pred')], axis = 1)
df_out.to_csv('df_preds.csv')
```

Model Selection Summary

- Linear models (Ridge, Lasso) underfit, confirming the need for nonlinear methods.
- Tree-based models (Random Forest, XGBoost) captured interactions effectively; XGBoost achieved the best generalization.
- Neural networks performed competitively.
- Optuna hyperparameter tuning improved efficiency and final model performance.

We select XGBoost as the final model due to its stability and efficiency.

Notebook 4: Iterative EDA and Refinement

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import  mean_squared_error, r2_score
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer,
TransformedTargetRegressor
from sklearn.preprocessing import OneHotEncoder, StandardScaler,
FunctionTransformer
from xgboost import XGBRegressor

import joblib
import math

from scipy.special import logit, expit
from utils import logit_func, expit_func

import optuna
import random
from datetime import datetime

import os

# Set max column width to None.
pd.set_option('display.max_colwidth', None)

# Import dataframe and model.
df = pd.read_csv('df_preds.csv', index_col = 0)
model = joblib.load('final_model.pkl')
```

This notebook demonstrates an iterative refinement process: identifying discrepancies between model and sportsbook predictions, investigating their causes, and refining data and model training accordingly. This aligns with best practices for error analysis and improvement.

In [5]:
```
# Create prediction differential column.
df['pred_diff'] = df['y_pred'] - df['vegas_home_wp']
# df['abs_pred_diff']  = abs(df['y_pred'] - df['vegas_home_wp'])
```

Iterative EDA

Our model's predictions uncovered additional discrepancies in the dataset. We

address them here and retrain the model as needed.

In [4]:
```
# View largest prediction differences.
df.sort_values(by = 'pred_diff', ascending = False).head(10)
```

Out[4]:

	game_id	home_pos	down	ydstogo	yardline_100_home	tir
1023155	2020_15_LAC_LV	1	2.0	7.0	24.0	
924633	2018_14_BAL_KC	1	1.0	10.0	12.0	
1023162	2020_15_LAC_LV	1	4.0	5.0	5.0	

1023156	2020_15_LAC_LV	1	3.0	1.0	18.0
386631	2007_07_CHI_PHI	1	1.0	10.0	65.0
870360	2017_11_KC_NYG	1	4.0	5.0	36.0
1081228	2021_20_BUF_KC	1	2.0	6.0	34.0
1044895	2021_05_IND_BAL	1	2.0	5.0	5.0
1044890	2021_05_IND_BAL	1	2.0	3.0	26.0

1023154	2020_15_LAC_LV	1	1.0	10.0		27.0

These investigations highlight cases where sportsbook probabilities may be unreliable. Rather than treating our model as wrong in these instances, we use discrepancies as a diagnostic tool to uncover data quality issues.

In [241…

```
# View example.
game = '2020_15_LAC_LV'
df[df['game_id'] == game][['game_id', 'game_seconds_remaining',
'home_score_differential', 'desc', 'play_type_nfl', 'vegas_home_wp',
'y_pred', 'pred_diff']].iloc[-8:-4]
```

Out [241…

	game_id	game_seconds_remaining	home_score_differential	
1023159	2020_15_LAC_LV	227.0	0.0	r
1023161	2020_15_LAC_LV	206.0	0.0	i
1023162	2020_15_LAC_LV	202.0	0.0	
1023164	2020_15_LAC_LV	198.0	3.0	i

The win probability of .10 for the above field goal is almost certainly invalid, as the plays before and after are .47 and .64, respectively.

Upon evaluating the model's predictions compared to vegas_home_wp, a small number of additional invalid probabilities were located.

Inspecting the largest 20 variances revealed that each play occurs in overtime.

In [242…

```
# Calculate proportion of overtime plays in dataset.
ot_plays = sum(df['time_weight']==1.0)
print(f'Number of overtime plays: {ot_plays}')
print(f'Proportion of overtime plays: {ot_plays/len(df)}')
```

```
Number of overtime plays: 5158
Proportion of overtime plays: 0.0062969324733955214
```

Because overtime probabilities are inconsistently defined in the dataset and represent a small portion of plays (< 1%), we exclude them to prevent noise from distorting model calibration.

In [208…
```
# Re-import cleaned dataset.
df_orig = pd.read_csv('df_filtered.csv', index_col = 0)
# df_orig.head()
```

In [209…
```
# Exclude overtime and confirm.
df_orig = df_orig[df_orig['time_weight'] < 1].copy()
print(f'Number of overtime plays dropped: {len(df)-len(df_orig)}')
```

```
Number of overtime plays dropped: 5158
```

We will again use Optuna to search for the optimal hyperparameters. This retraining ensures that hyperparameters are optimized for the refined dataset, rather than reusing values from earlier runs that included overtime plays.

In [210…
```
# Define preprocessing.
features_to_encode = [
    'down'
]

features_to_scale = [
    'yardline_100_home',
    'ydstogo',
    'home_score_differential',
    'home_spread_line'
]

passthrough_features = [
    'home_pos',
    'time_weight',
    'home_timeouts_remaining',
    'away_timeouts_remaining']


preprocessing = ColumnTransformer(
    transformers=[
        ('encoder', OneHotEncoder(drop='first', sparse_output=False),
features_to_encode),
        ('scaler', StandardScaler(), features_to_scale),
        ('noop', FunctionTransformer(validate=False),
passthrough_features)
    ],
    verbose_feature_names_out=False
).set_output(transform='pandas')
```

In [ ]:

```
# Set random seed for reproducibility.
np.random.seed(42)
random.seed(42)

X = df.drop('vegas_home_wp', axis = 1)
y = df['vegas_home_wp']

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = .2,
                                                    random_state =
42)
# Define objective.
def objective(trial):
    # Define hyperparameter space.
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 100, 500),
        'max_depth': trial.suggest_int('max_depth', 3, 15),
        'learning_rate': trial.suggest_float('learning_rate', .01,
.3),
        'subsample': trial.suggest_float('subsample', 0.5, 1.0),
        'colsample_bytree': trial.suggest_float('colsample_bytree',
0.5, 1.0),
        'colsample_bylevel': trial.suggest_float('colsample_bylevel',
0.5, 1.0),
        'colsample_bynode': trial.suggest_float('colsample_bynode',
0.5, 1.0),
        'reg_alpha': trial.suggest_float('reg_alpha', 0.0, 20.0),
        'reg_lambda': trial.suggest_float('reg_lambda', 0.0, 20.0),
        'min_child_weight':trial.suggest_int('min_child_weight', 1,
20),
        'gamma': trial.suggest_int('gamma', 0, 5),
        'random_state': 42

    }

    # Instantiate model.
    xgb_model_refined = XGBRegressor(random_state = 42)

    model_pipe = Pipeline([
        ('preprocessing', preprocessing),
        ('model', xgb_model_refined)
        ])

    # Wrap pipeline in transformed target regressor.
    final_model = TransformedTargetRegressor(
        regressor = model_pipe,
        func = logit_func,
        inverse_func = expit_func
        )

    # Extract parameters from dictionary.
```

```
    final_model.set_params(
        **{f'regressor__model__{key}':value for key, value in
params.items()})

    # Set scoring to cross validation with negative MSE.
    score = cross_val_score(
        model_pipe,
        X_train,
        y_train,
        cv=3,
        scoring='neg_mean_squared_error',
        n_jobs=1
    )

    return float(score.mean())

# Set save path.
cwd = os.getcwd()
db_path = f"sqlite:///{cwd}/xgb_tuning_04.db"

# Random seed in sampler for reproducibility.
sampler = optuna.samplers.TPESampler(seed=42)

# Instantiate study with sampler.
study = optuna.create_study(
    study_name="xgb_tuning_04",
    direction="maximize",
    storage=db_path,
    load_if_exists=True,
    sampler = sampler
    # skip_if_exists=True
)

# Optimize study.
study.optimize(objective, n_trials = 500)

# View best MSE.
study.best_value
```

In [ ]:
```
# View best hyperparameters.
best_params = study.best_params
best_params
```

In [ ]:
```
# Explicitly define best hyperparameters.
best_params = {
    'n_estimators': 486,
    'max_depth': 14,
    'learning_rate': 0.07134184802465568,
    'subsample': 0.8309921954982938,
    'colsample_bytree': 0.9371805471715066,
    'colsample_bylevel': 0.9704986984923443,
    'colsample_bynode': 0.8243247029432288,
```

```
    'reg_alpha': 0.011694190257938011,
    'reg_lambda': 18.292979039395572,
    'min_child_weight': 20,
    'gamma': 0}

# Print list copy-paste friendly list of hyperparameters.
for k, v in best_params.items():
    print(f'{k} = {v},')
```

Optuna gave us the hyperparameters

- n_estimators = 486,
- max_depth = 14,
- learning_rate = 0.07134184802465568,
- subsample = 0.8309921954982938,
- colsample_bytree = 0.9371805471715066,
- colsample_bylevel = 0.9704986984923443,
- colsample_bynode = 0.8243247029432288,
- reg_alpha = 0.011694190257938011,
- reg_lambda = 18.292979039395572,
- min_child_weight = 20,
- gamma = 0,

To simulate out-of-sample performance, we use the 2024 season as a holdout set. This prevents temporal leakage and evaluates how well the model generalizes to unseen seasons.

In [243…

```
# Training set 2023 and older.
df_train = df_orig[~df_orig['game_id'].str.contains("2024")].copy()

# Test set 2024.
df_test = df_orig[df_orig['game_id'].str.contains("2024")].copy()


X_train = df_train.drop('vegas_home_wp', axis = 1)
X_test = df_test.drop('vegas_home_wp', axis = 1)

y_train = df_train['vegas_home_wp']
y_test = df_test['vegas_home_wp']


# Instantiate model with optimal hyperparameters.
final_model = XGBRegressor(
    n_estimators = 486,
    max_depth = 14,
    learning_rate = 0.07134184802465568,
    subsample = 0.8309921954982938,
    colsample_bytree = 0.9371805471715066,
```

```
    colsample_bylevel = 0.9704986984923443,
    colsample_bynode = 0.8243247029432288,
    reg_alpha = 0.011694190257938011,
    reg_lambda = 18.292979039395572,
    min_child_weight = 20,
    gamma = 0,
)

# Wrap model in pipeline.
model_pipe = Pipeline([
    ('preprocessing', preprocessing),
    ('model', final_model)
])

# Feed pipeline into transformed target regressor.
ui_model = TransformedTargetRegressor(
    regressor = model_pipe,
    func = logit_func,
    inverse_func = expit_func,
    check_inverse = False
)

# Fit model
ui_model.fit(X_train, y_train)

# Predict.
y_pred = ui_model.predict(X_test)

# View Results.
print(f'Updated R²: {r2_score(y_test, y_pred)}')
print(f'Updated MSE: {mean_squared_error(y_test, y_pred)}')
```

```
Updated R²: 0.9966759519369123
Updated MSE: 0.0003253792843940762
```

We repeat the feature engineering from above and export our train and test dataframes, along with the pickled model.

In [247…

```
# Predict on training data (2003–2023).
y_pred_train = ui_model.predict(X_train)

# Create training dataframe with predictions.
df_03_23 = pd.concat([df_train, pd.Series(y_pred_train, index =
                                          df_train.index,
                                          name = 'y_pred')], axis = 1)


# Create test dataframe with predictions.
df_24 = pd.concat([df_test, pd.Series(y_pred, index =
                                      df_test.index,
                                      name = 'y_pred')], axis = 1)
```

```
# Create prediction differential columns in each dataframe.
df_03_23['pred_diff'] = df_03_23['y_pred'] -
df_03_23['vegas_home_wp']
df_24['pred_diff'] = df_24['y_pred'] - df_24['vegas_home_wp']

# Export train and test dataframes.
df_03_23.to_csv('df_03_23.csv')
df_24.to_csv('df_24.csv')

# Export model.
joblib.dump(ui_model, 'ui_model.pkl', compress=3)
```

Iterative Refinement Summary

- Model prediction discrepancies flagged potential data issues (e.g., invalid win probabilities on certain plays).
- Invalid win probabilities were isolated to overtime.
- Overtime plays were removed.
- Hyperparameters were re-optimized with Optuna on the refined dataset.
- A temporal split (2003-2023 training, 2024 testing) was adopted to ensure realistic generalization.

These refinements demonstrate how iterative error analysis strengthens dataset reliability.

Notebook 5: Results and Evaluation

In [3]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import  mean_squared_error, r2_score, log_loss,
brier_score_loss
import joblib

import math

# Set max column width to None.
pd.set_option('display.max_colwidth', None)

# Import training and test sets.
df_03_23 = pd.read_csv('df_03_23.csv', index_col = 0)
df_24 = pd.read_csv('df_24.csv', index_col = 0)
```

Log Loss and Brier Score Loss Comparison

In [2]:
```
# View training set metrics.
print(f'2003-2023 Log Loss (Model): {log_loss(df_03_23['home_win'],
df_03_23['y_pred']):.6f}')
print(f'2003-2023 Log Loss (Sportsbook):
{log_loss(df_03_23['home_win'], df_03_23['vegas_home_wp']):.6f}')
print()
print(f'2003-2023 Brier Score (Model):
{brier_score_loss(df_03_23['home_win'], df_03_23['y_pred']):.6f}')
print(f'2003-2023 Brier Score (Sportsbook):
{brier_score_loss(df_03_23['home_win'],
df_03_23['vegas_home_wp']):.6f}')
```

```
2003-2023 Log Loss (Model): 0.440330
2003-2023 Log Loss (Sportsbook): 0.439266

2003-2023 Brier Score (Model): 0.146132
2003-2023 Brier Score (Sportsbook): 0.145770
```

Log Loss and Brier Scores are excellent ways to measure how well predicted probabilities align with binary outcomes. A lower score is better. Our model's performance is close to the sportsbook baseline, indicating it reproduces market-level accuracy. However, the sportsbook maintains a slight edge, likely due to incorporating information beyond play-level features (e.g., injuries, weather).

We also compare the unseen 2024 predictions from our holdout set the same

way.

In [227…
```
# View test set metrics.
print(f'2024 Log Loss (Model): {log_loss(df_24['home_win'],
df_24['y_pred']):.6f}')
print(f'2024 Log Loss (Sportsbook): {log_loss(df_24['home_win'],
df_24['vegas_home_wp']):.6f}')
print()
print(f'2024 Brier Score (Model):
{brier_score_loss(df_24['home_win'], df_24['y_pred']):.6f}')
print(f'2024 Brier Score (Sportsbook):
{brier_score_loss(df_24['home_win'], df_24['vegas_home_wp']):.6f}')
```

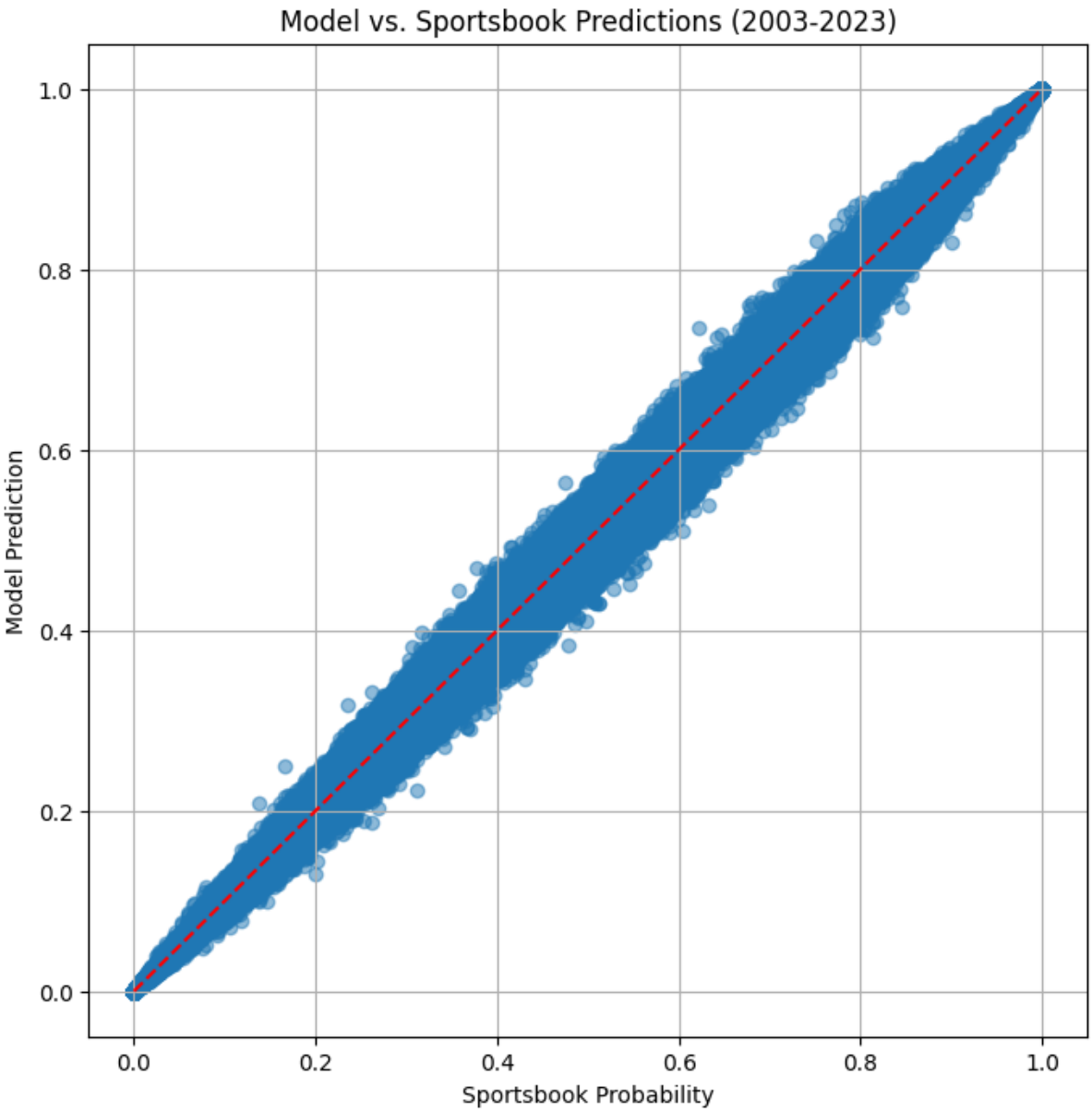
```
2024 Log Loss (Model): 0.416578
2024 Log Loss (Sportsbook): 0.414921

2024 Brier Score (Model): 0.137133
2024 Brier Score (Sportsbook): 0.136276
```

Scores are close again, but the sportsbook still has an edge.

We create a scatterplot to compare the model's predictions to the sportsbook probabilities.

Scatterplot

In [259…
```
# Create scatterplot.
plt.figure(figsize=(8, 8))
plt.scatter(df_03_23['y_pred'], df_03_23['vegas_home_wp'], alpha=0.5)

# Set grid area and diagonal.
plt.plot([0, 1], [0, 1], 'r--')

# Set labels and title.
plt.xlabel("Sportsbook Probability")
plt.ylabel("Model Prediction")
plt.title("Model vs. Sportsbook Predictions (2003–2023)")
plt.grid(True)
plt.show()
```



The clustering along the diagonal confirms our model tracks closely with sportsbook probabilities, with smaller differences near the extremes and larger differences around mid-range probabilities. There are a few outliers, but none extreme.

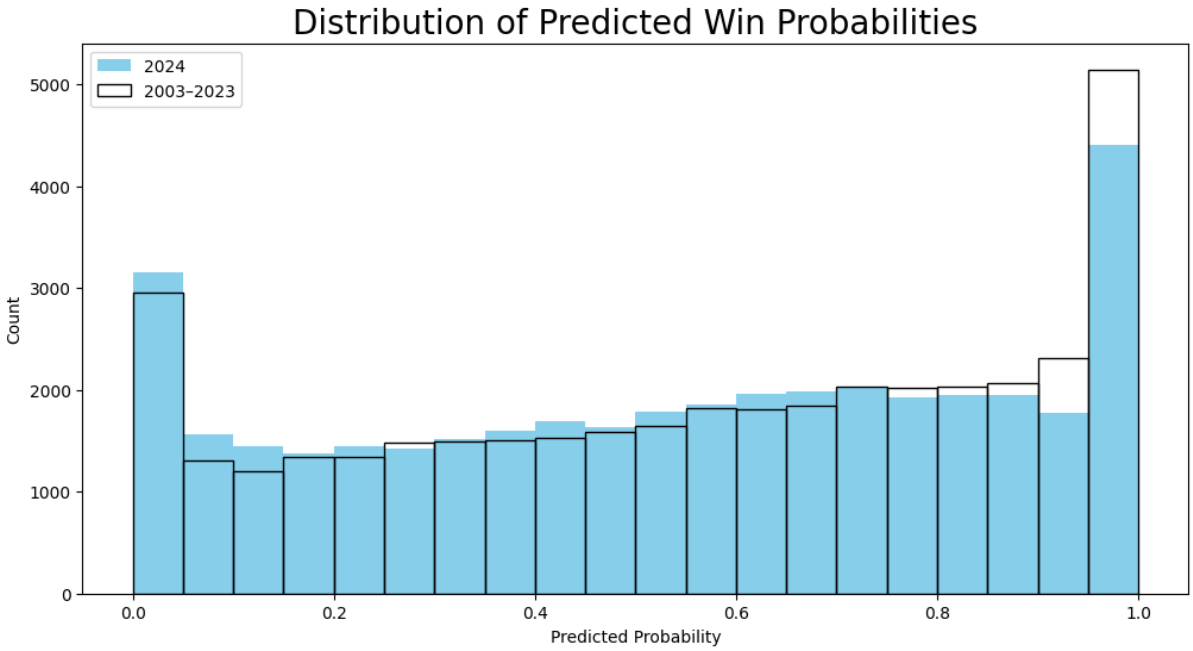
The wider spread in mid-range probabilities suggests our model may be less confident in balanced games, which could be an area for targeted refinement.

Distribution plot

In [257…

```
# Plot distribution.
plt.figure(figsize = (12,6))
plt.hist(df_24['y_pred'], bins=20, label='2024', color='skyblue')

# Sample training set to test set size.
```

```
plt.hist(df_03_23['y_pred'].sample(df_24.shape[0]), bins=20,
label='2003–2023', fill = None)

plt.title('Distribution of Predicted Win Probabilities', fontsize =
20)
plt.xlabel('Predicted Probability')
plt.ylabel('Count')
plt.legend()
plt.show()
```



The distribution of predicted probabilities is similar between the training and test sets, with the model appearing slightly more conservative on the 2024 data.

In [215…
```
print('2003–2023 Home Win Percentage:',
round(df_03_23.groupby('game_id')['home_win'].mean().mean()*100, 2))
print('2024 Home Win Percentage:', round(df_24.groupby('game_id')
['home_win'].mean().mean()*100, 2))
```

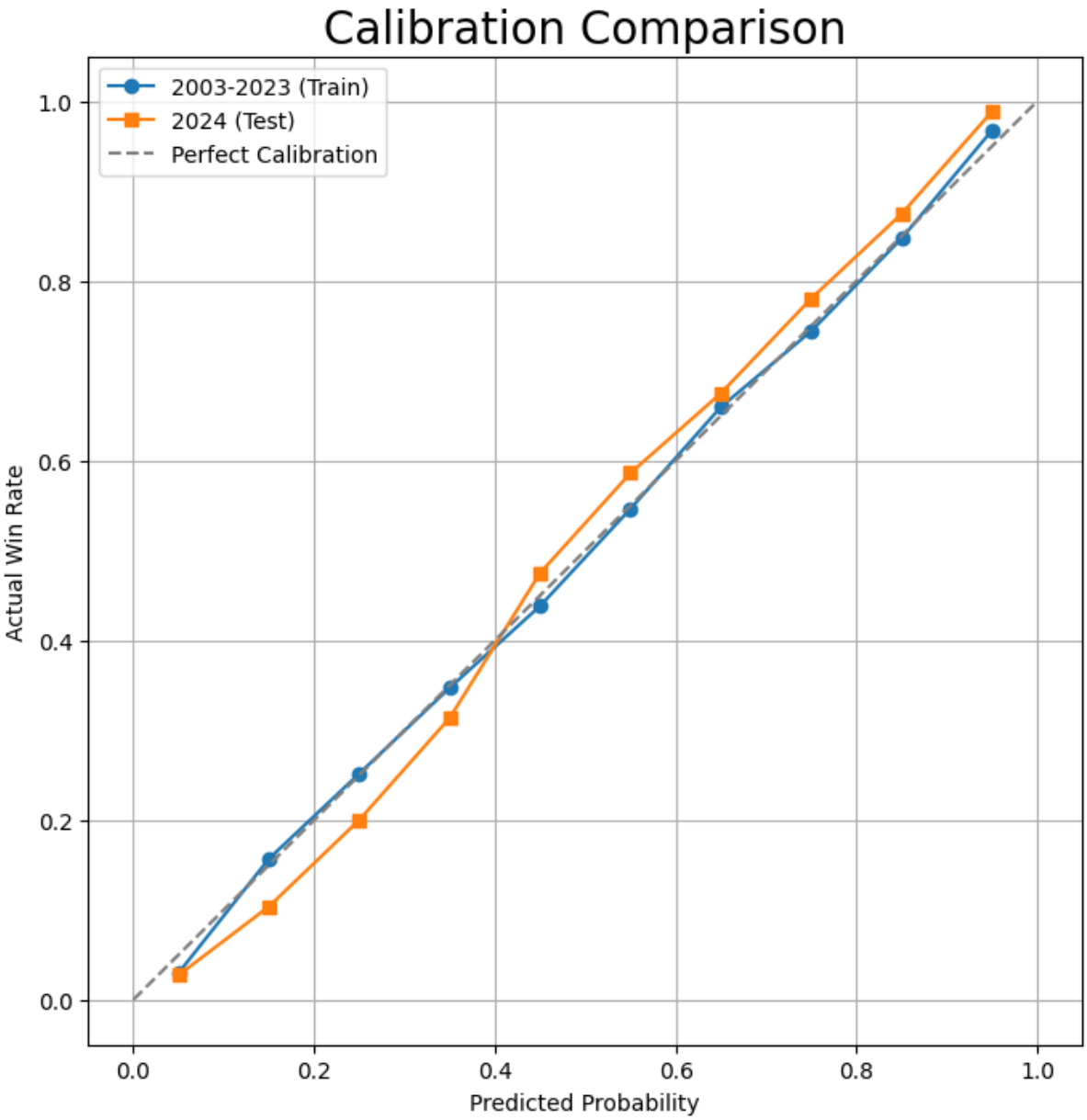
```
2003–2023 Home Win Percentage: 56.42
2024 Home Win Percentage: 54.74
```

Interestingly, the home team win rate dropped from 56.4% in 2003-2023 to 54.7% in 2024. This distribution shift highlights the importance of temporal validation: as underlying league dynamics change, the model adapts by lowering the predicted home win probabilities.

Calibration Plot

In [258…
```
# Set bins.
plt.figure(figsize = (8,8))
bins = np.linspace(0, 1, 11)
```

```
# Explicitly define centers of bins as our X variable.
centers = np.arange(.05, 1, .1)

# Bin data.
df_03_23['y_pred_bin'] = pd.cut(df_03_23['y_pred'], bins = bins)
df_train_grouped = df_03_23.groupby('y_pred_bin', observed=False)
['home_win'].mean().reset_index()

df_24['y_pred_bin'] = pd.cut(df_24['y_pred'], bins = bins)
df_test_grouped = df_24.groupby('y_pred_bin', observed=False)
['home_win'].mean().reset_index()

# Plot data.
plt.plot(centers, df_train_grouped['home_win'], marker = 'o', label =
'2003-2023 (Train)')
plt.plot(centers, df_test_grouped['home_win'], marker = 's', label =
'2024 (Test)')
plt.plot([0,1], [0, 1],  '--', color = 'gray', label = 'Perfect
Calibration')
plt.legend()
plt.xlabel('Predicted Probability')
plt.ylabel('Actual Win Rate')
plt.title('Calibration Comparison', fontsize = 20)
plt.grid(True)
```



Calibration is critical for probability models since users rely on predicted values as decision thresholds.

Our model shows good overall calibration, with minor bias (optimistic <0.4, conservative >0.4). This level of calibration compares favorably to published sports probability models, reinforcing its reliability.

To explore this further, we bin variables and visualize loss difference to identify areas where the model may struggle to generalize.

Loss Plots

In [8]:
```
# Bin data
df_03_23['pred_diff_bin'] = pd.cut(df_03_23['pred_diff'], bins =
```

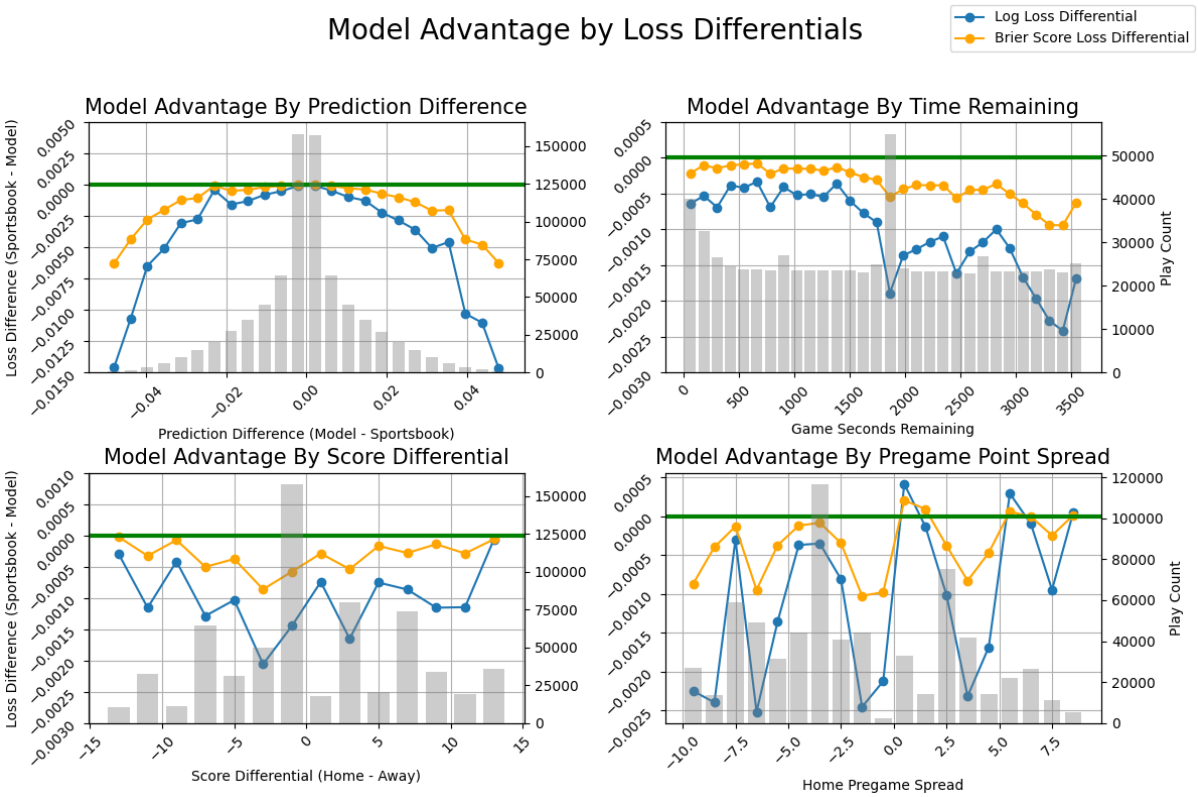
```
np.linspace(-.05, .05, 25))
df_03_23['time_bin'] = pd.cut(df_03_23['game_seconds_remaining'],
bins=30)
df_03_23['score_diff_bin'] =
pd.cut(df_03_23['home_score_differential'], bins=np.linspace(-14, 14,
15))
df_03_23['spread_bin'] = pd.cut(df_03_23['home_spread_line'],
bins=np.arange(-10, 10, 1))
# df_03_23['spread_bin'] =

loss_dict = {
    'loss_by_pred_diff':'pred_diff_bin',
    'loss_by_time':'time_bin',
    'loss_by_score_diff':'score_diff_bin',
    'loss_by_spread':'spread_bin'
}

# Reusable function for creating loss dataframes.
def create_groups(dataframe):
    loss_dataframes = {}
    for group, bin in loss_dict.items():
        loss_dataframes[group] = (
            dataframe.groupby(bin, observed=True, group_keys=False)
            .apply(lambda g: pd.Series({
                'model_log_loss': log_loss(g['home_win'],
g['y_pred'], labels=[0, 1]),
                'sportsbook_log_loss': log_loss(g['home_win'],
g['vegas_home_wp'], labels=[0, 1]),
                'model_brier_score_loss':
brier_score_loss(g['home_win'], g['y_pred'], labels = [0, 1]),
                'sportsbook_brier_score_loss':
brier_score_loss(g['home_win'], g['vegas_home_wp'], labels = [0, 1]),
                'count': len(g)
            }), include_groups = False)
            .reset_index()
        )
        loss_dataframes[group]['bin_center'] = loss_dataframes[group]
[bin].apply(lambda x:x.mid)
        loss_dataframes[group]['log_loss_diff'] =
loss_dataframes[group]['sportsbook_log_loss'] -
loss_dataframes[group]['model_log_loss']
        loss_dataframes[group]['brier_score_loss_diff'] =
loss_dataframes[group]['sportsbook_brier_score_loss'] -
loss_dataframes[group]['model_brier_score_loss']
    return loss_dataframes

loss_dataframes = create_groups(df_03_23)

# Extract grouped dataframes.
for name, df_binned in loss_dataframes.items():
    globals()[name] = df_binned
```

In [9]:

```
# Instantiate subplots.
fig, axs = plt.subplots(2, 2, figsize=(12,8)) #, constrained_layout =
True)

# By prediction difference.
axs[0, 0].plot(
    loss_by_pred_diff['bin_center'],
    loss_by_pred_diff['log_loss_diff'],
    marker='o'
)
axs[0, 0].plot(
    loss_by_pred_diff['bin_center'],
    loss_by_pred_diff['brier_score_loss_diff'],
    marker='o',
    color = 'orange'
)
axs[0, 0].axhline(0, color='green', linewidth = 3)
axs[0, 0].tick_params(rotation=45)
axs[0, 0].set_xlabel('Prediction Difference (Model - Sportsbook)')
axs[0, 0].set_ylabel('Loss Difference (Sportsbook - Model)')

ax0 = axs[0,0].twinx()
ax0.bar(
    loss_by_pred_diff['bin_center'],
    loss_by_pred_diff['count'],
    width = .003,
    color = 'gray',
    alpha = .4,
    label = 'Play Count'
)
# ax0.set_ylabel('Play Count')
axs[0, 0].set_title('Model Advantage By Prediction Difference',
fontsize = 15)
axs[0, 0].grid(True)


# By time remaining.
axs[0, 1].plot(
    loss_by_time['bin_center'],
    loss_by_time['log_loss_diff'],
    marker='o'
)
axs[0, 1].plot(
    loss_by_time['bin_center'],
    loss_by_time['brier_score_loss_diff'],
    marker='o',
    color = 'orange'
)
axs[0, 1].axhline(0, color='green', linewidth = 3)
axs[0, 1].tick_params(rotation=45)
axs[0, 1].set_xlabel('Game Seconds Remaining')
# axs[0, 1].set_ylabel('Log Loss Difference (Sportsbook - Model)')
```

```
ax1 = axs[0, 1].twinx()
ax1.bar(
    loss_by_time['bin_center'],
    loss_by_time['count'],
    width = 100,
    color = 'gray',
    alpha = .4,
    label = 'Play Count'
)
ax1.set_ylabel('Play Count')
axs[0, 1].set_title('Model Advantage By Time Remaining', fontsize =
15)
axs[0, 1].grid(True)


# By score differential.
axs[1, 0].plot(
    loss_by_score_diff['bin_center'],
    loss_by_score_diff['log_loss_diff'],
    marker='o'
)
axs[1, 0].plot(
    loss_by_score_diff['bin_center'],
    loss_by_score_diff['brier_score_loss_diff'],
    marker='o',
    color = 'orange'
)
axs[1, 0].axhline(0, color='green', linewidth = 3)
axs[1, 0].tick_params(rotation=45)
axs[1, 0].set_xlabel('Score Differential (Home - Away)')
axs[1, 0].set_ylabel('Loss Difference (Sportsbook - Model)')

ax2 = axs[1, 0].twinx()
ax2.bar(
    loss_by_score_diff['bin_center'],
    loss_by_score_diff['count'],
    width = 1.5,
    color = 'gray',
    alpha = .4,
    label = 'Play Count'
)
# ax2.set_ylabel('Play Count')
axs[1, 0].set_title('Model Advantage By Score Differential', fontsize
= 15)
axs[1, 0].grid(True)


# By point spread.
axs[1, 1].plot(
    loss_by_spread['bin_center'],
    loss_by_spread['log_loss_diff'],
    marker='o'
```

```
)
axs[1, 1].plot(
    loss_by_spread['bin_center'],
    loss_by_spread['brier_score_loss_diff'],
    marker='o',
    color = 'orange'
)

axs[1, 1].axhline(0, color='green', linewidth = 3)
axs[1, 1].tick_params(rotation=45)
axs[1, 1].set_xlabel('Home Pregame Spread')
# axs[1, 1].set_ylabel('Log Loss Difference (Sportsbook - Model)')

ax3 = axs[1,1].twinx()
ax3.bar(
    loss_by_spread['bin_center'],
    loss_by_spread['count'],
    width = .8,
    color = 'gray',
    alpha = .4,
    label = 'Play Count'
)

ax3.set_ylabel('Play Count')
axs[1, 1].set_title('Model Advantage By Pregame Point Spread',
fontsize = 15)
axs[1, 1].grid(True)

axs[0, 0].set_ylim(-.015, .005)
axs[0, 1].set_ylim(-.003, .0005)
axs[1, 0].set_ylim(-.003, .001)
# axs[1, 1].set_ylim(-.001, .005)

fig.legend(['Log Loss Differential', 'Brier Score Loss
Differential'], loc = 'upper right')
fig.tight_layout()
fig.subplots_adjust(top = .85, hspace = .4)
plt.suptitle('Model Advantage by Loss Differentials', fontsize = 20)
plt.show()
```

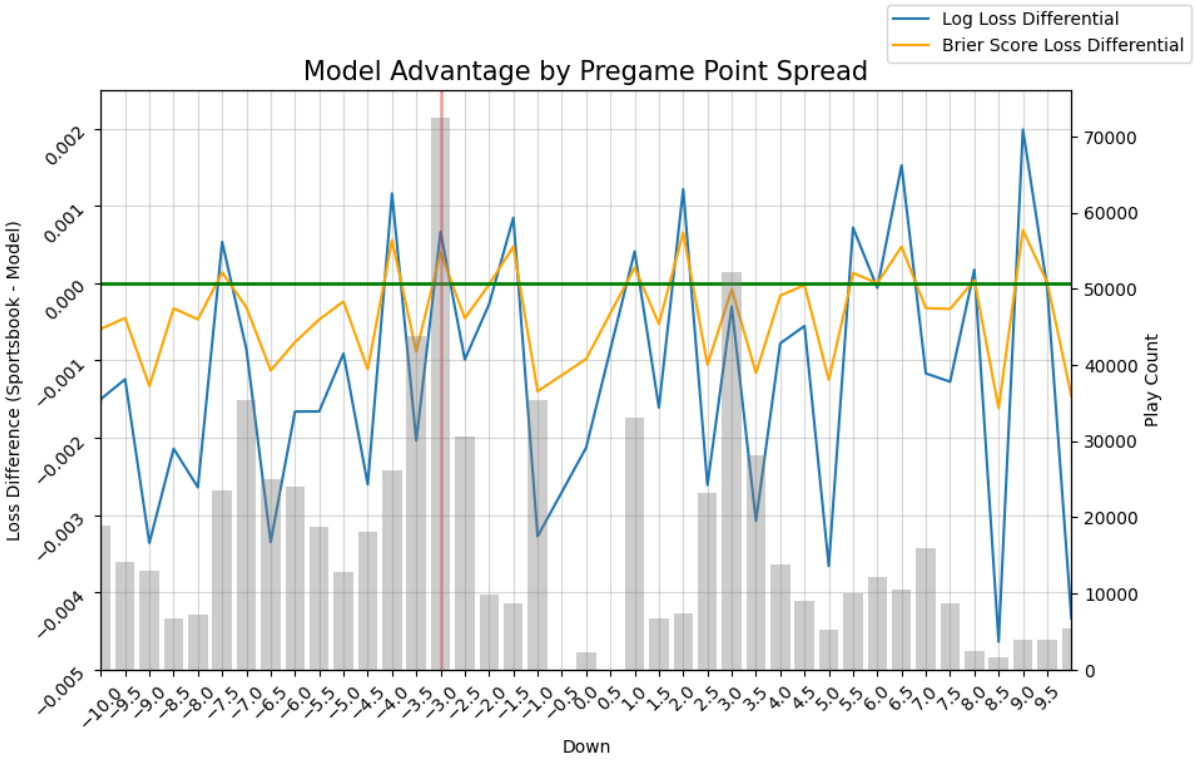


The model does not perform well on the full dataset compared to the sportsbook model. However there are a couple of areas in the point spread plot where both loss functions cross into the positive region.

The binning makes these areas unclear. We plot each point spread in greater detail to gain clarity.

In [5]:
```
# Group by point spread.
df_grouped = df_03_23.groupby('home_spread_line').apply(lambda g:
pd.Series({
    'model_spread_log_loss': log_loss(g['home_win'], g['y_pred'],
labels = [0, 1]),
    'sportsbook_spread_log_loss': log_loss(g['home_win'],
g['vegas_home_wp'], labels = [0, 1]),
    'model_spread_brier_score_loss': brier_score_loss(g['home_win'],
g['y_pred'], labels = [0, 1]),
    'sportsbook_spread_brier_score_loss':
brier_score_loss(g['home_win'], g['vegas_home_wp'], labels = [0, 1]),
    'count': len(g)
}), include_groups = False).reset_index()

# Calculate loss delta.
df_grouped['log_loss_diff'] =
df_grouped['sportsbook_spread_log_loss'] -
df_grouped['model_spread_log_loss']
df_grouped['brier_score_loss_diff'] =
df_grouped['sportsbook_spread_brier_score_loss'] -
df_grouped['model_spread_brier_score_loss']
```

```
# Instantiate plot.
fig, ax = plt.subplots(figsize = (10,6))
ax.plot(df_grouped['home_spread_line'],
        df_grouped['log_loss_diff'])
ax.plot(df_grouped['home_spread_line'],
        df_grouped['brier_score_loss_diff'],
        color = 'orange')

ax.axhline(0, color='green', linewidth = 2)

ax.grid(True, alpha = .5)
ax.set_xticks(np.arange(-10, 10, .5))
ax.tick_params(rotation = 45)
ax.set_xlim(-10, 10)

ax2 = ax.twinx()
ax2.bar(
    df_grouped['home_spread_line'],
    df_grouped['count'],
    width = .4,
    color = 'gray',
    alpha = .4,
    label = 'Play Count'
)

ax.set_ylim(-.005, .0025)
spreads = [-3]
for i in spreads:
    ax.axvline(x= i, color='red',  linewidth=2, alpha=0.3)

ax.set_xlabel('Down')
ax.set_ylabel('Loss Difference (Sportsbook - Model)')
ax2.set_ylabel('Play Count')
fig.legend(['Log Loss Differential', 'Brier Score Loss
Differential'])
# fig.tight_layout()
plt.title('Model Advantage by Pregame Point Spread', fontsize = 15)
plt.show()
```



3-point home favorites are not only the most common point spread, but our model also appears to generalize better than the sportsbook in this area.

We subset by point spreads of -3.

Log Loss and Brier Score Loss (2003-2023 Subset)

In [18]:
```
# Subset optimal spreads.
df_subset =
df_03_23[df_03_23['home_spread_line'].isin(spreads)].copy()

# View losses.
print(f'Subset Log Loss (Model): {log_loss(df_subset['home_win'],
df_subset['y_pred']):.6f}')
print(f'Subset Log Loss (Sportsbook):
{log_loss(df_subset['home_win'], df_subset['vegas_home_wp']):.6f}')
print()
print(f'Subset Brier Score (Model):
{brier_score_loss(df_subset['home_win'], df_subset['y_pred']):.6f}')
print(f'Subset Brier Score (Sportsbook):
{brier_score_loss(df_subset['home_win'],
df_subset['vegas_home_wp']):.6f}')
print()
print(f'Percentage of games:
{round(len(np.unique(df_subset['game_id']))/len(np.unique(df_03_23['g
ame_id']))*100, 3)}%')
```

```
Subset Log Loss (Model): 0.505990
Subset Log Loss (Sportsbook): 0.506655

Subset Brier Score (Model): 0.171637
Subset Brier Score (Sportsbook): 0.172034

Percentage of games: 9.364%
```

Our model is generalizing better than the sportsbook in this subset! We move forward with a similar subset on our unseen 2024 data.

Almost 10% of games are still included, which is over 1 per week.

Log Loss and Brier Score Loss (2024 Subset)

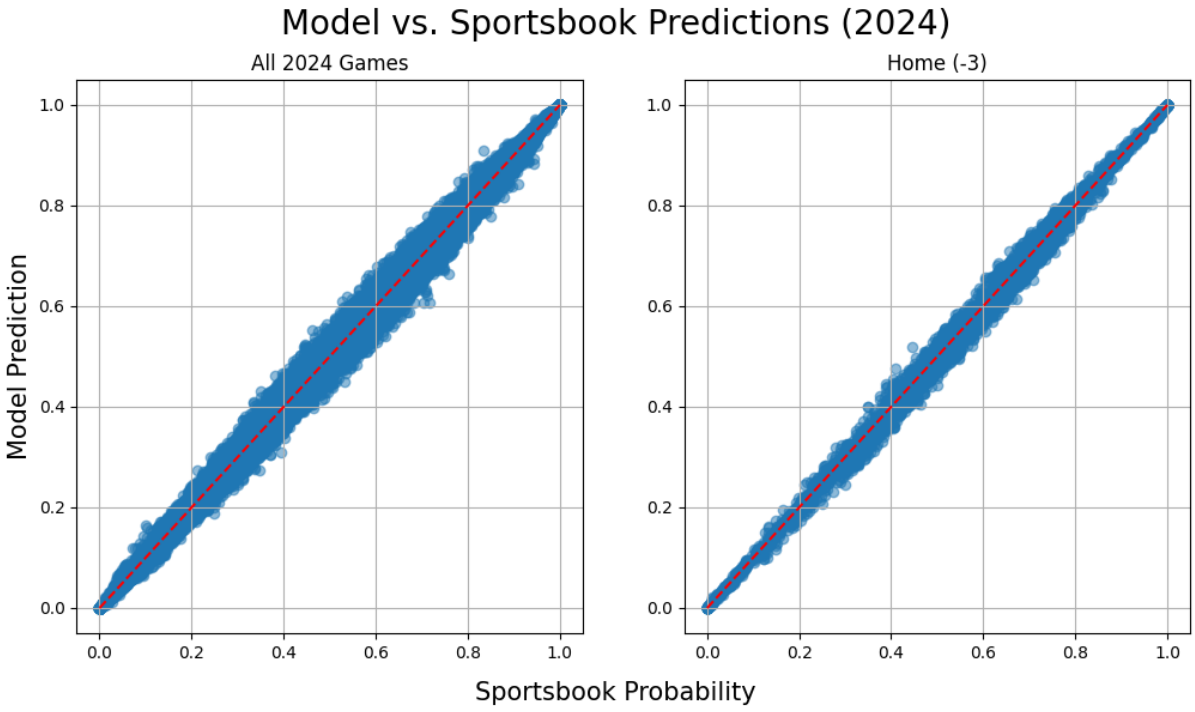
In [6]:
```
df_subset_24 = df_24[df_24['home_spread_line'].isin(spreads)].copy()
# df_subset_24 = df_subset_24[df_subset_24['game_seconds_remaining']
< 1800].copy()

# View losses.
print(f'2024 Subset Log Loss (Model):
{log_loss(df_subset_24['home_win'], df_subset_24['y_pred']):.6f}')
print(f'2024 Subset Log Loss (Sportsbook):
{log_loss(df_subset_24['home_win'],
df_subset_24['vegas_home_wp']):.6f}')
print()
print(f'2024 Subset Brier Score (Model):
{brier_score_loss(df_subset_24['home_win'],
df_subset_24['y_pred']):.6f}')
print(f'2024 Subset Brier Score (Sportsbook):
{brier_score_loss(df_subset_24['home_win'],
df_subset_24['vegas_home_wp']):.6f}')
print()
print(f'Subset Percentage of games:
{round(len(np.unique(df_subset_24['game_id']))/len(np.unique(df_24['g
ame_id']))*100, 3)}%')
```

```
2024 Subset Log Loss (Model): 0.496310
2024 Subset Log Loss (Sportsbook): 0.499148

2024 Subset Brier Score (Model): 0.171007
2024 Subset Brier Score (Sportsbook): 0.172185

Subset Percentage of games: 8.772%
```

Our model is still generalizing better than the sportsbook on unseen data, with just under 9% of 2024 games.

2024 Test Data Scatterplot

In [20]:
```
# Create scatterplot.
fig, ax = plt.subplots(1, 2, figsize = (12, 6))
# plt.figure(figsize=(8, 8))

ax[0].scatter(df_24['y_pred'], df_24['vegas_home_wp'], alpha=0.5)

# Set grid area and diagonal.
ax[0].plot([0, 1], [0, 1], 'r--')
ax[0].set_title('All 2024 Games')

ax[1].scatter(df_subset_24['y_pred'], df_subset_24['vegas_home_wp'],
alpha=0.5)

# Set grid area and diagonal.
ax[1].plot([0, 1], [0, 1], 'r--')
ax[1].set_title('Home (-3)')

# Set labels and title.
fig.supxlabel("Sportsbook Probability", fontsize = 15)
ax[0].set_ylabel("Model Prediction", fontsize = 15)
fig.suptitle("Model vs. Sportsbook Predictions (2024)", fontsize =
20)
ax[0].grid(True)
ax[1].grid(True)
plt.show()
```



Predictions still closely match the sportsbook.

We plot 2024 the same as above. We also visualize the advantage by down instead of point spread, as the latter is uniform across this subset.

2024 Test Data Loss Plots

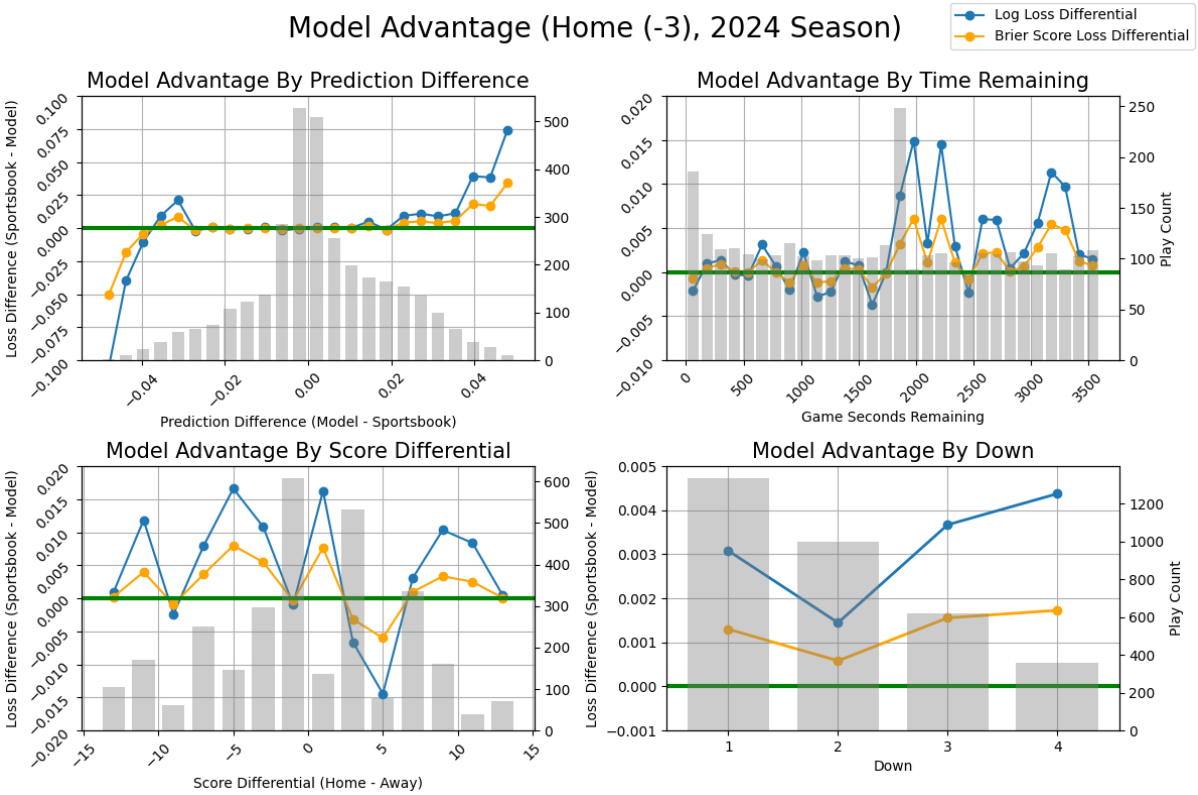
In [10]:
```
# Bin subset.
df_subset_24['pred_diff_bin'] = pd.cut(df_subset_24['pred_diff'],
bins = np.linspace(-.05, .05, 25))
df_subset_24['time_bin'] =
pd.cut(df_subset_24['game_seconds_remaining'], bins=30)
df_subset_24['score_diff_bin'] =
pd.cut(df_subset_24['home_score_differential'], bins=np.linspace(-14,
14, 15))
df_subset_24['spread_bin'] = pd.cut(df_subset_24['home_spread_line'],
bins=np.arange(-10, 10, 1))
# Create dictionary of binned dataframes.
subset_loss_dataframes = create_groups(df_subset_24)

# Extract grouped dataframes.
for name, dataframe in subset_loss_dataframes.items():
    globals()[name] = dataframe

# Create loss by down grouped dataframe.
loss_by_down = (
    df_subset_24.groupby('down', observed=True, group_keys=False)
    .apply(lambda g: pd.Series({
        'model_log_loss': log_loss(g['home_win'], g['y_pred'],
labels=[0, 1]),
        'sportsbook_log_loss': log_loss(g['home_win'],
g['vegas_home_wp'], labels=[0, 1]),
        'model_brier_score_loss': brier_score_loss(g['home_win'],
g['y_pred'], labels = [0, 1]),
        'sportsbook_brier_score_loss':
brier_score_loss(g['home_win'], g['vegas_home_wp'], labels = [0, 1]),
        'count': len(g)
    }), include_groups = False)
    .reset_index()
)


# Calculate loss differential.
loss_by_down['log_loss_diff'] = loss_by_down['sportsbook_log_loss'] -
loss_by_down['model_log_loss']
loss_by_down['brier_score_loss_diff'] =
loss_by_down['sportsbook_brier_score_loss'] -
loss_by_down['model_brier_score_loss']
```

In [11]:
```
# Configure subplots.
fig, axs = plt.subplots(2, 2, figsize=(12,8)) #, constrained_layout =
True)


# By prediction difference.
axs[0, 0].plot(
```

```
    loss_by_pred_diff['bin_center'],
    loss_by_pred_diff['log_loss_diff'],
    marker='o'
)
axs[0, 0].plot(
    loss_by_pred_diff['bin_center'],
    loss_by_pred_diff['brier_score_loss_diff'],
    marker='o',
    color = 'orange'
)
axs[0, 0].axhline(0, color='green', linewidth = 3)
axs[0, 0].tick_params(rotation=45)
axs[0, 0].set_xlabel('Prediction Difference (Model - Sportsbook)')
axs[0, 0].set_ylabel('Loss Difference (Sportsbook - Model)')
ax0 = axs[0,0].twinx()
ax0.bar(
    loss_by_pred_diff['bin_center'],
    loss_by_pred_diff['count'],
    width = .003,
    color = 'gray',
    alpha = .4,
    label = 'Play Count'
)
# ax0.set_ylabel('Play Count')
axs[0, 0].set_title('Model Advantage By Prediction Difference',
fontsize = 15)
axs[0, 0].grid(True)


# By time remaining.
axs[0, 1].plot(
    loss_by_time['bin_center'],
    loss_by_time['log_loss_diff'],
    marker='o'
)
axs[0, 1].plot(
    loss_by_time['bin_center'],
    loss_by_time['brier_score_loss_diff'],
    marker='o',
    color = 'orange'
)
axs[0, 1].axhline(0, color='green', linewidth = 3)
axs[0, 1].tick_params(rotation=45)
axs[0, 1].set_xlabel('Game Seconds Remaining')
# axs[0, 1].set_ylabel('Log Loss Difference (Sportsbook - Model)')

ax1 = axs[0, 1].twinx()
ax1.bar(
    loss_by_time['bin_center'],
    loss_by_time['count'],
    width = 100,
    color = 'gray',
```

```
    alpha = .4,
    label = 'Play Count'
)
ax1.set_ylabel('Play Count')
axs[0, 1].set_title('Model Advantage By Time Remaining', fontsize =
15)
axs[0, 1].grid(True)


# By score differential.
axs[1, 0].plot(
    loss_by_score_diff['bin_center'],
    loss_by_score_diff['log_loss_diff'],
    marker='o'
)
axs[1, 0].plot(
    loss_by_score_diff['bin_center'],
    loss_by_score_diff['brier_score_loss_diff'],
    marker='o',
    color = 'orange'
)
axs[1, 0].axhline(0, color='green', linewidth = 3)
axs[1, 0].tick_params(rotation=45)
axs[1, 0].set_xlabel('Score Differential (Home - Away)')
axs[1, 0].set_ylabel('Loss Difference (Sportsbook - Model)')

ax2 = axs[1, 0].twinx()
ax2.bar(
    loss_by_score_diff['bin_center'],
    loss_by_score_diff['count'],
    width = 1.5,
    color = 'gray',
    alpha = .4,
    label = 'Play Count'
)
# ax2.set_ylabel('Play Count')
axs[1, 0].set_title('Model Advantage By Score Differential', fontsize
= 15)
axs[1, 0].grid(True)


# By down.
axs[1, 1].plot(
    loss_by_down['down'],
    loss_by_down['log_loss_diff'],
    marker = 'o',
    linewidth = 2
)
axs[1, 1].plot(
    loss_by_down['down'],
    loss_by_down['brier_score_loss_diff'],
    color = 'orange',
```

```
    marker = 'o',
    linewidth = 2
)
axs[1, 1].axhline(0, color='green', linewidth = 3)
axs[1, 1].set_xlabel('Down')
axs[1, 1].set_ylabel('Loss Difference (Sportsbook - Model)')

# plt.title('Model Advantage Over Sportsbook by Pregame Spread')
ax3 = axs[1, 1].twinx()
ax3.bar(
    loss_by_down['down'],
    loss_by_down['count'],
    width = .75,
    color = 'gray',
    alpha = .4,
    label = 'Play Count'
)
ax3.set_xticks([1, 2, 3, 4])
ax3.set_ylabel('Play Count')
axs[1, 1].set_title('Model Advantage By Down', fontsize = 15)
axs[1, 1].grid(True)

axs[0, 0].set_ylim(-.1, .1)
axs[0, 1].set_ylim(-.01, .02)
axs[1, 0].set_ylim(-.02, .02)
axs[1, 1].set_ylim(-.001, .005)

fig.legend(['Log Loss Differential', 'Brier Score Loss
Differential'], loc = 'upper right')
fig.tight_layout()
fig.subplots_adjust(top = .88, hspace = .4)
plt.suptitle(f'Model Advantage (Home ({spreads[0]}), 2024 Season)',
fontsize = 20)
plt.show()
```



Much better performance on the 2024 3-point favorite subset, but it's unclear if this is a meaningful advantage.

It's worth noting that the model now performs better in the first half, as well as at higher prediction deltas and negative score differentials.

We simulate wagers to see if the model generalizes well enough to edge out the house spread.

Wager Simulation

In [12]:
```python
# Convert probability to implied probability with house edge.
def prob_to_market_prob(prob):
    hold = 0.0476190476190477
    # hold = 0.0476
    p1 = prob
    p2 = 1-p1
    overround = 1 + hold
    fair_total = p1 + p2
    vig_p1 = min((p1 / fair_total) * overround, 0.9999)
    vig_p2 = overround - vig_p1
    return vig_p1, vig_p2

# Convert probability to American odds.
def prob_to_odds(prob):
    if prob > 0.5:
```

```
        return round(-prob / (1 - prob) * 100)
    else:
        return round((1 - prob) / prob * 100)

# Confirm odds output.
home_p = 0.5

market_prob_home, market_prob_away = prob_to_market_prob(home_p)
market_odds_home = round(prob_to_odds(market_prob_home), -
(len(str(abs(prob_to_odds(market_prob_home))))-2))
market_odds_away = round(prob_to_odds(market_prob_away), -
(len(str(abs(prob_to_odds(market_prob_away))))-2))

print('Implied Odds:')
print(f'Home: {market_odds_home}, Away: {market_odds_away}')
```

```
Implied Odds:
Home: -110, Away: -110
```

Sportsbook odds are working as intended.

In [13]:
```
# Combine functions.
def prob_to_market_odds(prob):
    inflated_prob = prob_to_market_prob(prob)[0]
    market_odds = prob_to_odds(inflated_prob)
    rounded_odds = int(round(market_odds, -
(len(str(abs(market_odds)))-2)))

    # Cap underdog odds.
    if rounded_odds > 100:
        rounded_odds = min(rounded_odds, 5000)

    # Cap favorite odds.
    if rounded_odds < 100:
        rounded_odds = max(rounded_odds, -100000)
    if abs(rounded_odds) == 100:
        return f'(EVEN)'
    elif rounded_odds > 100:
        return f'(+{rounded_odds})'
    else:
        return f'({rounded_odds})'
```

We create a function that simulates wagers. We also set a threshold for prediction differences.

If `y_pred` is above .5 *and greater* than the sportsbook's probability + threshold, we have action on the home team.

If `y_pred` is below .5 *and less* than sportsbook probability - threshold, we have action on the away team.

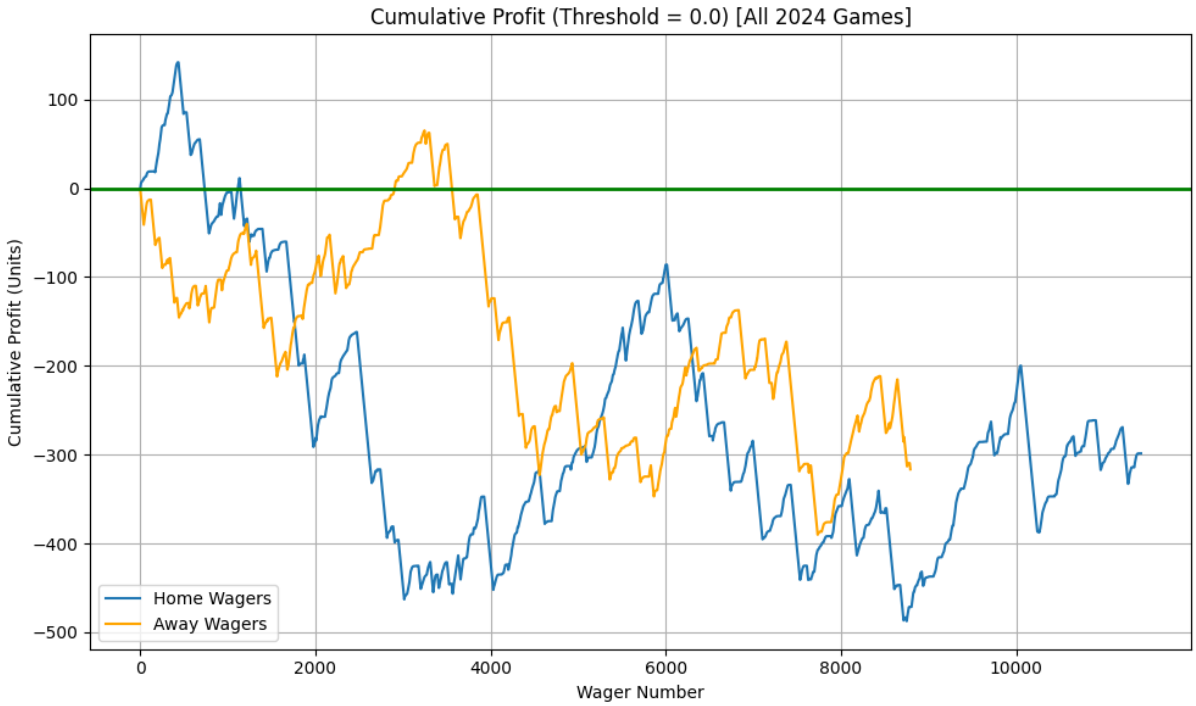
In [14]:
```
# Create wager simulation function.
def simulate_wagers(df = None, threshold=0.00, risk=0, label = None):
```

```
    df = df.copy()
    df['action_side'] = 0

    # Home action.
    df.loc[(df['y_pred'] > df['vegas_home_wp'] + threshold) &
(df['y_pred'] > 0.5), 'action_side'] = 1

    # Away action.
    df.loc[(df['y_pred'] < df['vegas_home_wp'] - threshold) &
(df['y_pred'] < 0.5), 'action_side'] = -1

    # Subset wager action.
    df = df[df['action_side'] != 0]

    # Evaluate wagers based on game result.
    df['settled_win'] = ((df['home_win'] == 1) & (df['action_side']
== 1)) | ((df['home_win'] == 0) & (df['action_side'] == -1))

    # Add market odds columns with probability to odds function.
    df['market_home_odds'] = df.apply(lambda
x:prob_to_market_odds(x['vegas_home_wp']), axis = 1)
    df['market_away_odds'] = df.apply(lambda
x:prob_to_market_odds(.9999999-x['vegas_home_wp']), axis = 1)

    # Calculate net profit per wager.
    def calc_profit(row):
        if row['action_side'] == 1:
            odds_val = row['market_home_odds']
        else:
            odds_val = row['market_away_odds']
        if odds_val == '(EVEN)':
            odds_val = '(+100)'
        odds_val = int(odds_val[1:-1])
        payout = (risk * abs(odds_val)) / 100 if odds_val > 0 else
(risk * 100) / abs(odds_val)
        return payout if row['settled_win'] else -risk

    df['profit'] = df.apply(calc_profit, axis=1)

    # Split dataframe into home and away action.
    df_home = df[df['action_side'] == 1].copy()
    df_away = df[df['action_side'] == -1].copy()

    # Calculate cumulative profit.
    df_home['cumulative_profit'] = df_home['profit'].cumsum()
    df_home['wager_number'] = range(1, len(df_home) + 1)

    df_away['cumulative_profit'] = df_away['profit'].cumsum()
    df_away['wager_number'] = range(1, len(df_away) + 1)

    # Plot results.
    plt.figure(figsize=(10, 6))
```

```
    plt.plot(df_home['wager_number'], df_home['cumulative_profit'],
label='Home Wagers')
    plt.plot(df_away['wager_number'], df_away['cumulative_profit'],
label='Away Wagers', color='orange')
    plt.axhline(0, color='green', linewidth = 2)
    plt.title(f'Cumulative Profit (Threshold = {threshold}) {label}')
    plt.xlabel('Wager Number')
    plt.ylabel('Cumulative Profit (Units)')
    plt.legend(loc = 'lower left')
    plt.grid(True)
    plt.tight_layout()
    plt.show()


    print(f"Total Wagers Simulated: {len(df)}")
    print(f"Home Wagers: {len(df_home)} | Win Rate:
{df_home['settled_win'].mean():.2%} | Profit:
{df_home['profit'].sum():+.2f} units")
    print(f"Away Wagers: {len(df_away)} | Win Rate:
{df_away['settled_win'].mean():.2%} | Profit:
{df_away['profit'].sum():+.2f} units")
    print(f"Total Profit: {df['profit'].sum():+.2f} units | Avg
Profit per Wager: {df['profit'].mean():+.2f} units")
```

We first simulate the full 2024 season.

In [15]:
```
# Simulate wagers on entire season.
simulate_wagers(
    df=df_24,
    threshold=0.0,
    # threshold=0.045,
    risk=1,
    label = '[All 2024 Games]'
)
```

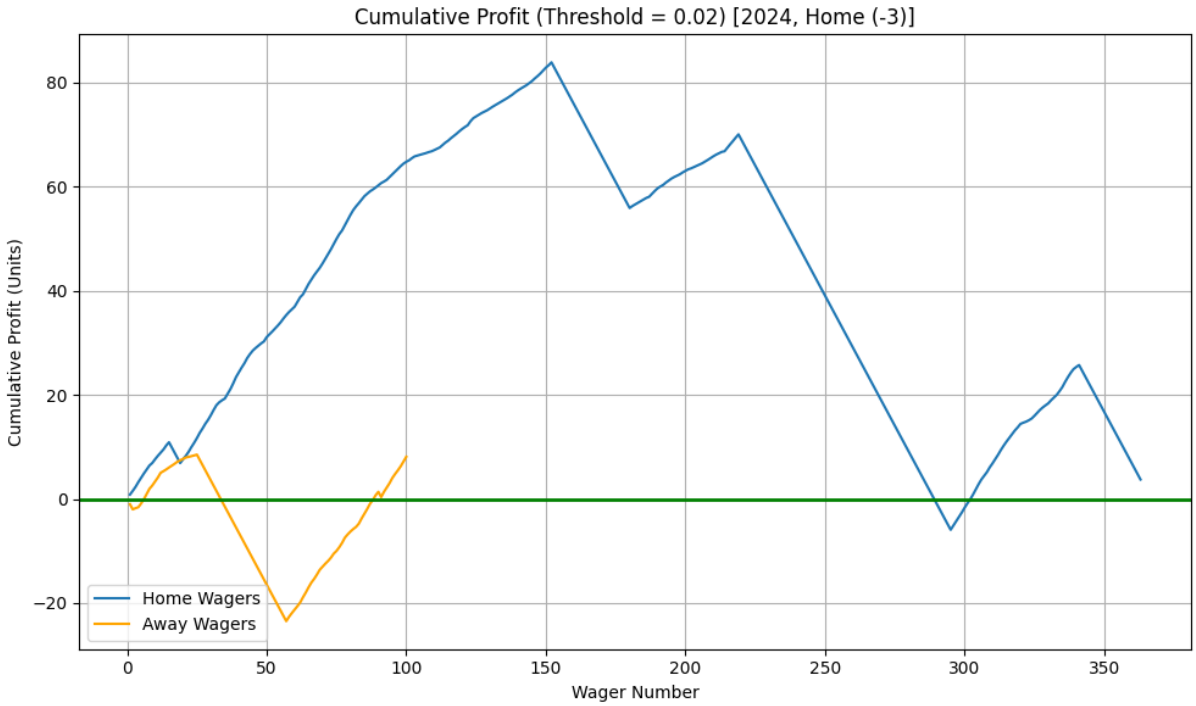



```
Total Wagers Simulated: 20219
Home Wagers: 11426 | Win Rate: 77.49% | Profit: -298.60 units
Away Wagers: 8793 | Win Rate: 76.28% | Profit: -316.52 units
Total Profit: -615.11 units | Avg Profit per Wager: -0.03 units
```

Net result is negative on the full season.

Now we simulate wagers on the 3-point favorite subset.

In [21]:
```
spreads = [-3]
df_subset_24 = df_24[df_24['home_spread_line'].isin(spreads)].copy()
# df_subset_24 = df_subset_24[df_subset_24['game_seconds_remaining']
< 1800].copy()

# Simulate wagers on subset.
simulate_wagers(
    df=df_subset_24,
    threshold=0.02,
    risk=1,
    label='[2024, Home (-3)]'
)
```




```
Total Wagers Simulated: 463
Home Wagers: 363 | Win Rate: 64.19% | Profit: +3.74 units
Away Wagers: 100 | Win Rate: 65.00% | Profit: +8.10 units
Total Profit: +11.84 units | Avg Profit per Wager: +0.03 units
```

Much better performance on the Home -3 subset! A small threshold yields a positive result, as does subsetting to the second half without a threshold.

Results Summary

- Model achieved Log Loss and Brier Scores close to sportsbook benchmarks.
- Scatterplots confirmed alignment with sportsbook probabilities, with slight underconfidence in balanced games.
- Temporal evaluation (2024 holdout set) showed our model adapts reasonably well to changes in league dynamics.
- Calibration analysis demonstrated reliable probability estimates with minor bias.
- Although profit margins are small, model yielded positive results with market-adjusted odds in certain scenarios such as 3-point home favorites with a small threshold.

Overall, the model reproduces sportsbook-level accuracy while remaining transparent and explainable. This supports its utility as a real-time win probability tool independent of proprietary sportsbook markets.

utils.py

In [ ]:
```
import numpy as np
from scipy.special import logit, expit
```

In [ ]:
```
def logit_func(y):
    tol = 1e-5
    return logit(np.clip(y, tol, 1-tol))
def expit_func(y):
    return expit(y)
```

```python
# Streamlit Home Page
import streamlit as st


st.title("Welcome to my Eastern University Capstone user interface!")
st.markdown("""
    Hello! My name is Michael Hart.

    I am a master's candidate in Data Science at Eastern University. My professional interests include machine learning model deployment and automation pipelines.

    Outside of data science, I enjoy football, motorcycles, aviation, and locally brewed beverages.

    """)

st.image("images/hart.png", caption="2015 NFC Championship (Jan. 24, 2016)")
```

```python
# Streamlit Resume Page
import streamlit as st

st.title("Resume")
st.header("Professional Summary")
st.markdown("""
            Results-oriented data professional with an acute attention to detail and a
passion for continuous learning.
            """)

st.divider()

st.header("Education")
col1, col2 = st.columns([3, 1])
with col1:
    st.markdown("""
                **Eastern University** - *Master of Science in Data Science*
                """)
with col2:
    st.markdown("""
                (In Progress)
                """)

col1, col2 = st.columns([3, 1])
with col1:
    st.markdown("""
                **Winthrop University** - *Bachelor of Science in Sociology*
                <span style="margin-left:20px;"> Minor in statistics </span>
                """, unsafe_allow_html=True)
with col2:
    st.markdown("""
                (2009)
                """)

st.divider()

st.header("Work Experience")

st.subheader("""
**Data Analyst** - *Journeyman Outfitter*,  2024-present
""")
st.markdown("""
<ul style="margin-left:20px;">
    <li>Collect, clean, transform, and prepare data for analysis </li>
    <li>Develop statistical models to gain insights from data</li>
</ul>
            """, unsafe_allow_html=True)

st.subheader("""
**Reporting Lead** - *AT&T*,  2014-2024
""")
st.markdown("""
<ul style="margin-left:20px;">
    <li>Developed and implemented dynamic calendar showing real-time
    forecast of scheduled installations at the team level</li>
    <li>Analyze fallout and performance trends to recommend new processes
```

```
    and strategies</li>
    <li>Partnered with cross-functional teams to ensure accurate and timely
    sales reporting </li>
    <li>Generate funnels, drill-down reports, and visualizations to communicate
    key insights and recommendations </li>
</ul>
""", unsafe_allow_html = True)

st.divider()

st.header("""
Skills and Attributes
             """)
st.markdown("""
<ul style="margin-left:20px;">
    <li>Data Science </li>
    <li>Data Analysis </li>
    <li>Statistics </li>
    <li>Python </li>
    <li>SQL </li>
    <li>R </li>
    <li>Time Magazine's 2006 Person of the Year</li>
</ul>
             """, unsafe_allow_html=True)

st.divider()

st.header("""
Certifications
             """)

col1, col2, col3, col4 = st.columns([1, 1, 1, 1])
with col1:
    st.image("images/Google Data Analytics.png", caption="Google Data Analytics", width =
120)

with col2:
    st.image("images/Google Advanced Data Analytics.png", caption="Google Advanced Data
Analytics", width = 120)
with col3:
    st.image("images/IBM Data Science.png", caption="IBM Data Science", width = 120)
with col4:
    st.image("images/SAS Programmer.png", caption="SAS Programmer", width = 120)
```

```python
# Streamlit Projects Page
import streamlit as st

st.title("Hart's Portfolio")

st.markdown("""
### [South Carolina DMV Wait Time Scraper]
(https://github.com/OMGHart/dmv_scraper/tree/main)
            """)
st.write("Automated Raspberry Pi script logging DMV wait times")

st.markdown("""
### [Spam Call Number Range Blocker](https://github.com/OMGHart/range-contact-generator)
            """)
st.write("Generates a blockable iPhone contact file with a complete range of numbers")

st.markdown("""
### [Housing Price Predictor](https://github.com/OMGHart/housing-price-predictor)
            """)
st.write("Kaggle competition predicting real estate prices")

st.markdown("""
### [Titanic Survival Predictor](https://github.com/OMGHart/titanic-survival)
            """)
st.write("Kaggle competition predicting passenger survival on the sinking of Titanic")

st.markdown("""
### [SpaceX Rocket Tracker](https://github.com/OMGHart/SpaceX-Dash/tree/main)
""")
st.write("Plotly Express dashboard tracking SpaceX launches")
```

```python
# Streamlit Model UI Page
import streamlit as st
import joblib
import numpy as np
import pandas as pd
from xgboost import XGBClassifier, XGBRegressor
from scipy.special import logit, expit
from utils import logit_func, expit_func

import streamlit as st

st.set_page_config(
    page_title="NFL Win Probability",
    layout="wide",
    initial_sidebar_state="collapsed",
)

st.title("Hart's NFL Win Probability Predictor")

model = joblib.load('ui_model.pkl')

# Introduction and instructions.
st.markdown("""
    This tool uses a machine learning model trained on over 20 years of NFL data to
estimate win probabilities in real time. Enter the current game state- possession, score,
time, time outs remaining, field position, down and distance, and pregame spread to see
the model's prediction alongside market-implied odds.
""")

# Features.
feature_columns = ['yardline_100_home',
                   'down',
                   'ydstogo',
                   'home_pos',
                   'home_score_differential',
                   'home_spread_line',
                   'time_weight',
                   'home_timeouts_remaining',
                   'away_timeouts_remaining',
                  ]

# Set defaults.
DEFAULTS = {
    "qtr": 1,
    "clock": 15,
    "yardline": 50,
    "home_pos": 1,
    "score_differential": 0,
    "home_spread": 0,
    "ydstogo": 10,
    "down": 1,
    "home_timeouts": 3,
    "away_timeouts": 3
}

for key, value in DEFAULTS.items():
```

```
    if key not in st.session_state:
        st.session_state[key] = value

# Reset button.
if st.session_state.get("reset_triggered", False):
    st.session_state['qtr'] = 1
    st.session_state['score_differential'] = 0
    st.session_state['clock'] = 15
    st.session_state['yardline'] = 50
    st.session_state['down'] = 1
    st.session_state['ydstogo'] = 10
    st.session_state['home_spread'] = 0
    st.session_state['reset_triggered'] = False
    st.session_state['home_timeouts'] = 3
    st.session_state['away_timeouts'] = 3
    st.rerun()

if st.button("🔄 Reset to Default"):
    st.session_state['reset_triggered'] = True
    st.rerun()

st.subheader("Game State")

# Home possession.
home_pos = st.radio(
    "Possession",
    options=[1, 0],
    format_func=lambda x: "Home" if x == 1 else "Away",
    horizontal = True,
    key="home_pos"
)

# Score differential.
score_differential = st.slider("Score Differential", 30,
    -30,
    key="score_differential")

# Reverse score differential for home/away position consistency.
score_differential = -score_differential

# Home/away labels.
label_html_left = (
    '<span style="display:inline-block; border:1px solid #48FF6A; border-radius:12px; '
    'padding:4px 12px; font-size:1em; color:#48FF6A;">'
    'Home Team Winning</span>'
)
label_html_right = (
    '<span style="display:inline-block; border:1px solid #48FF6A; border-radius:12px;'
    'padding:4px 12px; font-size:1em; color:#48FF6A;  ">'
    'Away Team Winning</span>'
)

st.markdown(
    f"""
    <table style="width:100%; border-collapse:inherit; border:none; ">
      <tr>
        <td style="text-align:left; border:none;">{label_html_left}</td>
```

```
        <td style="text-align:right; border:none;">{label_html_right}</td>
      </tr>
    </table>
    """,
    unsafe_allow_html=True,
)


st.divider()

# Time and timeouts.
st.subheader("Clock")

qtr = st.radio(
    "Quarter",
    options=[1, 2, 3, 4],
    horizontal = True,
    key="qtr")
clock = st.slider("Minutes Remaining", 0, 15, key="clock")

col1, col2, col3 = st.columns([1, 1, 1])
with col1:
    home_timeouts = st.radio(
        "Home Timeouts",
        options=[0, 1, 2, 3],
        horizontal=True,
        key="home_timeouts"
    )

with col3:
    away_timeouts = st.radio(
        "Away Timeouts",
        options=[0, 1, 2, 3],
        horizontal=True,
        key="away_timeouts"
    )

st.divider()

# Field position.
st.subheader("Field Position")

yardline = st.slider("Possession Team Distance From Score (Yards)", 0, 100,
key="yardline")

# Goal line marker.
st.markdown(
        """
        <div style="text-align:left; font-size:1em; color:48FF6A">
        <span style="display:inline-block; border:1px solid #48FF6A; border-radius:12px;
         padding:4px 12px; color:#48FF6A;">⬆ Goal Line
        </span>
        </div>
        """, unsafe_allow_html=True
    )

st.divider()
```

```
# Down and distance.
st.subheader("Down & Distance")

down = st.radio(
    "Down",
    options=[1, 2, 3, 4],
    horizontal = True,
    key="down"
)
ydstogo = st.slider("First Down Distance (Yards)", 0, 30, key="ydstogo")

st.divider()

# Pregame point spread.
st.subheader("Home Team Pregame Spread")
home_spread = st.slider(" ",min_value = -21.0,
    max_value = 21.0,
    step = .5,
format =  '%.1f', key="home_spread")

# Home/away labels.
label_html_left = (
    '<span style="display:inline-block; border:1px solid #48FF6A; border-radius:12px; '
    'padding:4px 12px; font-size:1em; color:#48FF6A;">'
    'Home Team Favored</span>'
)
label_html_right = (
    '<span style="display:inline-block; border:1px solid #48FF6A; border-radius:12px; '
    'padding:4px 12px; font-size:1em; color:#48FF6A;">'
    'Away Team Favored</span>'
)

st.markdown(
    f"""
    <table style="width:100%; border-collapse:inherit; border:none;  ">
      <tr>
        <td style="text-align:left; border:none;">{label_html_left}</td>
        <td style="text-align:right; border:none;">{label_html_right}</td>
      </tr>
    </table>
    """,
    unsafe_allow_html=True,
)

# Create time_weight feature.
game_seconds_remaining = ((4-qtr)*15+clock)*60
time_weight = 1-(game_seconds_remaining/3600)

# User inputs.
user_inputs = {
    'yardline_100_home': yardline,
    'down': down,
    'ydstogo': ydstogo,
    'home_pos': home_pos,
    'home_score_differential': score_differential,
    'home_spread_line': home_spread,
```

```
    'time_weight':time_weight,
    'home_timeouts_remaining':home_timeouts,
    'away_timeouts_remaining':away_timeouts,
 }

# Confirmation.
print("Model loaded:", type(model))

# Create dataframe of inputs.
X_input = pd.DataFrame([[user_inputs.get(col, 0) for col in user_inputs.keys()]],
                       columns=user_inputs.keys())

# Reverse field position if away possession.
X_input['yardline_100_home'] = X_input.apply(lambda X:(100-X['yardline_100_home']) if
X['home_pos'] == 0 else X['yardline_100_home'], axis = 1)

# Convert probability to market probability.
def prob_to_market_prob(prob):
    p1 = prob
    p2 = 1-p1
    hold = 0.0476
    overround = 1 + hold
    fair_total = p1 + p2
    vig_p1 = min((p1 / fair_total) * overround, 0.99999)
    vig_p2 = overround - vig_p1
    return vig_p1, vig_p2

# Convert probability to American odds.
def prob_to_odds(prob):
    if prob > 0.5:
        return round(-prob / (1 - prob) * 100)
    else:
        return round((1 - prob) / prob * 100)

# Combine functions.
def prob_to_market_odds(prob):
    inflated_prob = prob_to_market_prob(prob)[0]
    market_odds = prob_to_odds(inflated_prob)
    rounded_odds = int(round(market_odds, -(len(str(abs(market_odds)))-2)))
    if rounded_odds > 100:
        rounded_odds = min(rounded_odds, 5000)
    if rounded_odds < 100:
        rounded_odds = max(rounded_odds, -100000)
    if abs(rounded_odds) == 100:
        return f'(EVEN)'
    elif rounded_odds > 100:
        return f'(+{rounded_odds})'
    else:
        return f'({rounded_odds})'

# Instantiate variables.
home_win_prob = model.predict(X_input)[0]
home_odds = prob_to_market_odds(home_win_prob)
away_win_prob = 1-home_win_prob
away_odds = prob_to_market_odds(away_win_prob)
```

```
# CSS styling.
st.markdown(
    f"""
    <style>

    /* Add padding at bottom */
    .main, .block-container {{
        padding-bottom: 150px;
    }}

    /* Set default colors */
    body, .main, .block-container, .sidebar, .sidebar-content, .stButton > button {{
        background-color: #141e28;
        color: white;
    }}

    /* Set button attributes */
    .stButton > button {{
        border: 2px solid #444;
        border-radius: 1tpx solid #444;
        font-color:white;
    }}

    /* Set radio button color */
    .stSlider label, .stRadio label, .stRadio div {{
        color: white;
    }}

    /* Set slider width */
    .st-c7 {{
        height: .5rem;
    }}

    /* Enlarge slider buttons */
    .st-emotion-cache-1dj3ksd {{
       height: 1.5rem;
       width: 1.5rem;
    }}


    /* Floating panel attributes*/
    .fixed-2x2-panel {{
        position: fixed;
        left: 0;
        bottom: 0;
        width: 100vw;
        height: 150px;
        background: rgba(20,30,40,0.97);
        z-index: 9999;
        padding: 2px 0 24px 0;
        display: flex;
        justify-content: center;
    }}

    /* Grid attributes */
    .panel-grid {{
        display: grid;
```

```
        grid-template-columns: 1fr 1fr;
        grid-template-rows: 1fr 1fr;
        gap: 5px;
        width: 100%;
        height: 100px;
        max-width: 500px;
    }}

    /* Cell attributes */
    .panel-cell {{
        border-radius: 24px;
        font-size: 1em;
        font-weight: 500;
        color: #fff;
        padding: 10px 0; # 2px 0;
        text-align: center;
        box-shadow: 0 4px 28px #0006;
        min-width: 0;
        word-break: break-word;
        height: 45px;
        display: flex;
        align-items: center;
        justify-content: center;
    }}
    .st-emotion-cache-ujm5ma {{
    color: #48FF6A;
    }}


    /* Set cell background colors */
    .cell-home {{ background: #0525c5; }}
    .cell-away {{ background: #a61616; }}


    /* Add text */
    </style>
    <div class="fixed-2x2-panel">
      <div class="panel-grid">
        <div class="panel-cell cell-home">
            Home Win Probability: {home_win_prob*100:.2f}%
        </div>
        <div class="panel-cell cell-away">
            Away Win Probability:  {away_win_prob*100:.2f}%
        </div>
        <div class="panel-cell cell-home">
            Home Odds: {home_odds}
        </div>
        <div class="panel-cell cell-away">
            Away Odds: {away_odds}
        </div>
      </div>
    </div>
    """,
    unsafe_allow_html=True
)
```

```python
import streamlit as st
import numpy as np
import pandas as pd


st.title("Odds Calculator")
st.markdown("""
    Use this tool to convert between probability and sportsbook odds, with or without a
the sportsbook's built-in margin (vig).
            """)

prob = st.number_input("Probability",
                       value = .5,
                       step = .1e-4,
                       min_value = 1e-5,
                       max_value = 1-1e-5,
                       format = "%.5f",
                       width = 200)
house = st.number_input("House Advantage (Default: .04545)",
                       value = .04545,
                       step = .1e-4,
                       min_value = 0.0,
                       max_value = .99,
                       format = "%.5f",
                       width = 200)

hold = ((2*house)/(1-house))/2

def prob_to_odds(prob):
    if prob > 0.5:
        return round(-prob / (1 - prob) * 100)
    else:
        return round((1 - prob) / prob * 100)

def prob_to_market_prob(prob, hold):
    p1 = prob
    p2 = 1-p1
    # hold = 0.0476
    # hold = float(hold)
    overround = 1 + hold
    fair_total = p1 + p2
    vig_p1 = min((p1 / fair_total) * overround, 0.99999)
    vig_p2 = overround - vig_p1
    return vig_p1#vig_p2

# Combine functions.
def prob_to_market_odds(prob, hold):
    # prob = vig_p1
    inflated_prob = prob_to_market_prob(prob, hold)
    market_odds = prob_to_odds(inflated_prob)
    rounded_odds = int(round(market_odds, -(len(str(abs(market_odds)))-2)))
    # return rounded_odds
    if rounded_odds > 100:
        rounded_odds = min(rounded_odds, 5000)
    if rounded_odds < 100:
        rounded_odds = max(rounded_odds, -100000)
```

```
    if abs(rounded_odds) == 100:
        return f'(EVEN)'
    elif rounded_odds > 100:
        return f'(+{rounded_odds})'
    else:
        return f'({rounded_odds})'

if st.button("Calculate"):
    # st.write(type(prob)
    result = prob_to_market_odds(prob, hold)
    st.info(f'Odds: {result}')
```