Final-Models

October 10, 2025

[2]:
```
import os
import sqlite3
import pandas as pd
from IPython.display import display
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import xgboost as xgb
import graphviz
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

[ ]:
```
# Aggregate each game throughout seasons stats
# Train on previous 3 seasons (2020, 2021, 2022)
# Predict last season (2023)
# Predict week 1 of 2024 like its week 19 of 2023
```

Features

- current w/l record
- current week / number of games played
- last year w/l record
- home game or not
- divison game or not
- current qb ranking
- point differential in current season
  - use as part of elo score?
- win streak?

Data Preparation

[4]:
```
!rm -rf data-bak
!mv data data-bak
!mkdir data
!cp ../Scrapers/nfl.db data
```

[5]:
```python
conn = sqlite3.connect('data/nfl.db')

tables_query = "SELECT name FROM sqlite_master WHERE type='table';"
tables = conn.execute(tables_query).fetchall()
for table in tables:
    table_name = table[0]
    df = pd.read_sql_query(f"SELECT * FROM {table_name}", conn)
    csv_file_name = f"data/{table_name}.csv"
    df.to_csv(csv_file_name, index=False)
    print(f"Downloaded {table_name} to {csv_file_name}")

conn.close()
```

```
Downloaded Teams to data/Teams.csv
Downloaded Games to data/Games.csv
Downloaded PlayerStats to data/PlayerStats.csv
Downloaded Rosters to data/Rosters.csv
```

[6]:
```python
# Remove the rows from the 2024 season in Games.csv to UpcomingGames.csv

df = pd.read_csv('data/Games.csv')
upcoming_df = df[df['season'] == 2024]
upcoming_df.to_csv('data/UpcomingGames.csv', index=False)
df_cleaned = df[df['season'] != 2024]
df_cleaned.to_csv('data/Games.csv', index=False)
print("CSV files exported, cleaned Games.csv saved, and UpcomingGames.csv␣
↪created.")
```

```
CSV files exported, cleaned Games.csv saved, and UpcomingGames.csv created.
```

[7]:
```python
# Only keep 2019, 2020, 2021, 2022, 2023 seasons
# Remove postseason games

df = pd.read_csv('data/Games.csv')
df = df[df['season'] >= 2019]
df = df[df['game_type'] == 'REG']
df.to_csv('data/Games.csv', index=False)
print("Old seasons removed")
```

```
Old seasons removed
```

[8]:
```python
# Aggregate current wins/losses/ties per week

df = pd.read_csv('data/Games.csv')

# Initialize columns for "current wins," "current losses," and "current ties"
df['away_current_wins'] = 0
df['away_current_losses'] = 0
df['away_current_ties'] = 0
```

```python
df['home_current_wins'] = 0
df['home_current_losses'] = 0
df['home_current_ties'] = 0

# Get a list of all teams
teams = pd.concat([df['away_team'], df['home_team']]).unique()

# Iterate over each season and team
for season in df['season'].unique():
    for team in teams:
        # Filter the games for the current team and season, considering both home and away games
        team_games = df[((df['away_team'] == team) | (df['home_team'] == team)) & (df['season'] == season)]
        team_games = team_games.sort_values(by='date')  # Ensure the games are in chronological order

        # Initialize win/loss/tie counters
        wins, losses, ties = 0, 0, 0

        # Update the cumulative stats game by game
        for idx, game in team_games.iterrows():
            if game['away_team'] == team:  # If the team is playing away
                df.at[idx, 'away_current_wins'] = wins
                df.at[idx, 'away_current_losses'] = losses
                df.at[idx, 'away_current_ties'] = ties
                if game['away_score'] > game['home_score']:
                    wins += 1
                elif game['away_score'] < game['home_score']:
                    losses += 1
                else:
                    ties += 1
            elif game['home_team'] == team:  # If the team is playing at home
                df.at[idx, 'home_current_wins'] = wins
                df.at[idx, 'home_current_losses'] = losses
                df.at[idx, 'home_current_ties'] = ties
                if game['home_score'] > game['away_score']:
                    wins += 1
                elif game['home_score'] < game['away_score']:
                    losses += 1
                else:
                    ties += 1

# Save the updated DataFrame to a new CSV file
df.to_csv('data/Games.csv', index=False)

# Display the Dallas Cowboys' 2023 season rows with updated stats
```

```
cowboys_2023_df = df[(df['season'] == 2023) & ((df['away_team'] == 'DAL') |␣
↪(df['home_team'] == 'DAL'))]
display(cowboys_2023_df[['season', 'week', 'away_team', 'home_team',␣
↪'away_current_wins', 'away_current_losses', 'away_current_ties',␣
↪'home_current_wins', 'home_current_losses', 'home_current_ties']])
```

```
      season  week away_team home_team  away_current_wins  \
1069    2023     1       DAL       NYG                  0
1082    2023     2       NYJ       DAL                  1
1098    2023     3       DAL       ARI                  2
1115    2023     4        NE       DAL                  1
1131    2023     5       DAL        SF                  3
1147    2023     6       DAL       LAC                  3
1163    2023     8       LAR       DAL                  3
1188    2023     9       DAL       PHI                  5
1201    2023    10       NYG       DAL                  2
1206    2023    11       DAL       CAR                  6
1220    2023    12       WAS       DAL                  4
1235    2023    13       SEA       DAL                  6
1260    2023    14       PHI       DAL                 10
1276    2023    15       DAL       BUF                 10
1290    2023    16       DAL       MIA                 10
1296    2023    17       DET       DAL                 11
1325    2023    18       DAL       WAS                 11

      away_current_losses  away_current_ties  home_current_wins  \
1069                    0                  0                  0
1082                    0                  0                  1
1098                    0                  0                  0
1115                    2                  0                  2
1131                    1                  0                  4
1147                    2                  0                  2
1163                    4                  0                  4
1188                    2                  0                  7
1201                    7                  0                  5
1206                    3                  0                  1
1220                    7                  0                  7
1235                    5                  0                  8
1260                    2                  0                  9
1276                    3                  0                  7
1290                    4                  0                 10
1296                    4                  0                 10
1325                    5                  0                  4

      home_current_losses  home_current_ties
1069                    0                  0
1082                    0                  0
1098                    2                  0
```

1115	1	0
1131	0	0
1147	2	0
1163	2	0
1188	1	0
1201	3	0
1206	8	0
1220	3	0
1235	3	0
1260	3	0
1276	6	0
1290	4	0
1296	5	0
1325	12	0

[9]:

```
# Aggregate last season overall wins/losses/ties

df = pd.read_csv('data/Games.csv')

# Initialize columns for last year's wins, losses, and ties
df['away_team_last_year_wins'] = 0
df['away_team_last_year_losses'] = 0
df['away_team_last_year_ties'] = 0
df['home_team_last_year_wins'] = 0
df['home_team_last_year_losses'] = 0
df['home_team_last_year_ties'] = 0

# Calculate the overall wins, losses, and ties for each team by season
team_stats_last_year = {}

for season in df['season'].unique():
    # Filter the games for the previous season
    last_season = season - 1
    season_games = df[df['season'] == last_season]

    for team in pd.concat([season_games['away_team'],␣
↪season_games['home_team']]).unique():
        # Filter the games for the current team
        team_games = season_games[(season_games['away_team'] == team) |␣
↪(season_games['home_team'] == team)]

        # Calculate wins, losses, and ties
        wins = sum((team_games['away_team'] == team) &␣
↪(team_games['away_score'] > team_games['home_score'])) + \
               sum((team_games['home_team'] == team) &␣
↪(team_games['home_score'] > team_games['away_score']))
```

```
        losses = sum((team_games['away_team'] == team) &
↪(team_games['away_score'] < team_games['home_score'])) + \
                sum((team_games['home_team'] == team) &
↪(team_games['home_score'] < team_games['away_score']))

        ties = sum(team_games['away_score'] == team_games['home_score'])

        # Store the stats for this team and season
        team_stats_last_year[(team, season)] = {'wins': wins, 'losses': losses,
↪'ties': ties}

# Assign the last year's stats to the corresponding games
for idx, row in df.iterrows():
    current_season = row['season']
    away_team = row['away_team']
    home_team = row['home_team']

    # Set the last year's record for both away and home teams if it exists
    if (away_team, current_season) in team_stats_last_year:
        df.at[idx, 'away_team_last_year_wins'] =
↪team_stats_last_year[(away_team, current_season)]['wins']
        df.at[idx, 'away_team_last_year_losses'] =
↪team_stats_last_year[(away_team, current_season)]['losses']
        df.at[idx, 'away_team_last_year_ties'] =
↪team_stats_last_year[(away_team, current_season)]['ties']

    if (home_team, current_season) in team_stats_last_year:
        df.at[idx, 'home_team_last_year_wins'] =
↪team_stats_last_year[(home_team, current_season)]['wins']
        df.at[idx, 'home_team_last_year_losses'] =
↪team_stats_last_year[(home_team, current_season)]['losses']
        df.at[idx, 'home_team_last_year_ties'] =
↪team_stats_last_year[(home_team, current_season)]['ties']

# Now drop 2019 games
df = df[df['season'] >= 2020]

# Calculate win percentages for last season
df['away_team_last_year_win_pct'] = df['away_team_last_year_wins'] / (
    df['away_team_last_year_wins'] + df['away_team_last_year_losses'] +
↪df['away_team_last_year_ties']
)

df['home_team_last_year_win_pct'] = df['home_team_last_year_wins'] / (
    df['home_team_last_year_wins'] + df['home_team_last_year_losses'] +
↪df['home_team_last_year_ties']
```

```
)

# Calculate win percentages for current season up to the current game
df['away_team_current_win_pct'] = df['away_current_wins'] / (
    df['away_current_wins'] + df['away_current_losses'] +␣
  ↪df['away_current_ties']
)

df['home_team_current_win_pct'] = df['home_current_wins'] / (
    df['home_current_wins'] + df['home_current_losses'] +␣
  ↪df['home_current_ties']
)

# Remove all tied games
# df = df[df['home_score'] != df['away_score']]

df.to_csv('data/Games.csv')
print("Data saved back to Games.csv")

# Verify
df = pd.read_csv('data/Games.csv')
cowboys_df = df[(df['away_team'] == 'DAL')]
cowboys_df = cowboys_df.sort_values(by=['season', 'date'])
unique_last_season_stats = cowboys_df[['season', 'away_team',␣
  ↪'away_team_last_year_wins', 'away_team_last_year_losses',
                                       'away_team_last_year_win_pct']].
  ↪drop_duplicates()
display(unique_last_season_stats)
```

```
Data saved back to Games.csv

     season away_team  away_team_last_year_wins  away_team_last_year_losses  \
13     2020       DAL                         8                           8
256    2021       DAL                         6                          10
575    2022       DAL                        12                           5
813    2023       DAL                        12                           5

     away_team_last_year_win_pct
13                      0.500000
256                     0.375000
575                     0.705882
813                     0.705882
```

[8]:
```
# # Calculate the number of games played so far in the current season

# # df['away_team_games_played'] = df['away_current_wins'] +␣
  ↪df['away_current_losses'] + df['away_current_ties']
```

```
# # df['home_team_games_played'] = df['home_current_wins'] +␣
↪df['home_current_losses'] + df['home_current_ties']
```

[10]:

```
# Create division game flag

games_df = pd.read_csv('data/Games.csv')

# Define the division mapping for all NFL teams
division_mapping = {
    'ARI': 'NFC West', 'LAR': 'NFC West', 'SEA': 'NFC West', 'SF': 'NFC West',
    'ATL': 'NFC South', 'CAR': 'NFC South', 'NO': 'NFC South', 'TB': 'NFC␣
↪South',
    'CHI': 'NFC North', 'DET': 'NFC North', 'GB': 'NFC North', 'MIN': 'NFC␣
↪North',
    'DAL': 'NFC East', 'NYG': 'NFC East', 'PHI': 'NFC East', 'WAS': 'NFC East',
    'BUF': 'AFC East', 'MIA': 'AFC East', 'NE': 'AFC East', 'NYJ': 'AFC East',
    'BAL': 'AFC North', 'CIN': 'AFC North', 'CLE': 'AFC North', 'PIT': 'AFC␣
↪North',
    'HOU': 'AFC South', 'IND': 'AFC South', 'JAX': 'AFC South', 'TEN': 'AFC␣
↪South',
    'DEN': 'AFC West', 'KC': 'AFC West', 'LAC': 'AFC West', 'LVR': 'AFC West'
}

# Map teams to their respective divisions
games_df['home_division'] = games_df['home_team'].map(division_mapping)
games_df['away_division'] = games_df['away_team'].map(division_mapping)

# Determine if a game is a division game
games_df['division_game'] = (games_df['home_division'] ==␣
↪games_df['away_division']).astype(int)

# Drop the temporary division columns
games_df.drop(columns=['home_division', 'away_division'], inplace=True)

# Save the updated dataframe to a new CSV file if needed
games_df.to_csv('data/Games.csv', index=False)

# Optional: Display the first few rows to verify
display(games_df[['home_team', 'away_team', 'division_game']].head())
```

	home_team	away_team	division_game
0	KC	HOU	0
1	ATL	SEA	0
2	BAL	CLE	1
3	BUF	NYJ	1
4	CAR	LVR	0

[ ]:
```
!open data/Games.csv
```

Models

1 Logistic Regression

X: features = [‘spread_line’, ‘away_current_wins’, ‘away_current_losses’, ‘home_current_wins’, ‘home_current_losses’]

y: ‘home_win’

Try something else like favorite_team wins?

Take absolute value of spread_line and add favorite column?

Scale features?

Remove 1st/2nd weeks games?

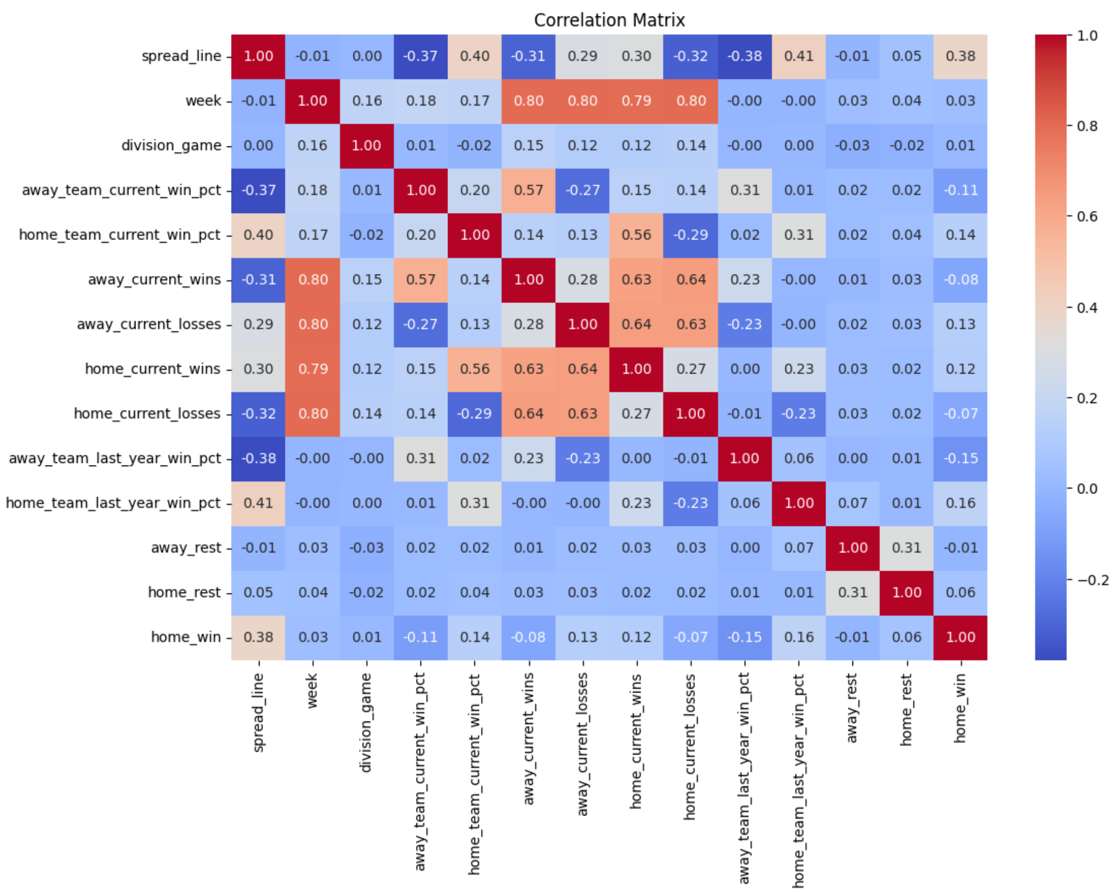
[12]:
```
# Predict home team winning
# Training on 2020, 2021, 2022
# Predicting 2023

df  = pd.read_csv('data/Games.csv')

# Remove all tied games
df = df[df['home_score'] != df['away_score']]

# Fill week 1 games
df['away_team_current_win_pct'] = df['away_team_current_win_pct'].fillna(0)
df['home_team_current_win_pct'] = df['home_team_current_win_pct'].fillna(0)

# Prepare the target variable: 1 if home_team wins, 0 if away_team wins
df['home_win'] = (df['home_score'] > df['away_score']).astype(int)

# Select features for the model
# features = [
#     'spread_line', 'week', 'division_game',
#     'away_team_current_win_pct', 'home_team_current_win_pct',
#     'away_team_last_year_win_pct', 'home_team_last_year_win_pct',
#     'away_rest', 'home_rest'
# ]
features = [
    'spread_line', 'week', 'division_game',
    'away_team_current_win_pct', 'home_team_current_win_pct',
    'away_current_wins', 'away_current_losses',
    'home_current_wins', 'home_current_losses',
    'away_team_last_year_win_pct', 'home_team_last_year_win_pct',
    'away_rest', 'home_rest'
]
```

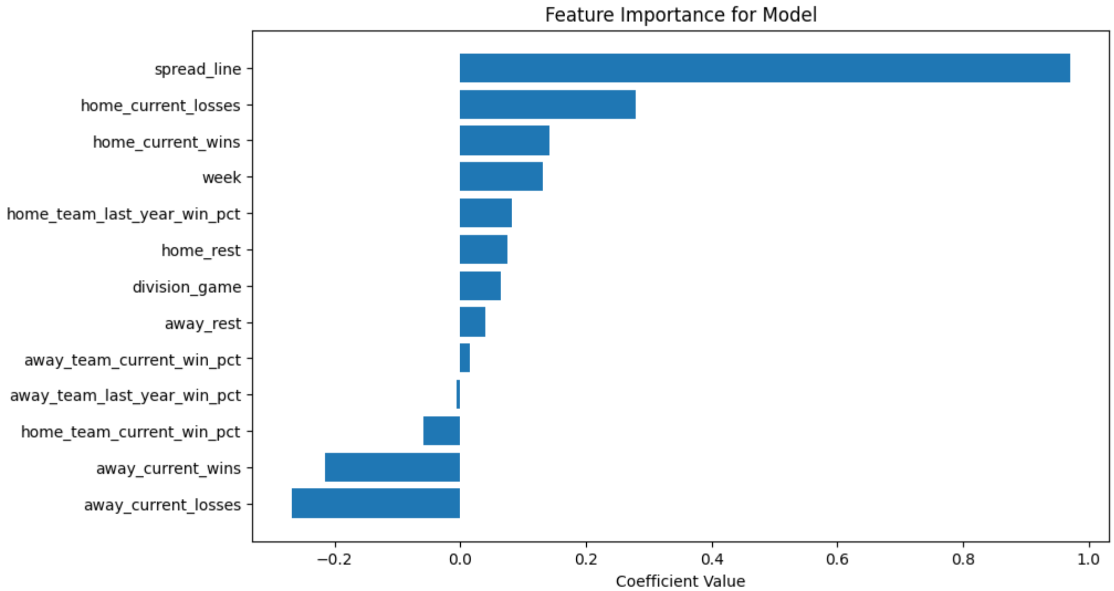
```python
# Remove week 1 games
# df = df[df['week'] != 1]

# Drop rows with missing values in the selected features
# df = df.dropna(subset=features)

# Keep only the columns needed plus any additional columns used in processing
columns_to_keep = features + ['home_team', 'away_team', 'home_win',␣
↪'home_score', 'away_score', 'season']
df = df[columns_to_keep]

# Split the data into training (2020-2022) and testing (2023)
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
train_df = df[df['season'].isin([2020, 2021, 2022])]
test_df = df[df['season'] == 2023]
X_train = train_df[features]
y_train = train_df['home_win']
X_test = test_df[features]
y_test = test_df['home_win']
print("training seasons >>", train_df['season'].unique())
print("test season >>", test_df['season'].unique())

# Initialize the scaler
scaler = StandardScaler()

# Fit the scaler on the training data and transform both training and test data
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train the logistic regression model
# model = LogisticRegression()
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Predict on the 2023 test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
coefficients = pd.Series(model.coef_[0], index=features)
coefficients = coefficients.sort_values()
intercept = model.intercept_
r_squared_train = model.score(X_train, y_train)
r_squared_test = model.score(X_test, y_test)
report = classification_report(y_test, y_pred)
```

```python
conf_matrix = confusion_matrix(y_test, y_pred)
correlations = df[features + ['home_win']].corr()
print("\nAccuracy: {:.2f}%\n".format(accuracy * 100))
print(f'R-squared (Training Set): {r_squared_train}')
print(f'R-squared (Test Set): {r_squared_test}')
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(report)
print("\nCoefficients:")
print(coefficients, "\n")
print(f'Intercept: {intercept}\n')

# Plot correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(correlations, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()

# Plot feature importance
plt.figure(figsize=(10, 6))
plt.barh(coefficients.index, coefficients.values)
plt.xlabel('Coefficient Value')
plt.title('Feature Importance for Model')
plt.show()
```

```
training seasons >> [2020 2021 2022]
test season >> [2023]

Accuracy: 68.38%

R-squared (Training Set): 0.6616352201257861
R-squared (Test Set): 0.6838235294117647

Confusion Matrix:
[[ 73  48]
 [ 38 113]]

Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.60      0.63       121
           1       0.70      0.75      0.72       151

    accuracy                           0.68       272
   macro avg       0.68      0.68      0.68       272
weighted avg       0.68      0.68      0.68       272
```

```
Coefficients:
away_current_losses            -0.268676
away_current_wins              -0.215731
home_team_current_win_pct      -0.058368
away_team_last_year_win_pct    -0.005210
away_team_current_win_pct       0.016093
away_rest                       0.040312
division_game                   0.064220
home_rest                       0.075751
home_team_last_year_win_pct     0.081393
week                            0.131864
home_current_wins               0.142195
home_current_losses             0.278944
spread_line                     0.970542
dtype: float64

Intercept: [0.12229403]
```





[13]:
```python
# Get predicted probabilities for the positive class (home team win)
y_pred_proba = model.predict_proba(X_test)[:, 1]

# Ensure test_df is a copy to avoid SettingWithCopyWarning
test_df = test_df.copy()

# Add the predicted probabilities to the test DataFrame
test_df['predict_proba'] = y_pred_proba

# Display the top predictions sorted by confidence
dispCols = ['home_team', 'away_team', 'spread_line', 'predict_proba']
display(test_df[dispCols].sort_values('predict_proba', ascending=False))
test_df.to_csv('data/test_df_results.csv')
```

	home_team	away_team	spread_line	predict_proba
945	DAL	NYG	17.5	0.921889
1042	BUF	NE	15.0	0.911153
1001	SF	SEA	14.5	0.899304
860	SF	ARI	15.0	0.896424
890	BUF	NYG	15.5	0.894103
1037	PHI	NYG	14.0	0.889988
956	MIA	LVR	14.0	0.885241
958	SF	TB	13.5	0.866832
1005	MIA	TEN	14.0	0.862420
1036	KC	LVR	11.0	0.861351
843	KC	CHI	13.0	0.854289
885	MIA	CAR	14.0	0.842557
964	DAL	WAS	13.0	0.834499

925	CLE	ARI	13.0	0.824796
1048	PHI	ARI	12.0	0.823696
868	MIA	NYG	12.5	0.815779
1058	CIN	CLE	7.5	0.809822
959	BUF	NYJ	8.0	0.792309
948	BUF	DEN	7.5	0.787598
905	BUF	TB	10.0	0.785451
991	JAX	CIN	10.0	0.781252
877	KC	DEN	10.5	0.780591
979	DAL	SEA	9.5	0.776128
856	PHI	WAS	9.5	0.775839
919	LAC	CHI	9.5	0.775217
801	BAL	HOU	9.5	0.775174
900	SEA	ARI	9.5	0.767063
866	DET	CAR	9.5	0.757671
994	BAL	LAR	7.5	0.756592
1053	KC	CIN	7.5	0.750443
817	BUF	LVR	7.5	0.749085
831	SF	NYG	10.5	0.747436
929	NO	CHI	9.0	0.747083
957	WAS	NYG	7.5	0.746851
858	LAC	LVR	7.0	0.731095
952	DET	CHI	8.0	0.725264
1014	MIA	NYJ	7.5	0.723481
826	DAL	NYJ	8.5	0.723296
836	JAX	HOU	7.5	0.722142
907	DAL	LAR	6.5	0.715316
1019	LAR	WAS	6.5	0.715274
910	MIA	NE	7.5	0.714427
1038	SF	BAL	6.5	0.714150
963	DET	GB	8.5	0.709844
887	LAR	ARI	7.0	0.704975
902	KC	LAC	5.5	0.704415
1065	LAC	KC	3.5	0.702906
1035	DEN	NE	7.0	0.698429
955	JAX	TEN	6.5	0.697714
937	BAL	CLE	6.0	0.697240
1068	SF	LAR	5.5	0.694290
807	WAS	ARI	7.0	0.692589
920	DET	LVR	7.0	0.688925
998	NO	CAR	5.5	0.683432
1016	NO	NYG	6.0	0.679251
832	BAL	IND	7.5	0.677425
962	KC	PHI	2.5	0.674383
924	BAL	SEA	6.0	0.673658
1040	DAL	DET	5.5	0.673294
859	DAL	NE	6.5	0.673236
1060	NE	NYJ	2.5	0.669752

938	CIN	HOU	5.5	0.669043
960	LAR	SEA	2.5	0.661683
1004	DAL	PHI	3.5	0.656336
818	CIN	BAL	3.5	0.652968
1051	SEA	PIT	4.5	0.652547
1066	LVR	DEN	3.5	0.649725
864	BUF	JAX	5.5	0.647799
1044	HOU	TEN	5.0	0.647671
881	CIN	SEA	3.0	0.645760
987	TB	CAR	4.0	0.645034
812	SEA	LAR	4.5	0.644304
946	SEA	WAS	6.0	0.642636
1003	LAC	DEN	3.0	0.642335
918	SF	CIN	4.0	0.640239
1061	NO	ATL	3.5	0.636315
1010	DET	DEN	5.5	0.635393
984	PIT	ARI	6.0	0.630399
1017	TEN	HOU	3.0	0.630358
1039	CLE	NYJ	6.5	0.629334
978	MIN	CHI	3.0	0.628739
1052	DEN	LAC	3.5	0.624309
932	PHI	DAL	3.5	0.623838
1033	CHI	ARI	4.5	0.623740
799	KC	DET	4.0	0.622821
884	JAX	IND	4.0	0.620489
846	CIN	LAR	3.0	0.619755
992	PIT	NE	6.0	0.618596
1013	GB	TB	4.0	0.618433
804	MIN	TB	4.0	0.617419
1049	TB	NO	2.5	0.613576
1059	DET	MIN	3.5	0.613317
899	LAR	PIT	3.5	0.613040
1026	ATL	IND	3.0	0.610636
996	CIN	IND	3.0	0.609276
837	MIA	DEN	6.0	0.609253
1064	GB	CHI	2.5	0.608330
841	SEA	CAR	5.0	0.603327
863	WAS	CHI	6.0	0.601372
815	PHI	MIN	6.0	0.598269
972	TEN	CAR	3.5	0.597338
1030	NYJ	WAS	3.0	0.597336
949	BAL	CIN	4.0	0.597087
800	ATL	CAR	3.5	0.596749
988	LAR	CLE	3.5	0.596390
827	DEN	WAS	4.0	0.595430
1020	BUF	DAL	2.0	0.594981
1008	CIN	MIN	3.0	0.594371
901	DEN	GB	1.5	0.594219

855	NO	TB	4.0	0.592827
849	BUF	MIA	2.5	0.592435
915	SEA	CLE	4.0	0.591398
970	IND	TB	2.5	0.590546
954	HOU	ARI	5.5	0.589984
1054	MIN	GB	1.0	0.587897
848	JAX	ATL	3.5	0.587248
926	GB	LAR	3.5	0.586036
1002	KC	BUF	2.0	0.585025
819	DET	SEA	4.5	0.584167
1043	CHI	ATL	3.0	0.583421
1046	JAX	CAR	3.5	0.582138
1007	LVR	LAC	3.0	0.578724
941	PIT	GB	3.0	0.577918
942	TB	TEN	2.5	0.576980
1045	IND	LVR	3.5	0.574241
923	ATL	MIN	3.5	0.572714
928	NE	WAS	2.5	0.572594
1023	LAR	NO	4.0	0.567295
809	DEN	LVR	3.0	0.566974
903	PHI	MIA	3.0	0.565050
951	CLE	PIT	2.5	0.564060
980	HOU	DEN	3.5	0.563968
810	LAC	MIA	3.0	0.563043
893	BAL	DET	3.0	0.560553
898	TB	ATL	3.0	0.560457
834	DET	ATL	3.0	0.558149
875	SF	DAL	3.5	0.554802
869	NE	NO	2.5	0.553385
1012	CLE	CHI	3.0	0.551910
1041	BAL	MIA	3.0	0.550302
993	ATL	TB	1.5	0.549087
997	CLE	JAX	2.5	0.549049
838	MIN	LAC	1.0	0.548073
805	NO	TEN	3.0	0.545659
833	CLE	TEN	3.5	0.543904
927	HOU	TB	2.5	0.537352
935	CHI	CAR	3.0	0.536624
921	PIT	TEN	3.5	0.534656
967	ATL	NO	-2.0	0.533716
886	LVR	NE	3.0	0.529678
933	CIN	BUF	1.5	0.528240
873	DEN	NYJ	2.5	0.527078
865	ATL	HOU	2.5	0.524752
844	LVR	PIT	3.0	0.522353
961	DEN	MIN	2.5	0.518835
876	LVR	GB	1.0	0.514608
974	DEN	CLE	1.5	0.514058

1034	MIA	DAL	1.5	0.511240
816	ATL	GB	3.0	0.508163
1009	IND	PIT	1.5	0.502777
922	KC	MIA	1.0	0.501037
976	PHI	BUF	2.5	0.499911
892	NO	JAX	2.5	0.493643
906	CAR	HOU	-3.0	0.492830
879	ATL	WAS	1.5	0.490045
968	CIN	PIT	-2.0	0.484649
1029	MIN	DET	-2.5	0.479822
808	CHI	GB	1.0	0.479361
822	TB	CHI	2.5	0.478672
1011	CAR	ATL	-3.0	0.471667
931	LVR	NYG	1.0	0.470975
985	TEN	IND	-1.5	0.467304
995	CHI	DET	-3.0	0.464916
1063	ARI	SEA	-2.5	0.459375
891	LAC	DAL	-1.5	0.458503
936	NE	IND	-1.0	0.457654
1000	LVR	MIN	-3.0	0.455856
908	GB	MIN	-1.0	0.434848
1062	TEN	JAX	-3.5	0.431486
973	ARI	LAR	-3.0	0.429538
947	LVR	NYJ	-1.0	0.428346
802	CLE	CIN	-1.0	0.427518
983	NYJ	ATL	-2.0	0.426973
830	PIT	CLE	-2.5	0.426210
911	NYG	NYJ	-3.0	0.425318
969	HOU	JAX	-1.0	0.424832
828	NE	MIA	-2.0	0.423719
939	JAX	SF	-3.0	0.420186
1056	IND	HOU	-1.5	0.418244
913	TEN	ATL	-2.5	0.417888
1067	NYG	PHI	-4.5	0.415472
1057	CAR	TB	-5.0	0.414270
820	HOU	IND	-1.0	0.413942
930	CAR	IND	-1.5	0.413312
943	ARI	ATL	-2.0	0.412336
1027	CAR	GB	-3.5	0.411908
897	NYG	WAS	-3.0	0.410634
1024	PIT	CIN	-3.0	0.410595
944	LAC	DET	-2.5	0.410074
1032	TB	JAX	-2.0	0.410008
862	NYG	SEA	-2.5	0.405526
1070	MIA	BUF	-2.5	0.398513
829	CAR	NO	-3.0	0.396197
977	LAC	BAL	-3.0	0.391254
806	PIT	SF	-1.5	0.388851

1031	TEN	SEA	-3.0	0.386686
835	GB	NO	-1.5	0.380743
971	NYG	NE	-4.5	0.379597
880	CHI	MIN	-3.0	0.378779
852	CLE	BAL	-2.0	0.376004
999	NYJ	HOU	-3.5	0.375635
814	NYJ	BUF	-2.5	0.373178
953	GB	LAC	-3.0	0.372358
909	IND	NO	-2.0	0.371831
940	MIN	NO	-3.0	0.370005
874	MIN	KC	-3.5	0.367834
1006	NYG	GB	-6.0	0.365915
889	TB	DET	-3.0	0.363397
894	CHI	LVR	-2.5	0.362600
867	IND	TEN	-2.5	0.355081
839	NYJ	NE	-2.5	0.354016
982	NO	DET	-4.0	0.353662
883	HOU	NO	-2.0	0.352726
989	PHI	SF	-3.0	0.352519
854	IND	LAR	-1.0	0.351166
912	PIT	JAX	-2.5	0.351099
813	NYG	DAL	-3.5	0.350540
823	TEN	LAC	-2.5	0.350262
981	NE	LAC	-4.5	0.350219
1055	BAL	PIT	-3.0	0.343008
934	NYJ	LAC	-3.0	0.340809
857	TEN	CIN	-2.5	0.340023
1022	SEA	PHI	-5.0	0.339028
1028	HOU	CLE	-3.0	0.338238
1021	JAX	BAL	-4.0	0.333786
851	CHI	DEN	-3.0	0.326069
847	GB	DET	-2.5	0.325653
870	PIT	BAL	-4.5	0.321589
871	ARI	CIN	-3.0	0.320826
895	IND	CLE	-3.5	0.319523
1047	NYG	LAR	-6.0	0.319449
990	GB	KC	-5.5	0.317965
853	HOU	PIT	-3.0	0.315962
811	NE	PHI	-3.5	0.314716
872	LAR	PHI	-3.5	0.309527
803	IND	JAX	-4.0	0.308004
850	CAR	MIN	-4.5	0.296177
821	JAX	KC	-3.5	0.285798
904	MIN	SF	-7.0	0.274497
986	WAS	MIA	-8.5	0.268707
917	DEN	KC	-7.0	0.262016
914	WAS	PHI	-7.0	0.261766
878	TEN	BAL	-5.5	0.261414

```
896         NE      BUF        -7.5      0.260712
824        ARI      NYG        -5.0      0.256403
888        NYJ      PHI        -6.5      0.243078
845         TB      PHI        -6.0      0.234880
965        SEA       SF        -7.0      0.225402
1015        NE       KC       -10.0      0.222598
840        WAS      BUF        -5.5      0.218901
975        LVR       KC        -9.0      0.215814
1018       ARI       SF       -12.5      0.198432
882        CLE       SF        -9.5      0.191604
950        CAR      DAL       -11.0      0.189415
966        NYJ      MIA        -9.5      0.188056
825        LAR       SF        -7.5      0.187695
916        ARI      BAL        -9.5      0.180374
1069       WAS      DAL       -13.0      0.173857
861        NYJ       KC        -9.5      0.160167
1025       LAC      BUF       -13.0      0.141943
1050       WAS       SF       -14.0      0.128460
842        ARI      DAL       -11.5      0.127613
```

[ ]:
```
!open data/test_df_results.csv
```

2 XGBoost

- Try both 'booster': 'gbtree' and 'booster': 'gblinear'

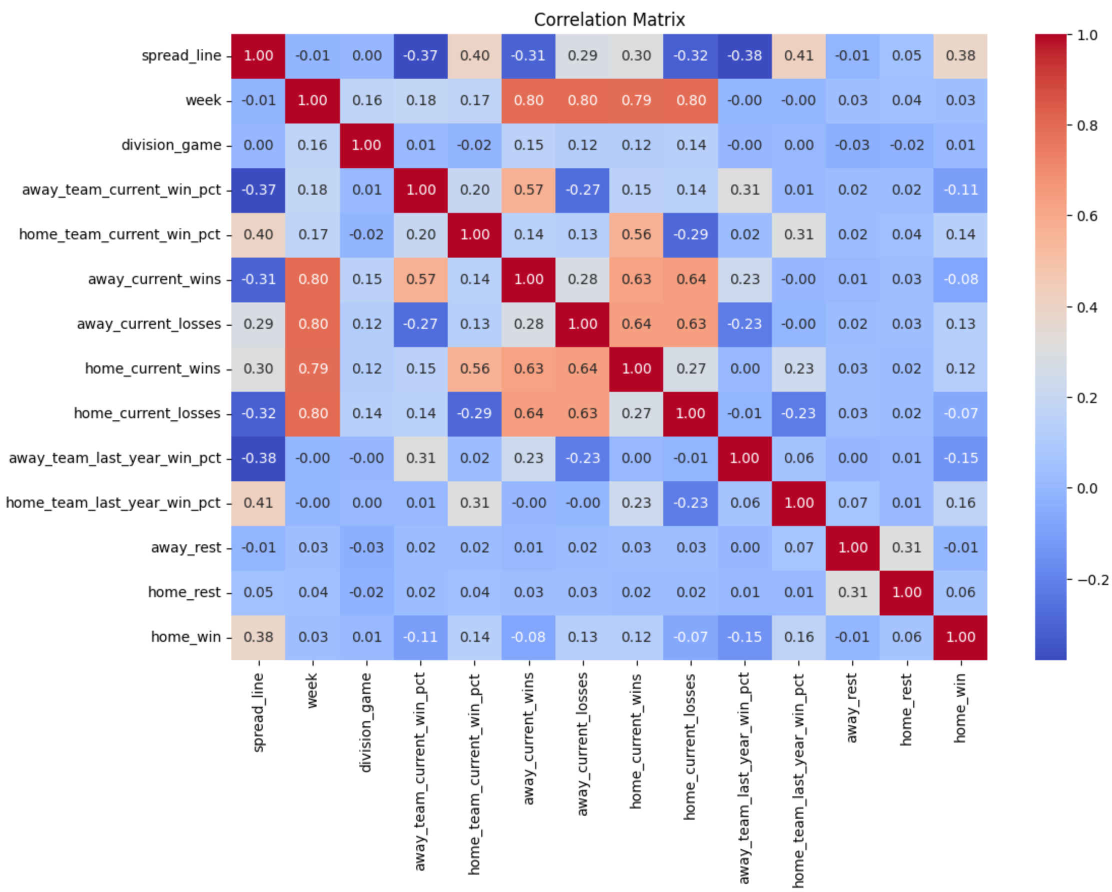
[27]:
```
df = pd.read_csv('data/Games.csv')

# Remove tied games
df = df[df['home_score'] != df['away_score']]

# Fill missing values
df['away_team_current_win_pct'] = df['away_team_current_win_pct'].fillna(0)
df['home_team_current_win_pct'] = df['home_team_current_win_pct'].fillna(0)

# Prepare target variable
df['home_win'] = (df['home_score'] > df['away_score']).astype(int)

# Select features
features = [
    'spread_line', 'week', 'division_game',
    'away_team_current_win_pct', 'home_team_current_win_pct',
    'away_current_wins', 'away_current_losses',
    'home_current_wins', 'home_current_losses',
    'away_team_last_year_win_pct', 'home_team_last_year_win_pct',
    'away_rest', 'home_rest'
]
```

```python
# Keep necessary columns
columns_to_keep = features + ['home_team', 'away_team', 'home_win',␣
↪'home_score', 'away_score', 'season']
df = df[columns_to_keep]

# Split the data into training and testing sets
train_df = df[df['season'].isin([2020, 2021, 2022])]
test_df = df[df['season'] == 2023]
X_train = train_df[features]
y_train = train_df['home_win']
X_test = test_df[features]
y_test = test_df['home_win']

# Initialize the scaler
# scaler = StandardScaler()
# X_train = scaler.fit_transform(X_train)
# X_test = scaler.transform(X_test)

# Convert data to DMatrix format
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# Define parameters
params = {
    'verbosity': 0,
    'objective': 'binary:logistic',
    # 'booster': 'gblinear',
    'booster': 'gbtree',
    # 'eval_metric': 'logloss',
    'learning_rate': 0.1
}

# Define evaluation set
evallist = [(dtrain, 'train'), (dtest, 'eval')]

# Train the XGBoost model
num_round = 1000
bst = xgb.train(params, dtrain, num_round, evallist, early_stopping_rounds=10)
# bst = xgb.train(params, dtrain, num_round, evallist)

# Predict on the test set
ypred = bst.predict(dtest)
# print(ypred)

# Convert predictions to a binary outcome (0 or 1) if using 'binary:logistic'
# Alternatively, if using 'binary:hinge', predictions are already binary
```

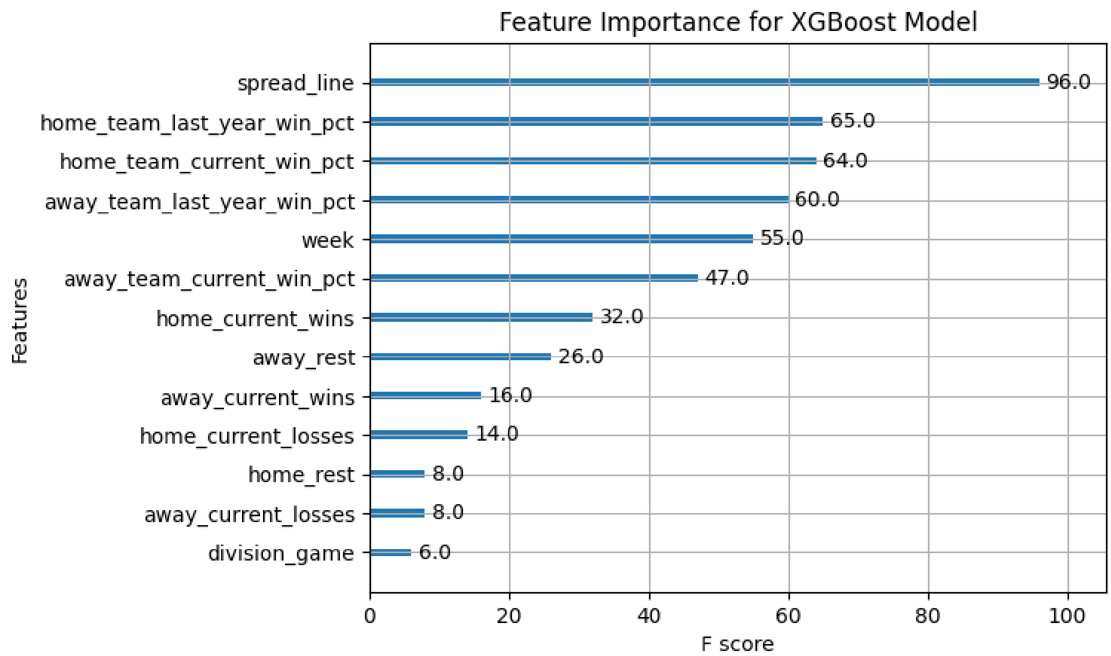
```python
# ypred_binary = ypred
ypred_binary = (ypred > 0.5).astype(int)

# Evaluate the model
accuracy = accuracy_score(y_test, ypred_binary)
precision = precision_score(y_test, ypred_binary) # Calculate precision
recall = recall_score(y_test, ypred_binary) # Calculate recall
f1 = f1_score(y_test, ypred_binary)
conf_matrix = confusion_matrix(y_test, ypred_binary)
class_report = classification_report(y_test, ypred_binary)
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)

# Plot correlation matrix
correlations = df[features + ['home_win']].corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlations, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()

# Plot feature importance
plt.figure(figsize=(10, 8))
xgb.plot_importance(bst, importance_type='weight')
plt.title('Feature Importance for XGBoost Model')
plt.show()
```

```
[0]	train-logloss:0.66059	eval-logloss:0.68178
[1]	train-logloss:0.63526	eval-logloss:0.67634
[2]	train-logloss:0.61451	eval-logloss:0.67222
[3]	train-logloss:0.59631	eval-logloss:0.66727
[4]	train-logloss:0.58043	eval-logloss:0.66848
[5]	train-logloss:0.56785	eval-logloss:0.66817
[6]	train-logloss:0.55204	eval-logloss:0.67112
[7]	train-logloss:0.54189	eval-logloss:0.67089
[8]	train-logloss:0.52835	eval-logloss:0.67158
[9]	train-logloss:0.51808	eval-logloss:0.67210
[10]	train-logloss:0.50995	eval-logloss:0.67385
[11]	train-logloss:0.50099	eval-logloss:0.67536
[12]	train-logloss:0.49389	eval-logloss:0.67780
[13]	train-logloss:0.48569	eval-logloss:0.67952
Accuracy: 0.5515
Precision: 0.6074
```

```
Recall: 0.5430
F1-Score: 0.5734

Confusion Matrix:
[[68 53]
 [69 82]]

Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.56      0.53       121
           1       0.61      0.54      0.57       151

    accuracy                           0.55       272
   macro avg       0.55      0.55      0.55       272
weighted avg       0.56      0.55      0.55       272
```




```
&lt;Figure size 1000x800 with 0 Axes&gt;
```



[28]:

```
# Analyze results

# Save to csv
results_df = test_df[['home_team', 'away_team', 'spread_line']].copy()
results_df['predicted_home_win'] = ypred
results_df.to_csv('data/xgboost_results.csv', index=False)
print("Predicted Probabilities saved to xgboost_results.csv")

# Highest and lowest predictions
df = pd.read_csv('data/xgboost_results.csv')
pd.set_option('display.max_columns', None)  # Show all columns
pd.set_option('display.max_rows', None)     # Show all rows (be cautious with␣
↪large dataframes)
pd.set_option('display.max_colwidth', None) # Show full column width
df_sorted = df[['home_team', 'away_team', 'spread_line', 'predicted_home_win']].
↪sort_values('predicted_home_win', ascending=False)
# display(df_sorted)
print("Top 20 Highest Predictions:")
display(df_sorted.head(20))
print("\nBottom 20 Lowest Predictions:")
display(df_sorted.tail(20))
```

```
Predicted Probabilities saved to xgboost_results.csv
Top 20 Highest Predictions:

    home_team away_team  spread_line  predicted_home_win
```

238	PHI	NYG	14.0	0.857556
243	BUF	NE	15.0	0.857556
44	KC	CHI	13.0	0.856618
91	BUF	NYG	15.5	0.856618
126	CLE	ARI	13.0	0.852965
165	DAL	WAS	13.0	0.847293
130	NO	CHI	9.0	0.843787
249	PHI	ARI	12.0	0.843401
159	SF	TB	13.5	0.839353
180	DAL	SEA	9.5	0.836372
78	KC	DEN	10.5	0.828886
86	MIA	CAR	14.0	0.823583
69	MIA	NYG	12.5	0.823583
202	SF	SEA	14.5	0.821938
146	DAL	NYG	17.5	0.819949
158	WAS	NYG	7.5	0.819019
120	LAC	CHI	9.5	0.816875
106	BUF	TB	10.0	0.815260
2	BAL	HOU	9.5	0.810756
111	MIA	NE	7.5	0.804493

Bottom 20 Lowest Predictions:

	home_team	away_team	spread_line	predicted_home_win
216	NE	KC	-10.0	0.266322
166	SEA	SF	-7.0	0.243613
115	WAS	PHI	-7.0	0.214279
105	MIN	SF	-7.0	0.213590
248	NYG	LAR	-6.0	0.171641
172	NYG	NE	-4.5	0.171641
226	LAC	BUF	-13.0	0.167954
79	TEN	BAL	-5.5	0.152062
118	DEN	KC	-7.0	0.152062
83	CLE	SF	-9.5	0.152062
71	PIT	BAL	-4.5	0.152062
89	NYJ	PHI	-6.5	0.152062
62	NYJ	KC	-9.5	0.152062
191	GB	KC	-5.5	0.141835
207	NYG	GB	-6.0	0.141835
167	NYJ	MIA	-9.5	0.141835
176	LVR	KC	-9.0	0.141835
223	SEA	PHI	-5.0	0.141835
187	WAS	MIA	-8.5	0.141835
251	WAS	SF	-14.0	0.141835

[ ]:

[ ]:

```

```

[ ]:

```

```

[ ]:

```
# Evaluate the model
# accuracy = accuracy_score(y_test, y_pred)
# report = classification_report(y_test, y_pred)
# conf_matrix = confusion_matrix(y_test, y_pred)
# print("\nAccuracy: {:.2f}%\n".format(accuracy * 100))
# print("\nConfusion Matrix:")
# print(conf_matrix)
# print("\nClassification Report:")
# print(report)


# Ensure test_df is a copy to avoid SettingWithCopyWarning
# test_df = test_df.copy()

# print("Predicted Probabilities:")
# print(ypred)

# # Plot feature importance
# plt.figure(figsize=(10, 6))
# xgb.plot_importance(bst, importance_type='weight')
# plt.title('Feature Importance for XGBoost Model')
# plt.show()

# Plot importance
plt.figure(figsize=(14, 10))  # Set the figure size to make the plot larger
xgb.plot_importance(bst, importance_type='weight', max_num_features=20)  # You
↪can adjust the parameters as needed
plt.title('Feature Importance for XGBoost Model')
plt.show()
```

[ ]:

```
# Analyze results

df = pd.read_csv('data/xgboost_results.csv')
pd.set_option('display.max_columns', None)  # Show all columns
pd.set_option('display.max_rows', None)     # Show all rows (be cautious with
↪large dataframes)
pd.set_option('display.max_colwidth', None) # Show full column width
df_sorted = df[['home_team', 'away_team', 'spread_line', 'predicted_prob']].
↪sort_values('predicted_prob', ascending=False)
display(df_sorted)
```

[ ]:

```

```

[ ]:

```python

```

[ ]:

```python

```

[ ]:

```python

```

[ ]:

```python

```

[ ]:

```python

```

[ ]:

```python

```

[ ]:

```python
# 3. Try a Different Solver
# Logistic Regression supports different solvers. The default is lbfgs, but you␣
↪can try saga, newton-cg, or liblinear:
# # Create a Logistic Regression model with a different solver
# model = LogisticRegression(solver='saga', max_iter=1000)
# model.fit(X_train_scaled, y_train)
```

[ ]:

```python
# Predict one week of current season
# iweek = 9

# Pick only this week's games for prediction
# dfTest = dfGamesTest[dfGamesTest.gameWeek == iweek]
```

[ ]:

```python

```

[ ]:

```python

```

[1]:

```python
!cp ../Analysis/data/Games.csv data/
```

[2]:

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, roc_auc_score
from xgboost import XGBClassifier

# Load the dataset
file_path = 'data/Games.csv'  # Make sure this path is correct for your local␣
↪system
data = pd.read_csv(file_path)

# Selecting relevant features for simplicity
features = [
    'away_score', 'home_score', 'spread_line', 'total_line',
    'away_rest', 'home_rest', 'temp', 'wind', 'miles_traveled'
]
target = 'result'  # We'll predict whether the home team won (result > 0)
```

```python
# Clean up the data by dropping rows with missing values in the relevant columns
data_cleaned = data[features + [target]].dropna()

# Define the feature matrix (X) and target vector (y)
X = data_cleaned[features]
y = (data_cleaned[target] > 0).astype(int)  # 1 if home team won, 0 otherwise

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Initialize the XGBoost classifier with reduced complexity for faster training
model = XGBClassifier(use_label_encoder=False, eval_metric='logloss',␣
↪n_estimators=50, max_depth=3)

# Train the model
model.fit(X_train, y_train)

# Predict the results for the test set
y_pred = model.predict(X_test)
y_proba = model.predict_proba(X_test)[:, 1]  # Get probabilities for the␣
↪positive class (home team wins)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_proba)

print(f"Accuracy: {accuracy:.2f}")
print(f"ROC-AUC Score: {roc_auc:.2f}")
```

```
Accuracy: 0.99
ROC-AUC Score: 1.00
```

```
/Users/tylerdurette/.pyenv/versions/3.12.0/lib/python3.12/site-
packages/xgboost/core.py:158: UserWarning: [17:21:07] WARNING:
/Users/runner/work/xgboost/xgboost/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

[3]:
```python
import pandas as pd

# Create the upcoming games data with 0's for all the features
upcoming_games_data = {
    'game_id': [
        "2024_02_BUF_MIA", "2024_02_LV_BAL", "2024_02_LAC_CAR",␣
↪"2024_02_NO_DAL", "2024_02_TB_DET",
```

```
        "2024_02_IND_GB", "2024_02_CLE_JAX", "2024_02_SF_MIN",␣
 ↪"2024_02_SEA_NE", "2024_02_NYJ_TEN",
        "2024_02_NYG_WAS", "2024_02_LA_ARI", "2024_02_PIT_DEN",␣
 ↪"2024_02_CIN_KC", "2024_02_CHI_HOU",
        "2024_02_ATL_PHI"
    ],
    'away_score': [0] * 16,  # Set all to 0
    'home_score': [0] * 16,  # Set all to 0
    'spread_line': [0] * 16,  # Set all to 0
    'total_line': [0] * 16,  # Set all to 0
    'away_rest': [0] * 16,  # Set all to 0
    'home_rest': [0] * 16,  # Set all to 0
    'temp': [0] * 16,  # Set all to 0
    'wind': [0] * 16,  # Set all to 0
    'miles_traveled': [0] * 16  # Set all to 0
}

# Convert to DataFrame
upcoming_games_df = pd.DataFrame(upcoming_games_data)

# Ensure that the columns match the training features
features = ['away_score', 'home_score', 'spread_line', 'total_line',␣
 ↪'away_rest', 'home_rest', 'temp', 'wind', 'miles_traveled']

# Make predictions on the upcoming games with zeroed features
predictions = model.predict(upcoming_games_df[features])
probabilities = model.predict_proba(upcoming_games_df[features])[:, 1]  # Get␣
 ↪probabilities for the home team winning

# Add predictions and probabilities to the DataFrame
upcoming_games_df['home_team_win_prediction'] = predictions
upcoming_games_df['home_team_win_probability'] = probabilities

# Output the predictions
print(upcoming_games_df[['game_id', 'home_team_win_prediction',␣
 ↪'home_team_win_probability']])
```

```
           game_id  home_team_win_prediction  home_team_win_probability
0    2024_02_BUF_MIA                         0                   0.290282
1     2024_02_LV_BAL                         0                   0.290282
2    2024_02_LAC_CAR                         0                   0.290282
3     2024_02_NO_DAL                         0                   0.290282
4     2024_02_TB_DET                         0                   0.290282
5     2024_02_IND_GB                         0                   0.290282
6    2024_02_CLE_JAX                         0                   0.290282
7     2024_02_SF_MIN                         0                   0.290282
8     2024_02_SEA_NE                         0                   0.290282
```

9	2024_02_NYJ_TEN	0	0.290282
10	2024_02_NYG_WAS	0	0.290282
11	2024_02_LA_ARI	0	0.290282
12	2024_02_PIT_DEN	0	0.290282
13	2024_02_CIN_KC	0	0.290282
14	2024_02_CHI_HOU	0	0.290282
15	2024_02_ATL_PHI	0	0.290282

[ ]:

```

```