In [1]:
```
import pandas as pd
from statsbombpy import sb
from mplsoccer import Pitch
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import networkx as nx
from tqdm import tqdm
from sklearn.preprocessing import LabelEncoder
```

In [2]:
```
df=sb.competitions()
```

In [3]:
```
df = df.sort_values('competition_id')
```

In [4]:
```
df.head(60)
```

Out[4]:

	competition_id	season_id	country_name	competition_name	competition_gender	c
64	2	27	England	Premier League	male	
65	2	44	England	Premier League	male	
58	7	235	France	Ligue 1	male	
59	7	108	France	Ligue 1	male	
60	7	27	France	Ligue 1	male	
0	9	281	Germany	1. Bundesliga	male	
1	9	27	Germany	1. Bundesliga	male	
48	11	22	Spain	La Liga	male	
55	11	278	Spain	La Liga	male	
47	11	23	Spain	La Liga	male	
54	11	37	Spain	La Liga	male	
53	11	38	Spain	La Liga	male	
52	11	39	Spain	La Liga	male	
51	11	40	Spain	La Liga	male	
50	11	41	Spain	La Liga	male	
49	11	21	Spain	La Liga	male	
39	11	42	Spain	La Liga	male	
40	11	4	Spain	La Liga	male	
41	11	1	Spain	La Liga	male	
42	11	2	Spain	La Liga	male	
43	11	27	Spain	La Liga	male	

	competition_id	season_id	country_name	competition_name	competition_gender	c
44	11	26	Spain	La Liga	male	
45	11	25	Spain	La Liga	male	
46	11	24	Spain	La Liga	male	
38	11	90	Spain	La Liga	male	
67	12	86	Italy	Serie A	male	
66	12	27	Italy	Serie A	male	
3	16	4	Europe	Champions League	male	
6	16	27	Europe	Champions League	male	
15	16	37	Europe	Champions League	male	
7	16	26	Europe	Champions League	male	
8	16	25	Europe	Champions League	male	
4	16	1	Europe	Champions League	male	
20	16	276	Europe	Champions League	male	
19	16	71	Europe	Champions League	male	
18	16	277	Europe	Champions League	male	
17	16	76	Europe	Champions League	male	
16	16	44	Europe	Champions League	male	
5	16	2	Europe	Champions League	male	
14	16	39	Europe	Champions League	male	
13	16	41	Europe	Champions League	male	
12	16	21	Europe	Champions League	male	

	competition_id	season_id	country_name	competition_name	competition_gender	c
11	16	22	Europe	Champions League	male	
10	16	23	Europe	Champions League	male	
9	16	24	Europe	Champions League	male	
70	35	75	Europe	UEFA Europa League	male	
26	37	42	England	FA Women's Super League	female	
25	37	90	England	FA Women's Super League	female	
27	37	4	England	FA Women's Super League	female	
36	43	269	International	FIFA World Cup	male	
35	43	270	International	FIFA World Cup	male	
34	43	272	International	FIFA World Cup	male	
33	43	51	International	FIFA World Cup	male	
32	43	54	International	FIFA World Cup	male	
31	43	55	International	FIFA World Cup	male	
30	43	3	International	FIFA World Cup	male	
29	43	106	International	FIFA World Cup	male	
61	44	107	United States of America	Major League Soccer	male	
63	49	3	United States of America	NWSL	female	
71	53	106	Europe	UEFA Women's Euro	female	

In [5]:
```
# Specify competition_id
competition_id = 11  # Replace with your desired competition ID

# Fetch all available seasons for the competition
competitions = sb.competitions()
seasons = competitions[competitions['competition_id'] == competition_id]['season
```

```
# List to store all matches and events
all_events = []

# Loop through each season and fetch matches and events
for season_id in tqdm(seasons, desc="Processing Seasons"):
    try:
        # Fetch matches for the current season
        matches = sb.matches(competition_id=competition_id, season_id=season_id)

        # Loop through matches and fetch events
        for match_id in matches['match_id']:
            try:
                # Fetch events for the current match
                events = sb.events(match_id=match_id)

                # Add match_id, home_team, and away_team to the events DataFrame
                home_team = matches[matches['match_id'] == match_id]['home_team'
                away_team = matches[matches['match_id'] == match_id]['away_team'

                events['match_id'] = match_id
                events['home_team'] = home_team
                events['away_team'] = away_team

                # Append the events to the list
                all_events.append(events)
            except Exception as e:
                print(f"Error fetching events for match_id {match_id}: {e}")
    except Exception as e:
        print(f"Error fetching matches for season_id {season_id}: {e}")

# Combine all events into a single DataFrame
if all_events:
    e_df = pd.concat(all_events, ignore_index=True)
    print(e_df.head())
else:
    print("No events data fetched.")
```

```
Processing Seasons: 100%|████████████████████████████████████████████████
██████| 18/18 [09:03<00:00, 30.19s/it]
```

```
  50_50 bad_behaviour_card ball_receipt_outcome ball_recovery_offensive  \
0   NaN                NaN                  NaN                      NaN
1   NaN                NaN                  NaN                      NaN
2   NaN                NaN                  NaN                      NaN
3   NaN                NaN                  NaN                      NaN
4   NaN                NaN                  NaN                      NaN

  ball_recovery_recovery_failure carry_end_location clearance_aerial_won  \
0                            NaN                NaN                   NaN
1                            NaN                NaN                   NaN
2                            NaN                NaN                   NaN
3                            NaN                NaN                   NaN
4                            NaN                NaN                   NaN

  clearance_body_part clearance_head clearance_left_foot  ... shot_redirect  \
0                 NaN            NaN                 NaN  ...           NaN
1                 NaN            NaN                 NaN  ...           NaN
2                 NaN            NaN                 NaN  ...           NaN
3                 NaN            NaN                 NaN  ...           NaN
4                 NaN            NaN                 NaN  ...           NaN

  shot_follows_dribble goalkeeper_success_in_play goalkeeper_lost_in_play  \
0                  NaN                       NaN                     NaN
1                  NaN                       NaN                     NaN
2                  NaN                       NaN                     NaN
3                  NaN                       NaN                     NaN
4                  NaN                       NaN                     NaN

  half_start_late_video_start player_off_permanent goalkeeper_lost_out  \
0                         NaN                  NaN                 NaN
1                         NaN                  NaN                 NaN
2                         NaN                  NaN                 NaN
3                         NaN                  NaN                 NaN
4                         NaN                  NaN                 NaN

   goalkeeper_success_out half_end_early_video_end goalkeeper_saved_to_post
0                     NaN                      NaN                      NaN
1                     NaN                      NaN                      NaN
2                     NaN                      NaN                      NaN
3                     NaN                      NaN                      NaN
4                     NaN                      NaN                      NaN

[5 rows x 121 columns]
```

In [6]:
```
e_df
```

Out[6]:

	50_50	bad_behaviour_card	ball_receipt_outcome	ball_recovery_offensive	ball_
0	NaN	NaN	NaN	NaN	
1	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	
...	...	...	...	...	
3136478	NaN	NaN	NaN	NaN	
3136479	NaN	NaN	NaN	NaN	
3136480	NaN	NaN	NaN	NaN	
3136481	NaN	NaN	NaN	NaN	
3136482	NaN	NaN	NaN	NaN	

3136483 rows × 121 columns

In [7]:
```
# Filter shot events
shots = e_df[e_df['type'] == 'Shot']
```

In [8]:
```
# Initialize home and away scores and game_state in the shots DataFrame using .l
shots.loc[:, 'home_score'] = 0
shots.loc[:, 'away_score'] = 0
shots.loc[:, 'game_state'] = 'tied'  # Default game state is 'tied'

# Loop through each match and update the game_state for each shot event in the s
for match_id in shots['match_id'].unique():
    # Extract shot events for the current match
    match_shots = shots[shots['match_id'] == match_id]

    home_goals = 0
    away_goals = 0

    # Loop through the shot events for this match to calculate game state
    for i, event in match_shots.iterrows():
        # Update scores if a goal is scored
        if event['shot_outcome'] == 'Goal':
            if event['team'] == event['home_team']:
                home_goals += 1
            elif event['team'] == event['away_team']:
                away_goals += 1

        # Set the current home and away scores for the shot event using .loc
        shots.loc[i, 'home_score'] = home_goals
        shots.loc[i, 'away_score'] = away_goals

        # Determine the game state based on the team taking the shot
        if event['team'] == event['home_team']:
            if home_goals > away_goals:
                shots.loc[i, 'game_state'] = 'leading'
```

```
            elif home_goals < away_goals:
                shots.loc[i, 'game_state'] = 'trailing'
            else:
                shots.loc[i, 'game_state'] = 'tied'

        elif event['team'] == event['away_team']:
            if away_goals > home_goals:
                shots.loc[i, 'game_state'] = 'leading'
            elif away_goals < home_goals:
                shots.loc[i, 'game_state'] = 'trailing'
            else:
                shots.loc[i, 'game_state'] = 'tied'

# Display the updated shots DataFrame with game_state
print(shots[['match_id', 'home_team', 'away_team', 'home_score', 'away_score', '
```

```
C:\Users\harsh\AppData\Local\Temp\ipykernel_12040\2881702795.py:2: PerformanceWar
ning: DataFrame is highly fragmented.  This is usually the result of calling `fra
me.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newfr
ame = frame.copy()`
  shots.loc[:, 'home_score'] = 0
C:\Users\harsh\AppData\Local\Temp\ipykernel_12040\2881702795.py:2: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  shots.loc[:, 'home_score'] = 0
C:\Users\harsh\AppData\Local\Temp\ipykernel_12040\2881702795.py:3: PerformanceWar
ning: DataFrame is highly fragmented.  This is usually the result of calling `fra
me.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newfr
ame = frame.copy()`
  shots.loc[:, 'away_score'] = 0
C:\Users\harsh\AppData\Local\Temp\ipykernel_12040\2881702795.py:3: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  shots.loc[:, 'away_score'] = 0
C:\Users\harsh\AppData\Local\Temp\ipykernel_12040\2881702795.py:4: PerformanceWar
ning: DataFrame is highly fragmented.  This is usually the result of calling `fra
me.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newfr
ame = frame.copy()`
  shots.loc[:, 'game_state'] = 'tied'  # Default game state is 'tied'
C:\Users\harsh\AppData\Local\Temp\ipykernel_12040\2881702795.py:4: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  shots.loc[:, 'game_state'] = 'tied'  # Default game state is 'tied'
```

	match_id	home_team	away_team	home_score	away_score	game_state
3805	3773386	Deportivo Alavés	Barcelona	0	0	tied
3806	3773386	Deportivo Alavés	Barcelona	0	0	tied
3807	3773386	Deportivo Alavés	Barcelona	0	0	tied
3808	3773386	Deportivo Alavés	Barcelona	0	0	tied
3809	3773386	Deportivo Alavés	Barcelona	1	0	leading

In [9]:
```
# Ensure 'timestamp' is in datetime format if it's not already
shots.loc[:, 'timestamp'] = pd.to_datetime(shots['timestamp'])

# Initialize the 'time_since_last_event' column to NaN using .loc
shots.loc[:, 'time_since_last_event'] = None

# Loop through each match to calculate the time difference
for match_id in shots['match_id'].unique():
    # Extract the shots for the current match
    match_shots = shots[shots['match_id'] == match_id]

    # Loop through the shots to calculate the time since the last event
    for i in range(1, len(match_shots)):
        current_event_time = match_shots.iloc[i]['timestamp']
        previous_event_time = match_shots.iloc[i - 1]['timestamp']

        # Calculate the time difference in seconds (you can change this to minut
        time_diff = (current_event_time - previous_event_time).total_seconds()

        # Update the 'time_since_last_event' for the current event using .loc
        shots.loc[match_shots.index[i], 'time_since_last_event'] = time_diff

# Display the updated shots DataFrame with 'time_since_last_event'
print(shots[['match_id', 'timestamp', 'time_since_last_event']].head())
```

```
C:\Users\harsh\AppData\Local\Temp\ipykernel_12040\2026288433.py:5: PerformanceWar
ning: DataFrame is highly fragmented.  This is usually the result of calling `fra
me.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newfr
ame = frame.copy()`
  shots.loc[:, 'time_since_last_event'] = None
C:\Users\harsh\AppData\Local\Temp\ipykernel_12040\2026288433.py:5: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  shots.loc[:, 'time_since_last_event'] = None
```

	match_id	timestamp	time_since_last_event
3805	3773386	2025-02-11 00:12:19.834000	None
3806	3773386	2025-02-11 00:16:07.385000	227.551
3807	3773386	2025-02-11 00:19:42.833000	215.448
3808	3773386	2025-02-11 00:22:26.180000	163.347
3809	3773386	2025-02-11 00:30:06.585000	460.405

In [10]:
```
# Function to calculate 2D or 3D distance based on the shot_end_location type
def calculate_shot_distance(row):
    if len(row['shot_end_location']) == 3:
        return np.sqrt(
            (row['location'][0] - row['shot_end_location'][0]) ** 2 +
            (row['location'][1] - row['shot_end_location'][1]) ** 2 +
            (0 - row['shot_end_location'][2]) ** 2
```

```
        )
    else:
        return np.sqrt(
            (row['location'][0] - row['shot_end_location'][0]) ** 2 +
            (row['location'][1] - row['shot_end_location'][1]) ** 2
        )

# Function to categorize shot placement into sections of the goal (Top-Corner, B
def categorize_goal_section(placement, y_value):
    # Left or right top/bottom corners (y-value is within 36-37 or 43-44)
    if 36 <= y_value <= 37 or 43 <= y_value <= 44:
        if placement <= 0.5:  # Bottom-Corner
            return 'Bottom-Corner'
        elif placement > 2.2:  # Top-Corner
            return 'Top-Corner'

    # Center section of the goal (y-value is between 37 and 43)
    elif 37 < y_value < 43:
        if placement <= 0.5:  # Bottom
            return 'Bottom'
        elif placement > 2.2:  # Top
            return 'Top'
        else:  # Middle
            return 'Middle'

    # Ensure that shots falling in the middle section (placement between 1.0 and
    elif 1.0 < placement <= 2.5:  # Middle section
        return 'Middle'

    # For shots outside the valid placement, categorize as off-target
    return 'Off-Target'

# Function to check if a shot is off-target based on x, y, and z ranges
def is_off_target(row):
    valid_x_range = (0, 120)
    valid_y_range = (36, 44)
    valid_z_range = (0, 2.67)

    # Check if x value is out of valid range
    if not (valid_x_range[0] <= row['shot_end_location'][0] <= valid_x_range[1]):
        return True

    # Check if y value is out of valid range
    if not (valid_y_range[0] <= row['shot_end_location'][1] <= valid_y_range[1]):
        return True

    # Check if z value is out of valid range, unless z = 0
    if len(row['shot_end_location']) == 3 and not (valid_z_range[0] <= row['shot
        # Only treat as off-target if z is out of range (except for z=0, which i
        if row['shot_end_location'][2] != 0:
            return True

    # If all checks pass, the shot is on target
    return False

# Function to process the shot and add additional features
def process_shot(shots):
    shots = shots.copy()

    # Calculate shot distance based on start and end location
```

```
    shots['shot_distance'] = shots.apply(calculate_shot_distance, axis=1)

    # Calculate shot angle in radians and convert to degrees
    shots['shot_angle'] = np.arctan2(
        shots['shot_end_location'].apply(lambda loc: loc[1] if len(loc) == 2 els
        shots['shot_end_location'].apply(lambda loc: loc[0] if len(loc) == 2 els
    )
    shots['shot_angle_degrees'] = np.degrees(shots['shot_angle'])

    # Handle Z-value explicitly when shot_end_location has 2 dimensions (set Z t
    shots['shot_placement'] = shots['shot_end_location'].apply(lambda loc: loc[2

    # Categorize shot placement into goal sections (Top-Corner, Bottom-Corner, T
    shots['goal_section'] = shots.apply(lambda row: categorize_goal_section(row[

    # Apply the off-target check
    shots['off_target'] = shots.apply(is_off_target, axis=1)

    # Final classification: If the shot is off-target, label as 'Off-Target', ot
    shots['shot_classification'] = np.where(shots['off_target'], 'Off-Target', s

    return shots


shots = process_shot(shots)
```

In [11]:
```
print(shots.dtypes)
```

```
50_50                            object
bad_behaviour_card               object
ball_receipt_outcome             object
ball_recovery_offensive          object
ball_recovery_recovery_failure   object
                                  ...
shot_angle_degrees              float64
shot_placement                  float64
goal_section                     object
off_target                         bool
shot_classification              object
Length: 132, dtype: object
```

In [12]:
```
# List of important columns you want to include in your model
important_columns = [
    'match_id', 'timestamp','location','shot_end_location','shot_first_time',
     'shot_technique', 'shot_body_part', 'shot_type','shot_outcome', 'under_pres
    'shot_distance', 'shot_angle_degrees','time_since_last_event', 'game_state',
    'home_score', 'away_score','position','period','shot_classification']


refined_model_data = shots[important_columns].copy()

refined_model_data.head(50)
```

Out[12]:

	match_id	timestamp	location	shot_end_location	shot_first_time	shot_techn
3805	3773386	2025-02-11 00:12:19.834000	[108.6, 28.0]	[120.0, 47.8, 0.0]	NaN	Nc
3806	3773386	2025-02-11 00:16:07.385000	[103.6, 51.0]	[115.8, 42.1, 0.0]	True	Nc
3807	3773386	2025-02-11 00:19:42.833000	[104.3, 33.9]	[120.0, 50.6, 0.0]	True	Nc
3808	3773386	2025-02-11 00:22:26.180000	[97.9, 44.3]	[119.8, 37.6]	NaN	Nc
3809	3773386	2025-02-11 00:30:06.585000	[118.3, 42.1]	[120.0, 40.7, 0.0]	NaN	Nc
3810	3773386	2025-02-11 00:35:25.546000	[113.1, 39.3]	[120.0, 38.5, 4.9]	NaN	Nc
3811	3773386	2025-02-11 00:42:34.293000	[101.3, 41.6]	[110.5, 39.5]	NaN	Nc
3812	3773386	2025-02-11 00:44:47.440000	[84.7, 42.1]	[120.0, 46.4, 5.6]	NaN	Nc
3813	3773386	2025-02-11 00:45:16.926000	[105.3, 32.9]	[118.3, 37.0, 0.0]	True	Nc
3814	3773386	2025-02-11 00:02:47.901000	[113.3, 26.1]	[118.2, 35.8, 1.3]	NaN	Nc
3815	3773386	2025-02-11 00:05:54.058000	[107.4, 27.2]	[117.1, 36.3, 0.5]	NaN	Nc
3816	3773386	2025-02-11 00:08:34.260000	[111.5, 31.6]	[120.0, 47.7, 0.7]	True	V
3817	3773386	2025-02-11 00:14:25.864000	[99.0, 24.7]	[120.0, 34.0, 5.3]	NaN	Nc
3818	3773386	2025-02-11 00:17:57.784000	[107.6, 31.3]	[120.0, 42.7, 0.4]	True	
3819	3773386	2025-02-11 00:19:34.950000	[108.4, 49.5]	[115.2, 43.3, 0.3]	NaN	Nc
3820	3773386	2025-02-11 00:22:39.838000	[98.2, 34.7]	[120.0, 25.8, 0.2]	True	Nc
3821	3773386	2025-02-11 00:23:34.135000	[106.6, 35.3]	[117.7, 39.8, 1.6]	NaN	Nc

	match_id	timestamp	location	shot_end_location	shot_first_time	shot_techn
3822	3773386	2025-02-11 00:28:44.317000	[103.2, 35.7]	[104.1, 36.0]	NaN	Half V
3823	3773386	2025-02-11 00:32:00.879000	[103.4, 24.7]	[116.5, 37.6, 2.0]	NaN	Nc
3824	3773386	2025-02-11 00:32:50.235000	[96.4, 46.8]	[98.2, 46.0]	NaN	Nc
3825	3773386	2025-02-11 00:37:44.138000	[97.5, 49.6]	[99.7, 48.4]	NaN	Nc
3826	3773386	2025-02-11 00:38:05.308000	[92.9, 28.0]	[102.5, 34.3]	NaN	Nc
3827	3773386	2025-02-11 00:42:02.502000	[106.4, 49.8]	[117.8, 41.2, 0.9]	True	Nc
3828	3773386	2025-02-11 00:42:38.083000	[88.7, 39.7]	[116.6, 39.3, 2.1]	NaN	Nc
3829	3773386	2025-02-11 00:43:25.053000	[114.5, 46.0]	[118.2, 40.8]	NaN	Nc
3830	3773386	2025-02-11 00:44:08.947000	[108.9, 44.1]	[120.0, 27.2, 0.0]	True	Nc
3831	3773386	2025-02-11 00:45:44.395000	[94.0, 49.1]	[100.2, 47.5]	NaN	Nc
3832	3773386	2025-02-11 00:48:11.821000	[99.0, 46.8]	[107.8, 45.0]	True	Nc
3833	3773386	2025-02-11 00:48:33.533000	[104.9, 49.8]	[120.0, 33.4, 1.9]	NaN	Nc
7516	3773565	2025-02-11 00:01:39.034000	[101.1, 50.9]	[117.2, 41.6, 1.6]	NaN	Nc
7517	3773565	2025-02-11 00:10:28.938000	[106.6, 34.6]	[107.6, 34.9]	NaN	Nc
7518	3773565	2025-02-11 00:11:06.691000	[110.4, 37.9]	[120.0, 43.4, 0.2]	NaN	Nc
7519	3773565	2025-02-11 00:15:41.304000	[103.1, 57.1]	[120.0, 33.8, 0.2]	NaN	Nc
7520	3773565	2025-02-11 00:16:47.548000	[92.6, 47.5]	[94.7, 47.1]	NaN	Nc
7521	3773565	2025-02-11 00:22:23.374000	[109.5, 55.4]	[120.0, 25.1, 0.1]	True	Nc

	match_id	timestamp	location	shot_end_location	shot_first_time	shot_techn
7522	3773565	2025-02-11 00:29:17.718000	[103.1, 31.4]	[104.2, 31.8]	NaN	Nc
7523	3773565	2025-02-11 00:32:38.713000	[99.2, 56.5]	[117.9, 43.2, 0.4]	NaN	Nc
7524	3773565	2025-02-11 00:34:58.025000	[103.9, 37.9]	[120.0, 36.6, 2.1]	NaN	Nc
7525	3773565	2025-02-11 00:38:23.131000	[100.5, 45.8]	[120.0, 40.9, 7.6]	True	Nc
7526	3773565	2025-02-11 00:41:54.600000	[100.8, 43.8]	[120.0, 36.1, 0.2]	NaN	Nc
7527	3773565	2025-02-11 00:04:55.141000	[104.0, 35.9]	[117.4, 38.4, 0.2]	NaN	Nc
7528	3773565	2025-02-11 00:10:47.410000	[102.1, 45.6]	[120.0, 37.7, 4.0]	NaN	Nc
7529	3773565	2025-02-11 00:14:25.059000	[111.1, 32.3]	[120.0, 38.6, 5.5]	True	Half V
7530	3773565	2025-02-11 00:17:58.795000	[113.2, 52.5]	[120.0, 36.5, 0.2]	NaN	Half V
7531	3773565	2025-02-11 00:21:18.935000	[107.6, 29.1]	[120.0, 44.3, 3.4]	NaN	Half V
7532	3773565	2025-02-11 00:29:15.684000	[103.5, 49.5]	[120.0, 43.4, 7.6]	True	Half V
7533	3773565	2025-02-11 00:34:14.506000	[100.2, 33.3]	[107.8, 35.0]	NaN	Nc
7534	3773565	2025-02-11 00:34:50.223000	[94.0, 44.4]	[101.8, 43.9]	NaN	Nc
7535	3773565	2025-02-11 00:35:57.390000	[100.6, 58.9]	[116.1, 42.1, 0.7]	NaN	Nc
7536	3773565	2025-02-11 00:39:45.616000	[103.8, 56.4]	[106.8, 53.9]	NaN	Nc

In [13]:
```
# Check for missing values in the important columns
missing_values = refined_model_data[important_columns].isnull().sum()
```

In [14]:
```
missing_values
```

Out[14]:
```
match_id                    0
timestamp                   0
location                    0
shot_end_location           0
shot_first_time         14505
shot_technique              0
shot_body_part              0
shot_type                   0
shot_outcome                0
under_pressure          17448
shot_distance               0
shot_angle_degrees          0
time_since_last_event     868
game_state                  0
home_score                  0
away_score                  0
position                    0
period                      0
shot_classification         0
dtype: int64
```

In [15]:
```
cleaned_model_data = refined_model_data.copy()


# Assuming `df` is your original dataframe
cleaned_model_data['time_since_last_event'] = pd.to_numeric(refined_model_data['

# Fill NaN values with the mean (you can also use median or any other method you
cleaned_model_data['time_since_last_event'].fillna(refined_model_data['time_sinc


# Display the cleaned data
cleaned_model_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 21210 entries, 3805 to 3136364
Data columns (total 19 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   match_id               21210 non-null  int64
 1   timestamp              21210 non-null  object
 2   location               21210 non-null  object
 3   shot_end_location      21210 non-null  object
 4   shot_first_time        6705 non-null   object
 5   shot_technique         21210 non-null  object
 6   shot_body_part         21210 non-null  object
 7   shot_type              21210 non-null  object
 8   shot_outcome           21210 non-null  object
 9   under_pressure         3762 non-null   object
 10  shot_distance          21210 non-null  float64
 11  shot_angle_degrees     21210 non-null  float64
 12  time_since_last_event  21210 non-null  float64
 13  game_state             21210 non-null  object
 14  home_score             21210 non-null  int64
 15  away_score             21210 non-null  int64
 16  position               21210 non-null  object
 17  period                 21210 non-null  int64
 18  shot_classification    21210 non-null  object
dtypes: float64(3), int64(4), object(12)
memory usage: 3.7+ MB
```

```
C:\Users\harsh\AppData\Local\Temp\ipykernel_12040\3237411695.py:8: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained as
signment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work becau
se the intermediate object on which we are setting values always behaves as a cop
y.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth
od({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to pe
rform the operation inplace on the original object.


  cleaned_model_data['time_since_last_event'].fillna(refined_model_data['time_sin
ce_last_event'].mean(), inplace=True)
```

In [16]:
```
# Function to convert NaN and non-boolean values to boolean
def to_boolean(x):
    if pd.isna(x):  # If the value is NaN
        return False
    return bool(x)

# Apply the function to convert to boolean
cleaned_model_data['shot_first_time'] = cleaned_model_data['shot_first_time'].ap
cleaned_model_data['under_pressure'] = cleaned_model_data['under_pressure'].appl
```

In [17]:
```
cleaned_model_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 21210 entries, 3805 to 3136364
Data columns (total 19 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   match_id               21210 non-null  int64
 1   timestamp              21210 non-null  object
 2   location               21210 non-null  object
 3   shot_end_location      21210 non-null  object
 4   shot_first_time        21210 non-null  bool
 5   shot_technique         21210 non-null  object
 6   shot_body_part         21210 non-null  object
 7   shot_type              21210 non-null  object
 8   shot_outcome           21210 non-null  object
 9   under_pressure         21210 non-null  bool
 10  shot_distance          21210 non-null  float64
 11  shot_angle_degrees     21210 non-null  float64
 12  time_since_last_event  21210 non-null  float64
 13  game_state             21210 non-null  object
 14  home_score             21210 non-null  int64
 15  away_score             21210 non-null  int64
 16  position               21210 non-null  object
 17  period                 21210 non-null  int64
 18  shot_classification    21210 non-null  object
dtypes: bool(2), float64(3), int64(4), object(10)
memory usage: 3.5+ MB
```

In [18]:
```
from sklearn.preprocessing import StandardScaler

# Initialize StandardScaler
scaler = StandardScaler()

# List of numerical columns to scale
```

```
numerical_columns = ['shot_distance', 'shot_angle_degrees']

# Apply StandardScaler to the numerical columns
cleaned_model_data[numerical_columns] = scaler.fit_transform(cleaned_model_data[

# Check the scaled data
cleaned_model_data[numerical_columns]
```

Out[18]:

	shot_distance	shot_angle_degrees
3805	0.716722	0.821318
3806	-0.003108	0.290491
3807	0.723585	1.170489
3808	0.721800	-0.488807
3809	-1.201807	-0.089012
...	...	...
3136360	-1.006759	1.544207
3136361	-0.344708	0.170933
3136362	0.605622	0.087028
3136363	-0.859126	-0.770692
3136364	-1.079093	1.797731

21210 rows × 2 columns

In [19]:
```
# Drop the columns that are not useful for the model
cleaned_model_data = cleaned_model_data.drop(columns=['timestamp', 'match_id','l
```

In [20]:
```
cleaned_model_data['Goal'] = cleaned_model_data['shot_outcome'].apply(lambda x:

cleaned_model_data.drop(columns=['shot_outcome'], inplace=True)
```

In [21]:
```
# Map integers to meaningful labels
period_mapping = {
    1: "First Half",
    2: "Second Half",
    3: "Third Period",
    4: "Fourth Period",
    5: "Penalties"
}

# Apply the mapping to the 'period' column
cleaned_model_data['period'] = cleaned_model_data['period'].map(period_mapping)

# Convert the column to a categorical type
cleaned_model_data['period'] = pd.Categorical(
    cleaned_model_data['period'],
    categories=["First Half", "Second Half", "Third Period", "Fourth Period", "P
    ordered=True
)
```

In [22]:
```
catboost_data=cleaned_model_data.copy()
```

In [23]:
```
# Apply One-Hot Encoding using pandas get_dummies
categorical_columns = ['shot_technique', 'shot_body_part', 'shot_type', 'game_st

# Perform One-Hot Encoding on the specified columns
cleaned_model_data = pd.get_dummies(cleaned_model_data, columns=categorical_colu



# Check the changes
cleaned_model_data
```

Out[23]:

	shot_first_time	under_pressure	shot_distance	shot_angle_degrees	time_since_
3805	False	False	0.716722	0.821318	1
3806	True	False	-0.003108	0.290491	2
3807	True	False	0.723585	1.170489	2
3808	False	False	0.721800	-0.488807	1
3809	False	False	-1.201807	-0.089012	4
...	...	...	...	...	
3136360	False	False	-1.006759	1.544207	4
3136361	False	True	-0.344708	0.170933	
3136362	False	False	0.605622	0.087028	2
3136363	True	False	-0.859126	-0.770692	4
3136364	False	False	-1.079093	1.797731	

21210 rows × 62 columns

In [24]:
```
cleaned_model_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 21210 entries, 3805 to 3136364
Data columns (total 62 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   shot_first_time                       21210 non-null  bool
 1   under_pressure                        21210 non-null  bool
 2   shot_distance                         21210 non-null  float64
 3   shot_angle_degrees                    21210 non-null  float64
 4   time_since_last_event                 21210 non-null  float64
 5   home_score                            21210 non-null  int64
 6   away_score                            21210 non-null  int64
 7   Goal                                  21210 non-null  int64
 8   shot_technique_Backheel               21210 non-null  bool
 9   shot_technique_Diving Header          21210 non-null  bool
 10  shot_technique_Half Volley            21210 non-null  bool
 11  shot_technique_Lob                    21210 non-null  bool
 12  shot_technique_Normal                 21210 non-null  bool
 13  shot_technique_Overhead Kick          21210 non-null  bool
 14  shot_technique_Volley                 21210 non-null  bool
 15  shot_body_part_Head                   21210 non-null  bool
 16  shot_body_part_Left Foot              21210 non-null  bool
 17  shot_body_part_Other                  21210 non-null  bool
 18  shot_body_part_Right Foot             21210 non-null  bool
 19  shot_type_Corner                      21210 non-null  bool
 20  shot_type_Free Kick                   21210 non-null  bool
 21  shot_type_Open Play                   21210 non-null  bool
 22  shot_type_Penalty                     21210 non-null  bool
 23  game_state_leading                    21210 non-null  bool
 24  game_state_tied                       21210 non-null  bool
 25  game_state_trailing                   21210 non-null  bool
 26  position_Center Attacking Midfield    21210 non-null  bool
 27  position_Center Back                  21210 non-null  bool
 28  position_Center Defensive Midfield    21210 non-null  bool
 29  position_Center Forward               21210 non-null  bool
 30  position_Center Midfield              21210 non-null  bool
 31  position_Goalkeeper                   21210 non-null  bool
 32  position_Left Attacking Midfield      21210 non-null  bool
 33  position_Left Back                    21210 non-null  bool
 34  position_Left Center Back             21210 non-null  bool
 35  position_Left Center Forward          21210 non-null  bool
 36  position_Left Center Midfield         21210 non-null  bool
 37  position_Left Defensive Midfield      21210 non-null  bool
 38  position_Left Midfield                21210 non-null  bool
 39  position_Left Wing                    21210 non-null  bool
 40  position_Left Wing Back               21210 non-null  bool
 41  position_Right Attacking Midfield     21210 non-null  bool
 42  position_Right Back                   21210 non-null  bool
 43  position_Right Center Back            21210 non-null  bool
 44  position_Right Center Forward         21210 non-null  bool
 45  position_Right Center Midfield        21210 non-null  bool
 46  position_Right Defensive Midfield     21210 non-null  bool
 47  position_Right Midfield               21210 non-null  bool
 48  position_Right Wing                   21210 non-null  bool
 49  position_Right Wing Back              21210 non-null  bool
 50  position_Secondary Striker            21210 non-null  bool
 51  period_First Half                     21210 non-null  bool
 52  period_Second Half                    21210 non-null  bool
 53  period_Third Period                   21210 non-null  bool
 54  period_Fourth Period                  21210 non-null  bool
```

```
 55  period_Penalties                    21210 non-null  bool
 56  shot_classification_Bottom          21210 non-null  bool
 57  shot_classification_Bottom-Corner   21210 non-null  bool
 58  shot_classification_Middle          21210 non-null  bool
 59  shot_classification_Off-Target      21210 non-null  bool
 60  shot_classification_Top             21210 non-null  bool
 61  shot_classification_Top-Corner      21210 non-null  bool
dtypes: bool(56), float64(3), int64(3)
memory usage: 2.8 MB
```

In [25]:
```
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

# Define your features (X) and target (y)
X = cleaned_model_data.drop(columns=['Goal'])  # Drop the target column
y = cleaned_model_data['Goal']  # Target column (goal or not)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Check the shapes of the splits
print(X_train.shape, X_test.shape)

smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

```
(16968, 61) (4242, 61)
```

Logistic Regression

In [26]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler



# Feature scaling (Logistic Regression often benefits from scaling)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_resampled)  # Use resampled data
X_test_scaled = scaler.transform(X_test)

# Initialize and fit the Logistic Regression model
log_reg = LogisticRegression(random_state=42, max_iter=1000, class_weight='balan
log_reg.fit(X_train_scaled, y_train_resampled)  # Fit on resampled data

# Predict on the test set
y_pred = log_reg.predict(X_test_scaled)

# Evaluate the model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[3572  135]
 [ 295  240]]

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.96      0.94      3707
           1       0.64      0.45      0.53       535

    accuracy                           0.90      4242
   macro avg       0.78      0.71      0.74      4242
weighted avg       0.89      0.90      0.89      4242
```

RandomForestClassifier

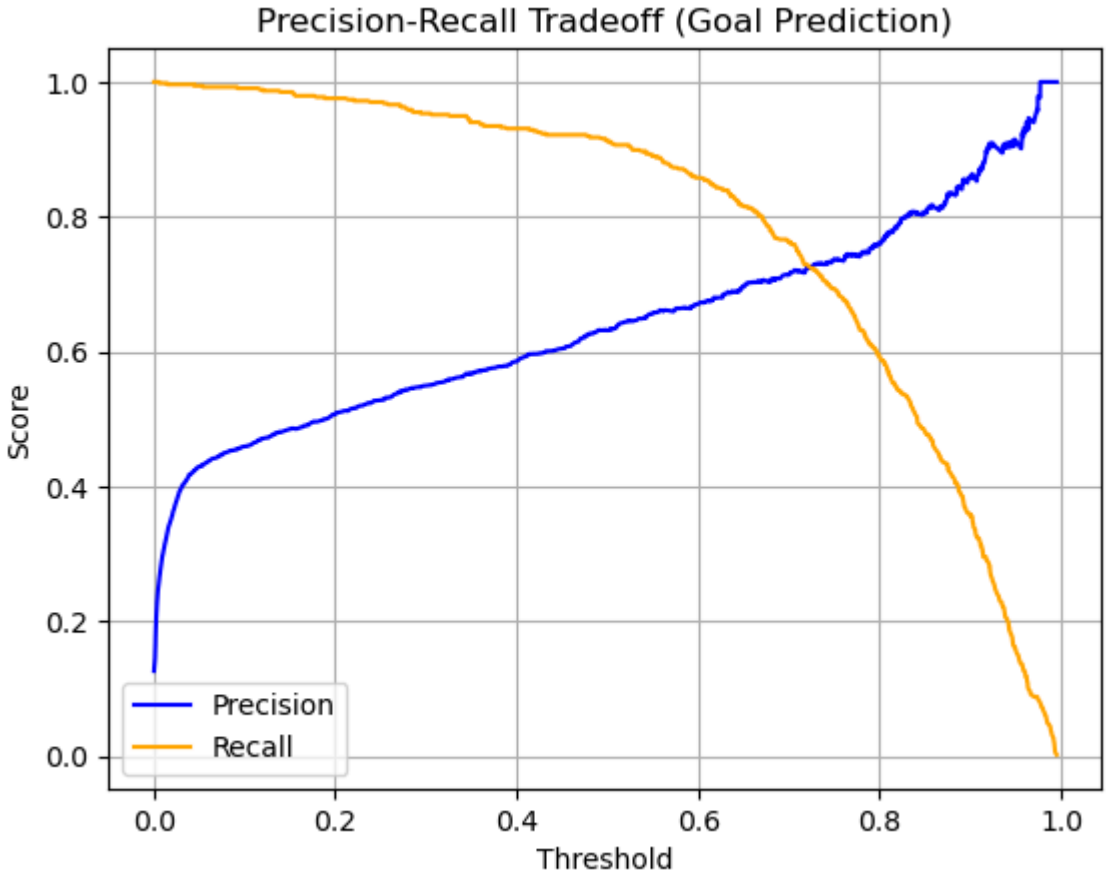
In [28]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_rep


# Initialize the Random Forest Classifier
model = RandomForestClassifier(random_state=42)

# Train the model on the resampled training data
model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print the evaluation metrics
print(f"Accuracy: {accuracy}")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)
```

```
Accuracy: 0.9125412541254125

Confusion Matrix:
[[3523  184]
 [ 187  348]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.95      0.95      3707
           1       0.65      0.65      0.65       535

    accuracy                           0.91      4242
   macro avg       0.80      0.80      0.80      4242
weighted avg       0.91      0.91      0.91      4242
```

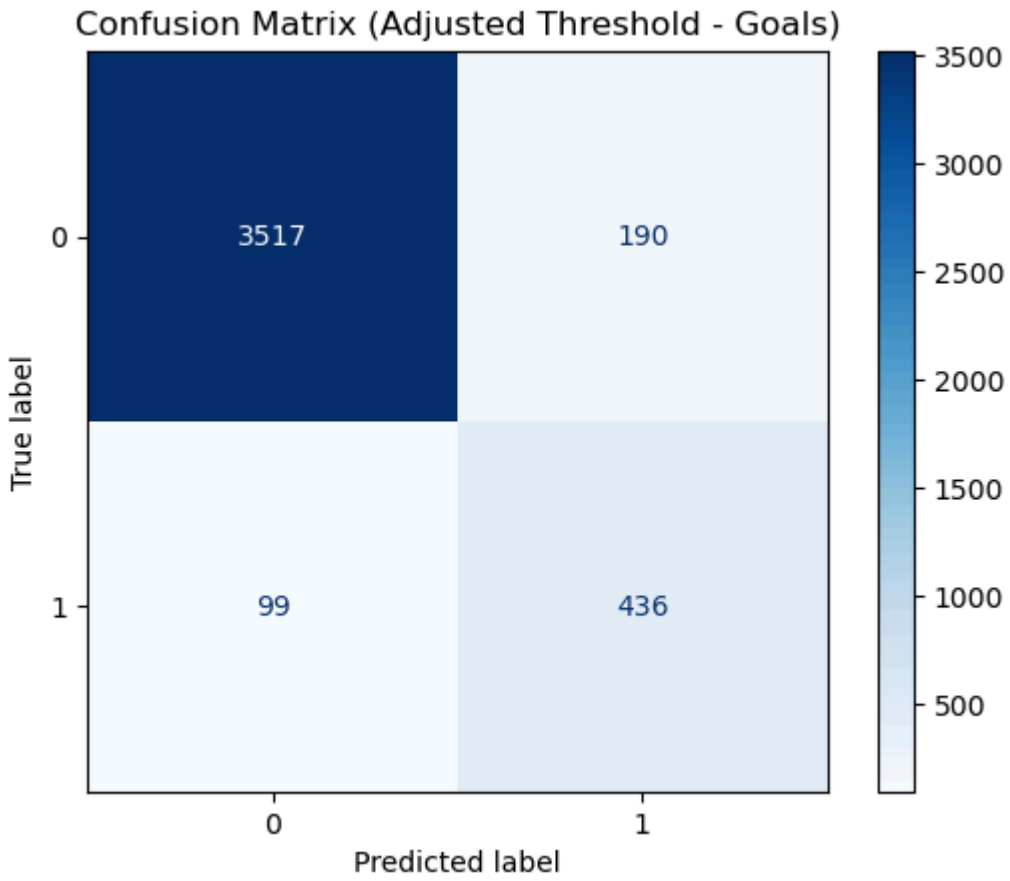
XGBoost

In [30]:
```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_rep
from xgboost import XGBClassifier
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve
from imblearn.over_sampling import SMOTE

# Apply SMOTE to oversample the minority class
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Define the model with class weights (scale_pos_weight)
xxgb_clf = XGBClassifier(
    objective='binary:logistic',
    eval_metric='logloss',
    max_depth=6,
    learning_rate=0.1,
    n_estimators=100,
    scale_pos_weight=len(y_train_resampled) / sum(y_train_resampled),  # Adding
    use_label_encoder=False
)

# Train the model
xxgb_clf.fit(X_train_resampled, y_train_resampled)

# Get probability scores
yy_prob_xgb = xxgb_clf.predict_proba(X_test)[:, 1]

# Compute Precision-Recall Curve
precision, recall, thresholds = precision_recall_curve(y_test, yy_prob_xgb)

# Plot the Precision-Recall curve
plt.plot(thresholds, precision[:-1], label="Precision", color="blue")
plt.plot(thresholds, recall[:-1], label="Recall", color="orange")
plt.xlabel("Threshold")
plt.ylabel("Score")
plt.legend()
plt.title("Precision-Recall Tradeoff (Goal Prediction)")
plt.grid()
plt.show()

# Set the threshold lower to maximize recall for class 1 (goals)
# For example, you can choose a threshold value where recall is higher but preci
selected_threshold = 0.65  # Adjust the threshold based on the curve

# Apply the new threshold
yy_pred_xgb_adjusted = (yy_prob_xgb >= selected_threshold).astype(int)

# Evaluate the model with the adjusted threshold
conf_matrix = confusion_matrix(y_test, yy_pred_xgb_adjusted)
print("Confusion Matrix (Adjusted Threshold - Prioritize Goals):\n", conf_matrix

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=[0, 1
```

```
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix (Adjusted Threshold - Goals)')
plt.show()

# Print classification report
print("Classification Report (Adjusted Threshold - Prioritize Recall for Goals):
```




```
Confusion Matrix (Adjusted Threshold - Prioritize Goals):
 [[3517  190]
 [  99  436]]
```




```
Classification Report (Adjusted Threshold - Prioritize Recall for Goals):
              precision    recall  f1-score   support

           0       0.97      0.95      0.96      3707
           1       0.70      0.81      0.75       535

    accuracy                           0.93      4242
   macro avg       0.83      0.88      0.86      4242
weighted avg       0.94      0.93      0.93      4242
```

Catboost

In [31]:
```python
from catboost import CatBoostClassifier


# Apply SMOTE to the training data to handle class imbalance
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Define and train the CatBoost model
model = CatBoostClassifier(
    iterations=500,
    learning_rate=0.1,
    depth=10,
    loss_function='Logloss',
    eval_metric='Accuracy',
    verbose=100
)

model.fit(X_train, y_train, eval_set=(X_test, y_test), early_stopping_rounds=10)
```

```
# Predict and evaluate
y_pred = model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
0:      learn: 0.8952735        test: 0.8903819 best: 0.8903819 (0)     total: 17
5ms     remaining: 1m 27s
Stopped by overfitting detector  (10 iterations wait)

bestTest = 0.9398868458
bestIteration = 55

Shrink model to first 56 iterations.
[[3609   98]
 [ 157  378]]
              precision    recall  f1-score   support

           0       0.96      0.97      0.97      3707
           1       0.79      0.71      0.75       535

    accuracy                           0.94      4242
   macro avg       0.88      0.84      0.86      4242
weighted avg       0.94      0.94      0.94      4242
```

In [ ]: