In [108]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score
import json
```

In [109]:

```
matches = pd.read_csv('matches_updated_mens_odi_upto_feb_2025.csv')
```

In [110]:

```
deliveries = pd.read_csv('deliveries_updated_mens_odi_upto_feb_2025.csv')
```

In [111]:

```
matches.head()
```

Out[111]:

	gender	season	toss_decision	team2	city	neutralvenue	player_of_match1	date3	umpir
0	male	2002/03	field	India	Napier	NaN	NaN	NaN	D Cov
1	male	2002/03	bat	New Zealand	Christchurch	NaN	NaN	NaN	D Cov
2	male	2002/03	field	New Zealand	Queenstown	NaN	NaN	NaN	D Cov
3	male	2002/03	bat	India	Wellington	NaN	NaN	NaN	Harp
4	male	2002/03	field	Australia	Sydney	NaN	NaN	NaN	S Tau

5 rows × 33 columns

In [116]:

```
deliveries.head()
```

Out[116]:

	matchId	inning	over_ball	over	ball	batting_team	bowling_team	batsman	non_striker	bowler	b
0	64814	1	0.1	0	1	New Zealand	India	SP Fleming	NJ Astle	J Srinath	
1	64814	1	0.2	0	2	New Zealand	India	NJ Astle	SP Fleming	J Srinath	
2	64814	1	0.3	0	3	New Zealand	India	NJ Astle	SP Fleming	J Srinath	
3	64814	1	0.4	0	4	New Zealand	India	NJ Astle	SP Fleming	J Srinath	
4	64814	1	0.5	0	5	New Zealand	India	SP Fleming	NJ Astle	J Srinath	

In [118]:

```
matches.columns
```

Out[118]:
```
Index(['gender', 'season', 'toss_decision', 'team2', 'city', 'neutralvenue',
       'player_of_match1', 'date3', 'umpire2', 'toss_winner', 'event', 'date1',
       'winner', 'team1', 'reserve_umpire1', 'venue', 'date2',
       'reserve_umpire2', 'winner_wickets', 'match_referee', 'balls_per_over',
       'method', 'match_number', 'umpire1', 'eliminator', 'outcome',
       'player_of_match', 'winner_runs', 'date', 'tv_umpire', 'reserve_umpire',
       'player_of_match2', 'matchId'],
      dtype='object')
```

In [120]:
```
deliveries.columns
```

Out[120]:
```
Index(['matchId', 'inning', 'over_ball', 'over', 'ball', 'batting_team',
       'bowling_team', 'batsman', 'non_striker', 'bowler', 'batsman_runs',
       'extras', 'isWide', 'isNoBall', 'Byes', 'LegByes', 'Penalty',
       'dismissal_kind', 'player_dismissed', 'date'],
      dtype='object')
```

In [122]:
```
# preprocess and clean the data

matches['match_date'] = pd.to_datetime(matches['date'])

matches = matches.dropna(subset=['winner']) # remove incomplete records
```

In [124]:
```
# target 1 if team 1 won, else 0

matches['result'] = (matches['winner'] == matches['team1']).astype(int)
```

In [126]:
```
# encode toss decision

matches['toss_decision_encode'] = matches['toss_decision'].map({'bat': 0, 'field': 1})
```

Team Rating Calculation

In [129]:
```
team_stats = matches[matches['winner'].notna()]
team_win_counts = team_stats['winner'].value_counts()
team_total_matches = pd.concat([team_stats['team1'], team_stats['team2']]).value_counts(
team_win_pct = (team_win_counts / team_total_matches).fillna(0)

matches['team1_rating'] = matches['team1'].map(team_win_pct)
matches['team2_rating'] = matches['team2'].map(team_win_pct)
```

In [131]:
```
# Batting: total runs scored by each batsman
batsman_runs = deliveries.groupby('batsman')['batsman_runs'].sum()

# Bowling: total wickets taken by each bowler
# player_dismissed is non-null for a dismissal, so we count those per bowler
wickets = deliveries[deliveries['player_dismissed'].notna()]
bowler_wickets = wickets.groupby('bowler')['player_dismissed'].count()
```

```
# Normalize ratings between 0 and 1 for comparability
# You can also scale differently depending on your model
batsman_rating = (batsman_runs - batsman_runs.min()) / (batsman_runs.max() - batsman_run
bowler_rating = (bowler_wickets - bowler_wickets.min()) / (bowler_wickets.max() - bowler

# Merge into a single player rating (can be weighted)
# Here, equal weight to batting and bowling (0.5 each) where applicable
all_players = pd.Index(batsman_rating.index.union(bowler_rating.index))
player_rating = pd.Series(index=all_players, dtype=float)

for player in all_players:
    bat_score = batsman_rating.get(player, 0)
    bowl_score = bowler_rating.get(player, 0)
    player_rating[player] = 0.5 * bat_score + 0.5 * bowl_score  # change weights as need

# Now map to a match-level dataset
# Example: map striker and bowler ratings into deliveries
deliveries['striker_rating'] = deliveries['batsman'].map(player_rating)
deliveries['bowler_rating'] = deliveries['bowler'].map(player_rating)
```

Home/Away Feature

In [133]:

```
def determine_home_team(row):
    if pd.notnull(row['venue']) and pd.notnull(row['team1']) and row['team1'].lower() in
        return row['team1']
    elif pd.notnull(row['venue']) and pd.notnull(row['team2']) and row['team2'].lower()
        return row['team2']
    else:
        return 'Neutral'

matches['home_team'] = matches.apply(determine_home_team, axis=1)
matches['is_home_team1'] = (matches['home_team'] == matches['team1']).astype(int)
matches['is_home_team2'] = (matches['home_team'] == matches['team2']).astype(int)
```

Toss Feature

In [136]:

```
matches['toss_winner_is_team1'] = (matches['toss_winner'] == matches['team1']).astype(in
matches['toss_decision_encoded'] = matches['toss_decision'].map({'bat': 0, 'field': 1})
```

Recent Form

In [140]:

```
def get_recent_form(team, match_date):
    recent_matches = matches[((matches['team1'] == team) | (matches['team2'] == team)) &
                             (matches['match_date'] < match_date)].sort_values('match_d
    return (recent_matches['winner'] == team).sum()

matches['match_date'] = pd.to_datetime(matches['match_date'])
matches = matches.sort_values('match_date')
```

```
matches['team1_recent_form'] = matches.apply(lambda x: get_recent_form(x['team1'], x['ma
matches['team2_recent_form'] = matches.apply(lambda x: get_recent_form(x['team2'], x['ma
```

Final Feature

In [142]:

```
# Drop matches without results
df = matches.dropna(subset=['winner'])

# Encode winner
df['label'] = np.where(df['winner'] == df['team1'], 0, 1)  # 0: team1 wins, 1: team2 win

features = ['team1_rating', 'team2_rating', 'is_home_team1', 'is_home_team2',
            'toss_winner_is_team1', 'toss_decision_encoded', 'team1_recent_form', 'team2
X = df[features]
y = df['label']
```

Train and Test Split

In [144]:

```
train = df[df['match_date'].dt.year < 2024]
test = df[df['match_date'].dt.year >= 2024]

X_train = train[features]
y_train = train['label']
X_test = test[features]
y_test = test['label']
```

Model Training

In [146]:

```
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {acc:.4f}")
print(f"F1 Score: {f1:.4f}")
```

```
Accuracy: 0.6044
F1 Score: 0.6538
```

In [152]:

```
def predict_win_prob(json_input):
    data = json.loads(json_input)

    team1 = data['team1']
    team2 = data['team2']
    toss_winner = data['toss_winner']
    toss_decision = data['toss_decision']
    venue = data['venue']
    match_date = pd.to_datetime(data['match_date'])

    row = {
        'team1_rating': team_win_pct.get(team1, 0.5),
        'team2_rating': team_win_pct.get(team2, 0.5),
        'is_home_team1': int(team1 in venue),
        'is_home_team2': int(team2 in venue),
        'toss_winner_is_team1': int(toss_winner == team1),
        'toss_decision_encoded': 0 if toss_decision == 'bat' else 1,
        'team1_recent_form': get_recent_form(team1, match_date),
        'team2_recent_form': get_recent_form(team2, match_date)
    }

    input_df = pd.DataFrame([row])
    probs = model.predict_proba(input_df)[0]
    return {team1: round(probs[0]*100, 2), team2: round(probs[1]*100, 2)}
```

In [154]:

```
json_input = json.dumps({
    "team1": "India",
    "team2": "England",
    "toss_winner": "India",
    "toss_decision": "bat",
    "venue": "Mumbai",
    "match_date": "2025-02-10"
})

predict_win_prob(json_input)
```

Out[154]:

```
{'India': 59.0, 'England': 41.0}
```

In [ ]: