Premliminary Analysis

Ken Kabeya & Aneesh Koppolu

Table of contents

Preliminary Analysis

Introduction

The Rugby World Cup (RWC) is the most prestigious tournament in men's international rugby, held every four years and featuring the top-performing nations from around the globe. Since its inception in 1987, all winning teams have come from the group of tier-one rugby nations—those with well-established professional structures and a history of competitive success. The World Cup's high stakes, unique pressure, and knockout format often produce different outcomes from regular international fixtures, yet the path to RWC success may still be written in a team's performance leading up to the tournament.

This project seeks to bridge our understanding of regular international performance with World Cup outcomes through a predictive modeling lens. Our guiding research question is:
> **Can regular fixture performance since 1999 reliably predict World Cup success for tier-one rugby nations?**

We expect that teams with consistently high win rates, dominant scoring margins, and strong rankings (e.g., Elo ratings) leading into a World Cup are more likely to progress further or win the tournament. Based on our model, and consistent with historical trends, we predict that **South Africa (RSA)**, **Ireland (IRE)**, or **New Zealand (NZ)** are the most likely candidates to win the 2027 Rugby World Cup.

Data

Data Source and Collection

The data for this project was sourced from an international rugby match dataset containing results from test matches played by men's national teams. The dataset includes **2,783 matches**, of which **1,230** were retained after filtering for: - Matches played **between 1999 and 2024**, inclusive - **Tier-one teams** only - Complete match records (with no missing scores or team names)

Cases and Variables

Each row in the dataset represents a single international rugby match and includes variables such as: - `date`: Match date - `home_team`, `away_team`: Competing teams - `home_score`, `away_score`: Points scored by each team - `competition`: Type of match (e.g., World Cup, Six Nations, etc.) - `neutral`: Boolean indicator for neutral venue - `world_cup`: Boolean indicator for whether the match is a World Cup fixture

Data Wrangling and Feature Engineering

To support our analysis, we created several new variables and tidied the dataset: - Extracted `year` from the match date for time-series analysis - Created binary indicators: `homeWin` and `awayWin` - Calculated `count` to aid in aggregating match totals - Grouped matches by team to calculate: - **Home and away win percentages - Average points scored and conceded (home and away) - World Cup vs. regular fixture performance** - Developed year-by-year win percentage timelines for each team

Variables for Modeling

We plan to include the following variables in our predictive models: - **Win Percentage (last 2 years before RWC)**: Measures recent performance - **Average Point Differential**: Offensive and defensive strength indicator - **Elo Rating** (if included or computed): Captures opponent-adjusted team strength - **Tournament Flag**: Helps compare regular matches to World Cup fixtures - **Home/Away Advantage**: Quantified through win rates and score differentials

These variables were chosen for their interpretability and demonstrated relevance in differentiating strong and weak tournament performers.

Methodology

Research Question

Our goal is to explore whether consistent performance in international fixtures since 1999 can predict a team's success at the Rugby World Cup (RWC). We focus on tier-one nations, and based on trends in both regular and tournament data, we aim to forecast the winner of the 2027 RWC. Our preliminary model predicts that **South Africa, Ireland, or New Zealand** are the most likely champions.

Data Overview and Cleaning

We used a dataset containing **2,783 international matches**, which was filtered to: - **Matches after 1998**, yielding **1,230 matches**. - **Tier-one teams only**, excluding matches with insufficient data or unclear team tier.

Cleaning Steps: - Extracted **year** from match dates for temporal analysis. - Created binary flags for **homeWin** and **awayWin**. - Dropped rows with missing values. - Created a **"count"** variable to aggregate matches played per team.

Feature Engineering and Wrangling

We created several new variables from existing match data: - **Home/Away Win Percentages**: Calculated by aggregating match results per team. - **Average Points Scored and Conceded**: For both World Cup and non-World Cup matches. - **World Cup Match Flag**: To compare regular vs. tournament performance. - **Yearly Win Percentage**: A normalized view of performance over time per team.

> Code implementation used pandas and NumPy, with seaborn and matplotlib for plotting.

Exclusions

We excluded: - All matches involving **non-tier-one nations**. - Matches with **missing scores** or unclear venues. - Non-competitive matches that could skew performance metrics (e.g., experimental squads).

Summary Statistics

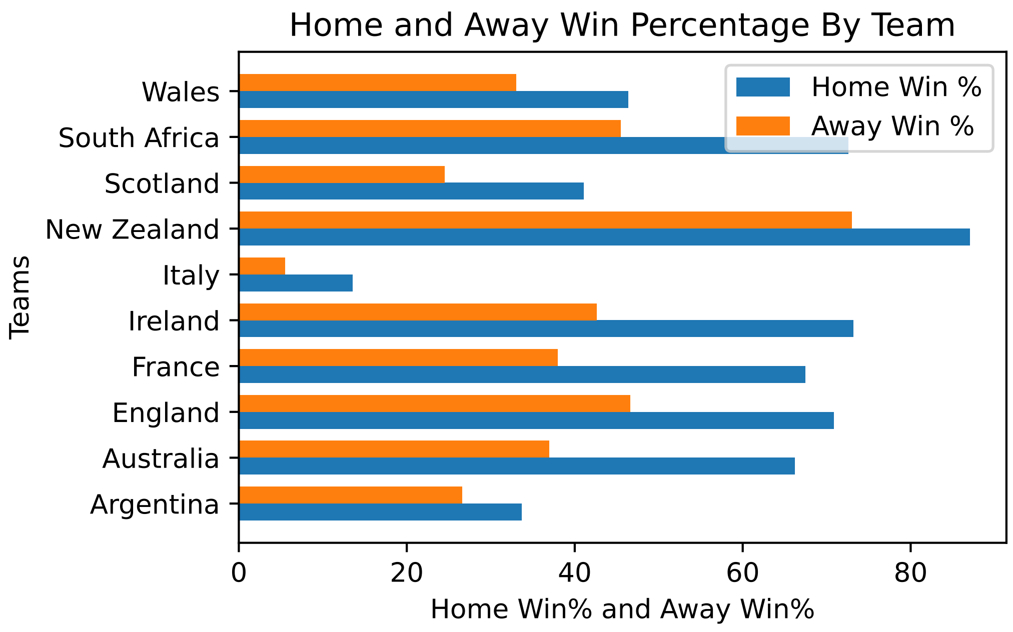
Variable	Mean	Std Dev	Min	Max
Home Score	25.07	13.15	0	101
Away Score	20.78	11.38	0	68
Home Win Rate	59.6%	—	0	1
Away Win Rate	38.4%	—	0	1
Year	2011.5	7.2	1999	2024

Observation:
On average, the **home team outperforms the away team by ~4.3 points**. The home win rate is significantly higher across all teams.

Key Visualizations
1. Home and Away Win Rates by Team
::: {.cell execution_count=1}
::: {.cell-output .cell-output-display execution_count=38}

::: {.cell execution_count=2} "' {.python .cell-code} #only keeping data after 1999 #will create a year col and then trim the data
rugbyDF['year'] = rugbyDF['date'].str[0:4].astype(int) rugbyDF = rugbyDF[rugbyDF.year > 1998] rugbyDF.dropna(inplace = True) rugbyDF.shape #new shape of dataframe: (1230,12) "'
::: {.cell-output .cell-output-display execution_count=39}
::: {.cell execution_count=3} "' {.python .cell-code} #creating variable to track if home or away team won #created weights for win, draw, and loss rugbyDF["homeWin"] = np.where(rugbyDF["home_score"] > rugbyDF["away_score"], 1,0) rugbyDF["awayWin"] = np.where(rugbyDF["home_score"] < rugbyDF["away_score"], 1,0)
#creating a count var to help with num of games played later on rugbyDF["count"] = 1 "' :::
::: {.cell execution_count=4} "' {.python .cell-code} for cols in rugbyDF.columns: print(cols + ":" + (str)(rugbyDF[cols].dtype))
#Since there are 2 numerical cols, i can get summary statistics on them #year variable will not be taken into account for summary statistics rugbyDF.describe() "'
::: {.cell-output .cell-output-stdout}
::: {.cell-output .cell-output-display execution_count=41}
"'{=html}
.dataframe tbody tr th { vertical-align: top; }
.dataframe thead th { text-align: right; }
::: :::
We can observe that the mean points scored by the home team was approximately 4.2 higher than the away team. All major quadrants have higher points for home_score than away_score, and the std is also greater.
We also notice that the home team won ~59.6%* of the games contested.
*This does not account for games that take place in neutral venues.
::: {.cell execution_count=5} "' {.python .cell-code} #create diff df by teams, then find the summary statistics for those teams. unq_hteams = rugbyDF['home_team'].unique() unq_ateams = rugbyDF['away_team'].unique() unq_teams = np.unique((unq_hteams, unq_ateams)) unq_teams
#shortens the df to only include awayteam, awaywin or hometeam, homewin rugbyDFH = rugbyDF[["home_team","homeWin","count"]] rugbyDFA = rugbyDF[["away_team","awayWin","count"]]
#groups the data by team groupedH = rugbyDFH.groupby('home_team').sum() groupedA = rugbyDFA.groupby('away_team').sum()
#renames the axis to the same thing groupedA = groupedA.rename_axis('team') groupedH = groupedH.rename_axis('team')
#find team win percentage by away and home groupedH["home_win%"] = groupedH["homeWin"]/groupedH["count"] *100 groupedA["away_win%"] = groupedA["awayWin"]/groupedA["count"]* 100

#renames count so that they dont clash when merging groupedH =
groupedH.rename(columns={'count': 'count_home'}) groupedA =
groupedA.rename(columns={'count': 'count_away'})
#merge the data frames teams_wins = pd.merge(groupedH, groupedA, on='team',
how='inner') teams_wins "'
::: {.cell-output .cell-output-display execution_count=42}
"'{=html}
.dataframe tbody tr th { vertical-align: top; }
.dataframe thead th { text-align: right; }
::: :::
This data frame represents the team wise performance breakdown from 1999-2024.
::: {.cell execution_count=6} "' {.python .cell-code} fig, ax = plt.subplots()
y = np.arange(len(unq_teams)) # Team positions width = 0.37 # Bar width
plt.barh(y - width/2, teams_wins["home_win%"], width, label='Home Win %') plt.barh(y +
width/2, teams_wins["away_win%"], width, label='Away Win %') plt.yticks(y, unq_teams)
# Add labels and title plt.ylabel('Teams') plt.xlabel('Home Win% and Away Win%')
plt.title('Home and Away Win Percentage By Team')
plt.legend() fig.tight_layout() plt.show() plt.close() "'
::: {.cell-output .cell-output-display}



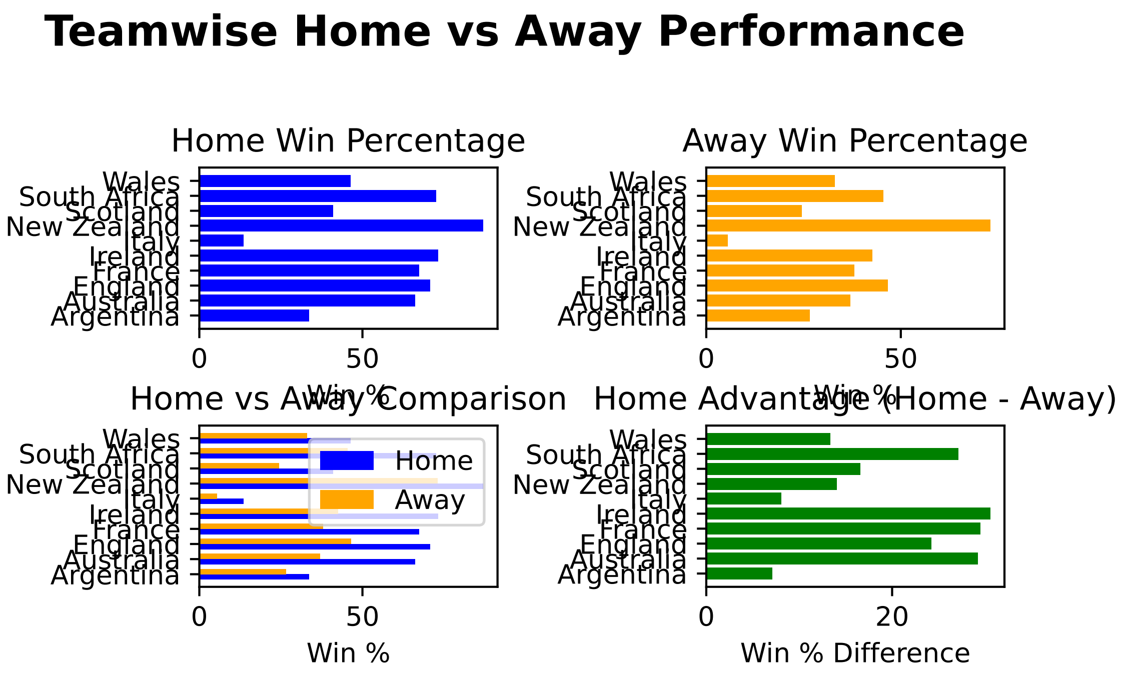
::: :::

- New Zealand dominates both at home and away. - All teams perform **better at home**, confirming home advantage. - Italy underperforms consistently.

2. Average Points Scored vs. Conceded - Teams like **New Zealand** and **South Africa** show high scores with low concession. - **Italy** and **Scotland** concede more than they score, especially in away games.

```
::: {.cell execution_count=7} “‘ {.python .cell-code} #a function to create a scoring dataset
def createscoringdataset(rugbyDF):
#shortens the df to only include awayteam, awaywin or hometeam, homewin
rugbyDFH_score = rugbyDF[[“home_team”,“home_score”,“away_score”,“count”]]
rugbyDFA_score = rugbyDF[[“away_team”,“away_score”,“home_score”,“count”]]
#renaming away_score and home_score to home_conceded and away_conceded
rugbyDFH_score = rugbyDFH_score.rename(columns={‘away_score’: ‘home_conceded’})
rugbyDFA_score = rugbyDFA_score.rename(columns={‘home_score’: ‘away_conceded’})
#groups the data by team groupedH_score =
rugbyDFH_score.groupby(‘home_team’).sum() groupedA_score =
rugbyDFA_score.groupby(‘away_team’).sum()
#renames the axis to the same thing groupedH_score =
groupedH_score.rename_axis(‘team’) groupedA_score =
groupedA_score.rename_axis(‘team’)
#merge the data frames teams_score = pd.merge(groupedH_score, groupedA_score,
on=‘team’, how=‘inner’)
#rename count_x and count_y to home_played and away_played teams_score =
teams_score.rename(columns={‘count_x’: ‘home_played’, ‘count_y’: ‘away_played’})
#find team win percentage by away and home teams_score[“home_score_avg”] =
teams_score[“home_score”]/teams_score[“home_played”] teams_score[“away_score_avg”]
= teams_score[“away_score”]/teams_score[“away_played”]
teams_score[“home_conceded_avg”] =
teams_score[“home_conceded”]/teams_score[“home_played”]
teams_score[“away_conceded_avg”] =
teams_score[“away_conceded”]/teams_score[“away_played”] return teams_score
teams_score = createscoringdataset(rugbyDF) “‘ :::
::: {.cell execution_count=8} “‘ {.python .cell-code} # Create a 2x2 grid of subplots fig, axs
= plt.subplots(2, 2) # 2 rows, 2 columns
# Add main title fig.suptitle(“Teamwise Home vs Away Performance”, fontsize=16, y=1.02,
# Adjust vertical position (1.0 = top of plot) fontweight=‘bold’)
# Plot 1: Home Win % axs[0, 0].barh(unq_teams, teams_wins[“home_win%”],
color=‘blue’) axs[0, 0].set_title(‘Home Win Percentage’) axs[0, 0].set_xlabel(‘Win %’)
# Plot 2: Away Win % axs[0, 1].barh(unq_teams, teams_wins[“away_win%”],
color=‘orange’) axs[0, 1].set_title(‘Away Win Percentage’) axs[0, 1].set_xlabel(‘Win %’)
# Plot 3: Home vs Away Comparison (side-by-side) y = range(len(unq_teams)) width =
0.35 axs[1, 0].barh([y - width/2 for y in y], teams_wins[“home_win%”], width,
label=‘Home’, color=‘blue’) axs[1, 0].barh([y + width/2 for y in y],
teams_wins[“away_win%”], width, label=‘Away’, color=‘orange’) axs[1, 0].set_title(‘Home
vs Away Comparison’) axs[1, 0].set_xlabel(‘Win %’) axs[1, 0].set_yticks(y) axs[1,
0].set_yticklabels(unq_teams) axs[1, 0].legend()
```

# Plot 4: Difference (Home - Away) axs[1, 1].barh(unq_teams, teams_wins[“home_win%”] - teams_wins[“away_win%”], color=‘green’) axs[1, 1].set_title(‘Home Advantage (Home - Away)’) axs[1, 1].set_xlabel(‘Win % Difference’)
plt.tight_layout(rect=[0.00, 0.0, 1, 1]) # Adjusts spacing between subplots
plt.subplots_adjust(wspace=0.7, hspace=0.6) plt.show() plt.close() “‘
::: {.cell-output .cell-output-display}



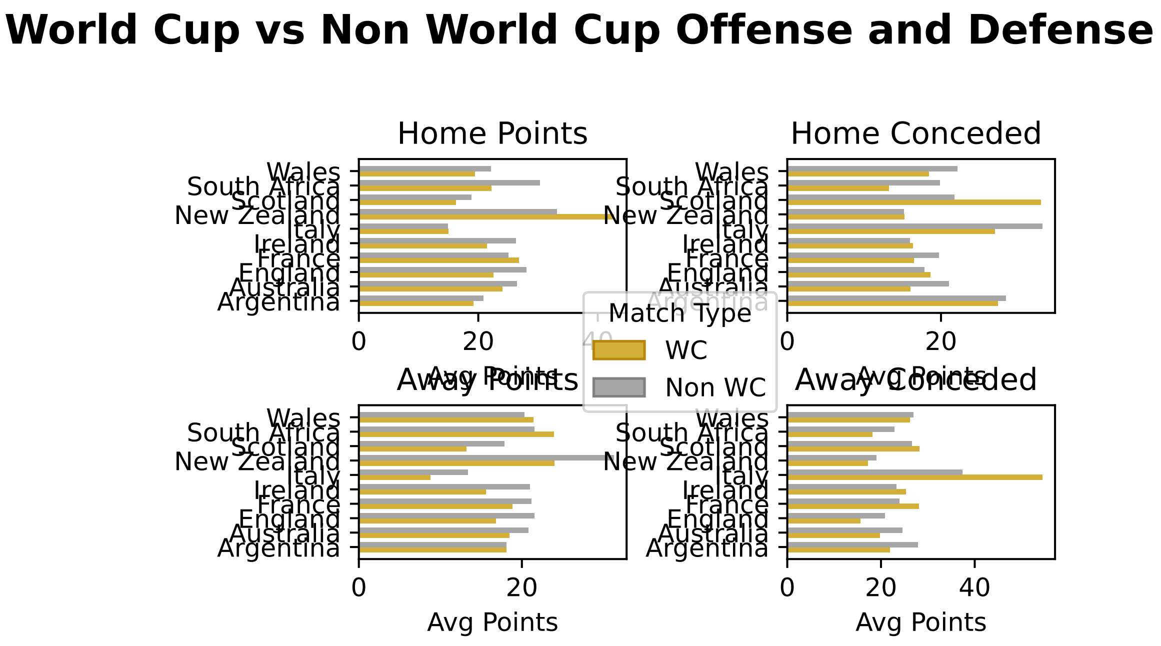
:::

:::

3. World Cup vs. Regular Match Comparison - **Ireland and South Africa** maintain strong performance in both contexts. - Some teams **drop significantly in performance during World Cups**, revealing tournament pressure or lack of depth.
::: {.cell execution_count=9} “‘ {.python .cell-code} #creates world cup scoring df worldCupDF = rugbyDF[rugbyDF.world_cup == True] nonWorldCup = rugbyDF[rugbyDF.world_cup == False] wc_scoring = createscoringdataset(worldCupDF) non_wc_scoring = createscoringdataset(nonWorldCup) “‘ :::
::: {.cell execution_count=10} “‘ {.python .cell-code} # Create a 2x2 grid of subplots fig, axs = plt.subplots(2, 2) # 2 rows, 2 columns
# Add main title fig.suptitle(“World Cup vs Non World Cup Offense and Defense”, fontsize=16, y=1.02, # Adjust vertical position (1.0 = top of plot) fontweight=‘bold’)

```
# Plot 1: World Cup vs Non World Cup home-score-avg y = range(len(unq_teams)) width
= 0.35 axs[0, 0].barh([y - width/2 for y in y], wc_scoring[‘home_score_avg’], width,
label=‘World Cup’, color=‘#D4AF37’) axs[0, 0].barh([y + width/2 for y in y],
non_wc_scoring[‘home_score_avg’], width, label=‘Non World Cup’, color=‘#A6A6A6’)
axs[0, 0].set_title(‘Home Points’) axs[0, 0].set_xlabel(‘Avg Points’) axs[0, 0].set_yticks(y)
axs[0, 0].set_yticklabels(unq_teams)
# Plot 2: World Cup vs Non World Cup away-score-avg axs[1, 0].barh([y - width/2 for y in
y], wc_scoring[‘away_score_avg’], width, label=‘World Cup’, color=‘#D4AF37’) axs[1,
0].barh([y + width/2 for y in y], non_wc_scoring[‘away_score_avg’], width, label=‘Non
World Cup’, color=‘#A6A6A6’) axs[1, 0].set_title(‘Away Points’) axs[1, 0].set_xlabel(‘Avg
Points’) axs[1, 0].set_yticks(y) axs[1, 0].set_yticklabels(unq_teams)
# Plot 3: World Cup vs Non World Cup home-conceded-avg axs[0, 1].barh([y - width/2 for y
in y], wc_scoring[‘home_conceded_avg’], width, label=‘World Cup’, color=‘#D4AF37’)
axs[0, 1].barh([y + width/2 for y in y], non_wc_scoring[‘home_conceded_avg’], width,
label=‘Non World Cup’, color=‘#A6A6A6’) axs[0, 1].set_title(‘Home Conceded’) axs[0,
1].set_xlabel(‘Avg Points’) axs[0, 1].set_yticks(y) axs[0, 1].set_yticklabels(unq_teams)
# Plot 4: World Cup vs Non World Cup away-conceded-avg axs[1, 1].barh([y - width/2 for y
in y], wc_scoring[‘away_conceded_avg’], width, label=‘World Cup’, color=‘#D4AF37’)
axs[1, 1].barh([y + width/2 for y in y], non_wc_scoring[‘away_conceded_avg’], width,
label=‘Non World Cup’, color=‘#A6A6A6’) axs[1, 1].set_title(‘Away Conceded’) axs[1,
1].set_xlabel(‘Avg Points’) axs[1, 1].set_yticks(y) axs[1, 1].set_yticklabels(unq_teams)
# Create invisible artist in center fig.patches.extend([plt.Rectangle((0.5, 0.5), 0.01, 0.01,
alpha=0, zorder=100, transform=fig.transFigure)])
# Create unified legend legend_elements = [ plt.Rectangle((0,0), 1, 1, fc=‘#D4AF37’,
ec=‘#B8860B’, lw=1, label=‘WC’), plt.Rectangle((0,0), 1, 1, fc=‘#A6A6A6’, ec=‘#808080’,
lw=1, label=‘Non WC’)]
# Place legend in absolute center legend = fig.legend(handles=legend_elements,
loc=‘center’, bbox_to_anchor=(0.60, 0.49), # Dead center bbox_transform=fig.transFigure,
frameon=True, title=‘Match Type’, borderaxespad=1)
plt.tight_layout(rect=[0.07, 0.0, 1, 1]) # Adjusts spacing between subplots
plt.subplots_adjust(wspace=0.6, hspace=0.6) plt.show() plt.close() “‘
```

::: {.cell-output .cell-output-display}

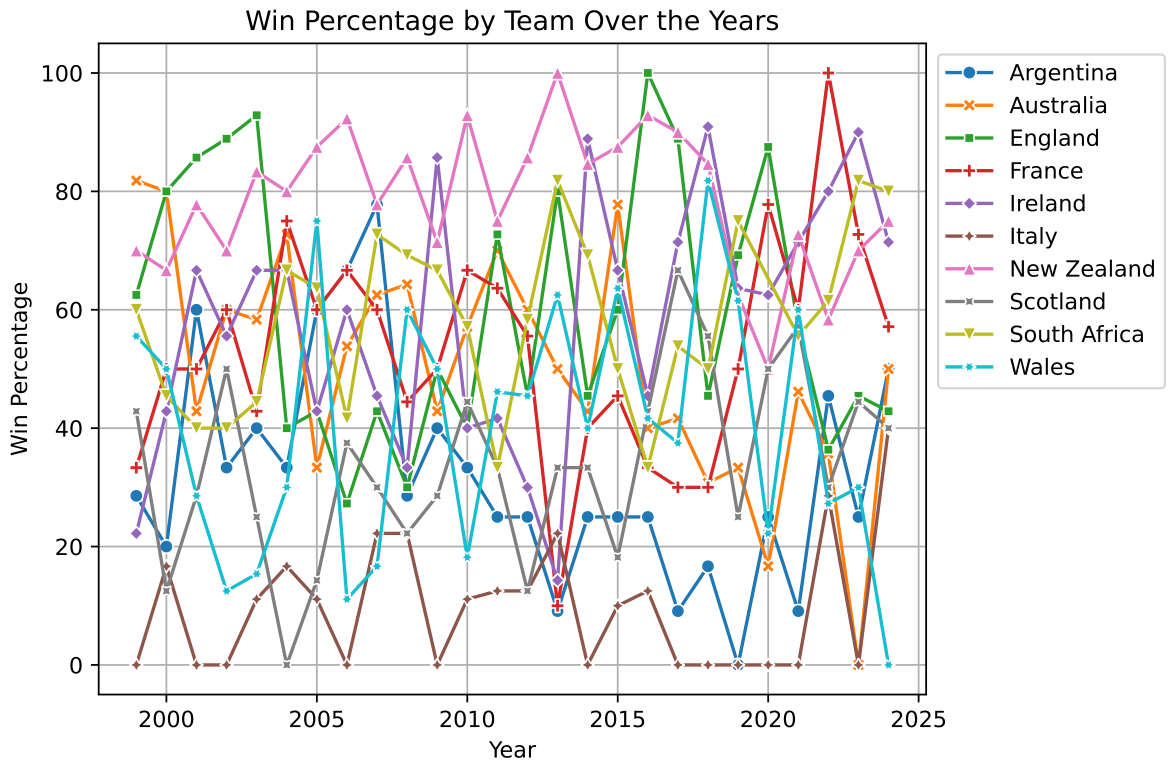


::: :::

4. Year-wise Win Percentage Trends - Shows **consistency and peaks** for contenders like **NZ, RSA, and IRE**. - Allows modeling team form trajectory over time.

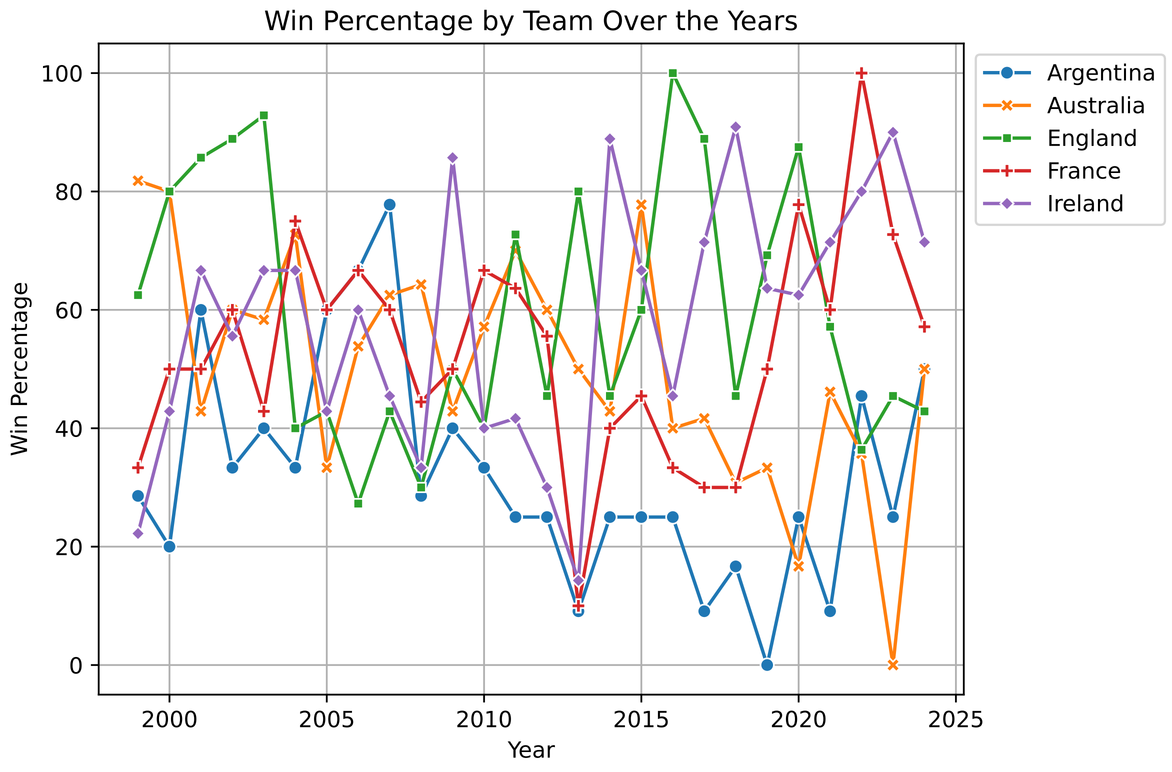
::: {.cell execution_count=11} "' {.python .cell-code} #only kees the year, awayWin, homeWin, and country year_wise = rugbyDF[["home_team","away_team","homeWin","awayWin","count","year"]] #drops the away in homeDf and home in awayDf home_year_wise = year_wise.drop(columns={"away_team","awayWin"}) away_year_wise = year_wise.drop(columns={"home_team","homeWin"}) #renames so that both df have the same col names for concatenation home_year_wise = home_year_wise.rename(columns={"home_team":"team","homeWin":"win"}) away_year_wise = away_year_wise.rename(columns={"away_team":"team","awayWin":"win"}) year_wise = pd.concat([home_year_wise,away_year_wise], axis=0).reset_index(drop=True) #groups the data by year and by team and then cerates a win% col for normalized results def getGroupedYW(year_wise): year_wise_grouped = year_wise.groupby(["team","year"]).sum() year_wise_grouped["win_%"] = year_wise_grouped["win"].div(year_wise_grouped["count"])*100 year_wise_grouped = year_wise_grouped.drop(columns={"win","count"}) return year_wise_grouped year_wise_grouped = getGroupedYW(year_wise) "' :::

::: {.cell execution_count=12} “‘ {.python .cell-code} def plotYearWiseData(year_wise_grouped): year_wise_grouped_reset = year_wise_grouped.reset_index()
plt.figure(figsize=(7.5, 5)) sns.lineplot(data=year_wise_grouped_reset, x=‘year’, y=‘win_%’, hue=‘team’, style=‘team’, markers=True, dashes=False)
plt.title(‘Win Percentage by Team Over the Years’) plt.xlabel(‘Year’) plt.ylabel(‘Win Percentage’) plt.legend(bbox_to_anchor=(1.001, 1), loc=‘upper left’) plt.grid(True)
plt.tight_layout() plt.show() plt.close()
plotYearWiseData(year_wise_grouped) “‘
::: {.cell-output .cell-output-display}



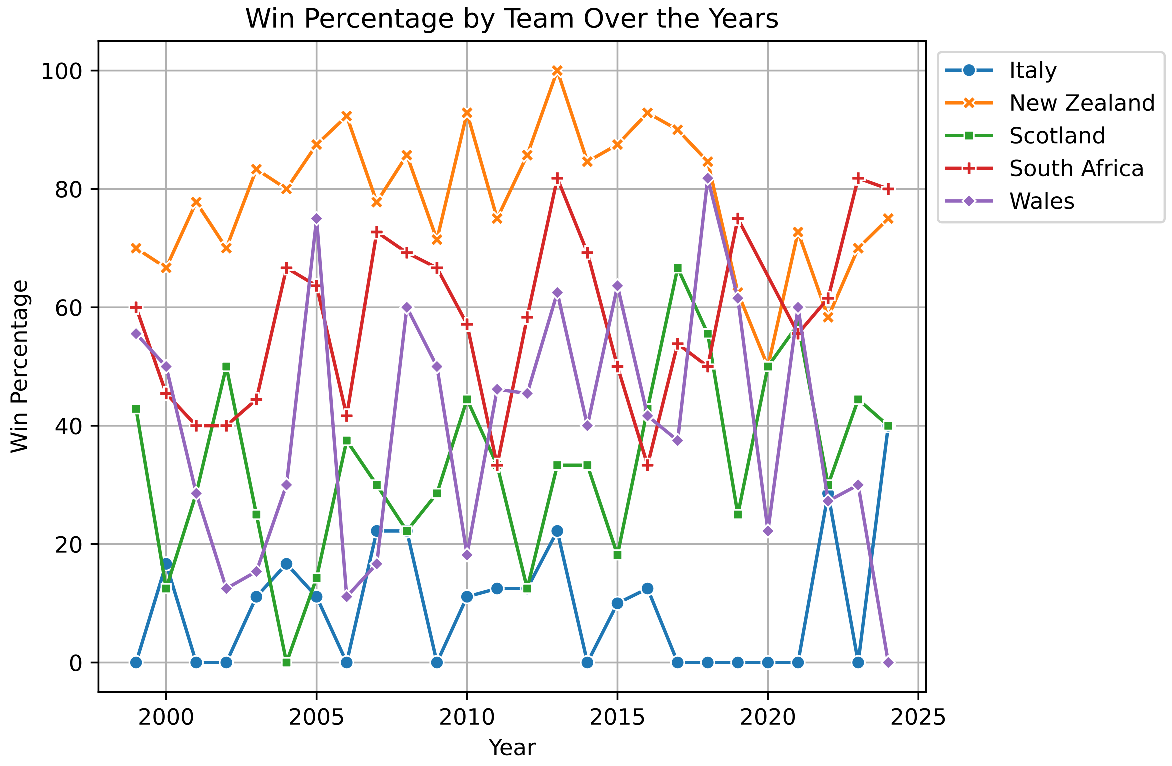
::: :::
Since this plot is chaotic, I will cut down on the number of countries plotted. 2 plots with 5 countries on each.
::: {.cell execution_count=13} “‘ {.python .cell-code} # Split teams into two halves half_idx = len(unq_teams) // 2 first_half_teams = unq_teams[:half_idx] second_half_teams = unq_teams[half_idx:]
# Create filtered DataFrames df_first_half = year_wise[year_wise[‘team’].isin(first_half_teams)] df_second_half = year_wise[year_wise[‘team’].isin(second_half_teams)]

#Checking if the teams do not overlap df_first_half[“team”].unique()
df_second_half[“team”].unique() “‘
::: {.cell-output .cell-output-display execution_count=50}
::: {.cell execution_count=14} “‘ {.python .cell-code} #uses the function to create the grouped data first_half_grouped = getGroupedYW(df_first_half)
#plots it plotYearWiseData(first_half_grouped) “‘
::: {.cell-output .cell-output-display}



::: :::
In this breakdown of ARG, AUS, ENG, FRA, and IRE we can see that Australia starts out really strong at the turn of the millennium and slowly starts to regress. 2023 saw them experience their worst year with no wins to show for. IRE and FRA have been the two teams that continue to improve on average. ENG and ARG are the two wildcard teams that can be either really good or really bad; they are wildly unpredictable.
::: {.cell execution_count=15} “‘ {.python .cell-code} #uses the function to create the grouped data second_half_grouped = getGroupedYW(df_second_half)
#plots it plotYearWiseData(second_half_grouped) “‘

::: {.cell-output .cell-output-display}



::: :::

Modeling Approach

Random Forest Classifier

- Used for **binary classification**: World Cup Winner vs. Non-Winner.
- Captures **non-linear patterns** and **interaction effects**.
- Feature importance helps identify key performance indicators.

Time-Series Analysis

- Year-wise win percentage trends inform about **form, slumps, and growth**.
- Useful for **pre-tournament prediction**, especially for 2027.

Why These Methods?

- **Logistic regression** works well with ordered categories.
- **Random forests** are robust to outliers and handle complex variable relationships.
- **Visualization-backed wrangling** ensured each model was driven by interpretable and meaningful variables.

XGBoost CLassifier with Monte Carlo Simulations

XGBoost (Extreme Gradient Boosting) is a highly efficient and scalable machine learning algorithm, especially for structured/tabular data. Its advantages include:

- High Predictive Accuracy: Often outperforms other algorithms (like logistic regression, random forests, or neural networks) on structured datasets.
- Handles Non-Linearity & Complex Relationships: Captures intricate patterns in data that simpler models might miss.
- Feature Importance: Provides insights into which variables most influence predictions.
- Robust to Overfitting: Includes regularization (L1/L2) and early stopping.
- Works Well with Imbalanced Data: Can handle classification tasks where one class is rare (e.g., fraud detection).

Monte Carlo (MC) simulations are used to model uncertainty and probabilistic outcomes by running thousands of random simulations. When combined with XGBoost, they help:

- Quantify Uncertainty: XGBoost gives a point prediction (e.g., probability of an event), but MC can simulate how reliable that prediction is under varying conditions.
- Risk Assessment: If inputs have randomness (e.g., stock prices, sensor noise), MC can propagate this uncertainty through the XGBoost model.
- Sensitivity Analysis: Test how small changes in input variables affect the output (e.g., “What if Feature X varies by ±10%?”).
- Decision-Making Under Uncertainty: Useful in finance (portfolio risk), healthcare (treatment outcomes), or engineering (system failures).

```
# Load the dataset
df = pd.read_csv('data/rugby.csv')
df['year'] = df['date'].str[0:4].astype(int)
df = df[df.year > 1998]
# Data preprocessing
# Filter to only international matches (remove club matches)
international_matches = df['competition'].str.contains('International|Championship|Nations|W
df = df.loc[international_matches].copy()

# Feature engineering
# Create target variable - winner of each match
conditions = [
    df['home_score'] > df['away_score'],
    df['home_score'] < df['away_score']
]
choices = [df['home_team'], df['away_team']]
df.loc[:, 'winner'] = np.select(conditions, choices, default='Draw')

# Remove draws for classification
df = df.loc[df['winner'] != 'Draw'].copy()

# Encode categorical variables
le = LabelEncoder()
df.loc[:, 'home_team_encoded'] = le.fit_transform(df['home_team'])
df.loc[:, 'away_team_encoded'] = le.transform(df['away_team'])
df.loc[:, 'winner_encoded'] = le.transform(df['winner'])

# Create features based on historical performance
def calculate_team_stats(df):
    team_stats = defaultdict(lambda: {'games': 0, 'wins': 0, 'points_for': 0, 'points_agains

    for _, row in df.iterrows():
        home_team = row['home_team']
        away_team = row['away_team']
        home_score = row['home_score']
        away_score = row['away_score']

        # Update home team stats
        team_stats[home_team]['games'] += 1
        team_stats[home_team]['points_for'] += home_score
        team_stats[home_team]['points_against'] += away_score
        team_stats[home_team]['wins'] += 1 if home_score > away_score else 0
        # Update away team stats
        team_stats[away_team]['games'] += 1
        team_stats[away_team]['points_for'] += away_score
        team_stats[away_team]['points_against'] += home_score
        team_stats[away_team]['wins'] += 1 if away_score > home_score else 0

    return team_stats

# Calculate rolling stats
team_stats = calculate_team_stats(df)

# Add features to dataframe
def add_features(df, team_stats):
    df = df.copy()
    df.loc[:, 'home_win_pct'] = df['home_team'].apply(
        lambda x: team_stats[x]['wins'] / team_stats[x]['games'] if team_stats[x]['games'] >
    df.loc[:, 'away_win_pct'] = df['away_team'].apply(
        lambda x: team_stats[x]['wins'] / team_stats[x]['games'] if team_stats[x]['games'] >
    df.loc[:, 'home_points_avg'] = df['home_team'].apply(
        lambda x: team_stats[x]['points_for'] / team_stats[x]['games'] if team_stats[x]['game
    df.loc[:, 'away_points_avg'] = df['away_team'].apply(
        lambda x: team_stats[x]['points_for'] / team_stats[x]['games'] if team_stats[x]['game
    df.loc[:, 'home_points_against_avg'] = df['home_team'].apply(
        lambda x: team_stats[x]['points_against'] / team_stats[x]['games'] if team_stats[x][
    df.loc[:, 'away_points_against_avg'] = df['away_team'].apply(
        lambda x: team_stats[x]['points_against'] / team_stats[x]['games'] if team_stats[x][

    # Add some interaction features
    df.loc[:, 'win_pct_diff'] = df['home_win_pct'] - df['away_win_pct']
    df.loc[:, 'points_diff'] = df['home_points_avg'] - df['away_points_avg']

    return df

df = add_features(df, team_stats)
```

```
# Split into features and target
feature_cols = ['home_team_encoded', 'away_team_encoded',
                'home_win_pct', 'away_win_pct',
                'home_points_avg', 'away_points_avg',
                'home_points_against_avg', 'away_points_against_avg',
                'win_pct_diff', 'points_diff']
```

```python
X = df.loc[:, feature_cols]
y = df.loc[:, 'winner_encoded']

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train XGBoost model
model = xgb.XGBClassifier(
    objective='multi:softprob',
    num_class=len(le.classes_),
    n_estimators=100,
    max_depth=5,
    learning_rate=0.1,
    random_state=42
)

model.fit(X_train, y_train)

# Evaluate model
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model accuracy: {accuracy:.2f}")
```

```
Model accuracy: 0.78
```

```python
# Monte Carlo simulation for World Cup prediction
def simulate_world_cup(teams, model, le, n_simulations=1000):
    # Get current top teams (from our dataset)
    top_teams = ['New Zealand', 'South Africa', 'Australia', 'England', 'France',
                 'Ireland', 'Wales', 'Scotland', 'Argentina', 'Italy']

    # Create a dictionary to store win counts
    win_counts = {team: 0 for team in top_teams}

    for _ in range(n_simulations):
        # Randomly select 2 teams to "play" in the final
        finalists = random.sample(top_teams, 2)
        team1, team2 = finalists

        # Create feature vector for this matchup
        try:
            team1_encoded = le.transform([team1])[0]
            team2_encoded = le.transform([team2])[0]
        except ValueError:
            # Skip if team not in our training data
            continue

        # Get team stats (simplified - in practice you'd want more recent stats)
        team1_stats = team_stats.get(team1, {'games': 1, 'wins': 0, 'points_for': 0, 'points_
        team2_stats = team_stats.get(team2, {'games': 1, 'wins': 0, 'points_for': 0, 'points_

        # Create feature row
        features = np.array([
            team1_encoded, team2_encoded,
            team1_stats['wins'] / max(1, team1_stats['games']),
            team2_stats['wins'] / max(1, team2_stats['games']),
            team1_stats['points_for'] / max(1, team1_stats['games']),
            team2_stats['points_for'] / max(1, team2_stats['games']),
            team1_stats['points_against'] / max(1, team1_stats['games']),
            team2_stats['points_against'] / max(1, team2_stats['games']),
            (team1_stats['wins'] / max(1, team1_stats['games'])) -
            (team2_stats['wins'] / max(1, team2_stats['games'])),
            (team1_stats['points_for'] / max(1, team1_stats['games'])) -
            (team2_stats['points_for'] / max(1, team2_stats['games']))
        ]).reshape(1, -1)

        # Predict probabilities
        try:
            probs = model.predict_proba(features)[0]

            # Sample winner based on probabilities
            winner_idx = np.random.choice(len(probs), p=probs)
            winner = le.inverse_transform([winner_idx])[0]

            if winner in win_counts:
                win_counts[winner] += 1
        except:
            continue

    # Convert counts to percentages
    total = max(1, sum(win_counts.values()))
    win_pct = {team: (count / total) * 100 for team, count in win_counts.items()}
```

```python
    return win_pct

# Run simulation
win_probabilities = simulate_world_cup(le.classes_, model, le, n_simulations=100000)

# Display results
print("\nPredicted 2027 Rugby World Cup Win Probabilities:")
for team, prob in sorted(win_probabilities.items(), key=lambda x: x[1], reverse=True):
    if prob > 0:
        print(f"{team}: {prob:.1f}%")
```

```
Predicted 2027 Rugby World Cup Win Probabilities:
New Zealand: 15.9%
England: 12.9%
France: 12.5%
South Africa: 11.5%
Australia: 11.5%
Ireland: 11.2%
Wales: 8.3%
Scotland: 8.0%
Argentina: 6.3%
Italy: 1.8%
```