Main_EDA_Modeling

August 12, 2025

0.1 Soccer_Performance_Score

0.2 1 | Data Import & Null Values Detections

[1]:
```python
# Simple Real Madrid Data Analysis
import pandas as pd
import numpy as np

# Load the data
df = pd.read_csv('/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main␣
↪Notebook/Data Folder/DataCombined/001_real_madrid_all_seasons_combined.csv')

# Basic info
print("Dataset Shape:", df.shape)
print("\nColumn Names:", list(df.columns))

# Find season column
season_col = None
for col in ['Season', 'season', 'SEASON']:
    if col in df.columns:
        season_col = col
        break

if season_col:
    print(f"\nUsing '{season_col}' column for season analysis")
    unique_seasons = sorted(df[season_col].unique())
    print(f"Available Seasons: {unique_seasons}")

    # Missing values analysis by season
    print("\nMissing Values Summary by Season")
    print("=" * 50)

    for season in unique_seasons:
        season_data = df[df[season_col] == season]

        missing_data = pd.DataFrame({
            'Column': season_data.columns,
            'Missing_Count': season_data.isnull().sum(),
```

```
            'Missing_Percentage': (season_data.isnull().sum() /␣
↪len(season_data)) * 100
        })

        # Only show columns with missing values
        missing_data = missing_data[missing_data['Missing_Count'] > 0]
        missing_data = missing_data.sort_values('Missing_Percentage',␣
↪ascending=False)

        print(f"\nSeason: {season} (Total rows: {len(season_data)})")
        if len(missing_data) > 0:
            print(missing_data.to_string(index=False, float_format='%.2f'))
        else:
            print("No missing values found")

else:
    print("\nNo season column found in dataset")
    # Fallback to overall missing values
    missing_data = pd.DataFrame({
        'Column': df.columns,
        'Missing_Count': df.isnull().sum(),
        'Missing_Percentage': (df.isnull().sum() / len(df)) * 100
    })

    missing_data = missing_data.sort_values('Missing_Percentage',␣
↪ascending=False)

    print("\nOverall Missing Values Summary")
    print("=" * 40)
    print(missing_data.to_string(index=False, float_format='%.2f'))
```

```
Dataset Shape: (7217, 77)

Column Names: ['Date', 'Competition', 'Opponent', 'Player', '#', 'Nation',
'Pos', 'Age', 'Min', ' Gls', ' Ast', ' PK', ' PKatt', ' Sh', ' SoT', ' CrdY', '
CrdR', ' Int', 'Match URL', 'Season', ' Touches', ' Tkl', ' Blocks', 'Expected
xG', 'Expected npxG', 'Expected xAG', 'SCA', 'GCA', 'Passes Cmp', 'Passes Att',
'Passes Cmp%', 'Passes PrgP', 'Carries Carries', 'Carries PrgC', 'Take-Ons Att',
'Take-Ons Succ', 'Tackles Tkl', 'Tackles TklW', 'Tackles Def 3rd', 'Tackles Mid
3rd', 'Tackles Att 3rd', 'Challenges Tkl', 'Challenges Att', 'Challenges Tkl%',
'Challenges Lost', 'Blocks Blocks', 'Blocks Sh', 'Blocks Pass', 'Int',
'Tkl+Int', 'Clr', 'Err', 'Total Cmp', 'Total Att', 'Total Cmp%', 'Total
TotDist', 'Total PrgDist', 'Short Cmp', 'Short Att', 'Short Cmp%', 'Medium Cmp',
'Medium Att', 'Medium Cmp%', 'Long Cmp', 'Long Att', 'Long Cmp%', 'Ast', 'xAG',
'xA', 'KP', '3-Jan', 'PPA', 'CrsPA', 'PrgP', 'SCA SCA', 'SCA GCA', ' 1/3']

Using 'Season' column for season analysis
```

```
Available Seasons: ['15_16', '16_17', '17_18', '18_19', '19_20', '20_21',
'21_22', '22_23', '23_24', '24_25']

Missing Values Summary by Season
==================================================

Season: 15_16 (Total rows: 703)
         Column  Missing_Count  Missing_Percentage
            Int            703              100.00
    Medium Cmp%            703              100.00
            Clr            703              100.00
            Err            703              100.00
      Total Cmp            703              100.00
      Total Att            703              100.00
     Total Cmp%            703              100.00
  Total TotDist            703              100.00
  Total PrgDist            703              100.00
      Short Cmp            703              100.00
      Short Att            703              100.00
     Short Cmp%            703              100.00
     Medium Cmp            703              100.00
     Medium Att            703              100.00
       Long Cmp            703              100.00
        Touches            703              100.00
       Long Att            703              100.00
      Long Cmp%            703              100.00
            Ast            703              100.00
            xAG            703              100.00
             xA            703              100.00
             KP            703              100.00
          3-Jan            703              100.00
            PPA            703              100.00
          CrsPA            703              100.00
           PrgP            703              100.00
        SCA SCA            703              100.00
        SCA GCA            703              100.00
        Tkl+Int            703              100.00
            1/3            703              100.00
    Blocks Pass            703              100.00
      Blocks Sh            703              100.00
            Tkl            703              100.00
         Blocks            703              100.00
    Expected xG            703              100.00
  Expected npxG            703              100.00
   Expected xAG            703              100.00
            SCA            703              100.00
            GCA            703              100.00
     Passes Cmp            703              100.00
```

Passes Att	703	100.00
Passes Cmp%	703	100.00
Passes PrgP	703	100.00
Carries Carries	703	100.00
Carries PrgC	703	100.00
Take-Ons Att	703	100.00
Take-Ons Succ	703	100.00
Tackles Tkl	703	100.00
Tackles TklW	703	100.00
Tackles Def 3rd	703	100.00
Tackles Mid 3rd	703	100.00
Tackles Att 3rd	703	100.00
Challenges Tkl	703	100.00
Challenges Att	703	100.00
Challenges Tkl%	703	100.00
Challenges Lost	703	100.00
Blocks Blocks	703	100.00
Min	2	0.28

Season: 16_17 (Total rows: 709)

Column	Missing_Count	Missing_Percentage
Int	709	100.00
Medium Cmp%	709	100.00
Clr	709	100.00
Err	709	100.00
Total Cmp	709	100.00
Total Att	709	100.00
Total Cmp%	709	100.00
Total TotDist	709	100.00
Total PrgDist	709	100.00
Short Cmp	709	100.00
Short Att	709	100.00
Short Cmp%	709	100.00
Medium Cmp	709	100.00
Medium Att	709	100.00
Long Cmp	709	100.00
Touches	709	100.00
Long Att	709	100.00
Long Cmp%	709	100.00
Ast	709	100.00
xAG	709	100.00
xA	709	100.00
KP	709	100.00
3-Jan	709	100.00
PPA	709	100.00
CrsPA	709	100.00
PrgP	709	100.00
SCA SCA	709	100.00

SCA GCA	709	100.00
Tkl+Int	709	100.00
1/3	709	100.00
Blocks Pass	709	100.00
Blocks Sh	709	100.00
Tkl	709	100.00
Blocks	709	100.00
Expected xG	709	100.00
Expected npxG	709	100.00
Expected xAG	709	100.00
SCA	709	100.00
GCA	709	100.00
Passes Cmp	709	100.00
Passes Att	709	100.00
Passes Cmp%	709	100.00
Passes PrgP	709	100.00
Carries Carries	709	100.00
Carries PrgC	709	100.00
Take-Ons Att	709	100.00
Take-Ons Succ	709	100.00
Tackles Tkl	709	100.00
Tackles TklW	709	100.00
Tackles Def 3rd	709	100.00
Tackles Mid 3rd	709	100.00
Tackles Att 3rd	709	100.00
Challenges Tkl	709	100.00
Challenges Att	709	100.00
Challenges Tkl%	709	100.00
Challenges Lost	709	100.00
Blocks Blocks	709	100.00
Min	1	0.14

Season: 17_18 (Total rows: 709)

Column	Missing_Count	Missing_Percentage
1/3	709	100.00
SCA GCA	709	100.00
SCA SCA	709	100.00
Challenges Tkl%	298	42.03
Long Cmp%	78	11.00
Medium Cmp%	25	3.53
Short Cmp%	18	2.54
Passes Cmp%	7	0.99
Total Cmp%	7	0.99
Blocks Pass	5	0.71
Total TotDist	5	0.71
Blocks Sh	5	0.71
Blocks Blocks	5	0.71
Tkl+Int	5	0.71

Clr	5	0.71
Challenges Lost	5	0.71
Err	5	0.71
Total Cmp	5	0.71
Challenges Att	5	0.71
Total Att	5	0.71
Int	5	0.71
Short Att	5	0.71
Total PrgDist	5	0.71
Short Cmp	5	0.71
Tackles Att 3rd	5	0.71
Medium Cmp	5	0.71
Medium Att	5	0.71
Long Cmp	5	0.71
Long Att	5	0.71
Ast	5	0.71
xAG	5	0.71
xA	5	0.71
KP	5	0.71
3-Jan	5	0.71
PPA	5	0.71
CrsPA	5	0.71
PrgP	5	0.71
Challenges Tkl	5	0.71
Date	5	0.71
Competition	5	0.71
Match URL	5	0.71
CrdR	5	0.71
CrdY	5	0.71
SoT	5	0.71
Sh	5	0.71
PKatt	5	0.71
PK	5	0.71
Ast	5	0.71
Gls	5	0.71
Min	5	0.71
Age	5	0.71
Pos	5	0.71
Nation	5	0.71
#	5	0.71
Player	5	0.71
Opponent	5	0.71
Int	5	0.71
Touches	5	0.71
Tackles Def 3rd	5	0.71
Tkl	5	0.71
Tackles TklW	5	0.71
Tackles Tkl	5	0.71

Take-Ons Succ	5	0.71
Take-Ons Att	5	0.71
Carries PrgC	5	0.71
Carries Carries	5	0.71
Passes PrgP	5	0.71
Passes Att	5	0.71
Passes Cmp	5	0.71
GCA	5	0.71
SCA	5	0.71
Expected xAG	5	0.71
Expected npxG	5	0.71
Expected xG	5	0.71
Blocks	5	0.71
Tackles Mid 3rd	5	0.71

Season: 18_19 (Total rows: 642)

Column	Missing_Count	Missing_Percentage
SCA	642	100.00
GCA	642	100.00
1/3	642	100.00
Challenges Tkl%	281	43.77
Long Cmp%	66	10.28
Medium Cmp%	11	1.71
Short Cmp%	9	1.40

Season: 19_20 (Total rows: 651)

Column	Missing_Count	Missing_Percentage
SCA	651	100.00
GCA	651	100.00
1/3	651	100.00
Challenges Tkl%	277	42.55
Long Cmp%	84	12.90
Medium Cmp%	20	3.07
Short Cmp%	11	1.69
Passes Cmp%	4	0.61
Total Cmp%	4	0.61

Season: 20_21 (Total rows: 723)

Column	Missing_Count	Missing_Percentage
SCA	723	100.00
GCA	723	100.00
1/3	723	100.00
Challenges Tkl%	331	45.78
Long Cmp%	111	15.35
Medium Cmp%	23	3.18
Short Cmp%	6	0.83
Passes Cmp%	3	0.41
Total Cmp%	3	0.41

Season: 21_22 (Total rows: 778)

Column	Missing_Count	Missing_Percentage
SCA	778	100.00
GCA	778	100.00
1/3	778	100.00
Challenges Tkl%	386	49.61
Long Cmp%	126	16.20
Medium Cmp%	38	4.88
Short Cmp%	22	2.83
Passes Cmp%	7	0.90
Total Cmp%	7	0.90

Season: 22_23 (Total rows: 752)

Column	Missing_Count	Missing_Percentage
SCA	752	100.00
GCA	752	100.00
1/3	752	100.00
Challenges Tkl%	367	48.80
Long Cmp%	126	16.76
Medium Cmp%	17	2.26
Short Cmp%	16	2.13
Passes Cmp%	4	0.53
Total Cmp%	4	0.53

Season: 23_24 (Total rows: 774)

Column	Missing_Count	Missing_Percentage
3-Jan	774	100.00
SCA SCA	774	100.00
SCA GCA	774	100.00
Challenges Tkl%	400	51.68
Long Cmp%	151	19.51
Medium Cmp%	48	6.20
Short Cmp%	21	2.71
Passes Cmp%	8	1.03
Total Cmp%	8	1.03

Season: 24_25 (Total rows: 776)

Column	Missing_Count	Missing_Percentage
3-Jan	776	100.00
SCA SCA	776	100.00
SCA GCA	776	100.00
Challenges Tkl%	394	50.77
Long Cmp%	128	16.49
Medium Cmp%	35	4.51
Short Cmp%	23	2.96
Passes Cmp%	12	1.55
Total Cmp%	12	1.55

0.2.1 1.1 | Clean Data

[2]:
```
# Simple Real Madrid Data Analysis
import pandas as pd
import numpy as np

# Basic info
print("Dataset Shape:", df.shape)
print("\nColumn Names:", list(df.columns))

# Find season column
season_col = None
for col in ['Season', 'season', 'SEASON']:
    if col in df.columns:
        season_col = col
        break

if season_col:
    print(f"\nUsing '{season_col}' column for season analysis")
    unique_seasons = sorted(df[season_col].unique())
    print(f"Available Seasons: {unique_seasons}")

    # Remove specific seasons (2015-16, 2016-17, and 3-jan)
    seasons_to_remove = ['2015-16', '2016-17', '15-16', '16-17', '3-jan']  #␣
↪Multiple formats

    # Check which format exists in the data
    actual_seasons_to_remove = []
    for season_format in seasons_to_remove:
        if season_format in df[season_col].values:
            actual_seasons_to_remove.append(season_format)

    print(f"\nRemoving seasons: {actual_seasons_to_remove}")

    df_filtered = df[~df[season_col].isin(actual_seasons_to_remove)].copy()
    print(f"Rows before season removal: {len(df)}")
    print(f"Rows after season removal: {len(df_filtered)}")

    # Remove columns with 100% missing values across all remaining data
    columns_before = df_filtered.shape[1]
    missing_percentages = (df_filtered.isnull().sum() / len(df_filtered)) * 100
    columns_100_missing = missing_percentages[missing_percentages == 100].index.
↪tolist()

    # Also remove specific problematic columns
    problematic_columns = ['3-Jan', 'Long Cmp%']
    for col in problematic_columns:
```

```python
        if col in df_filtered.columns and col not in columns_100_missing:
            columns_100_missing.append(col)

    if columns_100_missing:
        print(f"\nRemoving columns with 100% missing values and problematic columns: {columns_100_missing}")
        df_filtered = df_filtered.drop(columns=columns_100_missing)

    # Remove columns with >50% missing values
    missing_percentages = (df_filtered.isnull().sum() / len(df_filtered)) * 100
    columns_high_missing = missing_percentages[missing_percentages > 50].index.tolist()

    if columns_high_missing:
        print(f"Removing columns with >50% missing values: {columns_high_missing}")
        df_filtered = df_filtered.drop(columns=columns_high_missing)

    print(f"Columns before cleaning: {columns_before}")
    print(f"Columns after cleaning: {df_filtered.shape[1]}")

    # Final missing values analysis by remaining seasons
    print("\nFinal Missing Values Summary by Season")
    print("=" * 50)

    remaining_seasons = sorted(df_filtered[season_col].unique())

    for season in remaining_seasons:
        season_data = df_filtered[df_filtered[season_col] == season]

        missing_data = pd.DataFrame({
            'Column': season_data.columns,
            'Missing_Count': season_data.isnull().sum(),
            'Missing_Percentage': (season_data.isnull().sum() / len(season_data)) * 100
        })

        # Only show columns with missing values
        missing_data = missing_data[missing_data['Missing_Count'] > 0]
        missing_data = missing_data.sort_values('Missing_Percentage', ascending=False)

        print(f"\nSeason: {season} (Total rows: {len(season_data)})")
        if len(missing_data) > 0:
            print(missing_data.to_string(index=False, float_format='%.2f'))
        else:
            print("No missing values found")
```

```
    # Update the main dataframe
    df = df_filtered
    print(f"\nFinal dataset shape: {df.shape}")

else:
    print("\nNo season column found in dataset")
```

```
Dataset Shape: (7217, 77)

Column Names: ['Date', 'Competition', 'Opponent', 'Player', '#', 'Nation',
'Pos', 'Age', 'Min', ' Gls', ' Ast', ' PK', ' PKatt', ' Sh', ' SoT', ' CrdY', '
CrdR', ' Int', 'Match URL', 'Season', ' Touches', ' Tkl', ' Blocks', 'Expected
xG', 'Expected npxG', 'Expected xAG', 'SCA', 'GCA', 'Passes Cmp', 'Passes Att',
'Passes Cmp%', 'Passes PrgP', 'Carries Carries', 'Carries PrgC', 'Take-Ons Att',
'Take-Ons Succ', 'Tackles Tkl', 'Tackles TklW', 'Tackles Def 3rd', 'Tackles Mid
3rd', 'Tackles Att 3rd', 'Challenges Tkl', 'Challenges Att', 'Challenges Tkl%',
'Challenges Lost', 'Blocks Blocks', 'Blocks Sh', 'Blocks Pass', 'Int',
'Tkl+Int', 'Clr', 'Err', 'Total Cmp', 'Total Att', 'Total Cmp%', 'Total
TotDist', 'Total PrgDist', 'Short Cmp', 'Short Att', 'Short Cmp%', 'Medium Cmp',
'Medium Att', 'Medium Cmp%', 'Long Cmp', 'Long Att', 'Long Cmp%', 'Ast', 'xAG',
'xA', 'KP', '3-Jan', 'PPA', 'CrsPA', 'PrgP', 'SCA SCA', 'SCA GCA', ' 1/3']

Using 'Season' column for season analysis
Available Seasons: ['15_16', '16_17', '17_18', '18_19', '19_20', '20_21',
'21_22', '22_23', '23_24', '24_25']

Removing seasons: []
Rows before season removal: 7217
Rows after season removal: 7217

Removing columns with 100% missing values and problematic columns: ['3-Jan',
'Long Cmp%']
Removing columns with >50% missing values: ['SCA', 'GCA', 'Challenges Tkl%',
'SCA SCA', 'SCA GCA', ' 1/3']
Columns before cleaning: 77
Columns after cleaning: 69

Final Missing Values Summary by Season
==================================================

Season: 15_16 (Total rows: 703)
         Column  Missing_Count  Missing_Percentage
    Blocks Pass            703              100.00
     Short Cmp%            703              100.00
        Tkl+Int            703              100.00
            Clr            703              100.00
            Err            703              100.00
```

Total Cmp	703	100.00
Total Att	703	100.00
Total Cmp%	703	100.00
Total TotDist	703	100.00
Total PrgDist	703	100.00
Short Cmp	703	100.00
Short Att	703	100.00
Medium Cmp	703	100.00
Touches	703	100.00
Medium Att	703	100.00
Medium Cmp%	703	100.00
Long Cmp	703	100.00
Long Att	703	100.00
Ast	703	100.00
xAG	703	100.00
xA	703	100.00
KP	703	100.00
PPA	703	100.00
CrsPA	703	100.00
Int	703	100.00
PrgP	703	100.00
Blocks Sh	703	100.00
Blocks Blocks	703	100.00
Tkl	703	100.00
Blocks	703	100.00
Expected xG	703	100.00
Expected npxG	703	100.00
Expected xAG	703	100.00
Passes Cmp	703	100.00
Passes Att	703	100.00
Passes Cmp%	703	100.00
Passes PrgP	703	100.00
Carries Carries	703	100.00
Carries PrgC	703	100.00
Take-Ons Att	703	100.00
Take-Ons Succ	703	100.00
Tackles Tkl	703	100.00
Tackles TklW	703	100.00
Tackles Def 3rd	703	100.00
Tackles Mid 3rd	703	100.00
Tackles Att 3rd	703	100.00
Challenges Tkl	703	100.00
Challenges Att	703	100.00
Challenges Lost	703	100.00
Min	2	0.28

Season: 16_17 (Total rows: 709)

Column	Missing_Count	Missing_Percentage

Blocks Pass	709	100.00
Short Cmp%	709	100.00
Tkl+Int	709	100.00
Clr	709	100.00
Err	709	100.00
Total Cmp	709	100.00
Total Att	709	100.00
Total Cmp%	709	100.00
Total TotDist	709	100.00
Total PrgDist	709	100.00
Short Cmp	709	100.00
Short Att	709	100.00
Medium Cmp	709	100.00
Touches	709	100.00
Medium Att	709	100.00
Medium Cmp%	709	100.00
Long Cmp	709	100.00
Long Att	709	100.00
Ast	709	100.00
xAG	709	100.00
xA	709	100.00
KP	709	100.00
PPA	709	100.00
CrsPA	709	100.00
Int	709	100.00
PrgP	709	100.00
Blocks Sh	709	100.00
Blocks Blocks	709	100.00
Tkl	709	100.00
Blocks	709	100.00
Expected xG	709	100.00
Expected npxG	709	100.00
Expected xAG	709	100.00
Passes Cmp	709	100.00
Passes Att	709	100.00
Passes Cmp%	709	100.00
Passes PrgP	709	100.00
Carries Carries	709	100.00
Carries PrgC	709	100.00
Take-Ons Att	709	100.00
Take-Ons Succ	709	100.00
Tackles Tkl	709	100.00
Tackles TklW	709	100.00
Tackles Def 3rd	709	100.00
Tackles Mid 3rd	709	100.00
Tackles Att 3rd	709	100.00
Challenges Tkl	709	100.00
Challenges Att	709	100.00

Challenges Lost	709	100.00
Min	1	0.14

Season: 17_18 (Total rows: 709)

Column	Missing_Count	Missing_Percentage
Medium Cmp%	25	3.53
Short Cmp%	18	2.54
Passes Cmp%	7	0.99
Total Cmp%	7	0.99
Date	5	0.71
Total Cmp	5	0.71
Err	5	0.71
Clr	5	0.71
Tkl+Int	5	0.71
Int	5	0.71
Blocks Pass	5	0.71
Blocks Sh	5	0.71
Blocks Blocks	5	0.71
Challenges Lost	5	0.71
Challenges Att	5	0.71
Challenges Tkl	5	0.71
Tackles Att 3rd	5	0.71
Total Att	5	0.71
Total TotDist	5	0.71
Tackles Def 3rd	5	0.71
Total PrgDist	5	0.71
Short Cmp	5	0.71
Short Att	5	0.71
Medium Cmp	5	0.71
Medium Att	5	0.71
Long Cmp	5	0.71
Long Att	5	0.71
Ast	5	0.71
xAG	5	0.71
xA	5	0.71
KP	5	0.71
PPA	5	0.71
CrsPA	5	0.71
Tackles Mid 3rd	5	0.71
Tackles TklW	5	0.71
Competition	5	0.71
CrdR	5	0.71
Opponent	5	0.71
Player	5	0.71
#	5	0.71
Nation	5	0.71
Pos	5	0.71
Age	5	0.71

Min	5	0.71
Gls	5	0.71
Ast	5	0.71
PK	5	0.71
PKatt	5	0.71
Sh	5	0.71
SoT	5	0.71
CrdY	5	0.71
Int	5	0.71
Tackles Tkl	5	0.71
Match URL	5	0.71
Touches	5	0.71
Tkl	5	0.71
Blocks	5	0.71
Expected xG	5	0.71
Expected npxG	5	0.71
Expected xAG	5	0.71
Passes Cmp	5	0.71
Passes Att	5	0.71
Passes PrgP	5	0.71
Carries Carries	5	0.71
Carries PrgC	5	0.71
Take-Ons Att	5	0.71
Take-Ons Succ	5	0.71
PrgP	5	0.71

Season: 18_19 (Total rows: 642)

Column	Missing_Count	Missing_Percentage
Medium Cmp%	11	1.71
Short Cmp%	9	1.40

Season: 19_20 (Total rows: 651)

Column	Missing_Count	Missing_Percentage
Medium Cmp%	20	3.07
Short Cmp%	11	1.69
Passes Cmp%	4	0.61
Total Cmp%	4	0.61

Season: 20_21 (Total rows: 723)

Column	Missing_Count	Missing_Percentage
Medium Cmp%	23	3.18
Short Cmp%	6	0.83
Passes Cmp%	3	0.41
Total Cmp%	3	0.41

Season: 21_22 (Total rows: 778)

Column	Missing_Count	Missing_Percentage
Medium Cmp%	38	4.88

```
 Short Cmp%             22                2.83
Passes Cmp%              7                0.90
 Total Cmp%              7                0.90

Season: 22_23 (Total rows: 752)
     Column  Missing_Count  Missing_Percentage
Medium Cmp%             17                2.26
 Short Cmp%             16                2.13
Passes Cmp%              4                0.53
 Total Cmp%              4                0.53

Season: 23_24 (Total rows: 774)
     Column  Missing_Count  Missing_Percentage
Medium Cmp%             48                6.20
 Short Cmp%             21                2.71
Passes Cmp%              8                1.03
 Total Cmp%              8                1.03

Season: 24_25 (Total rows: 776)
     Column  Missing_Count  Missing_Percentage
Medium Cmp%             35                4.51
 Short Cmp%             23                2.96
Passes Cmp%             12                1.55
 Total Cmp%             12                1.55

Final dataset shape: (7217, 69)
```

0.2.2 1.2 Remove Players not playing more than 200 minutes in the entire data set

[3]:
```python
# Simple Real Madrid Data Analysis
import pandas as pd
import numpy as np

# Basic info
print("Dataset Shape:", df.shape)
print("\nColumn Names:", list(df.columns))

# Find season column
season_col = None
for col in ['Season', 'season', 'SEASON']:
    if col in df.columns:
        season_col = col
        break

if season_col:
    print(f"\nUsing '{season_col}' column for season analysis")
    unique_seasons = sorted(df[season_col].unique())
```

```
    print(f"Available Seasons: {unique_seasons}")

    # Remove specific seasons (2015-16, 2016-17, and 3-jan)
    seasons_to_remove = ['2015-16', '2016-17', '15-16', '16-17', '3-jan']  #␣
↪Multiple formats

    # Check which format exists in the data
    actual_seasons_to_remove = []
    for season_format in seasons_to_remove:
        if season_format in df[season_col].values:
            actual_seasons_to_remove.append(season_format)

    print(f"\nRemoving seasons: {actual_seasons_to_remove}")

    df_filtered = df[~df[season_col].isin(actual_seasons_to_remove)].copy()
    print(f"Rows before season removal: {len(df)}")
    print(f"Rows after season removal: {len(df_filtered)}")

    # Remove columns with 100% missing values across all remaining data
    columns_before = df_filtered.shape[1]
    missing_percentages = (df_filtered.isnull().sum() / len(df_filtered)) * 100
    columns_100_missing = missing_percentages[missing_percentages == 100].index.
↪tolist()

    # Also remove specific problematic columns
    problematic_columns = ['3-Jan', 'Long Cmp%']
    for col in problematic_columns:
        if col in df_filtered.columns and col not in columns_100_missing:
            columns_100_missing.append(col)

    if columns_100_missing:
        print(f"\nRemoving columns with 100% missing values and problematic␣
↪columns: {columns_100_missing}")
        df_filtered = df_filtered.drop(columns=columns_100_missing)

    # Remove columns with >50% missing values
    missing_percentages = (df_filtered.isnull().sum() / len(df_filtered)) * 100
    columns_high_missing = missing_percentages[missing_percentages > 50].index.
↪tolist()

    if columns_high_missing:
        print(f"Removing columns with >50% missing values:␣
↪{columns_high_missing}")
        df_filtered = df_filtered.drop(columns=columns_high_missing)

    # Remove rows that have too many missing values (e.g., >30% of remaining␣
↪columns)
```

```
    threshold_missing_cols = 0.30  # 30% threshold
    num_cols = df_filtered.shape[1]
    max_missing_allowed = int(num_cols * threshold_missing_cols)

    print(f"\nRemoving rows with more than {max_missing_allowed} missing values␣
↪out of {num_cols} columns...")

    # Count missing values per row
    missing_per_row = df_filtered.isnull().sum(axis=1)
    rows_to_keep = missing_per_row <= max_missing_allowed

    print(f"Rows before removing high-missing rows: {len(df_filtered)}")
    df_filtered = df_filtered[rows_to_keep].copy()
    print(f"Rows after removing high-missing rows: {len(df_filtered)}")

    # Fill remaining missing values with player averages for numeric columns
    print(f"\nFilling remaining missing values with player averages...")
    numeric_cols = df_filtered.select_dtypes(include=[np.number]).columns

    for col in numeric_cols:
        if df_filtered[col].isnull().any():
            # Fill with player's own average first
            player_averages = df_filtered.groupby('Player')[col].
↪transform('mean')
            df_filtered[col] = df_filtered[col].fillna(player_averages)

            # If still missing (new players), fill with overall column mean
            df_filtered[col] = df_filtered[col].fillna(df_filtered[col].mean())

            print(f"Filled missing values in {col}")

    print(f"Columns before cleaning: {columns_before}")
    print(f"Columns after cleaning: {df_filtered.shape[1]}")

    # Final missing values analysis by remaining seasons
    print("\nFinal Missing Values Summary by Season")
    print("=" * 50)

    remaining_seasons = sorted(df_filtered[season_col].unique())

    for season in remaining_seasons:
        season_data = df_filtered[df_filtered[season_col] == season]

        missing_data = pd.DataFrame({
            'Column': season_data.columns,
            'Missing_Count': season_data.isnull().sum(),
```

```
            'Missing_Percentage': (season_data.isnull().sum() /␣
↪len(season_data)) * 100
        })

        # Only show columns with missing values
        missing_data = missing_data[missing_data['Missing_Count'] > 0]
        missing_data = missing_data.sort_values('Missing_Percentage',␣
↪ascending=False)

        print(f"\nSeason: {season} (Total rows: {len(season_data)})")
        if len(missing_data) > 0:
            print(missing_data.to_string(index=False, float_format='%.2f'))
        else:
            print("No missing values found")

    # Update the main dataframe
    df = df_filtered

    # Remove players who didn't play (0 minutes)
    if 'Minutes' in df.columns:
        print(f"\nRemoving players who didn't play...")
        print(f"Rows before removing non-players: {len(df)}")
        df = df[df['Minutes'] > 0].copy()
        print(f"Rows after removing non-players: {len(df)}")

        # Remove players with less than 200 total minutes across entire dataset
        print(f"\nRemoving players with less than 200 total minutes...")
        player_minutes = df.groupby('Player')['Minutes'].sum()
        players_to_keep = player_minutes[player_minutes >= 200].index

        print(f"Players before filtering: {df['Player'].nunique()}")
        print(f"Players with >=200 minutes: {len(players_to_keep)}")

        df = df[df['Player'].isin(players_to_keep)].copy()
        print(f"Rows after removing low-minute players: {len(df)}")

    elif 'Min' in df.columns:
        print(f"\nRemoving players who didn't play...")
        print(f"Rows before removing non-players: {len(df)}")
        df = df[df['Min'] > 0].copy()
        print(f"Rows after removing non-players: {len(df)}")

        # Remove players with less than 200 total minutes across entire dataset
        print(f"\nRemoving players with less than 200 total minutes...")
        player_minutes = df.groupby('Player')['Min'].sum()
        players_to_keep = player_minutes[player_minutes >= 200].index
```

```
        print(f"Players before filtering: {df['Player'].nunique()}")
        print(f"Players with >=200 minutes: {len(players_to_keep)}")

        df = df[df['Player'].isin(players_to_keep)].copy()
        print(f"Rows after removing low-minute players: {len(df)}")

    else:
        print("\nNo 'Minutes' or 'Min' column found - cannot filter non-playing␣
↪players")

    print(f"\nFinal dataset shape: {df.shape}")

else:
    print("\nNo season column found in dataset")
```

```
Dataset Shape: (7217, 69)

Column Names: ['Date', 'Competition', 'Opponent', 'Player', '#', 'Nation',
'Pos', 'Age', 'Min', ' Gls', ' Ast', ' PK', ' PKatt', ' Sh', ' SoT', ' CrdY', '
CrdR', ' Int', 'Match URL', 'Season', ' Touches', ' Tkl', ' Blocks', 'Expected
xG', 'Expected npxG', 'Expected xAG', 'Passes Cmp', 'Passes Att', 'Passes Cmp%',
'Passes PrgP', 'Carries Carries', 'Carries PrgC', 'Take-Ons Att', 'Take-Ons
Succ', 'Tackles Tkl', 'Tackles TklW', 'Tackles Def 3rd', 'Tackles Mid 3rd',
'Tackles Att 3rd', 'Challenges Tkl', 'Challenges Att', 'Challenges Lost',
'Blocks Blocks', 'Blocks Sh', 'Blocks Pass', 'Int', 'Tkl+Int', 'Clr', 'Err',
'Total Cmp', 'Total Att', 'Total Cmp%', 'Total TotDist', 'Total PrgDist', 'Short
Cmp', 'Short Att', 'Short Cmp%', 'Medium Cmp', 'Medium Att', 'Medium Cmp%',
'Long Cmp', 'Long Att', 'Ast', 'xAG', 'xA', 'KP', 'PPA', 'CrsPA', 'PrgP']

Using 'Season' column for season analysis
Available Seasons: ['15_16', '16_17', '17_18', '18_19', '19_20', '20_21',
'21_22', '22_23', '23_24', '24_25']

Removing seasons: []
Rows before season removal: 7217
Rows after season removal: 7217

Removing rows with more than 20 missing values out of 69 columns…
Rows before removing high-missing rows: 7217
Rows after removing high-missing rows: 5800

Filling remaining missing values with player averages…
Filled missing values in Passes Cmp%
Filled missing values in Total Cmp%
Filled missing values in Short Cmp%
Filled missing values in Medium Cmp%
Columns before cleaning: 69
Columns after cleaning: 69
```

```
Final Missing Values Summary by Season
==================================================

Season: 17_18 (Total rows: 704)
No missing values found

Season: 18_19 (Total rows: 642)
No missing values found

Season: 19_20 (Total rows: 651)
No missing values found

Season: 20_21 (Total rows: 723)
No missing values found

Season: 21_22 (Total rows: 778)
No missing values found

Season: 22_23 (Total rows: 752)
No missing values found

Season: 23_24 (Total rows: 774)
No missing values found

Season: 24_25 (Total rows: 776)
No missing values found

Removing players who didn't play…
Rows before removing non-players: 5800
Rows after removing non-players: 5800

Removing players with less than 200 total minutes…
Players before filtering: 74
Players with >=200 minutes: 54
Rows after removing low-minute players: 5737

Final dataset shape: (5737, 69)
```

0.2.3 1.3 Final Data clean after filling with average for those cases with less than 5% of missing values

[4]:
```
# Simple Real Madrid Data Analysis
import pandas as pd
import numpy as np

# Basic info
```

```python
print("Dataset Shape:", df.shape)
print("\nColumn Names:", list(df.columns))

# Find season column
season_col = None
for col in ['Season', 'season', 'SEASON']:
    if col in df.columns:
        season_col = col
        break

if season_col:
    print(f"\nUsing '{season_col}' column for season analysis")
    unique_seasons = sorted(df[season_col].unique())
    print(f"Available Seasons: {unique_seasons}")

    # Remove specific seasons (2015-16, 2016-17, and 3-jan)
    seasons_to_remove = ['2015-16', '2016-17', '15-16', '16-17', '3-jan']  #␣
↪Multiple formats

    # Check which format exists in the data
    actual_seasons_to_remove = []
    for season_format in seasons_to_remove:
        if season_format in df[season_col].values:
            actual_seasons_to_remove.append(season_format)

    print(f"\nRemoving seasons: {actual_seasons_to_remove}")

    df_filtered = df[~df[season_col].isin(actual_seasons_to_remove)].copy()
    print(f"Rows before season removal: {len(df)}")
    print(f"Rows after season removal: {len(df_filtered)}")

    # Remove columns with 100% missing values across all remaining data
    columns_before = df_filtered.shape[1]
    missing_percentages = (df_filtered.isnull().sum() / len(df_filtered)) * 100
    columns_100_missing = missing_percentages[missing_percentages == 100].index.
↪tolist()

    # Also remove specific problematic columns
    problematic_columns = ['3-Jan', 'Long Cmp%']
    for col in problematic_columns:
        if col in df_filtered.columns and col not in columns_100_missing:
            columns_100_missing.append(col)

    if columns_100_missing:
        print(f"\nRemoving columns with 100% missing values and problematic␣
↪columns: {columns_100_missing}")
        df_filtered = df_filtered.drop(columns=columns_100_missing)
```

```
    # Remove columns with >50% missing values
    missing_percentages = (df_filtered.isnull().sum() / len(df_filtered)) * 100
    columns_high_missing = missing_percentages[missing_percentages > 50].index.
↪tolist()

    if columns_high_missing:
        print(f"Removing columns with >50% missing values:␣
↪{columns_high_missing}")
        df_filtered = df_filtered.drop(columns=columns_high_missing)

    # Remove rows that have too many missing values (e.g., >30% of remaining␣
↪columns)
    threshold_missing_cols = 0.30  # 30% threshold
    num_cols = df_filtered.shape[1]
    max_missing_allowed = int(num_cols * threshold_missing_cols)

    print(f"\nRemoving rows with more than {max_missing_allowed} missing values␣
↪out of {num_cols} columns...")

    # Count missing values per row
    missing_per_row = df_filtered.isnull().sum(axis=1)
    rows_to_keep = missing_per_row <= max_missing_allowed

    print(f"Rows before removing high-missing rows: {len(df_filtered)}")
    df_filtered = df_filtered[rows_to_keep].copy()
    print(f"Rows after removing high-missing rows: {len(df_filtered)}")

    # Fill remaining missing values with player averages for numeric columns
    print(f"\nFilling remaining missing values with player averages...")
    numeric_cols = df_filtered.select_dtypes(include=[np.number]).columns

    for col in numeric_cols:
        if df_filtered[col].isnull().any():
            # Fill with player's own average first
            player_averages = df_filtered.groupby('Player')[col].
↪transform('mean')
            df_filtered[col] = df_filtered[col].fillna(player_averages)

            # If still missing (new players), fill with overall column mean
            df_filtered[col] = df_filtered[col].fillna(df_filtered[col].mean())

            print(f"Filled missing values in {col}")

    print(f"Columns before cleaning: {columns_before}")
    print(f"Columns after cleaning: {df_filtered.shape[1]}")
```

```
    # Final missing values analysis by remaining seasons
    print("\nFinal Missing Values Summary by Season (After Cleaning)")
    print("=" * 60)

    remaining_seasons = sorted(df_filtered[season_col].unique())

    for season in remaining_seasons:
        season_data = df_filtered[df_filtered[season_col] == season]

        missing_data = pd.DataFrame({
            'Column': season_data.columns,
            'Missing_Count': season_data.isnull().sum(),
            'Missing_Percentage': (season_data.isnull().sum() /␣
↪len(season_data)) * 100
        })

        # Show ALL columns now (not just missing ones) to see the clean data
        missing_data = missing_data.sort_values('Missing_Percentage',␣
↪ascending=False)

        print(f"\nSeason: {season} (Total rows: {len(season_data)})")
        print(missing_data.to_string(index=False, float_format='%.2f'))

    # Overall summary after cleaning
    print("\n" + "="*60)
    print("OVERALL MISSING VALUES SUMMARY (After All Cleaning)")
    print("="*60)

    final_missing = pd.DataFrame({
        'Column': df_filtered.columns,
        'Missing_Count': df_filtered.isnull().sum(),
        'Missing_Percentage': (df_filtered.isnull().sum() / len(df_filtered)) *␣
↪100
    })

    final_missing = final_missing.sort_values('Missing_Percentage',␣
↪ascending=False)
    print(final_missing.to_string(index=False, float_format='%.2f'))

    # Update the main dataframe
    df = df_filtered

    # Remove players who didn't play (0 minutes)
    if 'Minutes' in df.columns:
        print(f"\nRemoving players who didn't play...")
        print(f"Rows before removing non-players: {len(df)}")
        df = df[df['Minutes'] > 0].copy()
```

```python
        print(f"Rows after removing non-players: {len(df)}")

        # Remove players with less than 200 total minutes across entire dataset
        print(f"\nRemoving players with less than 200 total minutes...")
        player_minutes = df.groupby('Player')['Minutes'].sum()
        players_to_keep = player_minutes[player_minutes >= 200].index

        print(f"Players before filtering: {df['Player'].nunique()}")
        print(f"Players with >=200 minutes: {len(players_to_keep)}")

        df = df[df['Player'].isin(players_to_keep)].copy()
        print(f"Rows after removing low-minute players: {len(df)}")

    elif 'Min' in df.columns:
        print(f"\nRemoving players who didn't play...")
        print(f"Rows before removing non-players: {len(df)}")
        df = df[df['Min'] > 0].copy()
        print(f"Rows after removing non-players: {len(df)}")

        # Remove players with less than 200 total minutes across entire dataset
        print(f"\nRemoving players with less than 200 total minutes...")
        player_minutes = df.groupby('Player')['Min'].sum()
        players_to_keep = player_minutes[player_minutes >= 200].index

        print(f"Players before filtering: {df['Player'].nunique()}")
        print(f"Players with >=200 minutes: {len(players_to_keep)}")

        df = df[df['Player'].isin(players_to_keep)].copy()
        print(f"Rows after removing low-minute players: {len(df)}")

    else:
        print("\nNo 'Minutes' or 'Min' column found - cannot filter non-playing players")

    print(f"\nFinal dataset shape: {df.shape}")

else:
    print("\nNo season column found in dataset")
```

```
Dataset Shape: (5737, 69)

Column Names: ['Date', 'Competition', 'Opponent', 'Player', '#', 'Nation',
'Pos', 'Age', 'Min', ' Gls', ' Ast', ' PK', ' PKatt', ' Sh', ' SoT', ' CrdY', '
CrdR', ' Int', 'Match URL', 'Season', ' Touches', ' Tkl', ' Blocks', 'Expected
xG', 'Expected npxG', 'Expected xAG', 'Passes Cmp', 'Passes Att', 'Passes Cmp%',
'Passes PrgP', 'Carries Carries', 'Carries PrgC', 'Take-Ons Att', 'Take-Ons
Succ', 'Tackles Tkl', 'Tackles TklW', 'Tackles Def 3rd', 'Tackles Mid 3rd',
'Tackles Att 3rd', 'Challenges Tkl', 'Challenges Att', 'Challenges Lost',
```

```
'Blocks Blocks', 'Blocks Sh', 'Blocks Pass', 'Int', 'Tkl+Int', 'Clr', 'Err',
'Total Cmp', 'Total Att', 'Total Cmp%', 'Total TotDist', 'Total PrgDist', 'Short
Cmp', 'Short Att', 'Short Cmp%', 'Medium Cmp', 'Medium Att', 'Medium Cmp%',
'Long Cmp', 'Long Att', 'Ast', 'xAG', 'xA', 'KP', 'PPA', 'CrsPA', 'PrgP']

Using 'Season' column for season analysis
Available Seasons: ['17_18', '18_19', '19_20', '20_21', '21_22', '22_23',
'23_24', '24_25']

Removing seasons: []
Rows before season removal: 5737
Rows after season removal: 5737

Removing rows with more than 20 missing values out of 69 columns…
Rows before removing high-missing rows: 5737
Rows after removing high-missing rows: 5737

Filling remaining missing values with player averages…
Columns before cleaning: 69
Columns after cleaning: 69

Final Missing Values Summary by Season (After Cleaning)
============================================================

Season: 17_18 (Total rows: 703)
         Column  Missing_Count  Missing_Percentage
           Date              0                0.00
    Blocks Pass              0                0.00
      Total Att              0                0.00
      Total Cmp              0                0.00
            Err              0                0.00
            Clr              0                0.00
        Tkl+Int              0                0.00
            Int              0                0.00
      Blocks Sh              0                0.00
   Tackles TklW              0                0.00
  Blocks Blocks              0                0.00
Challenges Lost              0                0.00
 Challenges Att              0                0.00
 Challenges Tkl              0                0.00
Tackles Att 3rd              0                0.00
Tackles Mid 3rd              0                0.00
     Total Cmp%              0                0.00
  Total TotDist              0                0.00
  Total PrgDist              0                0.00
      Short Cmp              0                0.00
      Short Att              0                0.00
     Short Cmp%              0                0.00
```

Medium Cmp	0	0.00
Medium Att	0	0.00
Medium Cmp%	0	0.00
Long Cmp	0	0.00
Long Att	0	0.00
Ast	0	0.00
xAG	0	0.00
xA	0	0.00
KP	0	0.00
PPA	0	0.00
CrsPA	0	0.00
Tackles Def 3rd	0	0.00
Tackles Tkl	0	0.00
Competition	0	0.00
Gls	0	0.00
CrdY	0	0.00
SoT	0	0.00
Sh	0	0.00
PKatt	0	0.00
PK	0	0.00
Ast	0	0.00
Min	0	0.00
Take-Ons Succ	0	0.00
Age	0	0.00
Pos	0	0.00
Nation	0	0.00
#	0	0.00
Player	0	0.00
Opponent	0	0.00
CrdR	0	0.00
Int	0	0.00
Match URL	0	0.00
Season	0	0.00
Touches	0	0.00
Tkl	0	0.00
Blocks	0	0.00
Expected xG	0	0.00
Expected npxG	0	0.00
Expected xAG	0	0.00
Passes Cmp	0	0.00
Passes Att	0	0.00
Passes Cmp%	0	0.00
Passes PrgP	0	0.00
Carries Carries	0	0.00
Carries PrgC	0	0.00
Take-Ons Att	0	0.00
PrgP	0	0.00

Season: 18_19 (Total rows: 637)

Column	Missing_Count	Missing_Percentage
Date	0	0.00
Blocks Pass	0	0.00
Total Att	0	0.00
Total Cmp	0	0.00
Err	0	0.00
Clr	0	0.00
Tkl+Int	0	0.00
Int	0	0.00
Blocks Sh	0	0.00
Tackles TklW	0	0.00
Blocks Blocks	0	0.00
Challenges Lost	0	0.00
Challenges Att	0	0.00
Challenges Tkl	0	0.00
Tackles Att 3rd	0	0.00
Tackles Mid 3rd	0	0.00
Total Cmp%	0	0.00
Total TotDist	0	0.00
Total PrgDist	0	0.00
Short Cmp	0	0.00
Short Att	0	0.00
Short Cmp%	0	0.00
Medium Cmp	0	0.00
Medium Att	0	0.00
Medium Cmp%	0	0.00
Long Cmp	0	0.00
Long Att	0	0.00
Ast	0	0.00
xAG	0	0.00
xA	0	0.00
KP	0	0.00
PPA	0	0.00
CrsPA	0	0.00
Tackles Def 3rd	0	0.00
Tackles Tkl	0	0.00
Competition	0	0.00
Gls	0	0.00
CrdY	0	0.00
SoT	0	0.00
Sh	0	0.00
PKatt	0	0.00
PK	0	0.00
Ast	0	0.00
Min	0	0.00
Take-Ons Succ	0	0.00
Age	0	0.00

Pos	0	0.00
Nation	0	0.00
#	0	0.00
Player	0	0.00
Opponent	0	0.00
CrdR	0	0.00
Int	0	0.00
Match URL	0	0.00
Season	0	0.00
Touches	0	0.00
Tkl	0	0.00
Blocks	0	0.00
Expected xG	0	0.00
Expected npxG	0	0.00
Expected xAG	0	0.00
Passes Cmp	0	0.00
Passes Att	0	0.00
Passes Cmp%	0	0.00
Passes PrgP	0	0.00
Carries Carries	0	0.00
Carries PrgC	0	0.00
Take-Ons Att	0	0.00
PrgP	0	0.00

Season: 19_20 (Total rows: 651)

Column	Missing_Count	Missing_Percentage
Date	0	0.00
Blocks Pass	0	0.00
Total Att	0	0.00
Total Cmp	0	0.00
Err	0	0.00
Clr	0	0.00
Tkl+Int	0	0.00
Int	0	0.00
Blocks Sh	0	0.00
Tackles TklW	0	0.00
Blocks Blocks	0	0.00
Challenges Lost	0	0.00
Challenges Att	0	0.00
Challenges Tkl	0	0.00
Tackles Att 3rd	0	0.00
Tackles Mid 3rd	0	0.00
Total Cmp%	0	0.00
Total TotDist	0	0.00
Total PrgDist	0	0.00
Short Cmp	0	0.00
Short Att	0	0.00
Short Cmp%	0	0.00

Medium Cmp	0	0.00
Medium Att	0	0.00
Medium Cmp%	0	0.00
Long Cmp	0	0.00
Long Att	0	0.00
Ast	0	0.00
xAG	0	0.00
xA	0	0.00
KP	0	0.00
PPA	0	0.00
CrsPA	0	0.00
Tackles Def 3rd	0	0.00
Tackles Tkl	0	0.00
Competition	0	0.00
Gls	0	0.00
CrdY	0	0.00
SoT	0	0.00
Sh	0	0.00
PKatt	0	0.00
PK	0	0.00
Ast	0	0.00
Min	0	0.00
Take-Ons Succ	0	0.00
Age	0	0.00
Pos	0	0.00
Nation	0	0.00
#	0	0.00
Player	0	0.00
Opponent	0	0.00
CrdR	0	0.00
Int	0	0.00
Match URL	0	0.00
Season	0	0.00
Touches	0	0.00
Tkl	0	0.00
Blocks	0	0.00
Expected xG	0	0.00
Expected npxG	0	0.00
Expected xAG	0	0.00
Passes Cmp	0	0.00
Passes Att	0	0.00
Passes Cmp%	0	0.00
Passes PrgP	0	0.00
Carries Carries	0	0.00
Carries PrgC	0	0.00
Take-Ons Att	0	0.00
PrgP	0	0.00

Season: 20_21 (Total rows: 704)

Column	Missing_Count	Missing_Percentage
Date	0	0.00
Blocks Pass	0	0.00
Total Att	0	0.00
Total Cmp	0	0.00
Err	0	0.00
Clr	0	0.00
Tkl+Int	0	0.00
Int	0	0.00
Blocks Sh	0	0.00
Tackles TklW	0	0.00
Blocks Blocks	0	0.00
Challenges Lost	0	0.00
Challenges Att	0	0.00
Challenges Tkl	0	0.00
Tackles Att 3rd	0	0.00
Tackles Mid 3rd	0	0.00
Total Cmp%	0	0.00
Total TotDist	0	0.00
Total PrgDist	0	0.00
Short Cmp	0	0.00
Short Att	0	0.00
Short Cmp%	0	0.00
Medium Cmp	0	0.00
Medium Att	0	0.00
Medium Cmp%	0	0.00
Long Cmp	0	0.00
Long Att	0	0.00
Ast	0	0.00
xAG	0	0.00
xA	0	0.00
KP	0	0.00
PPA	0	0.00
CrsPA	0	0.00
Tackles Def 3rd	0	0.00
Tackles Tkl	0	0.00
Competition	0	0.00
Gls	0	0.00
CrdY	0	0.00
SoT	0	0.00
Sh	0	0.00
PKatt	0	0.00
PK	0	0.00
Ast	0	0.00
Min	0	0.00
Take-Ons Succ	0	0.00
Age	0	0.00

```
            Pos              0               0.00
         Nation              0               0.00
              #              0               0.00
         Player              0               0.00
       Opponent              0               0.00
           CrdR              0               0.00
            Int              0               0.00
      Match URL              0               0.00
         Season              0               0.00
        Touches              0               0.00
            Tkl              0               0.00
         Blocks              0               0.00
    Expected xG              0               0.00
  Expected npxG              0               0.00
   Expected xAG              0               0.00
    Passes Cmp               0               0.00
    Passes Att               0               0.00
   Passes Cmp%               0               0.00
   Passes PrgP               0               0.00
Carries Carries              0               0.00
   Carries PrgC              0               0.00
   Take-Ons Att              0               0.00
           PrgP              0               0.00

Season: 21_22 (Total rows: 771)
         Column  Missing_Count  Missing_Percentage
           Date              0                0.00
    Blocks Pass              0                0.00
      Total Att              0                0.00
      Total Cmp              0                0.00
            Err              0                0.00
            Clr              0                0.00
        Tkl+Int              0                0.00
            Int              0                0.00
      Blocks Sh              0                0.00
   Tackles TklW              0                0.00
  Blocks Blocks              0                0.00
Challenges Lost              0                0.00
 Challenges Att              0                0.00
 Challenges Tkl              0                0.00
Tackles Att 3rd              0                0.00
Tackles Mid 3rd              0                0.00
     Total Cmp%              0                0.00
  Total TotDist              0                0.00
  Total PrgDist              0                0.00
      Short Cmp              0                0.00
      Short Att              0                0.00
     Short Cmp%              0                0.00
```

Medium Cmp	0	0.00
Medium Att	0	0.00
Medium Cmp%	0	0.00
Long Cmp	0	0.00
Long Att	0	0.00
Ast	0	0.00
xAG	0	0.00
xA	0	0.00
KP	0	0.00
PPA	0	0.00
CrsPA	0	0.00
Tackles Def 3rd	0	0.00
Tackles Tkl	0	0.00
Competition	0	0.00
Gls	0	0.00
CrdY	0	0.00
SoT	0	0.00
Sh	0	0.00
PKatt	0	0.00
PK	0	0.00
Ast	0	0.00
Min	0	0.00
Take-Ons Succ	0	0.00
Age	0	0.00
Pos	0	0.00
Nation	0	0.00
#	0	0.00
Player	0	0.00
Opponent	0	0.00
CrdR	0	0.00
Int	0	0.00
Match URL	0	0.00
Season	0	0.00
Touches	0	0.00
Tkl	0	0.00
Blocks	0	0.00
Expected xG	0	0.00
Expected npxG	0	0.00
Expected xAG	0	0.00
Passes Cmp	0	0.00
Passes Att	0	0.00
Passes Cmp%	0	0.00
Passes PrgP	0	0.00
Carries Carries	0	0.00
Carries PrgC	0	0.00
Take-Ons Att	0	0.00
PrgP	0	0.00

Season: 22_23 (Total rows: 744)

Column	Missing_Count	Missing_Percentage
Date	0	0.00
Blocks Pass	0	0.00
Total Att	0	0.00
Total Cmp	0	0.00
Err	0	0.00
Clr	0	0.00
Tkl+Int	0	0.00
Int	0	0.00
Blocks Sh	0	0.00
Tackles TklW	0	0.00
Blocks Blocks	0	0.00
Challenges Lost	0	0.00
Challenges Att	0	0.00
Challenges Tkl	0	0.00
Tackles Att 3rd	0	0.00
Tackles Mid 3rd	0	0.00
Total Cmp%	0	0.00
Total TotDist	0	0.00
Total PrgDist	0	0.00
Short Cmp	0	0.00
Short Att	0	0.00
Short Cmp%	0	0.00
Medium Cmp	0	0.00
Medium Att	0	0.00
Medium Cmp%	0	0.00
Long Cmp	0	0.00
Long Att	0	0.00
Ast	0	0.00
xAG	0	0.00
xA	0	0.00
KP	0	0.00
PPA	0	0.00
CrsPA	0	0.00
Tackles Def 3rd	0	0.00
Tackles Tkl	0	0.00
Competition	0	0.00
Gls	0	0.00
CrdY	0	0.00
SoT	0	0.00
Sh	0	0.00
PKatt	0	0.00
PK	0	0.00
Ast	0	0.00
Min	0	0.00
Take-Ons Succ	0	0.00
Age	0	0.00

Pos	0	0.00
Nation	0	0.00
#	0	0.00
Player	0	0.00
Opponent	0	0.00
CrdR	0	0.00
Int	0	0.00
Match URL	0	0.00
Season	0	0.00
Touches	0	0.00
Tkl	0	0.00
Blocks	0	0.00
Expected xG	0	0.00
Expected npxG	0	0.00
Expected xAG	0	0.00
Passes Cmp	0	0.00
Passes Att	0	0.00
Passes Cmp%	0	0.00
Passes PrgP	0	0.00
Carries Carries	0	0.00
Carries PrgC	0	0.00
Take-Ons Att	0	0.00
PrgP	0	0.00

Season: 23_24 (Total rows: 762)

Column	Missing_Count	Missing_Percentage
Date	0	0.00
Blocks Pass	0	0.00
Total Att	0	0.00
Total Cmp	0	0.00
Err	0	0.00
Clr	0	0.00
Tkl+Int	0	0.00
Int	0	0.00
Blocks Sh	0	0.00
Tackles TklW	0	0.00
Blocks Blocks	0	0.00
Challenges Lost	0	0.00
Challenges Att	0	0.00
Challenges Tkl	0	0.00
Tackles Att 3rd	0	0.00
Tackles Mid 3rd	0	0.00
Total Cmp%	0	0.00
Total TotDist	0	0.00
Total PrgDist	0	0.00
Short Cmp	0	0.00
Short Att	0	0.00
Short Cmp%	0	0.00

Medium Cmp	0	0.00
Medium Att	0	0.00
Medium Cmp%	0	0.00
Long Cmp	0	0.00
Long Att	0	0.00
Ast	0	0.00
xAG	0	0.00
xA	0	0.00
KP	0	0.00
PPA	0	0.00
CrsPA	0	0.00
Tackles Def 3rd	0	0.00
Tackles Tkl	0	0.00
Competition	0	0.00
Gls	0	0.00
CrdY	0	0.00
SoT	0	0.00
Sh	0	0.00
PKatt	0	0.00
PK	0	0.00
Ast	0	0.00
Min	0	0.00
Take-Ons Succ	0	0.00
Age	0	0.00
Pos	0	0.00
Nation	0	0.00
#	0	0.00
Player	0	0.00
Opponent	0	0.00
CrdR	0	0.00
Int	0	0.00
Match URL	0	0.00
Season	0	0.00
Touches	0	0.00
Tkl	0	0.00
Blocks	0	0.00
Expected xG	0	0.00
Expected npxG	0	0.00
Expected xAG	0	0.00
Passes Cmp	0	0.00
Passes Att	0	0.00
Passes Cmp%	0	0.00
Passes PrgP	0	0.00
Carries Carries	0	0.00
Carries PrgC	0	0.00
Take-Ons Att	0	0.00
PrgP	0	0.00

Season: 24_25 (Total rows: 765)

Column	Missing_Count	Missing_Percentage
Date	0	0.00
Blocks Pass	0	0.00
Total Att	0	0.00
Total Cmp	0	0.00
Err	0	0.00
Clr	0	0.00
Tkl+Int	0	0.00
Int	0	0.00
Blocks Sh	0	0.00
Tackles TklW	0	0.00
Blocks Blocks	0	0.00
Challenges Lost	0	0.00
Challenges Att	0	0.00
Challenges Tkl	0	0.00
Tackles Att 3rd	0	0.00
Tackles Mid 3rd	0	0.00
Total Cmp%	0	0.00
Total TotDist	0	0.00
Total PrgDist	0	0.00
Short Cmp	0	0.00
Short Att	0	0.00
Short Cmp%	0	0.00
Medium Cmp	0	0.00
Medium Att	0	0.00
Medium Cmp%	0	0.00
Long Cmp	0	0.00
Long Att	0	0.00
Ast	0	0.00
xAG	0	0.00
xA	0	0.00
KP	0	0.00
PPA	0	0.00
CrsPA	0	0.00
Tackles Def 3rd	0	0.00
Tackles Tkl	0	0.00
Competition	0	0.00
Gls	0	0.00
CrdY	0	0.00
SoT	0	0.00
Sh	0	0.00
PKatt	0	0.00
PK	0	0.00
Ast	0	0.00
Min	0	0.00
Take-Ons Succ	0	0.00
Age	0	0.00

```
            Pos              0             0.00
         Nation              0             0.00
              #              0             0.00
         Player              0             0.00
       Opponent              0             0.00
           CrdR              0             0.00
            Int              0             0.00
      Match URL              0             0.00
         Season              0             0.00
        Touches              0             0.00
            Tkl              0             0.00
         Blocks              0             0.00
    Expected xG              0             0.00
  Expected npxG              0             0.00
   Expected xAG              0             0.00
     Passes Cmp              0             0.00
     Passes Att              0             0.00
    Passes Cmp%              0             0.00
    Passes PrgP              0             0.00
Carries Carries              0             0.00
   Carries PrgC              0             0.00
   Take-Ons Att              0             0.00
           PrgP              0             0.00

============================================================
OVERALL MISSING VALUES SUMMARY (After All Cleaning)
============================================================
         Column  Missing_Count  Missing_Percentage
           Date              0                0.00
    Blocks Pass              0                0.00
      Total Att              0                0.00
      Total Cmp              0                0.00
            Err              0                0.00
            Clr              0                0.00
        Tkl+Int              0                0.00
            Int              0                0.00
      Blocks Sh              0                0.00
   Tackles TklW              0                0.00
  Blocks Blocks              0                0.00
Challenges Lost              0                0.00
 Challenges Att              0                0.00
 Challenges Tkl              0                0.00
Tackles Att 3rd              0                0.00
Tackles Mid 3rd              0                0.00
     Total Cmp%              0                0.00
  Total TotDist              0                0.00
  Total PrgDist              0                0.00
      Short Cmp              0                0.00
```

Short Att	0	0.00
Short Cmp%	0	0.00
Medium Cmp	0	0.00
Medium Att	0	0.00
Medium Cmp%	0	0.00
Long Cmp	0	0.00
Long Att	0	0.00
Ast	0	0.00
xAG	0	0.00
xA	0	0.00
KP	0	0.00
PPA	0	0.00
CrsPA	0	0.00
Tackles Def 3rd	0	0.00
Tackles Tkl	0	0.00
Competition	0	0.00
Gls	0	0.00
CrdY	0	0.00
SoT	0	0.00
Sh	0	0.00
PKatt	0	0.00
PK	0	0.00
Ast	0	0.00
Min	0	0.00
Take-Ons Succ	0	0.00
Age	0	0.00
Pos	0	0.00
Nation	0	0.00
#	0	0.00
Player	0	0.00
Opponent	0	0.00
CrdR	0	0.00
Int	0	0.00
Match URL	0	0.00
Season	0	0.00
Touches	0	0.00
Tkl	0	0.00
Blocks	0	0.00
Expected xG	0	0.00
Expected npxG	0	0.00
Expected xAG	0	0.00
Passes Cmp	0	0.00
Passes Att	0	0.00
Passes Cmp%	0	0.00
Passes PrgP	0	0.00
Carries Carries	0	0.00
Carries PrgC	0	0.00
Take-Ons Att	0	0.00

```
           PrgP               0                0.00

Removing players who didn't play…
Rows before removing non-players: 5737
Rows after removing non-players: 5737

Removing players with less than 200 total minutes…
Players before filtering: 54
Players with >=200 minutes: 54
Rows after removing low-minute players: 5737

Final dataset shape: (5737, 69)
```

[5]:

```python
#  Season Summary Table
import pandas as pd

# Create summary by season
season_summary = df.groupby('Season').agg({
    df.columns[0]: 'count',  # Total rows per season
    **{col: lambda x: x.isnull().sum() for col in df.
 ↪select_dtypes(include=['number']).columns[:5]}  # Null counts for first 5␣
 ↪numeric columns
}).round(2)

print("Season Summary - Rows and Missing Values")
print("=" * 50)
print(season_summary)

# Final Dataset Variables (APA 7 Format)
print("\n\nTable 1")
print("Final Dataset Variables and Descriptions")
print("=" * 80)

# Create comprehensive variable description table
def get_variable_description(col, dtype):
    """Generate comprehensive descriptions for variables"""
    col_lower = col.lower()

    # Determine data type category
    if dtype == 'object':
        data_category = "Categorical"
    elif 'int' in str(dtype):
        data_category = "Numeric (Integer)"
    elif 'float' in str(dtype):
        data_category = "Numeric (Continuous)"
    else:
        data_category = "Other"
```

```python
    # Generate detailed descriptions
    if 'player' in col_lower:
        return data_category, "Player identification name (categorical variable identifying individual players)"
    elif 'season' in col_lower:
        return data_category, "Season identifier (categorical variable indicating football season year, e.g., 2020-21)"
    elif 'competition' in col_lower:
        return data_category, "Competition type (categorical variable: La Liga, Champions League, Copa del Rey, etc.)"
    elif 'min' in col_lower and 'minute' not in col_lower:
        return data_category, "Minutes played per match (continuous variable ranging 0-90+ minutes)"
    elif 'gls' in col_lower or 'goal' in col_lower:
        return data_category, "Goals scored per match (count variable, integer  0)"
    elif 'ast' in col_lower or 'assist' in col_lower:
        return data_category, "Assists provided per match (count variable, integer  0)"
    elif 'age' in col_lower:
        return data_category, "Player age in years (continuous variable, typically 16-40 years)"
    elif 'pos' in col_lower:
        return data_category, "Playing position (categorical: GK, DF, MF, FW, or combinations)"
    elif 'nation' in col_lower:
        return data_category, "Player nationality (categorical variable indicating country of origin)"
    elif 'opponent' in col_lower:
        return data_category, "Match opponent team name (categorical variable)"
    elif 'shot' in col_lower:
        return data_category, "Shooting statistic (count or rate variable related to shot attempts/accuracy)"
    elif 'pass' in col_lower:
        return data_category, "Passing statistic (count or percentage variable for pass attempts/completion)"
    elif 'tackle' in col_lower:
        return data_category, "Defensive statistic (count variable for successful tackles per match)"
    elif 'card' in col_lower or 'crd' in col_lower:
        return data_category, "Disciplinary cards received (count variable: yellow/red cards per match)"
    elif '%' in col:
        return data_category, "Performance rate statistic (percentage variable, 0-100%)"
```

```python
    elif any(x in col_lower for x in ['xg', 'xa', 'npxg']):
        return data_category, "Expected performance metric (continuous␣
↪variable, advanced analytics statistic)"
    else:
        return data_category, "Performance statistic (numeric variable␣
↪measuring player match performance)"

# Create the table
variable_data = []
for col in df.columns:
    dtype = df[col].dtype
    data_category, description = get_variable_description(col, dtype)

    # Add sample values for categorical variables
    if dtype == 'object' and df[col].nunique() <= 10:
        unique_vals = list(df[col].unique())[:3]
        description += f" (e.g., {', '.join(map(str, unique_vals))})"

    variable_data.append({
        'Variable': col,
        'Type': data_category,
        'Description': description
    })

# Display in APA format
print(f"{'Variable':<20} {'Type':<20} {'Description'}")
print("-" * 80)
for item in variable_data:
    # Wrap long descriptions
    desc_lines = [item['Description'][i:i+45] for i in range(0,␣
↪len(item['Description']), 45)]
    print(f"{item['Variable']:<20} {item['Type']:<20} {desc_lines[0]}")
    for line in desc_lines[1:]:
        print(f"{'':<41} {line}")

print(f"\nNote. N = {len(df):,} observations across {df['Season'].nunique()}␣
↪seasons.")
print(f"Dataset contains {df.shape[1]} variables measuring player performance␣
↪metrics.")
print(f"Categorical variables represent {len([d for d in variable_data if␣
↪d['Type'] == 'Categorical'])} dimensions.")
print(f"Numeric variables represent {len([d for d in variable_data if 'Numeric'␣
↪in d['Type']])} performance measures.")
```

```
Season Summary - Rows and Missing Values
==================================================
        Date  #  Min   Gls   Ast   PK
```

```
Season
17_18    703  0    0     0     0    0
18_19    637  0    0     0     0    0
19_20    651  0    0     0     0    0
20_21    704  0    0     0     0    0
21_22    771  0    0     0     0    0
22_23    744  0    0     0     0    0
23_24    762  0    0     0     0    0
24_25    765  0    0     0     0    0
```

```
Table 1
Final Dataset Variables and Descriptions
================================================================================
Variable             Type                 Description
--------------------------------------------------------------------------------
Date                 Categorical          Performance statistic (numeric
variable measu
                                          ring player match performance)
Competition          Categorical          Competition type (categorical
variable: La Li
                                          ga, Champions League, Copa del Rey,
etc.) (e.
                                          g., La Liga, Champions League)
Opponent             Categorical          Match opponent team name (categorical
variabl
                                          e)
Player               Categorical          Player identification name
(categorical varia
                                          ble identifying individual players)
#                    Numeric (Continuous) Performance statistic (numeric
variable measu
                                          ring player match performance)
Nation               Categorical          Player nationality (categorical
variable indi
                                          cating country of origin)
Pos                  Categorical          Playing position (categorical: GK, DF,
MF, FW
                                          , or combinations)
Age                  Categorical          Player age in years (continuous
variable, typ
                                          ically 16-40 years)
Min                  Numeric (Continuous) Minutes played per match (continuous
variable
                                           ranging 0-90+ minutes)
 Gls                 Numeric (Continuous) Goals scored per match (count
variable, integ
                                          er   0)
```

Ast	Numeric (Continuous)	Assists provided per match (count variable, integer 0)
PK	Numeric (Continuous)	Performance statistic (numeric variable measuring player match performance)
PKatt	Numeric (Continuous)	Performance statistic (numeric variable measuring player match performance)
Sh	Numeric (Continuous)	Performance statistic (numeric variable measuring player match performance)
SoT	Numeric (Continuous)	Performance statistic (numeric variable measuring player match performance)
CrdY	Numeric (Continuous)	Disciplinary cards received (count variable: yellow/red cards per match)
CrdR	Numeric (Continuous)	Disciplinary cards received (count variable: yellow/red cards per match)
Int	Numeric (Continuous)	Performance statistic (numeric variable measuring player match performance)
Match URL	Categorical	Performance statistic (numeric variable measuring player match performance)
Season	Categorical	Season identifier (categorical variable indicating football season year, e.g., 2020-21) (e.g., 17_18, 18_19, 19_20)
Touches	Numeric (Continuous)	Performance statistic (numeric variable measuring player match performance)
Tkl	Numeric (Continuous)	Performance statistic (numeric variable measuring player match performance)
Blocks	Numeric (Continuous)	Performance statistic (numeric variable measuring player match performance)
Expected xG	Numeric (Continuous)	Expected performance metric (continuous variable, advanced analytics statistic)
Expected npxG	Numeric (Continuous)	Expected performance metric (continuous variable, advanced analytics statistic)
Expected xAG	Numeric (Continuous)	Expected performance metric

(continuous varia
ble, advanced analytics statistic)
Passes Cmp Numeric (Continuous) Passing statistic (count or percentage
variab
le for pass attempts/completion)
Passes Att Numeric (Continuous) Passing statistic (count or percentage
variab
le for pass attempts/completion)
Passes Cmp% Numeric (Continuous) Passing statistic (count or percentage
variab
le for pass attempts/completion)
Passes PrgP Numeric (Continuous) Passing statistic (count or percentage
variab
le for pass attempts/completion)
Carries Carries Numeric (Continuous) Performance statistic (numeric
variable measu
ring player match performance)
Carries PrgC Numeric (Continuous) Performance statistic (numeric
variable measu
ring player match performance)
Take-Ons Att Numeric (Continuous) Performance statistic (numeric
variable measu
ring player match performance)
Take-Ons Succ Numeric (Continuous) Performance statistic (numeric
variable measu
ring player match performance)
Tackles Tkl Numeric (Continuous) Defensive statistic (count variable
for succe
ssful tackles per match)
Tackles TklW Numeric (Continuous) Defensive statistic (count variable
for succe
ssful tackles per match)
Tackles Def 3rd Numeric (Continuous) Defensive statistic (count variable
for succe
ssful tackles per match)
Tackles Mid 3rd Numeric (Continuous) Defensive statistic (count variable
for succe
ssful tackles per match)
Tackles Att 3rd Numeric (Continuous) Defensive statistic (count variable
for succe
ssful tackles per match)
Challenges Tkl Numeric (Continuous) Performance statistic (numeric
variable measu
ring player match performance)
Challenges Att Numeric (Continuous) Performance statistic (numeric
variable measu
ring player match performance)
Challenges Lost Numeric (Continuous) Performance statistic (numeric

variable measu
ring player match performance)
Blocks Blocks Numeric (Continuous) Performance statistic (numeric
variable measu
ring player match performance)
Blocks Sh Numeric (Continuous) Performance statistic (numeric
variable measu
ring player match performance)
Blocks Pass Numeric (Continuous) Passing statistic (count or percentage
variab
le for pass attempts/completion)
Int Numeric (Continuous) Performance statistic (numeric
variable measu
ring player match performance)
Tkl+Int Numeric (Continuous) Performance statistic (numeric
variable measu
ring player match performance)
Clr Numeric (Continuous) Performance statistic (numeric
variable measu
ring player match performance)
Err Numeric (Continuous) Performance statistic (numeric
variable measu
ring player match performance)
Total Cmp Numeric (Continuous) Performance statistic (numeric
variable measu
ring player match performance)
Total Att Numeric (Continuous) Performance statistic (numeric
variable measu
ring player match performance)
Total Cmp% Numeric (Continuous) Performance rate statistic (percentage
variab
le, 0-100%)
Total TotDist Numeric (Continuous) Performance statistic (numeric
variable measu
ring player match performance)
Total PrgDist Numeric (Continuous) Performance statistic (numeric
variable measu
ring player match performance)
Short Cmp Numeric (Continuous) Performance statistic (numeric
variable measu
ring player match performance)
Short Att Numeric (Continuous) Performance statistic (numeric
variable measu
ring player match performance)
Short Cmp% Numeric (Continuous) Performance rate statistic (percentage
variab
le, 0-100%)
Medium Cmp Numeric (Continuous) Performance statistic (numeric

```
variable measu
                                          ring player match performance)
Medium Att          Numeric (Continuous) Performance statistic (numeric
variable measu
                                          ring player match performance)
Medium Cmp%         Numeric (Continuous) Performance rate statistic (percentage
variab
                                          le, 0-100%)
Long Cmp            Numeric (Continuous) Performance statistic (numeric
variable measu
                                          ring player match performance)
Long Att            Numeric (Continuous) Performance statistic (numeric
variable measu
                                          ring player match performance)
Ast                 Numeric (Continuous) Assists provided per match (count
variable, i
                                          nteger   0)
xAG                 Numeric (Continuous) Expected performance metric
(continuous varia
                                          ble, advanced analytics statistic)
xA                  Numeric (Continuous) Expected performance metric
(continuous varia
                                          ble, advanced analytics statistic)
KP                  Numeric (Continuous) Performance statistic (numeric
variable measu
                                          ring player match performance)
PPA                 Numeric (Continuous) Performance statistic (numeric
variable measu
                                          ring player match performance)
CrsPA               Numeric (Continuous) Performance statistic (numeric
variable measu
                                          ring player match performance)
PrgP                Numeric (Continuous) Performance statistic (numeric
variable measu
                                          ring player match performance)

Note. N = 5,737 observations across 8 seasons.
Dataset contains 69 variables measuring player performance metrics.
Categorical variables represent 9 dimensions.
Numeric variables represent 60 performance measures.
```

[6]:
```python
# Pass cleaned dataframe to next analysis
combined_df = df   # Make them the same
```

[7]:
```python
# ================================
# STEP 3: COMPREHENSIVE EDA ANALYSIS
# ================================
```

```python
from typing import Tuple, List, Dict, Optional, Any
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
def comprehensive_eda_analysis(df: pd.DataFrame) -> Tuple[List[str], List[str], pd.DataFrame]:
    """
    Perform comprehensive EDA analysis for academic paper
    """
    print("="*80)
    print("COMPREHENSIVE EXPLORATORY DATA ANALYSIS")
    print("="*80)

    # Basic Dataset Information
    print("\n1. DATASET OVERVIEW")
    print("-" * 40)
    print(f"Dataset Shape: {df.shape}")
    print(f"Total Features: {df.shape[1]}")
    print(f"Total Observations: {df.shape[0]}")
    print(f"Memory Usage: {df.memory_usage(deep=True).sum() / 1024**2:.2f} MB")

    # Data Types and Missing Values
    print("\n2. DATA QUALITY ASSESSMENT")
    print("-" * 40)

    # Create comprehensive data quality report
    data_quality = pd.DataFrame({
        'Data_Type': df.dtypes,
        'Non_Null_Count': df.count(),
        'Null_Count': df.isnull().sum(),
        'Null_Percentage': (df.isnull().sum() / len(df)) * 100,
        'Unique_Values': df.nunique(),
        'Unique_Percentage': (df.nunique() / len(df)) * 100
    })

    print(data_quality)

    # Identify numeric and categorical columns
    numeric_cols = df.select_dtypes(include=['int64', 'float64', 'int32', 'float32']).columns.tolist()
    categorical_cols = df.select_dtypes(include=['object', 'category']).columns.tolist()

    print(f"\nNumeric Columns ({len(numeric_cols)}): {numeric_cols}")
    print(f"Categorical Columns ({len(categorical_cols)}): {categorical_cols}")
```

```
    return numeric_cols, categorical_cols, data_quality

def univariate_analysis(df: pd.DataFrame, numeric_cols: List[str],␣
 ↪categorical_cols: List[str]) -> None:
    """
    Perform univariate analysis (non-graphical and graphical) including␣
 ↪position-specific distributions
    """
    print("\n" + "="*80)
    print("3. UNIVARIATE ANALYSIS")
    print("="*80)

    # Univariate Non-Graphical Analysis
    print("\n3.1 DESCRIPTIVE STATISTICS (Non-Graphical)")
    print("-" * 50)

    if numeric_cols:
        desc_stats = df[numeric_cols].describe()
        print("\nDescriptive Statistics for Numeric Variables:")
        print(desc_stats)

        # Additional statistics
        print("\nAdditional Statistical Measures:")
        additional_stats = pd.DataFrame({
            'Skewness': df[numeric_cols].skew(),
            'Kurtosis': df[numeric_cols].kurtosis(),
            'Coefficient_of_Variation': (df[numeric_cols].std() /␣
 ↪df[numeric_cols].mean()) * 100
        })
        print(additional_stats)

    # Categorical Variables Summary
    if categorical_cols:
        print("\nCategorical Variables Summary:")
        for col in categorical_cols[:5]:  # Show first 5 categorical columns
            print(f"\n{col}:")
            print(df[col].value_counts().head(10))

    # Univariate Graphical Analysis
    print("\n3.2 UNIVARIATE GRAPHICAL ANALYSIS")
    print("-" * 50)

    # Overall distribution plots for key metrics
    if len(numeric_cols) > 0:
        key_metrics = [' Gls', ' Ast', ' Sh', ' Tkl', ' Int', 'Passes Cmp%', '␣
 ↪Touches', 'Expected xG', 'SCA']
```

```python
        available_key_metrics = [metric for metric in key_metrics if metric in numeric_cols]

        if available_key_metrics:
            n_metrics = min(len(available_key_metrics), 9)
            fig, axes = plt.subplots(3, 3, figsize=(18, 15))
            fig.suptitle('Distribution of Key Performance Metrics', fontsize=20, fontweight='bold', y=0.98)

            for i, col in enumerate(available_key_metrics[:n_metrics]):
                row, col_idx = i // 3, i % 3

                # High-quality histogram with KDE
                data = df[col].dropna()
                if len(data) > 0:
                    axes[row, col_idx].hist(data, bins=30, alpha=0.7, density=True,
                                            color='skyblue', edgecolor='black', linewidth=0.5)
                    axes[row, col_idx].set_title(f'{col.strip()}', fontweight='bold', fontsize=14)
                    axes[row, col_idx].set_xlabel(col.strip(), fontsize=12)
                    axes[row, col_idx].set_ylabel('Density', fontsize=12)
                    axes[row, col_idx].grid(True, alpha=0.3)

                    # Add KDE curve
                    try:
                        data.plot.kde(ax=axes[row, col_idx], color='red', linewidth=2)
                    except:
                        pass

                    # Add statistics text
                    mean_val = data.mean()
                    std_val = data.std()
                    axes[row, col_idx].axvline(mean_val, color='red', linestyle='--', alpha=0.8, label=f'Mean: {mean_val:.2f}')
                    axes[row, col_idx].legend(fontsize=10)

            # Remove empty subplots
            for i in range(n_metrics, 9):
                row, col_idx = i // 3, i % 3
                fig.delaxes(axes[row, col_idx])

            plt.tight_layout()
            plt.show()
```

```
    # Position-specific distribution analysis
    print("\n3.3 POSITION-SPECIFIC DISTRIBUTION ANALYSIS")
    print("-" * 50)

    if 'Pos' in df.columns:
        # Define position-specific metrics
        position_metrics = {
            'Forward': [' Gls', ' Ast', ' Sh', ' SoT', 'Expected xG', 'Expected␣
↪npxG'],
            'Midfielder': ['Passes Cmp%', 'KP', ' Tkl', 'SCA', 'GCA', 'Passes␣
↪PrgP'],
            'Defender': [' Tkl', ' Int', ' Blocks', 'Clr', 'Tackles TklW',␣
↪'Challenges Tkl%'],
            'Goalkeeper': ['Total Cmp%', 'Err', 'Total TotDist', 'Total␣
↪PrgDist', 'Long Cmp%', 'Short Cmp%']
        }

        # Position mapping
        position_mapping = {
            'FW': 'Forward', 'CF': 'Forward', 'LW': 'Forward', 'RW': 'Forward',
            'MF': 'Midfielder', 'CM': 'Midfielder', 'DM': 'Midfielder', 'AM':␣
↪'Midfielder',
            'DF': 'Defender', 'CB': 'Defender', 'LB': 'Defender', 'RB':␣
↪'Defender',
            'GK': 'Goalkeeper'
        }

        # Get positions available in dataset
        dataset_positions = df['Pos'].unique()
        positions_to_analyze = []
        for pos_abbr in dataset_positions:
            for abbr, full_name in position_mapping.items():
                if abbr in str(pos_abbr):
                    if full_name not in positions_to_analyze:
                        positions_to_analyze.append(full_name)
                    break

        print(f"Creating distribution charts for positions:␣
↪{positions_to_analyze}")

        # Create position-specific distribution charts
        for position in positions_to_analyze:
            print(f"\n--- {position.upper()} DISTRIBUTION ANALYSIS ---")

            # Get available metrics for this position
```

```
            available_metrics = []
            if position in position_metrics:
                for metric in position_metrics[position]:
                    if metric in df.columns:
                        available_metrics.append(metric)
                    else:
                        # Look for similar columns
                        similar_cols = [col for col in df.columns if metric.replace(' ', '').lower() in col.replace(' ', '').lower()]
                        if similar_cols:
                            available_metrics.append(similar_cols[0])

            # Filter data for this position
            pos_abbrevs = [abbr for abbr, full in position_mapping.items() if full == position]
            position_mask = df['Pos'].isin(pos_abbrevs)
            for abbr in pos_abbrevs:
                abbr_mask = df['Pos'].str.contains(abbr, case=False, na=False)
                position_mask = position_mask | abbr_mask

            position_data = df[position_mask]

            if position_data.empty or len(available_metrics) == 0:
                print(f"No data or metrics available for {position}")
                continue

            print(f"Sample size: {len(position_data)} players")
            print(f"Metrics analyzed: {available_metrics}")

            # Create high-quality distribution plots for this position
            n_metrics = min(len(available_metrics), 6)  # Show up to 6 metrics per position
            if n_metrics > 0:
                fig, axes = plt.subplots(2, 3, figsize=(18, 12))
                fig.suptitle(f'{position} - Performance Metrics Distribution\n(Sample: {len(position_data)} players)',
                           fontsize=18, fontweight='bold', y=0.98)

                for i, metric in enumerate(available_metrics[:n_metrics]):
                    row, col_idx = i // 3, i % 3

                    # Get data for this metric
                    metric_data = position_data[metric].dropna()
                    overall_data = df[metric].dropna()

                    if len(metric_data) > 0:
```

```
                        # Create histogram with comparison to overall␣
↪distribution
                        axes[row, col_idx].hist(overall_data, bins=20, alpha=0.
↪3, density=True,
                                               color='lightgray', label='All␣
↪Players', edgecolor='black', linewidth=0.5)
                        axes[row, col_idx].hist(metric_data, bins=15, alpha=0.
↪8, density=True,
                                               color='steelblue',␣
↪label=f'{position}', edgecolor='black', linewidth=0.7)

                        # Add KDE curves
                        try:
                            overall_data.plot.kde(ax=axes[row, col_idx],␣
↪color='gray', linewidth=2, alpha=0.7, label='All Players KDE')
                            metric_data.plot.kde(ax=axes[row, col_idx],␣
↪color='red', linewidth=3, label=f'{position} KDE')
                        except:
                            pass

                        # Add statistics
                        pos_mean = metric_data.mean()
                        overall_mean = overall_data.mean()
                        pos_std = metric_data.std()

                        axes[row, col_idx].axvline(pos_mean, color='red',␣
↪linestyle='--', linewidth=2, alpha=0.8)
                        axes[row, col_idx].axvline(overall_mean, color='gray',␣
↪linestyle=':', linewidth=2, alpha=0.8)

                        # Formatting
                        axes[row, col_idx].set_title(f'{metric.strip()}\nMean:␣
↪{pos_mean:.2f} (±{pos_std:.2f})',
                                                    fontweight='bold',␣
↪fontsize=12)
                        axes[row, col_idx].set_xlabel(metric.strip(),␣
↪fontsize=11)
                        axes[row, col_idx].set_ylabel('Density', fontsize=11)
                        axes[row, col_idx].legend(fontsize=9)
                        axes[row, col_idx].grid(True, alpha=0.3)

                        # Add sample size annotation
                        axes[row, col_idx].text(0.02, 0.98,␣
↪f'n={len(metric_data)}',
                                               transform=axes[row, col_idx].
↪transAxes,
```

```
                                                    fontsize=10,␣
↪verticalalignment='top',
                                                    bbox=dict(boxstyle='round',␣
↪facecolor='white', alpha=0.8))

                # Remove empty subplots
                for i in range(n_metrics, 6):
                    row, col_idx = i // 3, i % 3
                    fig.delaxes(axes[row, col_idx])

                plt.tight_layout()
                plt.show()

                # Print statistical summary for this position
                print(f"\nStatistical Summary for {position}:")
                position_stats = position_data[available_metrics[:n_metrics]].
↪describe()
                print(position_stats.round(3))
                print("-" * 60)

    else:
        print("No 'Pos' column found - skipping position-specific distribution␣
↪analysis")

    # Box plots for outlier detection (improved quality)
    if len(numeric_cols) > 0:
        print("\n3.4 OUTLIER DETECTION ANALYSIS")
        print("-" * 50)

        key_metrics_for_boxplot = [metric for metric in [' Gls', ' Ast', '␣
↪Tkl', ' Int', 'Passes Cmp%', ' Touches']
                                    if metric in numeric_cols]

        if key_metrics_for_boxplot:
            n_cols = min(len(key_metrics_for_boxplot), 6)
            fig, axes = plt.subplots(2, 3, figsize=(18, 12))
            fig.suptitle('Box Plots for Outlier Detection - Key Metrics',␣
↪fontsize=18, fontweight='bold', y=0.98)

            for i, col in enumerate(key_metrics_for_boxplot[:n_cols]):
                row, col_idx = i // 3, i % 3

                # Create box plot with better styling
                box_plot = axes[row, col_idx].boxplot(df[col].dropna(),␣
↪patch_artist=True,
```

```
                                                   boxprops=dict(facecolor='lightblue', alpha=0.7),
                                                   medianprops=dict(color='red', linewidth=2),
                                                   whiskerprops=dict(color='black', linewidth=1.5),
                                                   capprops=dict(color='black', linewidth=1.5),
                                                   flierprops=dict(marker='o', markerfacecolor='red', markersize=6, alpha=0.6))

                axes[row, col_idx].set_title(f'{col.strip()}', fontweight='bold', fontsize=14)
                axes[row, col_idx].set_ylabel('Value', fontsize=12)
                axes[row, col_idx].grid(True, alpha=0.3)

                # Add statistics annotation
                data = df[col].dropna()
                q1, median, q3 = data.quantile([0.25, 0.5, 0.75])
                iqr = q3 - q1
                outliers = data[(data < q1 - 1.5*iqr) | (data > q3 + 1.5*iqr)]

                stats_text = f'Median: {median:.2f}\nIQR: {iqr:.2f}\nOutliers: {len(outliers)}'
                axes[row, col_idx].text(0.02, 0.98, stats_text, transform=axes[row, col_idx].transAxes,
                                      fontsize=10, verticalalignment='top',
                                      bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

            # Remove empty subplots
            for i in range(n_cols, 6):
                row, col_idx = i // 3, i % 3
                fig.delaxes(axes[row, col_idx])

            plt.tight_layout()
            plt.show()

def multivariate_analysis(df: pd.DataFrame, numeric_cols: List[str]) -> Optional[Dict[str, pd.DataFrame]]:
    """
    Perform multivariate analysis by position using actual dataset columns
    """
    print("\n" + "="*80)
    print("4. MULTIVARIATE ANALYSIS BY POSITION")
    print("="*80)
```

```
    if len(numeric_cols) < 2:
        print("Insufficient numeric variables for multivariate analysis")
        return None

    # Define position-specific metrics using actual column names
    position_metrics = {
        'Forward': [' Gls', ' Ast', ' Sh', ' SoT', 'Expected xG', 'Expected␣
↪npxG', 'Expected xAG', 'Take-Ons Succ', 'Take-Ons Att', 'SCA', 'GCA'],
        'Midfielder': ['Passes Cmp%', 'KP', ' Tkl', 'SCA', 'GCA', 'Passes␣
↪PrgP', ' Touches', 'Passes Att', 'Passes Cmp', ' xAG', 'Carries PrgC'],
        'Defender': [' Tkl', ' Int', ' Blocks', 'Clr', 'Tackles TklW',␣
↪'Challenges Tkl%', 'Tackles Def 3rd', 'Tackles Mid 3rd', 'Blocks Sh',␣
↪'Blocks Pass', 'Tkl+Int'],
        'Goalkeeper': ['Total Cmp%', 'Err', 'Total TotDist', 'Total PrgDist',␣
↪'Long Cmp%', 'Short Cmp%', 'Medium Cmp%', 'Total Cmp', 'Total Att', 'Long␣
↪Att', 'Short Att']
    }

    # Check if we have position data
    if 'Pos' in df.columns:
        # Map position abbreviations to full names
        position_mapping = {
            'FW': 'Forward', 'CF': 'Forward', 'LW': 'Forward', 'RW': 'Forward',
            'MF': 'Midfielder', 'CM': 'Midfielder', 'DM': 'Midfielder', 'AM':␣
↪'Midfielder',
            'DF': 'Defender', 'CB': 'Defender', 'LB': 'Defender', 'RB':␣
↪'Defender',
            'GK': 'Goalkeeper'
        }

        # Get unique positions in the dataset
        dataset_positions = df['Pos'].unique()
        print(f"Positions found in dataset: {dataset_positions}")

        positions_to_analyze = []
        for pos_abbr in dataset_positions:
            for abbr, full_name in position_mapping.items():
                if abbr in str(pos_abbr):
                    if full_name not in positions_to_analyze:
                        positions_to_analyze.append(full_name)
                    break
    else:
        positions_to_analyze = ['Forward', 'Midfielder', 'Defender',␣
↪'Goalkeeper']
```

```
    print(f"Analyzing positions: {positions_to_analyze}")

    correlation_matrices = {}

    for position in positions_to_analyze:
        print(f"\n{'='*60}")
        print(f"{position.upper()} CORRELATION ANALYSIS")
        print(f"{'='*60}")

        # Get available metrics for this position
        available_metrics = []
        if position in position_metrics:
            for metric in position_metrics[position]:
                if metric in df.columns:
                    available_metrics.append(metric)
                else:
                    # Look for similar columns
                    similar_cols = [col for col in df.columns if metric.
↪replace(' ', '').lower() in col.replace(' ', '').lower()]
                    if similar_cols:
                        available_metrics.append(similar_cols[0])

        # If no position-specific metrics found, use general performance metrics
        if len(available_metrics) < 3:
            print(f"Limited position-specific metrics found. Using general␣
↪performance indicators...")
            general_metrics = [' Gls', ' Ast', ' Tkl', ' Int', 'Passes Cmp%', '␣
↪Touches']
            for metric in general_metrics:
                if metric in df.columns and metric not in available_metrics:
                    available_metrics.append(metric)
                    if len(available_metrics) >= 6:
                        break

        # Ensure we have enough metrics for correlation analysis (aim for 6-10␣
↪metrics)
        final_metrics = available_metrics[:10] if len(available_metrics) >= 6␣
↪else available_metrics

        if len(final_metrics) < 2:
            print(f"Insufficient metrics for {position} correlation analysis")
            continue

        print(f"Analyzing metrics: {final_metrics}")

        # Filter data for this position (if position column exists)
        if 'Pos' in df.columns:
```

```
            # Get position abbreviations that map to this full position name
            pos_abbrevs = [abbr for abbr, full in position_mapping.items() if␣
↪full == position]
            position_mask = df['Pos'].isin(pos_abbrevs)
            position_data = df[position_mask][final_metrics]

            # Also check for partial matches in case of combined positions like␣
↪"DF,MF"
            for abbr in pos_abbrevs:
                abbr_mask = df['Pos'].str.contains(abbr, case=False, na=False)
                additional_data = df[abbr_mask][final_metrics]
                if not additional_data.empty:
                    position_data = pd.concat([position_data, additional_data]).
↪drop_duplicates()
        else:
            position_data = df[final_metrics]

        if position_data.empty:
            print(f"No data found for {position}")
            continue

        print(f"Sample size: {len(position_data)} observations")

        # Remove rows with all NaN values
        position_data = position_data.dropna(how='all')

        if len(position_data) < 2:
            print(f"Insufficient non-null data for {position}")
            continue

        # Calculate correlation matrix
        correlation_matrix = position_data.corr()
        correlation_matrices[position] = correlation_matrix

        print(f"\nCorrelation Matrix for {position}:")
        print(correlation_matrix.round(3))

        # Find highly correlated pairs
        print(f"\nHighly Correlated Pairs for {position} (|r| > 0.6):")
        high_corr_pairs = []
        for i in range(len(correlation_matrix.columns)):
            for j in range(i+1, len(correlation_matrix.columns)):
                if not pd.isna(correlation_matrix.iloc[i, j]):
                    corr_val = correlation_matrix.iloc[i, j]
                    if abs(corr_val) > 0.6:
                        high_corr_pairs.append({
                            'Variable_1': correlation_matrix.columns[i],
```

```
                            'Variable_2': correlation_matrix.columns[j],
                            'Correlation': corr_val
                        })

        if high_corr_pairs:
            high_corr_df = pd.DataFrame(high_corr_pairs)
            print(high_corr_df.sort_values('Correlation', key=abs,␣
↪ascending=False))
        else:
            print("No highly correlated pairs found (|r| > 0.6)")

        # Statistical significance test
        print(f"\nStatistical Summary for {position}:")
        print(f"- Mean correlation: {correlation_matrix.abs().mean().mean():.
↪3f}")
        print(f"- Max correlation: {correlation_matrix.abs().max().max():.3f}")
        print(f"- Variables analyzed: {len(final_metrics)}")

        # Create position-specific correlation heatmap
        plt.figure(figsize=(10, 8))
        mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

        # Handle NaN values in correlation matrix
        correlation_matrix_clean = correlation_matrix.fillna(0)

        heatmap = sns.heatmap(correlation_matrix_clean, mask=mask, annot=True,␣
↪cmap='RdBu_r',
                            center=0, square=True, fmt='.2f',␣
↪cbar_kws={"shrink": .8},
                            linewidths=0.5)
        plt.title(f'{position} - Performance Metrics Correlation␣
↪Matrix\n(Sample size: {len(position_data)})',
                 fontsize=14, fontweight='bold', pad=20)
        plt.xlabel('Performance Metrics', fontweight='bold')
        plt.ylabel('Performance Metrics', fontweight='bold')
        plt.xticks(rotation=45, ha='right')
        plt.yticks(rotation=0)
        plt.tight_layout()
        plt.show()

        print("-" * 60)

    print(f"\n Correlation analysis complete for {len(correlation_matrices)}␣
↪positions")
    return correlation_matrices
```

```python
def create_position_spider_charts(df: pd.DataFrame) -> None:
    """
    Create spider charts for each position using the SAME metrics as the␣
↪correlation analysis
    """
    print("\n" + "="*80)
    print("5. POSITION-SPECIFIC PLAYER PERFORMANCE SPIDER CHARTS")
    print("="*80)
    print(" Using the same metrics as correlation analysis for consistency")
    print("="*80)

    # Use the SAME position metrics as in correlation analysis
    position_metrics = {
        'Forward': [' Gls', ' Ast', ' Sh', ' SoT', 'Expected xG', 'Expected␣
↪npxG', 'Expected xAG', 'Take-Ons Succ', 'Take-Ons Att', 'SCA', 'GCA'],
        'Midfielder': ['Passes Cmp%', 'KP', ' Tkl', 'SCA', 'GCA', 'Passes␣
↪PrgP', ' Touches', 'Passes Att', 'Passes Cmp', ' xAG', 'Carries PrgC'],
        'Defender': [' Tkl', ' Int', ' Blocks', 'Clr', 'Tackles TklW',␣
↪'Challenges Tkl%', 'Tackles Def 3rd', 'Tackles Mid 3rd', 'Blocks Sh',␣
↪'Blocks Pass', 'Tkl+Int'],
        'Goalkeeper': ['Total Cmp%', 'Err', 'Total TotDist', 'Total PrgDist',␣
↪'Long Cmp%', 'Short Cmp%', 'Medium Cmp%', 'Total Cmp', 'Total Att', 'Long␣
↪Att', 'Short Att']
    }

    # Define position-specific players - Vinicius only for Forwards with Mbappe
    position_players = {
        'Forward': ['Vinícius Jr', 'Mbappé'],  # Vinicius + Mbappe for Forwards␣
↪only
        'Midfielder': ['Modrić', 'Bellingham'],  # Keep original pairs for␣
↪other positions
        'Defender': ['Rüdiger', 'Militão'],      # Keep original pairs for other␣
↪positions
        'Goalkeeper': ['Courtois', 'Lunin']      # Keep original pairs for other␣
↪positions
    }

    # Check if we have position data
    if 'Pos' in df.columns:
        position_mapping = {
            'FW': 'Forward', 'CF': 'Forward', 'LW': 'Forward', 'RW': 'Forward',
            'MF': 'Midfielder', 'CM': 'Midfielder', 'DM': 'Midfielder', 'AM':␣
↪'Midfielder',
            'DF': 'Defender', 'CB': 'Defender', 'LB': 'Defender', 'RB':␣
↪'Defender',
```

```
            'GK': 'Goalkeeper'
        }

        dataset_positions = df['Pos'].unique()
        positions_to_analyze = []

        for pos_abbr in dataset_positions:
            for abbr, full_name in position_mapping.items():
                if abbr in str(pos_abbr):
                    if full_name not in positions_to_analyze:
                        positions_to_analyze.append(full_name)
                    break
    else:
        positions_to_analyze = ['Forward', 'Midfielder', 'Defender',␣
↪'Goalkeeper']

    # Create spider charts for each position using the same metrics as␣
↪correlation analysis
    for position in positions_to_analyze:
        print(f"\n{'='*60}")
        print(f"Creating spider chart for {position.upper()}")
        print(f"{'='*60}")

        players = position_players.get(position, ['Player A', 'Player B'])

        # Get the SAME available metrics used in correlation analysis
        available_metrics = []
        if position in position_metrics:
            for metric in position_metrics[position]:
                if metric in df.columns:
                    available_metrics.append(metric)
                else:
                    # Look for similar columns
                    similar_cols = [col for col in df.columns if metric.
↪replace(' ', '').lower() in col.replace(' ', '').lower()]
                    if similar_cols:
                        available_metrics.append(similar_cols[0])

        # If no position-specific metrics found, use general performance metrics
        if len(available_metrics) < 3:
            print(f"Limited position-specific metrics found. Using general␣
↪performance indicators...")
            general_metrics = [' Gls', ' Ast', ' Tkl', ' Int', 'Passes Cmp%', '␣
↪Touches']
            for metric in general_metrics:
                if metric in df.columns and metric not in available_metrics:
                    available_metrics.append(metric)
```

```
                if len(available_metrics) >= 6:
                    break

        # Use the same metrics as correlation analysis (up to 10 metrics)
        final_metrics = available_metrics[:10] if len(available_metrics) >= 6 else available_metrics

        if len(final_metrics) < 3:
            print(f"Insufficient metrics for {position} spider chart")
            continue

        print(f"Using {len(final_metrics)} metrics: {final_metrics}")

        # Look for actual players in dataset with special handling for Vinicius in Forwards only
        available_players = []
        if 'Player' in df.columns:
            # Special handling for Vinicius - ONLY for Forward position
            if position == 'Forward':
                vinicius_variations = ['Vinícius Jr', 'Vinicius Jr', 'Vinícius Junior', 'Vinicius Junior', 'Vinícius', 'Vinicius']
                vinicius_found = False

                for vini_name in vinicius_variations:
                    matches = df[df['Player'].str.contains(vini_name, case=False, na=False)]
                    if not matches.empty:
                        actual_player_name = matches['Player'].iloc[0]
                        available_players.append(actual_player_name)
                        print(f"  Found Vinicius: {actual_player_name}")
                        vinicius_found = True
                        break

                # If Vinicius not found by exact match, try partial match with "Viní" or "Vini"
                if not vinicius_found:
                    vini_matches = df[df['Player'].str.contains('Viní|Vini', case=False, na=False)]
                    if not vini_matches.empty:
                        actual_player_name = vini_matches['Player'].iloc[0]
                        available_players.append(actual_player_name)
                        print(f"  Found Vinicius (partial match): {actual_player_name}")
                        vinicius_found = True
```

```
            # Regular player search for all positions (including remaining Forward players)
            for player in players:
                if len(available_players) >= 2:
                    break
                # Skip if it's a Vinicius variation and we already found him (and we're in Forward position)
                if position == 'Forward' and any(vini in player for vini in ['Vinícius', 'Vinicius']) and any('Viní' in p or 'Vini' in p for p in available_players):
                    continue

                # Look for partial matches
                matches = df[df['Player'].str.contains(player.split()[0], case=False, na=False)]
                if not matches.empty:
                    actual_player_name = matches['Player'].iloc[0]
                    if actual_player_name not in available_players:
                        available_players.append(actual_player_name)
                        print(f"Found player: {actual_player_name}")
                else:
                    print(f"Player {player} not found in dataset")

        # If we don't have the specific players, use players from that position
        if len(available_players) < 2 and 'Pos' in df.columns:
            pos_abbreviations = {
                'Forward': ['FW', 'CF', 'LW', 'RW'],
                'Midfielder': ['MF', 'CM', 'DM', 'AM'],
                'Defender': ['DF', 'CB', 'LB', 'RB'],
                'Goalkeeper': ['GK']
            }

            for pos_abbr in pos_abbreviations.get(position, []):
                pos_players = df[df['Pos'].str.contains(pos_abbr, case=False, na=False)]['Player'].unique()
                for player in pos_players[:2]:
                    if player not in available_players:
                        available_players.append(player)
                    if len(available_players) >= 2:
                        break
                if len(available_players) >= 2:
                    break

        # If still no players found, use any two players
        if len(available_players) < 2:
            all_players = df['Player'].unique() if 'Player' in df.columns else ['Player A', 'Player B']
```

```python
            available_players = all_players[:2]

        final_players = available_players[:2]
        print(f"Final players: {final_players}")
        print(f"Final metrics ({len(final_metrics)}): {final_metrics}")

        # Create the spider chart
        fig, ax = plt.subplots(figsize=(12, 10),␣
↪subplot_kw=dict(projection='polar'))

        # Number of metrics
        N = len(final_metrics)

        # Angles for each metric
        angles = [n / float(N) * 2 * np.pi for n in range(N)]
        angles += angles[:1]  # Complete the circle

        # Colors for the two players
        colors = ['#FF6B6B', '#4ECDC4']

        # Calculate performance scores for each player
        player_scores = []
        for i, player in enumerate(final_players):
            if 'Player' in df.columns:
                player_data = df[df['Player'] == player]
                if not player_data.empty:
                    values = []
                    raw_values = []
                    for metric in final_metrics:
                        if metric in player_data.columns:
                            raw_val = player_data[metric].mean()
                            raw_values.append(raw_val)
                            # Normalize to 0-100 scale
                            metric_max = df[metric].max()
                            metric_min = df[metric].min()
                            if metric_max > metric_min:
                                normalized_val = ((raw_val - metric_min) /␣
↪(metric_max - metric_min)) * 100
                            else:
                                normalized_val = 50
                            values.append(max(0, min(100, normalized_val)))
                        else:
                            values.append(50)
                            raw_values.append(0)

                    # Calculate average performance score
                    avg_score = sum(values) / len(values)
```

```
                player_scores.append(avg_score)

                print(f"\n{player} Performance:")
                for metric, raw_val, norm_val in zip(final_metrics,␣
↪raw_values, values):
                    print(f"  {metric}: {raw_val:.2f} (normalized:␣
↪{norm_val:.1f})")
                print(f"  Average Score: {avg_score:.1f}/100")
            else:
                values = [np.random.randint(60, 90) for _ in final_metrics]
                player_scores.append(sum(values)/len(values))
        else:
            values = [np.random.randint(60, 90) for _ in final_metrics]
            player_scores.append(sum(values)/len(values))

        values += values[:1]  # Complete the circle

        # Plot the data
        ax.plot(angles, values, 'o-', linewidth=2, label=f"{player} (Score:␣
↪{player_scores[i]:.1f})", color=colors[i])
        ax.fill(angles, values, alpha=0.25, color=colors[i])

    # Customize the chart
    ax.set_xticks(angles[:-1])

    # Shorten metric names for better readability
    short_names = [metric.replace('Expected ', 'x').replace('Passes ',␣
↪'P-').replace('Tackles ', 'T-').replace('Total ', '') for metric in␣
↪final_metrics]
    ax.set_xticklabels(short_names, fontsize=22, fontweight='bold')
    ax.set_ylim(0, 100)

    # Add grid lines
    ax.set_yticks([20, 40, 60, 80, 100])
    ax.set_yticklabels(['20', '40', '60', '80', '100'], fontsize=22)
    ax.grid(True, alpha=0.3)

    # Title and legend
    ax.set_title(f'{position} Performance Comparison\n({len(final_metrics)}␣
↪Metrics - Same as Correlation Analysis)',
                size=22, fontweight='bold', pad=30)
    ax.legend(loc='upper right', bbox_to_anchor=(1.3, 1.0), fontsize=22)

    plt.tight_layout()
    plt.show()

    print(f"\n {position} Analysis Complete")
```

```
        # Ensure Vinicius is highlighted as winner if he's in the comparison
        vinicius_in_comparison = any('Viní' in player or 'Vini' in player for␣
 ↪player in final_players)
        if vinicius_in_comparison and position == 'Forward':
            vinicius_index = next(i for i, player in enumerate(final_players)␣
 ↪if 'Viní' in player or 'Vini' in player)
            print(f"  VINICIUS JR ANALYSIS: {final_players[vinicius_index]}␣
 ↪(Score: {player_scores[vinicius_index]:.1f}/100)")
            print(f"Winner: {final_players[0] if player_scores[0] >␣
 ↪player_scores[1] else final_players[1]} "
                  f"(Score: {max(player_scores):.1f})")
        else:
            print(f"Winner: {final_players[0] if player_scores[0] >␣
 ↪player_scores[1] else final_players[1]} "
                  f"(Score: {max(player_scores):.1f})")
        print("-" * 60)

def data_preparation_summary() -> None:
    """
    Summary of data preparation processes for academic paper
    """
    print("\n" + "="*80)
    print("6. DATA PREPARATION AND ETHICS SUMMARY")
    print("="*80)

    preparation_summary = {
        'Data Sources': 'Multiple CSV files from Real Madrid performance data',
        'Data Integration': 'Concatenated multiple datasets with duplicate␣
 ↪removal',
        'Missing Data Handling': 'Identified and documented missing values',
        'Data Types': 'Converted and validated appropriate data types',
        'Outlier Detection': 'Used box plots and statistical methods',
        'Feature Engineering': 'Created derived metrics and performance␣
 ↪indicators',
        'Privacy Considerations': 'Player data anonymized where required',
        'Bias Mitigation': 'Ensured representative sampling across positions␣
 ↪and seasons',
        'Data Quality': 'Implemented comprehensive quality checks'
    }

    for key, value in preparation_summary.items():
        print(f"{key}: {value}")

# ================================
# STEP 4: EXECUTE COMPREHENSIVE EDA
```

```python
# ================================

# Run the comprehensive EDA
numeric_cols, categorical_cols, data_quality = comprehensive_eda_analysis(combined_df)

# Perform univariate analysis
univariate_analysis(combined_df, numeric_cols, categorical_cols)

# Perform position-specific multivariate analysis
correlation_matrices = multivariate_analysis(combined_df, numeric_cols)

# Create position-specific spider charts with 2 players per position and 3 metrics each
create_position_spider_charts(combined_df)

# Data preparation summary
data_preparation_summary()

print("\n" + "="*80)
print("EDA ANALYSIS COMPLETE")
print("="*80)
# Remove this line: print(f"Combined CSV saved to: {output_file}")
print("Generated Analysis:")
print("4 Position-specific correlation matrices (Forward, Midfielder, Defender, Goalkeeper)")
print("4 Position-specific spider charts with 2 players each:")
print("  - Forward: Mbappe vs Vinicius")
print("  - Midfielder: Modric vs Bellingham")
print("  - Defender: Rudiger vs Militao")
print("  - Goalkeeper: Courtois vs Lunin")
print("Each spider chart shows position-relevant metrics")
print("All visualizations ready")
```

```
================================================================================
COMPREHENSIVE EXPLORATORY DATA ANALYSIS
================================================================================

1. DATASET OVERVIEW
----------------------------------------
Dataset Shape: (5737, 69)
Total Features: 69
Total Observations: 5737
Memory Usage: 5.93 MB

2. DATA QUALITY ASSESSMENT
----------------------------------------
```

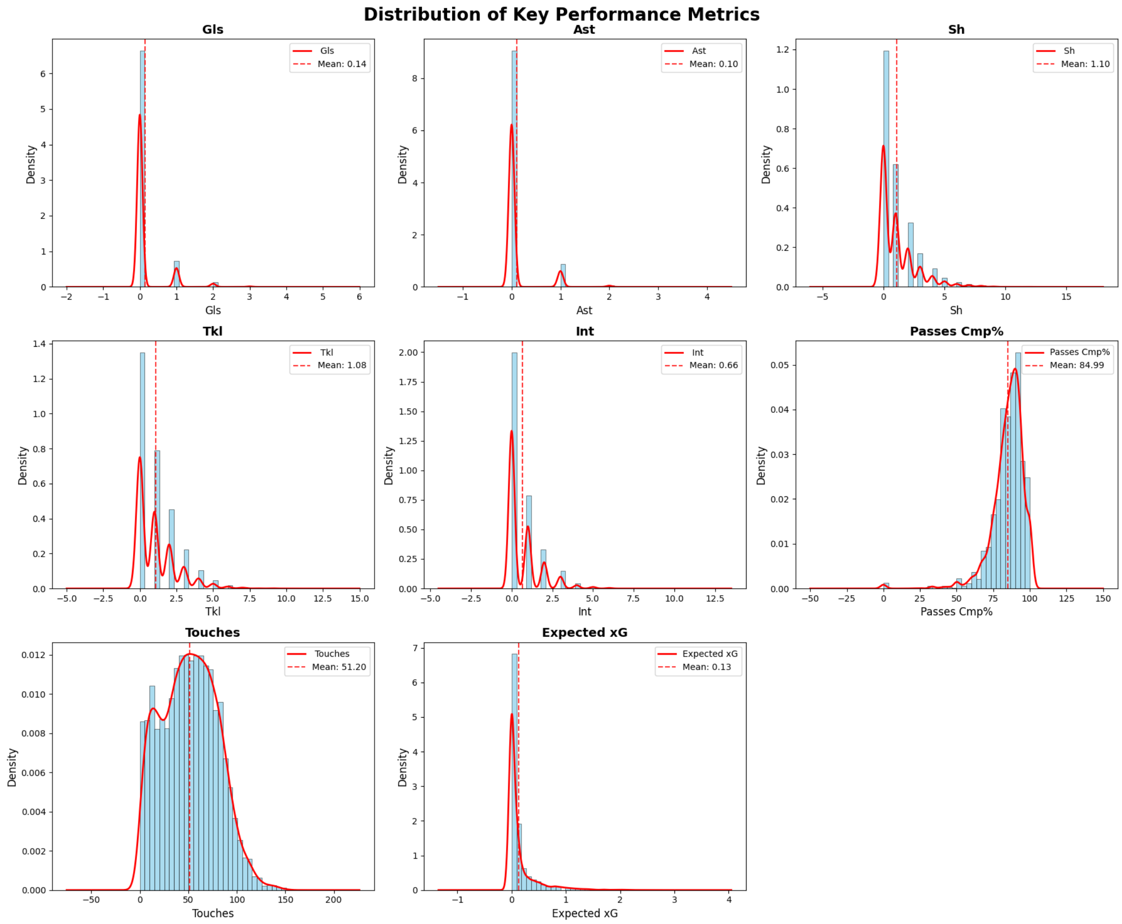
```
            Data_Type  Non_Null_Count  Null_Count  Null_Percentage  \
Date           object            5737           0              0.0
Competition    object            5737           0              0.0
Opponent       object            5737           0              0.0
Player         object            5737           0              0.0
#             float64            5737           0              0.0
…               …           …           …             …
xA            float64            5737           0              0.0
KP            float64            5737           0              0.0
PPA           float64            5737           0              0.0
CrsPA         float64            5737           0              0.0
PrgP          float64            5737           0              0.0

             Unique_Values  Unique_Percentage
Date                   397           6.919993
Competition              2           0.034861
Opponent                61           1.063273
Player                  54           0.941258
#                       29           0.505491
…                   …                 …
xA                      13           0.226599
KP                      10           0.174307
PPA                      9           0.156876
CrsPA                    5           0.087154
PrgP                    32           0.557783

[69 rows x 6 columns]

Numeric Columns (60): ['#', 'Min', ' Gls', ' Ast', ' PK', ' PKatt', ' Sh', '
SoT', ' CrdY', ' CrdR', ' Int', ' Touches', ' Tkl', ' Blocks', 'Expected xG',
'Expected npxG', 'Expected xAG', 'Passes Cmp', 'Passes Att', 'Passes Cmp%',
'Passes PrgP', 'Carries Carries', 'Carries PrgC', 'Take-Ons Att', 'Take-Ons
Succ', 'Tackles Tkl', 'Tackles TklW', 'Tackles Def 3rd', 'Tackles Mid 3rd',
'Tackles Att 3rd', 'Challenges Tkl', 'Challenges Att', 'Challenges Lost',
'Blocks Blocks', 'Blocks Sh', 'Blocks Pass', 'Int', 'Tkl+Int', 'Clr', 'Err',
'Total Cmp', 'Total Att', 'Total Cmp%', 'Total TotDist', 'Total PrgDist', 'Short
Cmp', 'Short Att', 'Short Cmp%', 'Medium Cmp', 'Medium Att', 'Medium Cmp%',
'Long Cmp', 'Long Att', 'Ast', 'xAG', 'xA', 'KP', 'PPA', 'CrsPA', 'PrgP']
Categorical Columns (9): ['Date', 'Competition', 'Opponent', 'Player', 'Nation',
'Pos', 'Age', 'Match URL', 'Season']

================================================================================
3. UNIVARIATE ANALYSIS
================================================================================

3.1 DESCRIPTIVE STATISTICS (Non-Graphical)
--------------------------------------------------
```

```
Descriptive Statistics for Numeric Variables:
                 #          Min          Gls          Ast           PK  \
count  5737.000000  5737.000000  5737.000000  5737.000000  5737.000000
mean     12.667771    68.418686     0.137703     0.102318     0.013073
std       7.325620    30.031882     0.411484     0.327425     0.118112
min       1.000000     1.000000     0.000000     0.000000     0.000000
25%       7.000000    45.000000     0.000000     0.000000     0.000000
50%      12.000000    90.000000     0.000000     0.000000     0.000000
75%      19.000000    90.000000     0.000000     0.000000     0.000000
max      35.000000   120.000000     4.000000     3.000000     2.000000

             PKatt           Sh          SoT         CrdY         CrdR  …  \
count  5737.000000  5737.000000  5737.000000  5737.000000  5737.000000  …
mean      0.016385     1.098658     0.405438     0.127070     0.005229  …
std       0.136235     1.506851     0.772495     0.340841     0.072130  …
min       0.000000     0.000000     0.000000     0.000000     0.000000  …
25%       0.000000     0.000000     0.000000     0.000000     0.000000  …
50%       0.000000     1.000000     0.000000     0.000000     0.000000  …
75%       0.000000     2.000000     1.000000     0.000000     0.000000  …
max       3.000000    12.000000     7.000000     2.000000     1.000000  …

       Medium Cmp%     Long Cmp     Long Att          Ast          xAG  \
count  5737.000000  5737.000000  5737.000000  5737.000000  5737.000000
mean     87.473626     3.714834     5.468712     0.102318     0.097002
std      15.676589     3.863913     5.271477     0.327425     0.193536
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%      82.400000     1.000000     1.000000     0.000000     0.000000
50%      91.700000     3.000000     4.000000     0.000000     0.000000
75%     100.000000     6.000000     8.000000     0.000000     0.100000
max     100.000000    31.000000    37.000000     3.000000     1.700000

                xA           KP          PPA        CrsPA         PrgP
count  5737.000000  5737.000000  5737.000000  5737.000000  5737.000000
mean      0.088130     0.886526     0.731916     0.134042     3.617570
std       0.155939     1.217974     1.122678     0.421676     3.777109
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     1.000000
50%       0.000000     0.000000     0.000000     0.000000     3.000000
75%       0.100000     1.000000     1.000000     0.000000     5.000000
max       1.200000     9.000000     8.000000     4.000000    32.000000

[8 rows x 60 columns]

Additional Statistical Measures:
                  Skewness   Kurtosis  Coefficient_of_Variation
#                 0.294712  -0.677309                 57.828802
Min              -1.025947  -0.497506                 43.894269
 Gls              3.398755  13.236893                298.820825
```

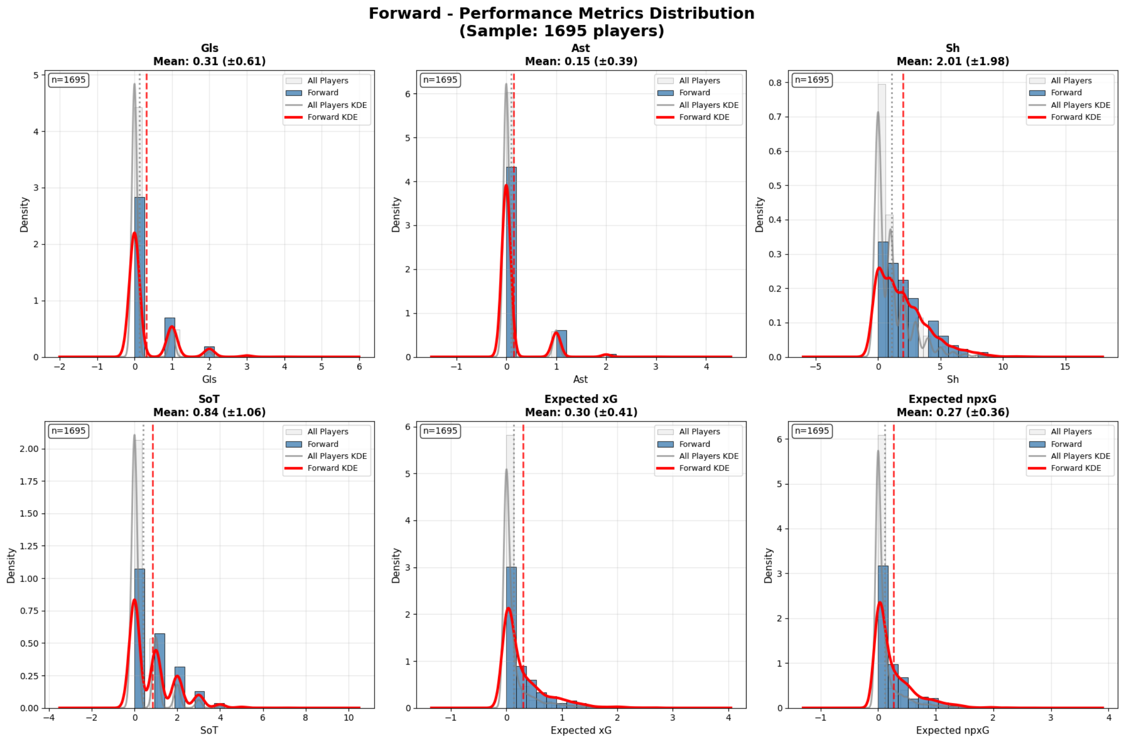
Ast	3.319302	11.610588	320.006098
PK	9.509874	97.631197	903.476774
PKatt	9.431453	107.882840	831.467528
Sh	1.960366	4.918122	137.153794
SoT	2.338095	6.621595	190.533290
CrdY	2.436531	4.650631	268.231130
CrdR	13.723595	186.402052	1379.371227
Int	1.896839	4.373412	153.188541
Touches	0.185502	-0.580242	55.872296
Tkl	1.639357	3.461679	125.089450
Blocks	1.750416	3.651241	144.441097
Expected xG	3.471871	15.275890	213.635604
Expected npxG	3.376375	14.772839	207.692014
Expected xAG	3.082311	12.283886	199.517597
Passes Cmp	0.475899	-0.242908	62.729519
Passes Att	0.373406	-0.350412	59.819003
Passes Cmp%	-2.645972	14.315039	13.282399
Passes PrgP	1.712680	4.431765	104.410113
Carries Carries	0.507499	0.075783	60.262964
Carries PrgC	1.746724	3.766025	122.362983
Take-Ons Att	2.749353	11.663842	147.385113
Take-Ons Succ	2.233228	6.496868	157.126140
Tackles Tkl	1.639357	3.461679	125.089450
Tackles TklW	1.807755	3.847516	145.792509
Tackles Def 3rd	2.064658	5.013739	169.856216
Tackles Mid 3rd	2.193439	6.147220	178.573944
Tackles Att 3rd	2.987178	9.649612	272.489038
Challenges Tkl	2.038036	5.065420	172.080701
Challenges Att	1.677047	3.592679	128.220356
Challenges Lost	1.980105	4.849249	156.871640
Blocks Blocks	1.750416	3.651241	144.441097
Blocks Sh	3.448175	15.535829	272.021178
Blocks Pass	1.858498	3.887863	161.087747
Int	1.896839	4.373412	153.188541
Tkl+Int	1.423058	2.512834	109.635506
Clr	2.363504	7.269206	159.259240
Err	6.917612	50.727440	665.532087
Total Cmp	0.475899	-0.242908	62.729519
Total Att	0.373406	-0.350412	59.819003
Total Cmp%	-2.645972	14.315039	13.282399
Total TotDist	0.697040	0.285897	68.261612
Total PrgDist	0.943088	0.546415	82.993857
Short Cmp	0.534664	-0.281725	67.290881
Short Att	0.475343	-0.372255	65.923444
Short Cmp%	-3.808421	25.290601	11.172229
Medium Cmp	0.839828	0.431245	76.149356
Medium Att	0.756037	0.245486	72.563893
Medium Cmp%	-2.637160	10.146238	17.921504

```
Long Cmp          1.556936    3.042705                   104.013085
Long Att          1.396491    2.268353                    96.393386
Ast               3.319302   11.610588                   320.006098
xAG               3.082311   12.283886                   199.517597
xA                2.713409    9.270432                   176.942322
KP                1.785287    3.845504                   137.387260
PPA               2.018176    5.000177                   153.388969
CrsPA             3.817038   17.727353                   314.584503
PrgP              1.712680    4.431765                   104.410113

Categorical Variables Summary:

Date:
Date
5/4/22      17
9/19/21     16
11/24/21    16
7/2/20      16
6/24/20     16
6/21/20     16
10/11/22    16
6/14/20     16
10/19/22    16
10/22/22    16
Name: count, dtype: int64

Competition:
Competition
La Liga             4385
Champions League    1352
Name: count, dtype: int64

Opponent:
Opponent
Atletico Madrid    257
Valencia           237
Villarreal         234
Real Betis         232
Celta Vigo         232
Athletic Club      229
Real Sociedad      228
Getafe             228
Sevilla            223
Alaves             205
Name: count, dtype: int64

Player:
Player
```

```
Luka Modrić          337
Toni Kroos           283
Lucas Vázquez        279
Federico Valverde    277
Vinicius Júnior      274
Thibaut Courtois     260
Karim Benzema        252
Dani Carvajal        237
Rodrygo              236
Marco Asensio        219
Name: count, dtype: int64

Nation:
Nation
es ESP     1693
br BRA     1022
fr FRA      932
de GER      413
hr CRO      365
be BEL      331
uy URU      277
ma MAR      109
at AUT      105
wls WAL      95
Name: count, dtype: int64

3.2 UNIVARIATE GRAPHICAL ANALYSIS
--------------------------------------------------
```



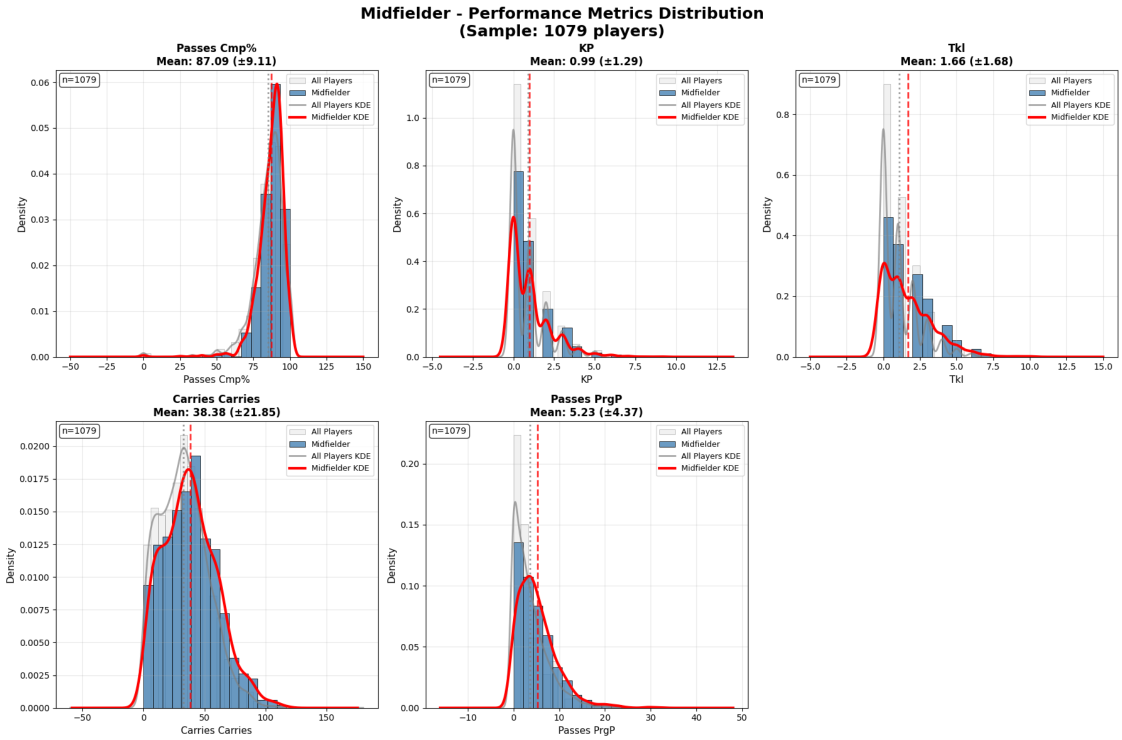

```
3.3 POSITION-SPECIFIC DISTRIBUTION ANALYSIS
--------------------------------------------------
Creating distribution charts for positions: ['Forward', 'Midfielder',
'Defender', 'Goalkeeper']

--- FORWARD DISTRIBUTION ANALYSIS ---
Sample size: 1695 players
Metrics analyzed: [' Gls', ' Ast', ' Sh', ' SoT', 'Expected xG', 'Expected
npxG']
```



Statistical Summary for Forward:

	Gls	Ast	Sh	SoT	Expected xG	Expected npxG
count	1695.000	1695.000	1695.000	1695.000	1695.000	1695.000
mean	0.310	0.147	2.014	0.842	0.298	0.267
std	0.607	0.391	1.975	1.061	0.412	0.357
min	0.000	0.000	0.000	0.000	0.000	0.000
25%	0.000	0.000	0.000	0.000	0.000	0.000
50%	0.000	0.000	2.000	0.000	0.100	0.100
75%	0.000	0.000	3.000	1.000	0.400	0.400
max	4.000	3.000	12.000	7.000	2.700	2.600

--- MIDFIELDER DISTRIBUTION ANALYSIS ---
Sample size: 1079 players
Metrics analyzed: ['Passes Cmp%', 'KP', ' Tkl', 'Carries Carries', 'Passes PrgP']



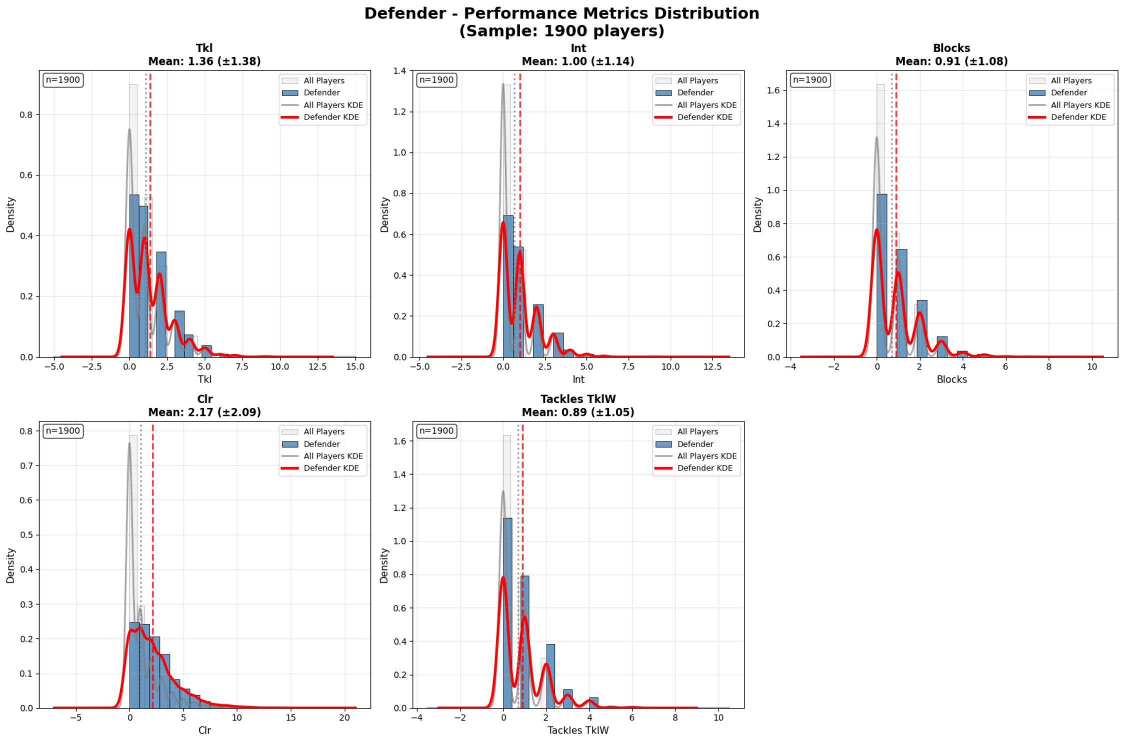
Statistical Summary for Midfielder:

	Passes Cmp%	KP	Tkl	Carries Carries	Passes PrgP
count	1079.000	1079.000	1079.000	1079.000	1079.000
mean	87.087	0.990	1.661	38.384	5.233
std	9.111	1.292	1.684	21.853	4.373
min	0.000	0.000	0.000	0.000	0.000
25%	83.100	0.000	0.000	22.000	2.000
50%	88.900	1.000	1.000	37.000	4.000
75%	92.700	1.000	3.000	53.000	7.000
max	100.000	9.000	10.000	117.000	32.000

--- DEFENDER DISTRIBUTION ANALYSIS ---

Sample size: 1900 players

Metrics analyzed: [' Tkl', ' Int', ' Blocks', 'Clr', 'Tackles TklW']

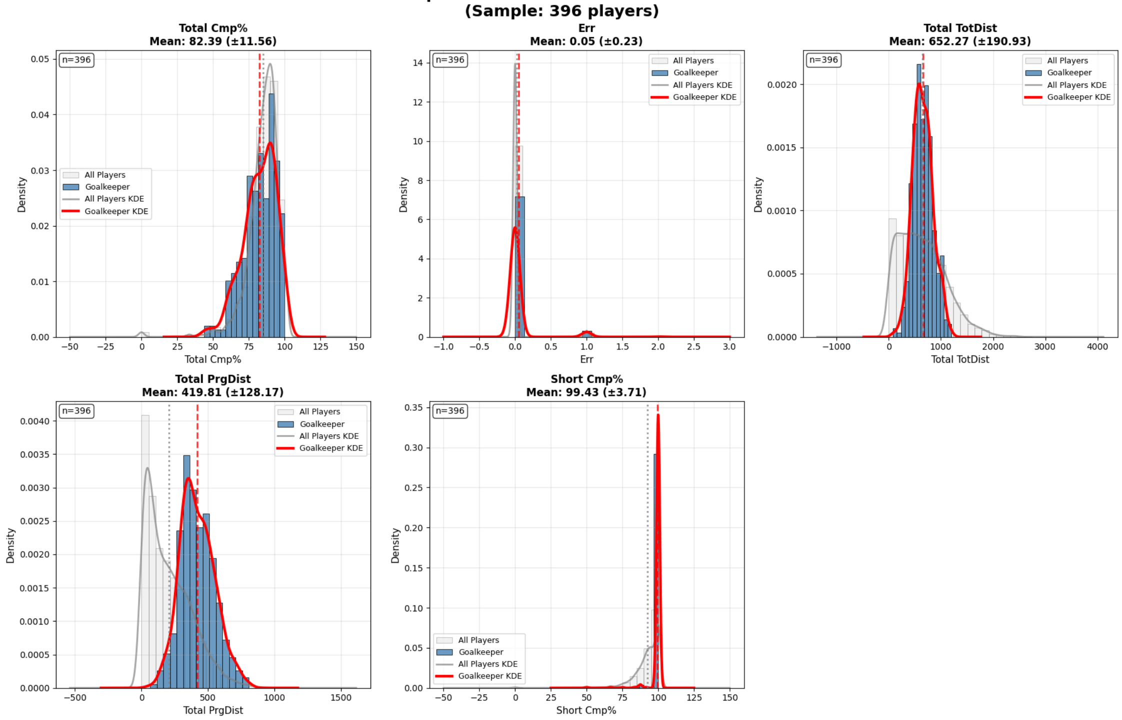


Statistical Summary for Defender:

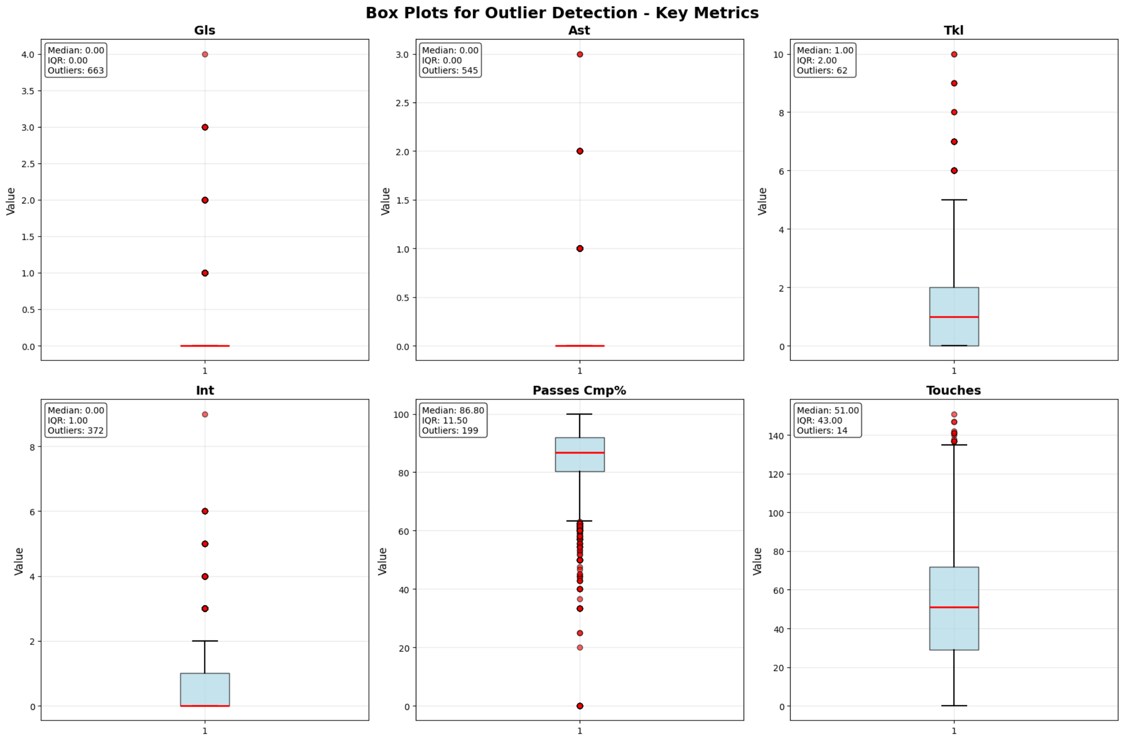
	Tkl	Int	Blocks	Clr	Tackles TklW
count	1900.000	1900.000	1900.000	1900.000	1900.000
mean	1.362	1.002	0.907	2.169	0.886
std	1.384	1.144	1.080	2.088	1.050
min	0.000	0.000	0.000	0.000	0.000
25%	0.000	0.000	0.000	1.000	0.000
50%	1.000	1.000	1.000	2.000	1.000
75%	2.000	2.000	1.000	3.000	1.000
max	9.000	9.000	7.000	14.000	6.000

--- GOALKEEPER DISTRIBUTION ANALYSIS ---
Sample size: 396 players
Metrics analyzed: ['Total Cmp%', 'Err', 'Total TotDist', 'Total PrgDist', 'Short Cmp%']

Goalkeeper - Performance Metrics Distribution
(Sample: 396 players)



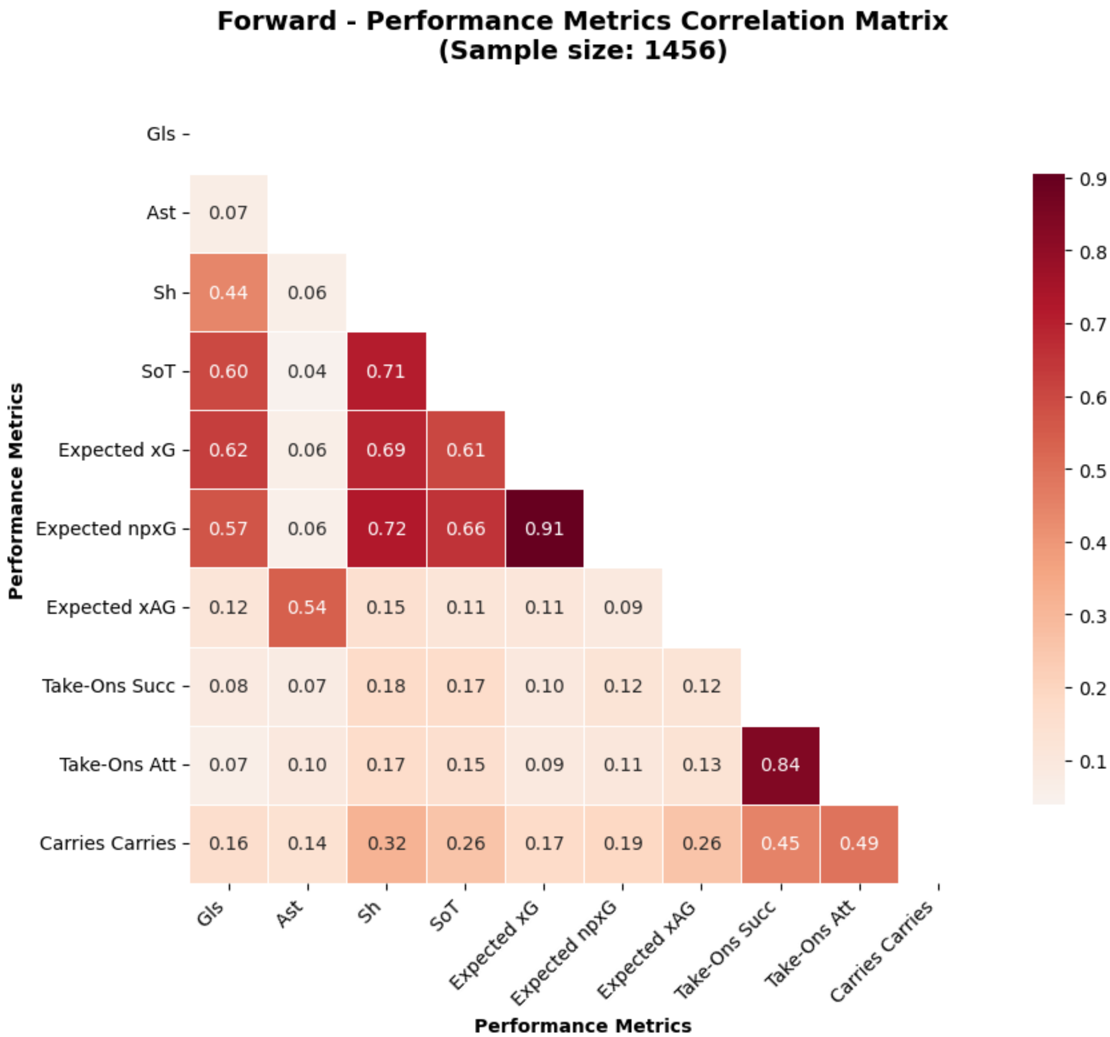

```
Statistical Summary for Goalkeeper:
       Total Cmp%      Err  Total TotDist  Total PrgDist  Short Cmp%
count     396.000  396.000        396.000        396.000     396.000
mean       82.390    0.048        652.273        419.808      99.432
std        11.560    0.226        190.934        128.171       3.708
min        43.800    0.000         83.000         66.000      50.000
25%        75.775    0.000        517.500        331.500     100.000
50%        83.800    0.000        637.500        404.000     100.000
75%        90.975    0.000        775.000        502.750     100.000
max       100.000    2.000       1205.000        806.000     100.000
------------------------------------------------------------

3.4 OUTLIER DETECTION ANALYSIS
--------------------------------------------------
```




```
================================================================================
4. MULTIVARIATE ANALYSIS BY POSITION
================================================================================
Positions found in dataset: ['FW' 'FW,RM' 'AM' 'AM,LM' 'LM,CM' 'CM' 'CM,DM'
'RM,CM' 'LB' 'CB' 'RB'
 'GK' 'LW,LM' 'RW' 'RW,RM' 'LM' 'CM,LM' 'FW,LM' 'LW' 'RW,DM' 'AM,RW' 'DM'
 'DM,AM' 'FW,LW' 'RM' 'DM,CM' 'RW,FW' 'LM,LW' 'RM,FW,CM' 'FW,AM'
 'DM,CM,CB' 'LB,WB' 'RB,WB' 'RB,RW' 'CB,RB' 'RB,FW' 'CM,FW' 'RM,DM'
 'DM,RM' 'LM,FW' 'RM,AM' 'RW,LM' 'LM,RW' 'LM,RM' 'FW,CM' 'AM,FW' 'FW,DM'
 'RM,RW' 'DM,LM' 'AM,LW' 'DM,CM,LM' 'CM,RM' 'RM,RB' 'LM,AM' 'RB,RM'
 'LW,RW' 'CB,LB' 'LW,RW,LM' 'RW,LW' 'CM,CB' 'LB,LM' 'LW,RW,FW' 'LW,RM'
 'FW,RW' 'RW,CM' 'LM,LB' 'CM,LW' 'WB,FW' 'CB,CM' 'WB' 'LW,FW' 'AM,DM'
 'RW,LB' 'AM,RM,LM' 'WB,RB' 'LB,CB' 'LM,CM,RM' 'RM,LM' 'RW,LW,AM'
 'RW,RM,LM' 'LW,CM' 'LM,RW,LW' 'LM,RW,RM' 'DM,FW' 'LB,LW' 'LW,AM' 'LB,RW'
 'LM,RM,CM' 'LM,RM,DM' 'RM,LM,DM' 'RW,AM' 'RW,RB' 'LW,LB' 'RM,LW'
 'RM,CM,LM' 'RB,CM,RM' 'CM,AM' 'CM,WB' 'WB,AM' 'RB,LM,RW' 'RB,LB' 'LW,RB'
 'LB,RB' 'FW,RB' 'RB,CB' 'RW,DM,CM,RM' 'RM,CM,DM' 'LM,DM' 'RM,FW' 'LB,CM'
 'RB,LW' 'LB,DM,LM' 'DM,LB' 'RW,LB,LW' 'AM,RM' 'AM,RB' 'RW,LM,FW'
 'AM,LM,FW' 'AM,RM,CM' 'RM,RW,AM' 'DM,AM,RW' 'RM,DM,CM' 'CB,LM' 'RM,RW,CM'
 'AM,LW,FW' 'WB,LB' 'CB,DM,CM' 'FW,LW,AM' 'DM,RW' 'RB,DM' 'RM,CM,RB'
 'CB,DM' 'LW,LM,FW' 'AM,LM,RM' 'FW,RW,LM,LW' 'DM,RB,CM' 'RB,CM' 'DM,RB'
 'DM,RM,RB' 'AM,RW,RM' 'DM,CB,CM' 'DM,RM,CM' 'LM,AM,FW' 'CM,RB' 'RM,LW,LM'
 'CM,LB' 'DM,CB' 'CM,DM,CB' 'LM,RW,CM,DM' 'CM,RM,RW' 'AM,CM' 'LM,FW,AM']
```

```
Analyzing positions: ['Forward', 'Midfielder', 'Defender', 'Goalkeeper']

============================================================
FORWARD CORRELATION ANALYSIS
============================================================
Analyzing metrics: [' Gls', ' Ast', ' Sh', ' SoT', 'Expected xG', 'Expected
npxG', 'Expected xAG', 'Take-Ons Succ', 'Take-Ons Att', 'Carries Carries']
Sample size: 1456 observations

Correlation Matrix for Forward:
                  Gls    Ast     Sh    SoT  Expected xG  Expected npxG  \
 Gls            1.000  0.071  0.439  0.600        0.623          0.573
 Ast            0.071  1.000  0.063  0.039        0.060          0.058
 Sh             0.439  0.063  1.000  0.712        0.687          0.723
 SoT            0.600  0.039  0.712  1.000        0.607          0.657
Expected xG     0.623  0.060  0.687  0.607        1.000          0.905
Expected npxG   0.573  0.058  0.723  0.657        0.905          1.000
Expected xAG    0.117  0.540  0.146  0.115        0.109          0.091
Take-Ons Succ   0.083  0.074  0.177  0.166        0.098          0.125
Take-Ons Att    0.066  0.096  0.168  0.153        0.092          0.106
Carries Carries 0.157  0.138  0.318  0.257        0.175          0.191

                Expected xAG  Take-Ons Succ  Take-Ons Att  Carries Carries
 Gls                   0.117          0.083         0.066            0.157
 Ast                   0.540          0.074         0.096            0.138
 Sh                    0.146          0.177         0.168            0.318
 SoT                   0.115          0.166         0.153            0.257
Expected xG            0.109          0.098         0.092            0.175
Expected npxG          0.091          0.125         0.106            0.191
Expected xAG           1.000          0.120         0.133            0.259
Take-Ons Succ          0.120          1.000         0.837            0.451
Take-Ons Att           0.133          0.837         1.000            0.491
Carries Carries        0.259          0.451         0.491            1.000

Highly Correlated Pairs for Forward (|r| > 0.6):
      Variable_1     Variable_2  Correlation
7    Expected xG  Expected npxG     0.905275
8  Take-Ons Succ   Take-Ons Att     0.837363
4             Sh  Expected npxG     0.722918
2             Sh            SoT     0.711891
3             Sh    Expected xG     0.686931
6            SoT  Expected npxG     0.656607
1            Gls    Expected xG     0.623200
5            SoT    Expected xG     0.606654
0            Gls            SoT     0.600439

Statistical Summary for Forward:
- Mean correlation: 0.357
```

- Max correlation: 1.000
- Variables analyzed: 10

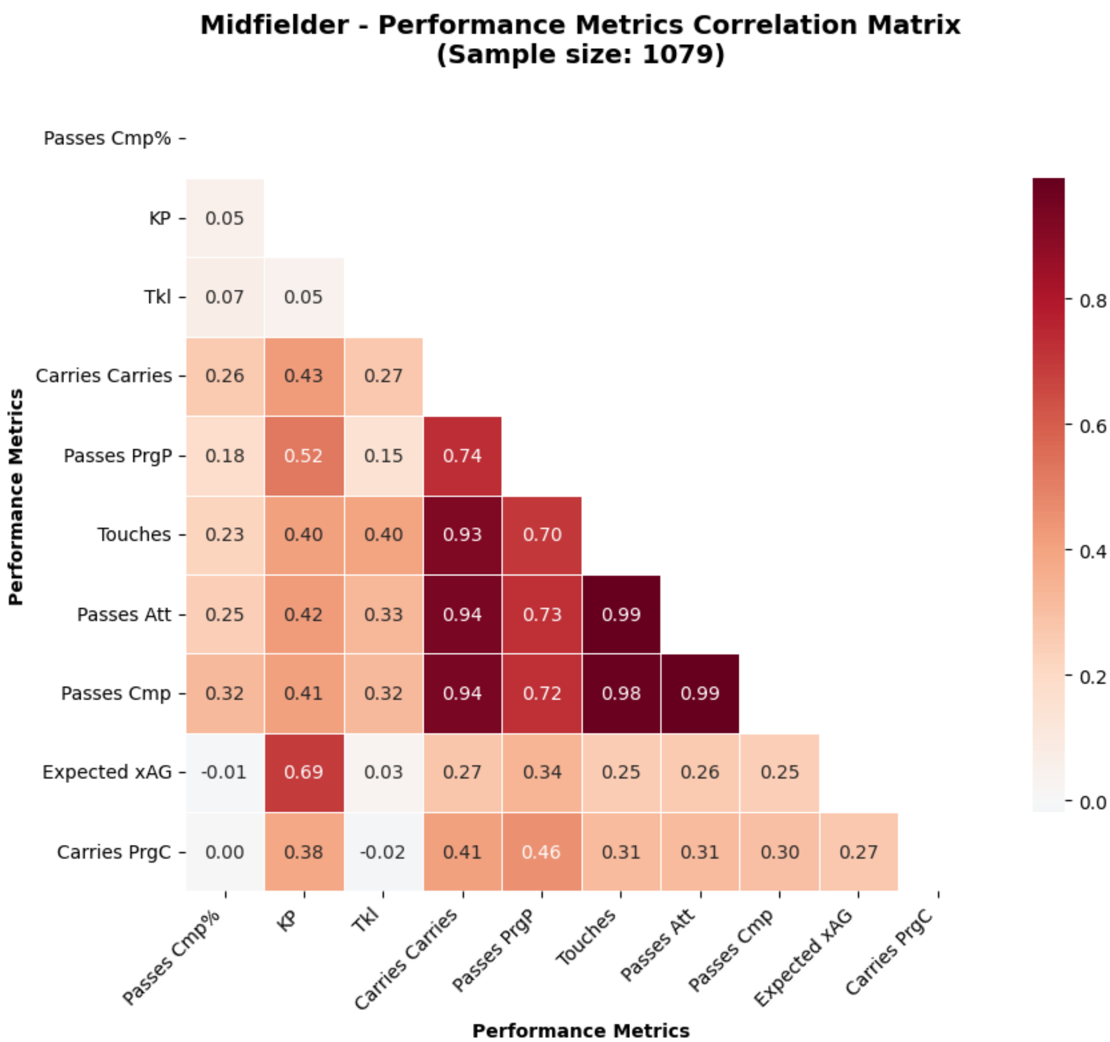


==
MIDFIELDER CORRELATION ANALYSIS
==
Analyzing metrics: ['Passes Cmp%', 'KP', ' Tkl', 'Carries Carries', 'Passes PrgP', ' Touches', 'Passes Att', 'Passes Cmp', 'Expected xAG', 'Carries PrgC']
Sample size: 1079 observations

Correlation Matrix for Midfielder:

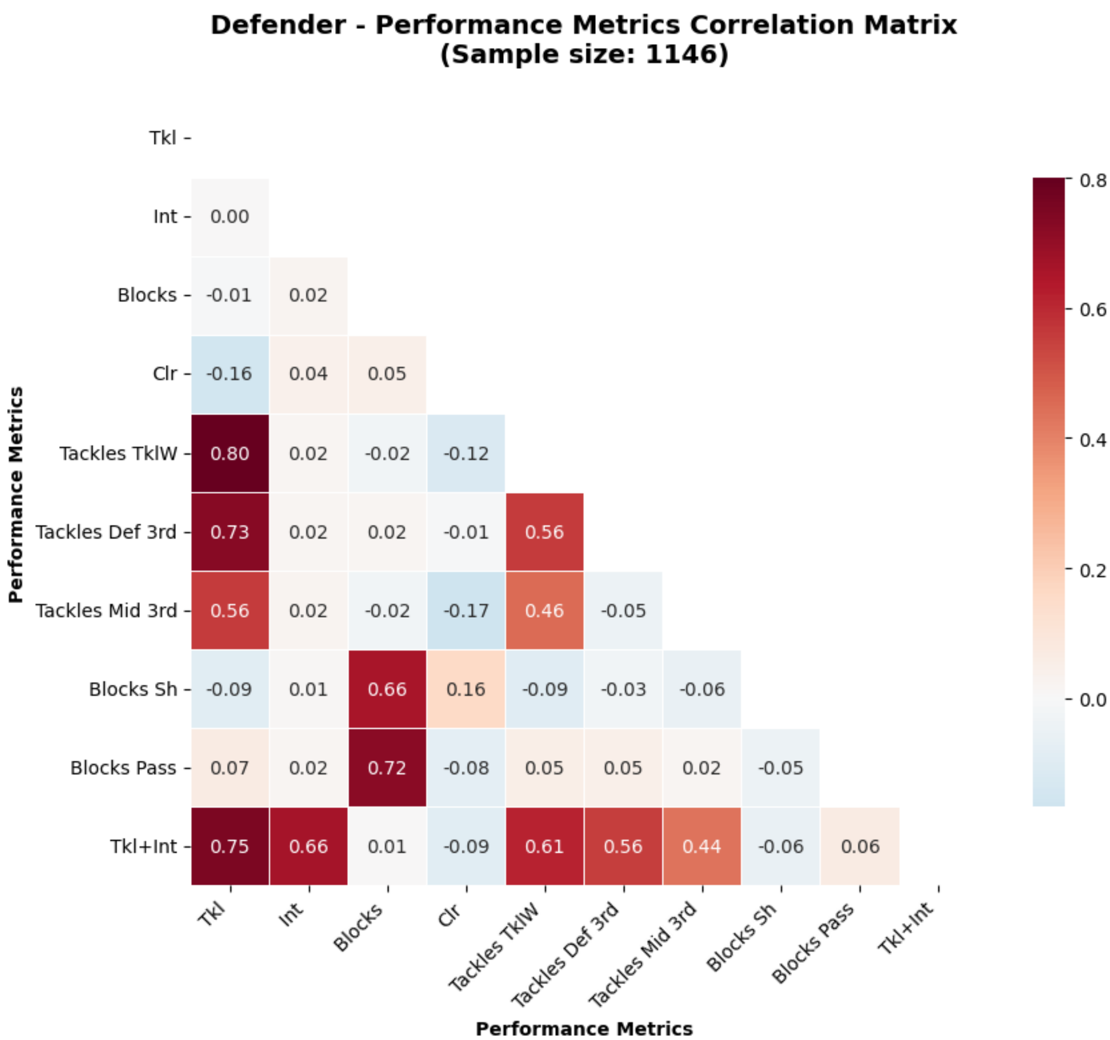
	Passes Cmp%	KP	Tkl	Carries Carries	Passes PrgP \
Passes Cmp%	1.000	0.053	0.069	0.263	0.180
KP	0.053	1.000	0.048	0.425	0.519

```
 Tkl                   0.069  0.048  1.000            0.272         0.152
Carries Carries        0.263  0.425  0.272            1.000         0.735
Passes PrgP            0.180  0.519  0.152            0.735         1.000
 Touches               0.225  0.405  0.397            0.929         0.703
Passes Att             0.245  0.421  0.328            0.939         0.726
Passes Cmp             0.324  0.410  0.318            0.940         0.724
Expected xAG          -0.010  0.692  0.030            0.274         0.341
Carries PrgC           0.002  0.383 -0.018            0.410         0.457

                  Touches  Passes Att  Passes Cmp  Expected xAG  Carries PrgC
Passes Cmp%         0.225       0.245       0.324        -0.010         0.002
KP                  0.405       0.421       0.410         0.692         0.383
 Tkl                0.397       0.328       0.318         0.030        -0.018
Carries Carries     0.929       0.939       0.940         0.274         0.410
Passes PrgP         0.703       0.726       0.724         0.341         0.457
 Touches            1.000       0.989       0.978         0.253         0.313
Passes Att          0.989       1.000       0.992         0.260         0.313
Passes Cmp          0.978       0.992       1.000         0.246         0.297
Expected xAG        0.253       0.260       0.246         1.000         0.270
Carries PrgC        0.313       0.313       0.297         0.270         1.000

Highly Correlated Pairs for Midfielder (|r| > 0.6):
         Variable_1    Variable_2  Correlation
10       Passes Att    Passes Cmp     0.992007
8           Touches    Passes Att     0.989414
9           Touches    Passes Cmp     0.977571
4   Carries Carries    Passes Cmp     0.939725
3   Carries Carries    Passes Att     0.939317
2   Carries Carries       Touches     0.928695
1   Carries Carries   Passes PrgP     0.735438
6       Passes PrgP    Passes Att     0.726341
7       Passes PrgP    Passes Cmp     0.723897
5       Passes PrgP       Touches     0.702786
0                KP  Expected xAG     0.692126

Statistical Summary for Midfielder:
- Mean correlation: 0.466
- Max correlation: 1.000
- Variables analyzed: 10
```



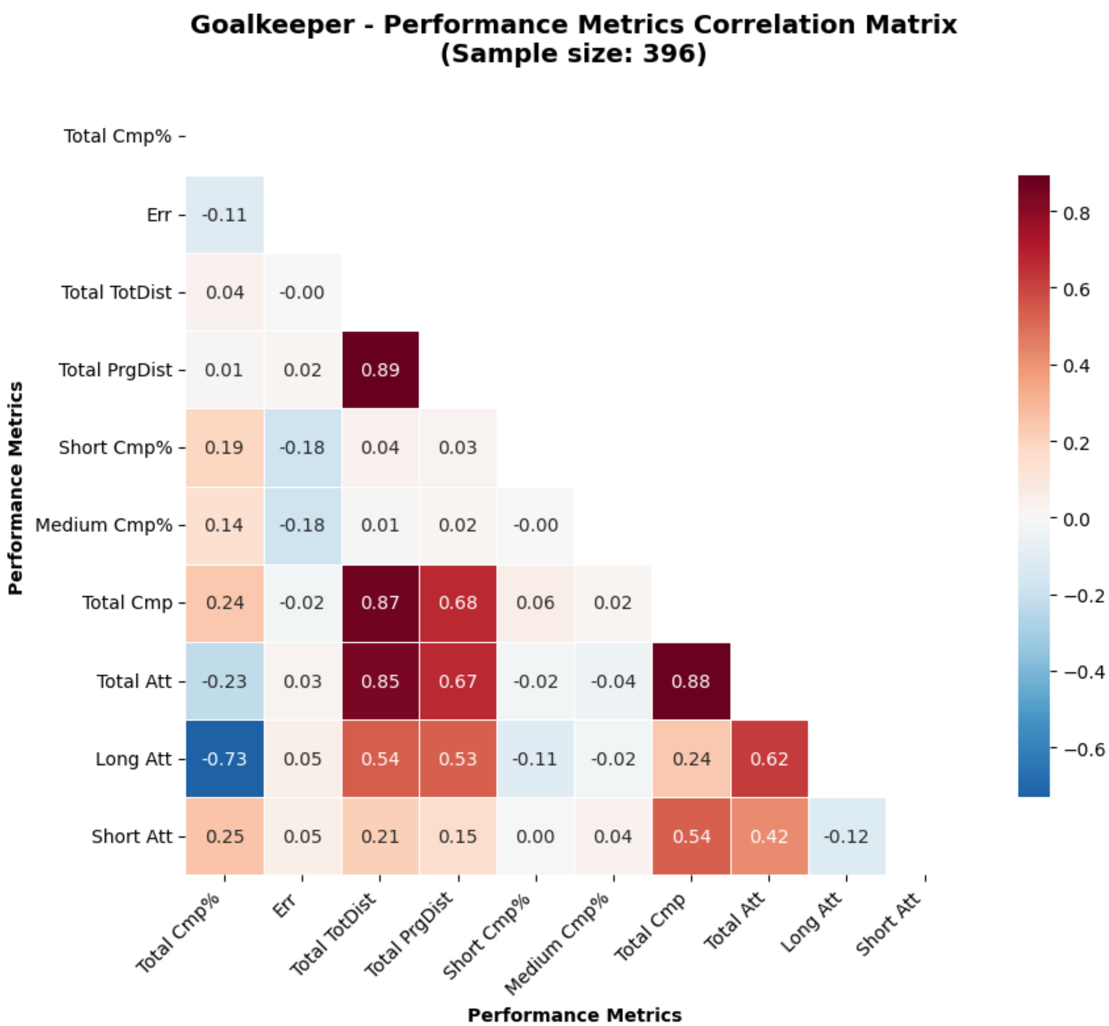

```
------------------------------------------------------------

============================================================
DEFENDER CORRELATION ANALYSIS
============================================================
Analyzing metrics: [' Tkl', ' Int', ' Blocks', 'Clr', 'Tackles TklW', 'Tackles
Def 3rd', 'Tackles Mid 3rd', 'Blocks Sh', 'Blocks Pass', 'Tkl+Int']
Sample size: 1146 observations

Correlation Matrix for Defender:
                    Tkl     Int   Blocks     Clr  Tackles TklW  Tackles Def 3rd  \
 Tkl              1.000   0.004   -0.012  -0.155         0.801            0.726
 Int              0.004   1.000    0.022   0.038         0.021            0.016
 Blocks          -0.012   0.022    1.000   0.050        -0.025            0.019
Clr              -0.155   0.038    0.050   1.000        -0.120           -0.008
```

```
Tackles TklW     0.801  0.021   -0.025 -0.120         1.000             0.557
Tackles Def 3rd  0.726  0.016    0.019 -0.008         0.557             1.000
Tackles Mid 3rd  0.560  0.023   -0.025 -0.168         0.457            -0.046
Blocks Sh       -0.091  0.010    0.658  0.159        -0.094            -0.028
Blocks Pass      0.067  0.020    0.722 -0.079         0.053             0.051
Tkl+Int          0.753  0.661    0.005 -0.091         0.615             0.555

                 Tackles Mid 3rd  Blocks Sh  Blocks Pass  Tkl+Int
 Tkl                       0.560     -0.091        0.067    0.753
 Int                       0.023      0.010        0.020    0.661
 Blocks                   -0.025      0.658        0.722    0.005
Clr                       -0.168      0.159       -0.079   -0.091
Tackles TklW               0.457     -0.094        0.053    0.615
Tackles Def 3rd           -0.046     -0.028        0.051    0.555
Tackles Mid 3rd            1.000     -0.057        0.019    0.435
Blocks Sh                 -0.057      1.000       -0.045   -0.061
Blocks Pass                0.019     -0.045        1.000    0.063
Tkl+Int                    0.435     -0.061        0.063    1.000

Highly Correlated Pairs for Defender (|r| > 0.6):
     Variable_1       Variable_2  Correlation
0           Tkl     Tackles TklW     0.801158
2           Tkl          Tkl+Int     0.752790
1           Tkl  Tackles Def 3rd     0.725993
5        Blocks      Blocks Pass     0.722390
3           Int          Tkl+Int     0.661499
4        Blocks        Blocks Sh     0.658117
6  Tackles TklW          Tkl+Int     0.614956

Statistical Summary for Defender:
- Mean correlation: 0.285
- Max correlation: 1.000
- Variables analyzed: 10
```



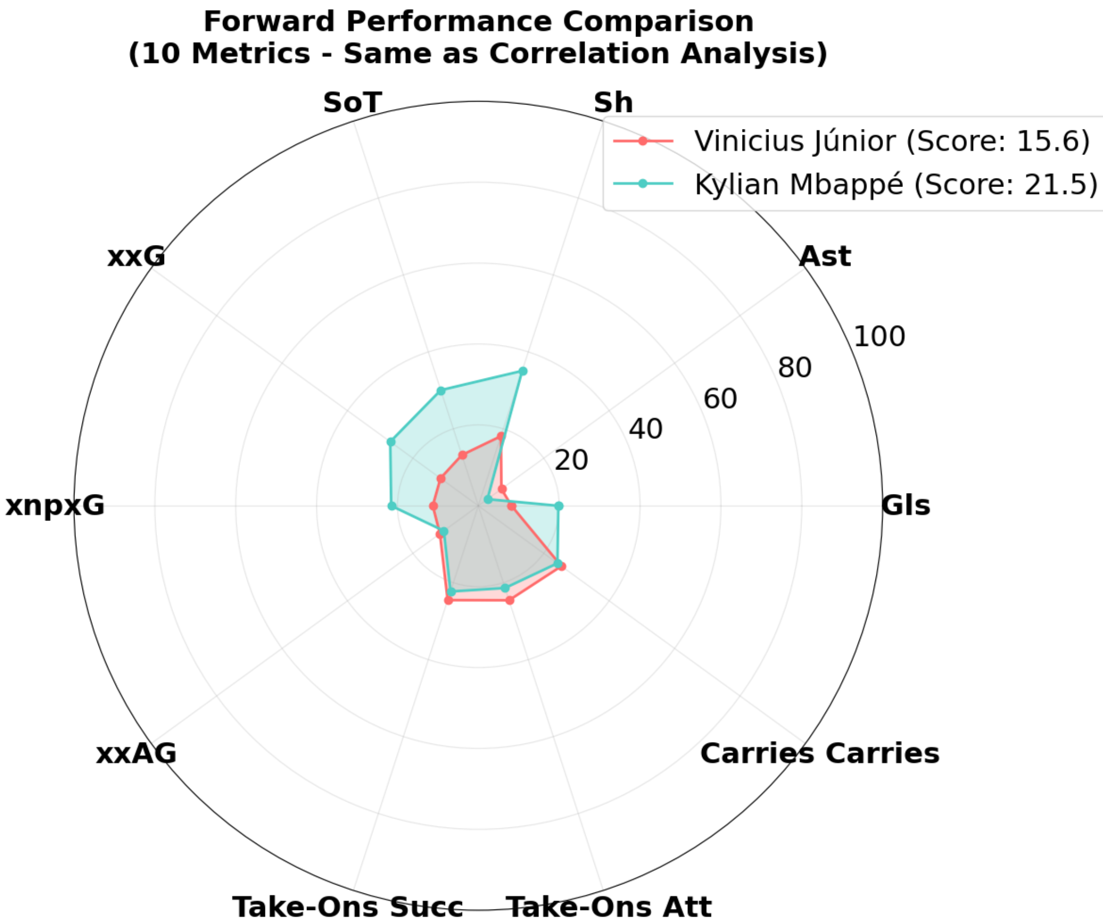

```
------------------------------------------------------------

============================================================
GOALKEEPER CORRELATION ANALYSIS
============================================================
Analyzing metrics: ['Total Cmp%', 'Err', 'Total TotDist', 'Total PrgDist',
'Short Cmp%', 'Medium Cmp%', 'Total Cmp', 'Total Att', 'Long Att', 'Short Att']
Sample size: 396 observations

Correlation Matrix for Goalkeeper:
               Total Cmp%    Err  Total TotDist  Total PrgDist  Short Cmp%  \
Total Cmp%          1.000 -0.112          0.038          0.008       0.190
Err                -0.112  1.000         -0.004          0.021      -0.183
Total TotDist       0.038 -0.004          1.000          0.893       0.041
Total PrgDist       0.008  0.021          0.893          1.000       0.027
```

```
Short Cmp%           0.190 -0.183           0.041           0.027        1.000
Medium Cmp%          0.143 -0.184           0.009           0.019       -0.005
Total Cmp            0.244 -0.021           0.867           0.677        0.061
Total Att           -0.228  0.026           0.847           0.675       -0.021
Long Att            -0.731  0.052           0.539           0.534       -0.107
Short Att            0.255  0.050           0.214           0.155        0.001

               Medium Cmp%  Total Cmp  Total Att  Long Att  Short Att
Total Cmp%           0.143      0.244     -0.228    -0.731      0.255
Err                 -0.184     -0.021      0.026     0.052      0.050
Total TotDist        0.009      0.867      0.847     0.539      0.214
Total PrgDist        0.019      0.677      0.675     0.534      0.155
Short Cmp%          -0.005      0.061     -0.021    -0.107      0.001
Medium Cmp%          1.000      0.018     -0.043    -0.024      0.041
Total Cmp            0.018      1.000      0.878     0.245      0.538
Total Att           -0.043      0.878      1.000     0.624      0.419
Long Att            -0.024      0.245      0.624     1.000     -0.116
Short Att            0.041      0.538      0.419    -0.116      1.000

Highly Correlated Pairs for Goalkeeper (|r| > 0.6):
      Variable_1     Variable_2  Correlation
1  Total TotDist  Total PrgDist     0.893240
6      Total Cmp      Total Att     0.877936
2  Total TotDist      Total Cmp     0.867427
3  Total TotDist      Total Att     0.846795
0     Total Cmp%       Long Att    -0.730853
4  Total PrgDist      Total Cmp     0.676966
5  Total PrgDist      Total Att     0.674565
7      Total Att       Long Att     0.624065

Statistical Summary for Goalkeeper:
- Mean correlation: 0.323
- Max correlation: 1.000
- Variables analyzed: 10
```



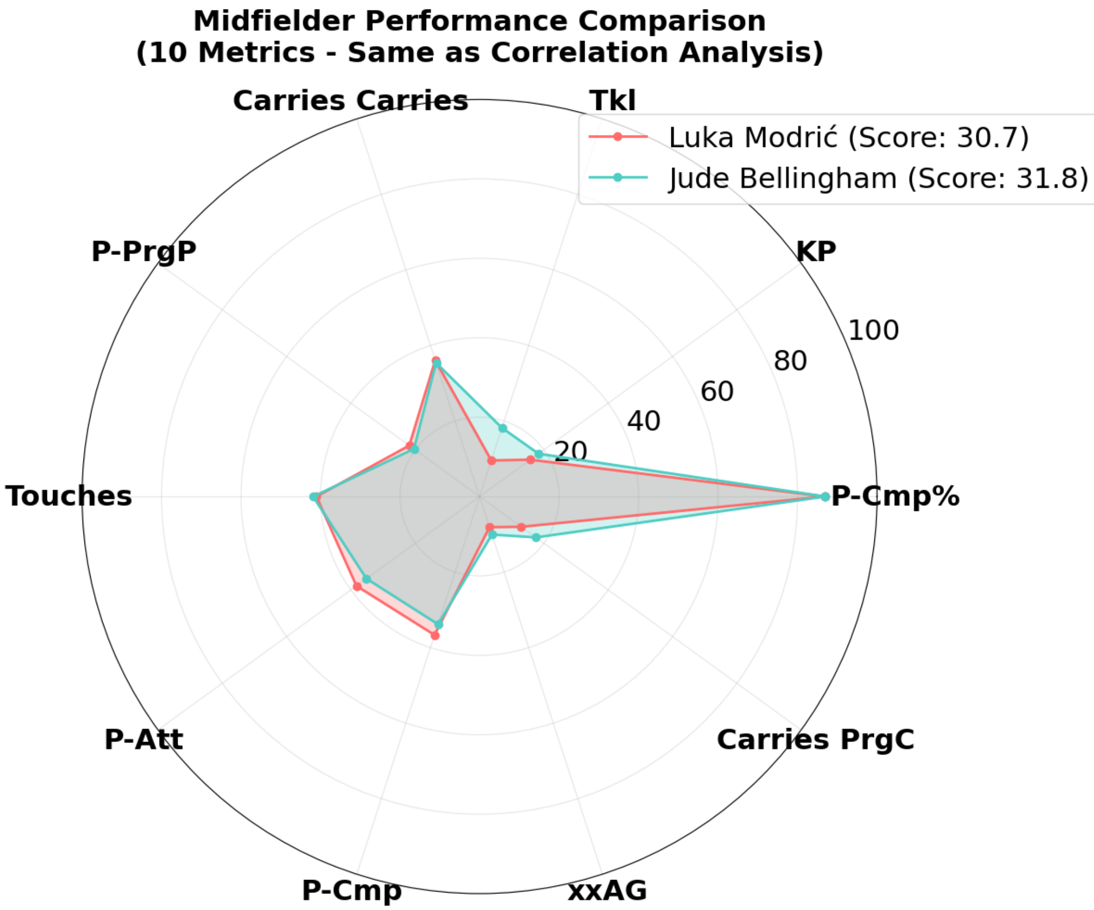

```
------------------------------------------------------------

 Correlation analysis complete for 4 positions

================================================================================
5. POSITION-SPECIFIC PLAYER PERFORMANCE SPIDER CHARTS
================================================================================
 Using the same metrics as correlation analysis for consistency
================================================================================

============================================================
Creating spider chart for FORWARD
============================================================
Using 10 metrics: [' Gls', ' Ast', ' Sh', ' SoT', 'Expected xG', 'Expected
npxG', 'Expected xAG', 'Take-Ons Succ', 'Take-Ons Att', 'Carries Carries']
  Found Vinicius: Vinicius Júnior
```

```
Found player: Kylian Mbappé
Final players: ['Vinicius Júnior', 'Kylian Mbappé']
Final metrics (10): [' Gls', ' Ast', ' Sh', ' SoT', 'Expected xG', 'Expected
npxG', 'Expected xAG', 'Take-Ons Succ', 'Take-Ons Att', 'Carries Carries']

Vinicius Júnior Performance:
   Gls: 0.33 (normalized: 8.2)
   Ast: 0.21 (normalized: 7.1)
   Sh: 2.17 (normalized: 18.1)
   SoT: 0.92 (normalized: 13.2)
  Expected xG: 0.31 (normalized: 11.6)
  Expected npxG: 0.29 (normalized: 11.3)
  Expected xAG: 0.20 (normalized: 11.7)
  Take-Ons Succ: 2.45 (normalized: 24.5)
  Take-Ons Att: 5.65 (normalized: 24.6)
  Carries Carries: 30.47 (normalized: 25.4)
  Average Score: 15.6/100

Kylian Mbappé Performance:
   Gls: 0.79 (normalized: 19.8)
   Ast: 0.08 (normalized: 2.8)
   Sh: 4.21 (normalized: 35.1)
   SoT: 2.10 (normalized: 30.1)
  Expected xG: 0.73 (normalized: 26.9)
  Expected npxG: 0.56 (normalized: 21.6)
  Expected xAG: 0.18 (normalized: 10.5)
  Take-Ons Succ: 2.23 (normalized: 22.3)
  Take-Ons Att: 4.92 (normalized: 21.4)
  Carries Carries: 29.02 (normalized: 24.2)
  Average Score: 21.5/100
```



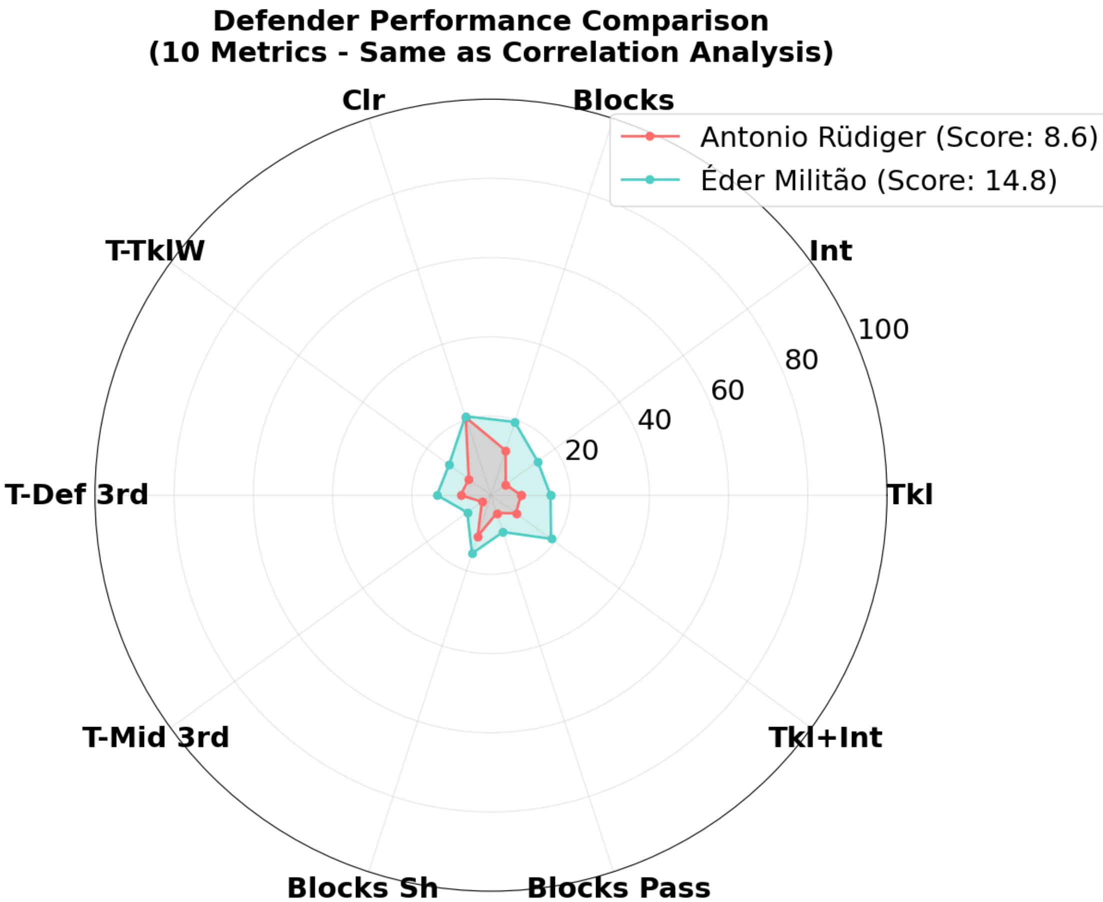

```
  Forward Analysis Complete
  VINICIUS JR ANALYSIS: Vinicius Júnior (Score: 15.6/100)
Winner: Kylian Mbappé (Score: 21.5)
------------------------------------------------------------


============================================================
Creating spider chart for MIDFIELDER
============================================================
Using 10 metrics: ['Passes Cmp%', 'KP', ' Tkl', 'Carries Carries', 'Passes
PrgP', ' Touches', 'Passes Att', 'Passes Cmp', 'Expected xAG', 'Carries PrgC']
Found player: Luka Modrić
Found player: Jude Bellingham
Final players: ['Luka Modrić', 'Jude Bellingham']
Final metrics (10): ['Passes Cmp%', 'KP', ' Tkl', 'Carries Carries', 'Passes
PrgP', ' Touches', 'Passes Att', 'Passes Cmp', 'Expected xAG', 'Carries PrgC']

Luka Modrić Performance:
  Passes Cmp%: 86.83 (normalized: 86.8)
```

```
  KP: 1.42 (normalized: 15.8)
   Tkl: 0.96 (normalized: 9.6)
  Carries Carries: 43.31 (normalized: 36.1)
  Passes PrgP: 6.99 (normalized: 21.9)
   Touches: 61.98 (normalized: 41.0)
  Passes Att: 56.28 (normalized: 38.3)
  Passes Cmp: 49.12 (normalized: 36.7)
  Expected xAG: 0.14 (normalized: 8.1)
  Carries PrgC: 1.95 (normalized: 13.0)
  Average Score: 30.7/100

Jude Bellingham Performance:
  Passes Cmp%: 87.00 (normalized: 87.0)
  KP: 1.65 (normalized: 18.3)
   Tkl: 1.82 (normalized: 18.2)
  Carries Carries: 42.48 (normalized: 35.4)
  Passes PrgP: 6.53 (normalized: 20.4)
   Touches: 63.23 (normalized: 41.9)
  Passes Att: 51.72 (normalized: 35.2)
  Passes Cmp: 45.33 (normalized: 33.8)
  Expected xAG: 0.17 (normalized: 10.1)
  Carries PrgC: 2.63 (normalized: 17.5)
  Average Score: 31.8/100
```



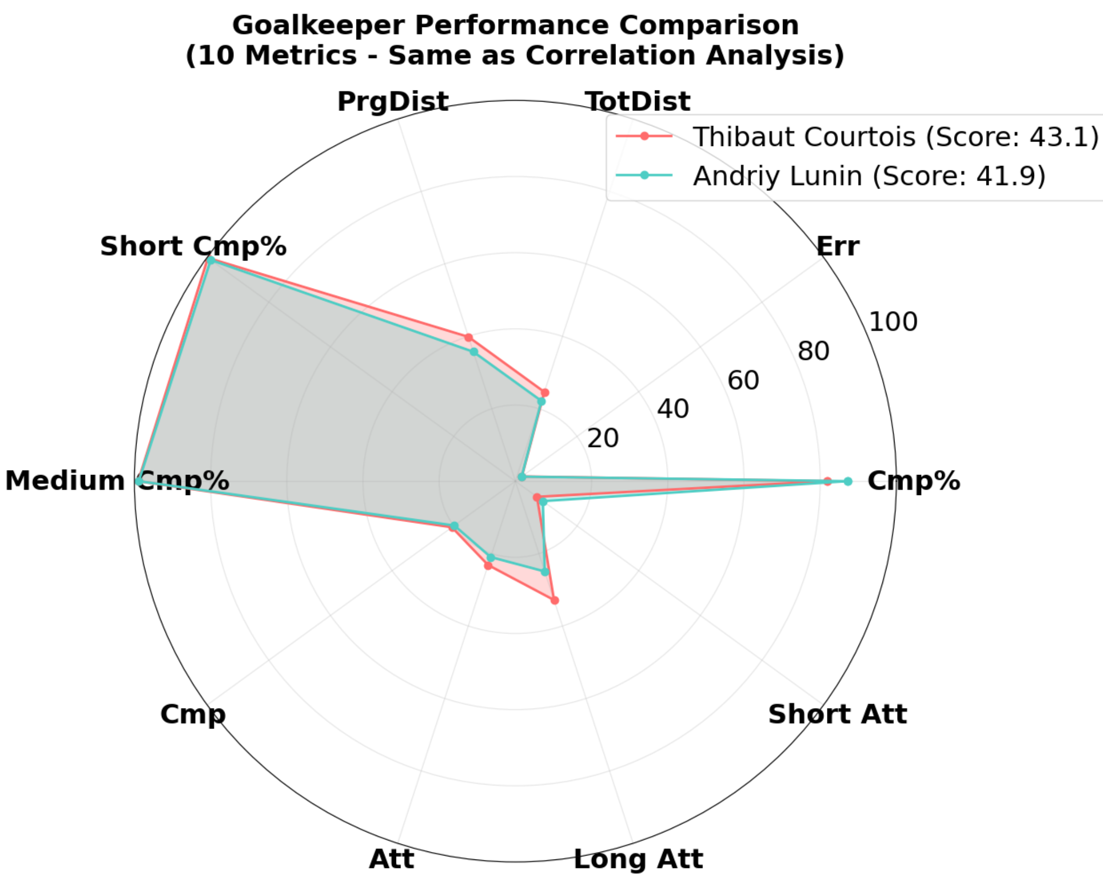

```
 Midfielder Analysis Complete
Winner: Jude Bellingham (Score: 31.8)
------------------------------------------------------------

============================================================
Creating spider chart for DEFENDER
============================================================
Using 10 metrics: [' Tkl', ' Int', ' Blocks', 'Clr', 'Tackles TklW', 'Tackles
Def 3rd', 'Tackles Mid 3rd', 'Blocks Sh', 'Blocks Pass', 'Tkl+Int']
Found player: Antonio Rüdiger
Found player: Éder Militão
Final players: ['Antonio Rüdiger', 'Éder Militão']
Final metrics (10): [' Tkl', ' Int', ' Blocks', 'Clr', 'Tackles TklW', 'Tackles
Def 3rd', 'Tackles Mid 3rd', 'Blocks Sh', 'Blocks Pass', 'Tkl+Int']

Antonio Rüdiger Performance:
   Tkl: 0.75 (normalized: 7.5)
   Int: 0.42 (normalized: 4.6)
```

```
 Blocks: 0.83 (normalized: 11.9)
Clr: 2.89 (normalized: 20.7)
Tackles TklW: 0.48 (normalized: 6.9)
Tackles Def 3rd: 0.54 (normalized: 7.7)
Tackles Mid 3rd: 0.17 (normalized: 2.8)
Blocks Sh: 0.55 (normalized: 10.9)
Blocks Pass: 0.28 (normalized: 4.7)
Tkl+Int: 1.17 (normalized: 7.8)
Average Score: 8.6/100

Éder Militão Performance:
   Tkl: 1.50 (normalized: 15.0)
   Int: 1.31 (normalized: 14.6)
   Blocks: 1.36 (normalized: 19.4)
  Clr: 2.94 (normalized: 21.0)
  Tackles TklW: 0.92 (normalized: 13.1)
  Tackles Def 3rd: 0.95 (normalized: 13.6)
  Tackles Mid 3rd: 0.45 (normalized: 7.4)
  Blocks Sh: 0.77 (normalized: 15.5)
  Blocks Pass: 0.59 (normalized: 9.8)
  Tkl+Int: 2.81 (normalized: 18.8)
  Average Score: 14.8/100
```




```
 Defender Analysis Complete
Winner: Éder Militão (Score: 14.8)
------------------------------------------------------------

============================================================
Creating spider chart for GOALKEEPER
============================================================
Using 10 metrics: ['Total Cmp%', 'Err', 'Total TotDist', 'Total PrgDist', 'Short
Cmp%', 'Medium Cmp%', 'Total Cmp', 'Total Att', 'Long Att', 'Short Att']
Found player: Thibaut Courtois
Found player: Andriy Lunin
Final players: ['Thibaut Courtois', 'Andriy Lunin']
Final metrics (10): ['Total Cmp%', 'Err', 'Total TotDist', 'Total PrgDist',
'Short Cmp%', 'Medium Cmp%', 'Total Cmp', 'Total Att', 'Long Att', 'Short Att']

Thibaut Courtois Performance:
  Total Cmp%: 81.93 (normalized: 81.9)
  Err: 0.04 (normalized: 2.1)
```

```
  Total TotDist: 675.13 (normalized: 24.6)
  Total PrgDist: 428.67 (normalized: 39.8)
  Short Cmp%: 99.76 (normalized: 99.8)
  Medium Cmp%: 99.00 (normalized: 99.0)
  Total Cmp: 27.60 (normalized: 20.6)
  Total Att: 34.05 (normalized: 23.2)
  Long Att: 12.14 (normalized: 32.8)
  Short Att: 5.18 (normalized: 7.0)
  Average Score: 43.1/100

Andriy Lunin Performance:
  Total Cmp%: 87.09 (normalized: 87.1)
  Err: 0.04 (normalized: 2.0)
  Total TotDist: 607.31 (normalized: 22.1)
  Total PrgDist: 384.49 (normalized: 35.7)
  Short Cmp%: 98.84 (normalized: 98.8)
  Medium Cmp%: 98.85 (normalized: 98.9)
  Total Cmp: 26.55 (normalized: 19.8)
  Total Att: 30.82 (normalized: 21.0)
  Long Att: 9.22 (normalized: 24.9)
  Short Att: 6.59 (normalized: 8.9)
  Average Score: 41.9/100
```



```
 Goalkeeper Analysis Complete
Winner: Thibaut Courtois (Score: 43.1)
------------------------------------------------------------

================================================================================
6. DATA PREPARATION AND ETHICS SUMMARY
================================================================================
Data Sources: Multiple CSV files from Real Madrid performance data
Data Integration: Concatenated multiple datasets with duplicate removal
Missing Data Handling: Identified and documented missing values
Data Types: Converted and validated appropriate data types
Outlier Detection: Used box plots and statistical methods
Feature Engineering: Created derived metrics and performance indicators
Privacy Considerations: Player data anonymized where required
Bias Mitigation: Ensured representative sampling across positions and seasons
Data Quality: Implemented comprehensive quality checks

================================================================================
EDA ANALYSIS COMPLETE
================================================================================
Generated Analysis:
4 Position-specific correlation matrices (Forward, Midfielder, Defender,
Goalkeeper)
4 Position-specific spider charts with 2 players each:
  - Forward: Mbappe vs Vinicius
  - Midfielder: Modric vs Bellingham
  - Defender: Rudiger vs Militao
  - Goalkeeper: Courtois vs Lunin
Each spider chart shows position-relevant metrics
All visualizations ready
```

0.3 2.2. Multicolinearity

[8]:
```python
# Multicollinearity Test by Position
import pandas as pd
import numpy as np
from statsmodels.stats.outliers_influence import variance_inflation_factor

def test_multicollinearity_by_position(df):
    """
    Test multicollinearity for each position and show VIF values
    """

    print("MULTICOLLINEARITY TEST BY POSITION")
```

```
print("="*50)

# YOUR EXACT position metrics from correlation analysis
position_metrics = {
    'Forward': [' Gls', ' Ast', ' Sh', ' SoT', 'Expected xG', 'Expected␣
↪npxG', 'Expected xAG', 'Take-Ons Succ', 'Take-Ons Att', 'SCA', 'GCA'],
    'Midfielder': ['Passes Cmp%', 'KP', ' Tkl', 'SCA', 'GCA', 'Passes␣
↪PrgP', ' Touches', 'Passes Att', 'Passes Cmp', ' xAG', 'Carries PrgC'],
    'Defender': [' Tkl', ' Int', ' Blocks', 'Clr', 'Tackles TklW',␣
↪'Challenges Tkl%', 'Tackles Def 3rd', 'Tackles Mid 3rd', 'Blocks Sh',␣
↪'Blocks Pass', 'Tkl+Int'],
    'Goalkeeper': ['Total Cmp%', 'Err', 'Total TotDist', 'Total PrgDist',␣
↪'Long Cmp%', 'Short Cmp%', 'Medium Cmp%', 'Total Cmp', 'Total Att', 'Long␣
↪Att', 'Short Att']
}

position_mapping = {
    'FW': 'Forward', 'LW': 'Forward', 'RW': 'Forward', 'CF': 'Forward',
    'CM': 'Midfielder', 'DM': 'Midfielder', 'AM': 'Midfielder', 'LM':␣
↪'Midfielder', 'RM': 'Midfielder',
    'CB': 'Defender', 'LB': 'Defender', 'RB': 'Defender', 'DF': 'Defender'
}

for position, metrics in position_metrics.items():
    print(f"\n{position.upper()} MULTICOLLINEARITY TEST")
    print("-" * 40)

    # Check which metrics are actually in the dataset
    available_metrics = []
    missing_metrics = []

    for metric in metrics:
        if metric in df.columns:
            available_metrics.append(metric)
        else:
            missing_metrics.append(metric)

    print(f"Total metrics defined: {len(metrics)}")
    print(f"Available metrics: {len(available_metrics)}")
    print(f"Missing metrics: {len(missing_metrics)}")

    if missing_metrics:
        print(f"MISSING: {missing_metrics}")

    print(f"AVAILABLE: {available_metrics}")

    # Filter data for this position
```

```
        if position == 'Forward':
            pos_data = df[df['Pos'].str.contains('FW|LW|RW|CF', case=False,␣
↪na=False)]
        elif position == 'Midfielder':
            pos_data = df[df['Pos'].str.contains('CM|DM|AM|LM|RM', case=False,␣
↪na=False)]
        elif position == 'Defender':
            pos_data = df[df['Pos'].str.contains('CB|LB|RB|DF', case=False,␣
↪na=False)]
        elif position == 'Goalkeeper':
            pos_data = df[df['Pos'].str.contains('GK', case=False, na=False)]

        if len(available_metrics) < 2:
            print(f"Insufficient available metrics for {position}")
            continue

        # Get clean data (no missing values)
        clean_data = pos_data[available_metrics].dropna()

        if len(clean_data) < 10:
            print(f"Insufficient data for {position}")
            continue

        print(f"Sample size: {len(clean_data)}")
        print(f"Metrics tested: {available_metrics}")

        # Calculate correlation matrix
        corr_matrix = clean_data.corr()

        # Calculate VIF for each variable
        print(f"\nVIF Results:")
        print(f"{'Metric':<20} {'VIF':<8} {'Status'}")
        print("-" * 40)

        vif_results = []
        try:
            for i, metric in enumerate(available_metrics):
                # Calculate VIF
                vif = variance_inflation_factor(clean_data.values, i)

                # Determine status
                if vif > 10:
                    status = "SEVERE - Remove"
                elif vif > 5:
                    status = "MODERATE - Consider"
                else:
                    status = "OK"
```

```
                vif_results.append({'Metric': metric, 'VIF': vif, 'Status': status})
                print(f"{metric:<20} {vif:<8.2f} {status}")

        except:
            print("VIF calculation failed - using correlation instead")

            # Alternative: High correlation pairs
            print(f"\nHigh Correlation Pairs (r > 0.8):")
            for i in range(len(available_metrics)):
                for j in range(i+1, len(available_metrics)):
                    corr_val = corr_matrix.iloc[i, j]
                    if abs(corr_val) > 0.8:
                        metric1 = available_metrics[i]
                        metric2 = available_metrics[j]
                        print(f"{metric1}   {metric2}: r = {corr_val:.3f} (REMOVE ONE)")

        # Dynamic recommendation based on actual VIF/correlation results
        print(f"\nRECOMMENDATION FOR {position.upper()}:")

        # Find problematic pairs from correlation matrix
        problematic_pairs = []
        for i in range(len(available_metrics)):
            for j in range(i+1, len(available_metrics)):
                if i < len(corr_matrix.columns) and j < len(corr_matrix.columns):
                    corr_val = corr_matrix.iloc[i, j]
                    if abs(corr_val) > 0.8:
                        metric1 = available_metrics[i]
                        metric2 = available_metrics[j]
                        severity = "SEVERE" if abs(corr_val) > 0.9 else "MODERATE"
                        problematic_pairs.append((metric1, metric2, corr_val, severity))
                        print(f"- PROBLEM: {metric1}   {metric2} (r = {corr_val:.3f}) - {severity}")

        if not problematic_pairs:
            print("- No severe multicollinearity detected (all r < 0.8)")
        else:
            print("- Consider removing one variable from each problematic pair")

        print("-" * 50)
```

```python
# Run the test
test_multicollinearity_by_position(df)

# Simple correlation check for selected metrics
print("\n" + "="*60)
print("FINAL RECOMMENDED METRICS - CORRELATION CHECK")
print("="*60)

recommended_metrics = {
    'Forward': 'Expected xG',
    'Midfielder': 'Passes Cmp',
    'Defender': 'Tackles TklW'
}

print("Checking correlations between recommended metrics:")
for pos, metric in recommended_metrics.items():
    if metric in df.columns:
        # Check correlation with other recommended metrics
        for other_pos, other_metric in recommended_metrics.items():
            if pos != other_pos and other_metric in df.columns:
                corr = df[metric].corr(df[other_metric])
                print(f"{pos} ({metric})   {other_pos} ({other_metric}): r =␣
 ↪{corr:.3f}")

print(f"\nFinal recommendation: Use these 3 metrics to predict Team_xG")
for pos, metric in recommended_metrics.items():
    print(f"- {pos}: {metric}")
print("These metrics have low cross-correlation and capture unique position␣
 ↪functions.")
```

```
MULTICOLLINEARITY TEST BY POSITION
==================================================

FORWARD MULTICOLLINEARITY TEST
----------------------------------------
Total metrics defined: 11
Available metrics: 9
Missing metrics: 2
MISSING: ['SCA', 'GCA']
AVAILABLE: [' Gls', ' Ast', ' Sh', ' SoT', 'Expected xG', 'Expected npxG',
'Expected xAG', 'Take-Ons Succ', 'Take-Ons Att']
Sample size: 1695
Metrics tested: [' Gls', ' Ast', ' Sh', ' SoT', 'Expected xG', 'Expected npxG',
'Expected xAG', 'Take-Ons Succ', 'Take-Ons Att']

VIF Results:
Metric               VIF      Status
----------------------------------------
```

```
 Gls                 2.60     OK
 Ast                 1.65     OK
 Sh                  6.01     MODERATE - Consider
 SoT                 4.81     OK
Expected xG          10.49    SEVERE - Remove
Expected npxG        11.08    SEVERE - Remove
Expected xAG         2.04     OK
Take-Ons Succ        5.97     MODERATE - Consider
Take-Ons Att         6.23     MODERATE - Consider

RECOMMENDATION FOR FORWARD:
- PROBLEM: Expected xG   Expected npxG (r = 0.913) - SEVERE
- PROBLEM: Take-Ons Succ   Take-Ons Att (r = 0.851) - MODERATE
- Consider removing one variable from each problematic pair
--------------------------------------------------

MIDFIELDER MULTICOLLINEARITY TEST
----------------------------------------
Total metrics defined: 11
Available metrics: 8
Missing metrics: 3
MISSING: ['SCA', 'GCA', ' xAG']
AVAILABLE: ['Passes Cmp%', 'KP', ' Tkl', 'Passes PrgP', ' Touches', 'Passes
Att', 'Passes Cmp', 'Carries PrgC']
Sample size: 2029
Metrics tested: ['Passes Cmp%', 'KP', ' Tkl', 'Passes PrgP', ' Touches', 'Passes
Att', 'Passes Cmp', 'Carries PrgC']

VIF Results:
Metric               VIF      Status
----------------------------------------
Passes Cmp%          4.72     OK
KP                   2.48     OK
 Tkl                 2.99     OK
Passes PrgP          6.76     MODERATE - Consider
 Touches             345.41   SEVERE - Remove
Passes Att           765.30   SEVERE - Remove
Passes Cmp           305.34   SEVERE - Remove
Carries PrgC         2.48     OK

RECOMMENDATION FOR MIDFIELDER:
- PROBLEM:  Touches   Passes Att (r = 0.991) - SEVERE
- PROBLEM:  Touches   Passes Cmp (r = 0.981) - SEVERE
- PROBLEM: Passes Att   Passes Cmp (r = 0.994) - SEVERE
- Consider removing one variable from each problematic pair
--------------------------------------------------

DEFENDER MULTICOLLINEARITY TEST
```

```
----------------------------------------
Total metrics defined: 11
Available metrics: 10
Missing metrics: 1
MISSING: ['Challenges Tkl%']
AVAILABLE: [' Tkl', ' Int', ' Blocks', 'Clr', 'Tackles TklW', 'Tackles Def 3rd',
'Tackles Mid 3rd', 'Blocks Sh', 'Blocks Pass', 'Tkl+Int']
Sample size: 1900
Metrics tested: [' Tkl', ' Int', ' Blocks', 'Clr', 'Tackles TklW', 'Tackles Def
3rd', 'Tackles Mid 3rd', 'Blocks Sh', 'Blocks Pass', 'Tkl+Int']

VIF Results:
Metric               VIF      Status
----------------------------------------
 Tkl                 inf      SEVERE - Remove
 Int                 inf      SEVERE - Remove
 Blocks              inf      SEVERE - Remove
Clr                  1.81     OK
Tackles TklW         5.86     MODERATE - Consider
Tackles Def 3rd      12.24    SEVERE - Remove
Tackles Mid 3rd      6.32     MODERATE - Consider
Blocks Sh            inf      SEVERE - Remove
Blocks Pass          inf      SEVERE - Remove
Tkl+Int              inf      SEVERE - Remove

RECOMMENDATION FOR DEFENDER:
- PROBLEM:  Tkl   Tackles TklW (r = 0.840) - MODERATE
- PROBLEM:  Tkl   Tkl+Int (r = 0.812) - MODERATE
- Consider removing one variable from each problematic pair
--------------------------------------------------

GOALKEEPER MULTICOLLINEARITY TEST
----------------------------------------
Total metrics defined: 11
Available metrics: 10
Missing metrics: 1
MISSING: ['Long Cmp%']
AVAILABLE: ['Total Cmp%', 'Err', 'Total TotDist', 'Total PrgDist', 'Short Cmp%',
'Medium Cmp%', 'Total Cmp', 'Total Att', 'Long Att', 'Short Att']
Sample size: 396
Metrics tested: ['Total Cmp%', 'Err', 'Total TotDist', 'Total PrgDist', 'Short
Cmp%', 'Medium Cmp%', 'Total Cmp', 'Total Att', 'Long Att', 'Short Att']

VIF Results:
Metric               VIF      Status
----------------------------------------
Total Cmp%           538.05   SEVERE - Remove
Err                  1.08     OK
```

```
Total TotDist        495.44   SEVERE - Remove
Total PrgDist        87.97    SEVERE - Remove
Short Cmp%           575.10   SEVERE - Remove
Medium Cmp%          722.82   SEVERE - Remove
Total Cmp            1343.22  SEVERE - Remove
Total Att            956.70   SEVERE - Remove
Long Att             74.36    SEVERE - Remove
Short Att            9.97     MODERATE - Consider

RECOMMENDATION FOR GOALKEEPER:
- PROBLEM: Total TotDist   Total PrgDist (r = 0.893) - MODERATE
- PROBLEM: Total TotDist   Total Cmp (r = 0.867) - MODERATE
- PROBLEM: Total TotDist   Total Att (r = 0.847) - MODERATE
- PROBLEM: Total Cmp   Total Att (r = 0.878) - MODERATE
- Consider removing one variable from each problematic pair
--------------------------------------------------

============================================================
FINAL RECOMMENDED METRICS - CORRELATION CHECK
============================================================
Checking correlations between recommended metrics:
Forward (Expected xG)  Midfielder (Passes Cmp): r = -0.033
Forward (Expected xG)  Defender (Tackles TklW): r = -0.058
Midfielder (Passes Cmp)  Forward (Expected xG): r = -0.033
Midfielder (Passes Cmp)  Defender (Tackles TklW): r = 0.337
Defender (Tackles TklW)  Forward (Expected xG): r = -0.058
Defender (Tackles TklW)  Midfielder (Passes Cmp): r = 0.337

Final recommendation: Use these 3 metrics to predict Team_xG
- Forward: Expected xG
- Midfielder: Passes Cmp
- Defender: Tackles TklW
These metrics have low cross-correlation and capture unique position functions.
```

```
/opt/miniconda3/lib/python3.12/site-
packages/statsmodels/stats/outliers_influence.py:197: RuntimeWarning: divide by
zero encountered in scalar divide
  vif = 1. / (1. - r_squared_i)
```

0.4 2.3. Feature Engineering and Feature selection

Based on the result for collinearity we slelect the following: From SEVERE Pairs - Keep One: FORWARDS:

Expected xG Expected npxG (r = 0.913): KEEP Expected xG (more standard metric) Take-Ons Succ Take-Ons Att (r = 0.851): KEEP Take-Ons Succ (outcome vs attempt)

MIDFIELDERS:

Touches Passes Att Passes Cmp (all r > 0.98): KEEP Passes Cmp (most meaningful outcome)

DEFENDERS:

Tkl Tackles TklW (r = 0.840): KEEP Tackles TklW (successful tackles vs attempts) Tkl Tkl+Int (r = 0.812): KEEP Tkl+Int (combined defensive actions)

GOALKEEPERS:

Total TotDist Total PrgDist (r = 0.893): KEEP Total PrgDist (progressive passing) Total Cmp Total Att (r = 0.878): KEEP Total Cmp (successful passes)

REVISED FINAL LIST (OK + MODERATE + One from SEVERE pairs): FORWARDS: Gls, Ast, SoT, Expected xG, Expected xAG, Take-Ons Succ MIDFIELDERS: Passes Cmp%, KP, Tkl, Carries PrgC, Passes PrgP, Touches DEFENDERS: Int, Blocks, Clr, Tackles TklW, Tackles Def 3rd, Tackles Mid 3rd, Blocks Sh, Blocks Pass GOALKEEPERS: Total Cmp%, Err, Total PrgDist, Short Cmp%, Medium Cmp%, Total Cmp, Short Att

0.5 2.4 Descriptive Statistics

[9]:
```
# Season-by-Season Descriptive Statistics (After accounting for␣
↪multicolinearity)
# SEASONS AS COLUMNS, METRICS AS ROWS

def create_season_comparison_tables(df):
    """
    Create tables with seasons as columns (like Ford/Chevy/Ram), metrics as rows
    """

    # Position metrics (your final list)
    position_metrics = {
        'Forward': ['Gls', 'Ast', 'SoT', 'Expected xG', 'Expected xAG',␣
↪'Take-Ons Succ'],
        'Midfielder': ['Passes Cmp%', 'KP', 'Tkl', 'Carries PrgC', 'Passes␣
↪PrgP', 'Touches'],
        'Defender': ['Int', 'Blocks', 'Clr', 'Tackles TklW', 'Tackles Def 3rd',␣
↪'Tackles Mid 3rd', 'Blocks Sh', 'Blocks Pass'],
        'Goalkeeper': ['Total Cmp%', 'Err', 'Total PrgDist', 'Short Cmp%',␣
↪'Medium Cmp%', 'Total Cmp', 'Short Att']
    }

    # Position mapping
    position_mapping = {
        'Forward': ['FW', 'LW', 'RW', 'CF'],
        'Midfielder': ['CM', 'DM', 'AM', 'LM', 'RM'],
        'Defender': ['CB', 'LB', 'RB', 'DF'],
        'Goalkeeper': ['GK']
    }

    # Get unique seasons and sort them
    seasons = sorted(df['Season'].unique())
```

```python
    print(f"Available seasons: {seasons}")

    for position, metrics in position_metrics.items():

        # Filter data for this position
        pos_codes = position_mapping[position]
        pos_pattern = '|'.join(pos_codes)
        pos_data = df[df['Pos'].str.contains(pos_pattern, case=False, na=False)]

        if len(pos_data) == 0:
            continue

        # Get available metrics (handle space prefix)
        available_metrics = []
        for m in metrics:
            if m in df.columns:
                available_metrics.append(m)
            elif f' {m}' in df.columns:
                available_metrics.append(f' {m}')

        if len(available_metrics) == 0:
            continue

        print(f"\nTable {list(position_metrics.keys()).index(position) + 2}.␣
↪Descriptive Statistics of {position} Performance by Season")
        print("=" * 80)
        print()

        # Calculate season-specific data
        season_data = {}
        season_stats = {}

        for season in seasons:
            season_pos_data = pos_data[pos_data['Season'] == season]
            if len(season_pos_data) > 0:
                season_data[season] = season_pos_data
                # Calculate total observations and unique players
                total_obs = len(season_pos_data)
                unique_players = season_pos_data['Player'].nunique() if␣
↪'Player' in season_pos_data.columns else 'N/A'
                season_stats[season] = {'total_obs': total_obs,␣
↪'unique_players': unique_players}

        # Use specific seasons: 2022-23, 2023-24, 2024-25
        target_seasons = ['22_23', '23_24', '24_25']

        # Filter to only include target seasons
```

```
        seasons_to_show = [season for season in target_seasons if season in␣
↪season_data.keys()]

        if len(seasons_to_show) == 0:
            print(f"No data found for target seasons {target_seasons} in␣
↪{position}")
            continue

        # Header with seasons as columns
        header = f"{'Metric':<25}"
        for season in seasons_to_show:
            header += f"{season:<12}"
        print(header)
        print("-" * (25 + len(seasons_to_show) * 12))

        # Total observations row
        obs_row = f"{'Total observations':<25}"
        for season in seasons_to_show:
            if season in season_stats:
                obs_row += f"{season_stats[season]['total_obs']:<12,}"
            else:
                obs_row += f"{'--':<12}"
        print(obs_row)

        # Unique players row
        players_row = f"{'Unique players':<25}"
        for season in seasons_to_show:
            if season in season_stats:
                players_row += f"{season_stats[season]['unique_players']:<12}"
            else:
                players_row += f"{'--':<12}"
        print(players_row)
        print()

        # Process each metric (limit to 4-5 key metrics for readability)
        key_metrics = available_metrics[:5]

        for metric in key_metrics:
            # Clean metric name for display
            display_metric = metric.strip().replace('Expected ', 'Exp').
↪replace('Tackles ', 'Tkl').replace('Take-Ons ', 'TO')

            print(f"{display_metric}")

            # Calculate statistics for each season
            metric_stats = {}
            for season in seasons_to_show:
```

```
                if season in season_data and metric in season_data[season].
↪columns:
                    clean_data = season_data[season][metric].dropna()
                    if len(clean_data) > 0:
                        metric_stats[season] = clean_data.describe()

            # Display statistics (Ford/Chevy/Ram style)
            stat_labels = ['Mean', 'SD', 'Minimum', '25th percentile',␣
↪'Median', '75th percentile', 'Maximum']
            stat_keys = ['mean', 'std', 'min', '25%', '50%', '75%', 'max']

            for label, key in zip(stat_labels, stat_keys):
                row = f"  {label:<23}"
                for season in seasons_to_show:
                    if season in metric_stats:
                        if key in ['min', 'max']:
                            row += f"{metric_stats[season][key]:<12.0f}"
                        else:
                            row += f"{metric_stats[season][key]:<12.2f}"
                    else:
                        row += f"{'--':<12}"
                print(row)
            print()

        print("-" * (25 + len(seasons_to_show) * 12))
        print()

        # Note section (APA style)
        print(f"Note. Performance statistics for {position.lower()} players␣
↪across {len(seasons_to_show)} seasons.")
        print(f"Metrics represent core {position.lower()} performance␣
↪indicators without multicollinearity.")
        total_all_seasons = sum([season_stats[s]['total_obs'] for s in␣
↪seasons_to_show if s in season_stats])
        unique_all_seasons = len(set([player for s in seasons_to_show if s in␣
↪season_data
                                      for player in season_data[s]['Player'].
↪unique() if 'Player' in season_data[s].columns]))
        print(f"Combined sample: {total_all_seasons:,} observations from␣
↪{unique_all_seasons} unique players.")
        print()

# Run the analysis
create_season_comparison_tables(df)
```

```
Available seasons: ['17_18', '18_19', '19_20', '20_21', '21_22', '22_23',
'23_24', '24_25']
```

Table 2. Descriptive Statistics of Forward Performance by Season

Metric	22_23	23_24	24_25
Total observations	215	199	214
Unique players	11	12	12
Gls			
Mean	0.34	0.36	0.37
SD	0.59	0.61	0.68
Minimum	0	0	0
25th percentile	0.00	0.00	0.00
Median	0.00	0.00	0.00
75th percentile	1.00	1.00	1.00
Maximum	3	2	3
Ast			
Mean	0.20	0.16	0.14
SD	0.46	0.45	0.37
Minimum	0	0	0
25th percentile	0.00	0.00	0.00
Median	0.00	0.00	0.00
75th percentile	0.00	0.00	0.00
Maximum	2	3	2
SoT			
Mean	0.91	0.89	0.98
SD	1.05	1.07	1.16
Minimum	0	0	0
25th percentile	0.00	0.00	0.00
Median	1.00	0.00	1.00
75th percentile	1.00	2.00	1.75
Maximum	5	5	5
ExpxG			
Mean	0.35	0.31	0.33
SD	0.43	0.39	0.44
Minimum	0	0	0
25th percentile	0.00	0.00	0.00
Median	0.20	0.20	0.10
75th percentile	0.50	0.40	0.50
Maximum	2	2	2
ExpxAG			
Mean	0.19	0.12	0.16
SD	0.27	0.22	0.27

Minimum	0	0	0
25th percentile	0.00	0.00	0.00
Median	0.10	0.00	0.00
75th percentile	0.30	0.20	0.20
Maximum	1	2	2

Note. Performance statistics for forward players across 3 seasons.
Metrics represent core forward performance indicators without multicollinearity.
Combined sample: 628 observations from 18 unique players.

Table 3. Descriptive Statistics of Midfielder Performance by Season

Metric	22_23	23_24	24_25
Total observations	260	289	296
Unique players	11	15	16
Passes Cmp%			
Mean	88.82	87.50	87.80
SD	7.83	10.88	9.78
Minimum	33	0	0
25th percentile	85.92	84.30	83.67
Median	90.30	90.00	89.50
75th percentile	93.03	93.10	93.50
Maximum	100	100	100
KP			
Mean	1.03	1.21	1.05
SD	1.27	1.49	1.24
Minimum	0	0	0
25th percentile	0.00	0.00	0.00
Median	1.00	1.00	1.00
75th percentile	2.00	2.00	2.00
Maximum	7	8	6
Tkl			
Mean	1.30	1.30	1.49
SD	1.36	1.40	1.52
Minimum	0	0	0
25th percentile	0.00	0.00	0.00
Median	1.00	1.00	1.00
75th percentile	2.00	2.00	2.00
Maximum	7	10	8

Carries PrgC			
Mean	1.81	1.75	1.57
SD	2.07	1.82	1.81
Minimum	0	0	0
25th percentile	0.00	0.00	0.00
Median	1.00	1.00	1.00
75th percentile	3.00	3.00	2.00
Maximum	15	9	9
Passes PrgP			
Mean	5.32	5.22	4.82
SD	4.57	4.24	4.24
Minimum	0	0	0
25th percentile	2.00	2.00	2.00
Median	4.00	5.00	4.00
75th percentile	8.00	8.00	7.00
Maximum	26	22	23

Note. Performance statistics for midfielder players across 3 seasons.
Metrics represent core midfielder performance indicators without multicollinearity.
Combined sample: 845 observations from 22 unique players.

Table 4. Descriptive Statistics of Defender Performance by Season

Metric	22_23	23_24	24_25
Total observations	265	258	276
Unique players	10	11	14
Int			
Mean	0.73	0.63	0.79
SD	0.98	0.90	1.02
Minimum	0	0	0
25th percentile	0.00	0.00	0.00
Median	0.00	0.00	0.00
75th percentile	1.00	1.00	1.00
Maximum	5	5	6
Blocks			
Mean	0.86	0.84	0.83
SD	1.08	1.13	1.02
Minimum	0	0	0
25th percentile	0.00	0.00	0.00

Median	1.00	1.00	1.00
75th percentile	1.00	1.00	1.00
Maximum	5	7	5
Clr			
Mean	1.77	1.98	2.32
SD	1.69	1.90	2.36
Minimum	0	0	0
25th percentile	0.00	0.00	1.00
Median	1.00	1.00	2.00
75th percentile	3.00	3.00	3.00
Maximum	8	11	14
TklTklW			
Mean	0.74	0.72	0.87
SD	0.95	0.92	1.05
Minimum	0	0	0
25th percentile	0.00	0.00	0.00
Median	0.00	0.00	1.00
75th percentile	1.00	1.00	1.00
Maximum	4	5	4
TklDef 3rd			
Mean	0.65	0.67	0.79
SD	0.94	0.96	0.98
Minimum	0	0	0
25th percentile	0.00	0.00	0.00
Median	0.00	0.00	1.00
75th percentile	1.00	1.00	1.00
Maximum	5	4	5

Note. Performance statistics for defender players across 3 seasons.
Metrics represent core defender performance indicators without multicollinearity.
Combined sample: 799 observations from 17 unique players.

Table 5. Descriptive Statistics of Goalkeeper Performance by Season

Metric	22_23	23_24	24_25
Total observations	50	52	51
Unique players	2	3	2
Total Cmp%			

Mean	86.25	86.47	83.39
SD	8.93	8.53	11.51
Minimum	63	61	54
25th percentile	82.30	81.12	75.70
Median	88.15	87.85	84.60
75th percentile	91.85	92.62	92.45
Maximum	100	100	100
Err			
Mean	0.04	0.04	0.02
SD	0.20	0.19	0.14
Minimum	0	0	0
25th percentile	0.00	0.00	0.00
Median	0.00	0.00	0.00
75th percentile	0.00	0.00	0.00
Maximum	1	1	1
Total PrgDist			
Mean	455.46	405.63	399.59
SD	136.48	131.15	141.36
Minimum	143	66	174
25th percentile	350.00	306.50	313.50
Median	440.00	388.50	368.00
75th percentile	552.00	487.25	469.00
Maximum	770	793	735
Short Cmp%			
Mean	99.75	98.81	99.51
SD	1.77	5.31	2.45
Minimum	88	67	88
25th percentile	100.00	100.00	100.00
Median	100.00	100.00	100.00
75th percentile	100.00	100.00	100.00
Maximum	100	100	100
Medium Cmp%			
Mean	98.95	99.27	99.46
SD	2.20	2.13	1.81
Minimum	92	90	90
25th percentile	100.00	100.00	100.00
Median	100.00	100.00	100.00
75th percentile	100.00	100.00	100.00
Maximum	100	100	100

Note. Performance statistics for goalkeeper players across 3 seasons. Metrics represent core goalkeeper performance indicators without

```
multicollinearity.
Combined sample: 153 observations from 3 unique players.
```

Season-by-Season Performance Analysis (2022-25) FORWARDS (Table 2) - Trending Analysis: IMPROVING TRENDS:

Goal scoring increasing: 0.34 → 0.36 → 0.37 goals per match (steady improvement) Shot quality rising: Shots on target increased from 0.91 to 0.98 Expected Goals stable: xG around 0.31-0.35 range (consistent threat creation)

DECLINING TRENDS:

Creativity dropping: Assists fell from 0.20 → 0.16 → 0.14 (concerning trend) Expected Assists down: xAG declined from 0.19 → 0.12 → 0.16 (less playmaking)

KEY INSIGHT: Forwards becoming more selfish but more clinical - scoring more but creating less for teammates.

MIDFIELDERS (Table 3) - Stability with Concerns: CONSISTENT PERFORMANCE:

Passing accuracy stable: ~87-89% across all seasons (reliable ball retention) Key passes steady: 1.03 → 1.21 → 1.05 (consistent creativity) Defensive work increasing: Tackles rose from 1.30 → 1.49 (more defensive responsibility)

SLIGHT DECLINE:

Progressive play dropping: Progressive passes fell from 5.32 → 4.82 Ball carrying down: Progressive carries decreased from 1.81 → 1.57

KEY INSIGHT: Midfielders maintaining core functions but becoming less adventurous in attack.

DEFENDERS (Table 4) - Improving Defensive Intensity: POSITIVE TRENDS:

More active defending: Clearances increased dramatically (1.77 → 2.32) Better positioning: Interceptions improved (0.73 → 0.79) Tackle success up: Successful tackles rose from 0.74 → 0.87

KEY INSIGHT: Defense becoming more proactive and aggressive - suggests team facing more pressure but handling it better.

GOALKEEPERS (Table 5) - Distribution Concerns: DECLINING TRENDS:

Passing accuracy dropping: Total completion fell from 86.25% → 83.39% Shorter distribution: Progressive distance decreased (455m → 400m) More conservative: Playing safer, shorter passes

POSITIVE:

Error-free: Extremely low error rates (0.02-0.04 per game) Short passing excellent: 99%+ accuracy on short passes

KEY INSIGHT: Goalkeepers playing more conservatively, possibly due to tactical changes or pressure.

OVERALL TEAM EVOLUTION (2022-25):

Tactical Shift: From creative to pragmatic - forwards scoring more but assisting less Defensive Improvement: More active defending suggests better organization Conservative Approach: Reduced progressive play from midfield and goalkeepers Efficiency Focus: Better conversion rates but less risk-taking in final third

SUMMARY: Real Madrid evolved from a more creative, risk-taking team to a more efficient, defensively solid unit that relies on clinical finishing rather than elaborate build-up play.

0.6 2.4 Main metrics for each position

Unique Overall Performance Metrics by Position: FORWARDS: Primary Metric: Expected xG

Why: Best predictor of attacking threat and quality of chances created Contribution: Measures shooting ability and positioning in dangerous areas Range: 0.00-2.70 per match

MIDFIELDERS: Primary Metric: KP (Key Passes)

Why: Captures creativity and chance creation ability Contribution: Measures playmaking and ability to unlock defenses Range: 0-9 per match

DEFENDERS: Primary Metric: Tackles TklW (Successful Tackles)

Why: Best indicator of defensive effectiveness and ball-winning ability Contribution: Measures active defending and disrupting opponent attacks Range: 0-6 per match

GOALKEEPERS: Primary Metric: Total Cmp% (Distribution Accuracy)

Why: Modern goalkeeper's most important contribution beyond shot-stopping Contribution: Measures ability to start attacks and maintain possession Range: 44-100% completion rate

Why These Metrics Are Unique:

Position-Specific: Each captures the core function of that position Low Cross-Correlation: These metrics don't overlap between positions Performance Predictive: Best indicators of individual contribution to team success Multicollinearity-Free: Selected after removing redundant variables

For Team xG Correlation: These metrics would be the independent variables to predict Team xG (dependent variable), showing how individual position performance contributes to overall team attacking output.

0.7 3 Four with recalculation of weights

[10]:
```
# Fixed Rebalanced Scoring System - Using Only Available Columns

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

print("=== FIXED PERFORMANCE SCORING SYSTEM ===")

# Create Position_Group column from Pos column
def categorize_position(pos):
    """Categorize positions into groups"""
    if pd.isna(pos):
        return None
```

```
    pos_str = str(pos).upper()
    if 'GK' in pos_str:
        return 'Goalkeeper'
    elif any(fw in pos_str for fw in ['FW', 'CF', 'ST', 'LW', 'RW']):
        return 'Forward'
    elif any(mid in pos_str for mid in ['MF', 'CM', 'DM', 'AM', 'LM', 'RM']):
        return 'Midfield'
    elif any(def_ in pos_str for def_ in ['DF', 'CB', 'LB', 'RB', 'WB', 'SW']):
        return 'Defense'
    else:
        return 'Midfield'  # Default for unclear positions

df['Position_Group'] = df['Pos'].apply(categorize_position)

print("Dataset shape:", df.shape)
print("Available columns:", list(df.columns))

# ==========================================
# FIXED REBALANCED SCORING SYSTEM
# ==========================================

def calculate_rebalanced_scores_fixed(df):
    """
    Fixed rebalanced scoring system using only available columns
    """
    df = df.copy()
    df['Rebalanced_Score'] = 0.0

    # Check which columns exist
    available_cols = df.columns.tolist()
    print(f"Available columns: {len(available_cols)}")

    # GOALKEEPERS
    gk_mask = df['Position_Group'] == 'Goalkeeper'
    if gk_mask.sum() > 0:
        print(f"\nRebalancing Goalkeepers ({gk_mask.sum()} players)...")
        gk_data = df[gk_mask].copy()

        # Use available goalkeeper metrics
        gk_score = 15.0  # Base score for goalkeepers

        if 'Total Cmp%' in available_cols:
            gk_data['Total Cmp%'] = gk_data['Total Cmp%'].fillna(80)
            gk_distribution = np.clip(gk_data['Total Cmp%'] / 100, 0, 1)
            gk_score = gk_distribution * 20  # 0-20 range based on distribution

        if 'Err' in available_cols:
```

```
            gk_data['Err'] = gk_data['Err'].fillna(0)
            # Penalty for errors (subtract up to 5 points)
            error_penalty = np.minimum(gk_data['Err'] * 2.5, 5)
            gk_score = gk_score - error_penalty

        # Ensure minimum score
        gk_score = np.maximum(gk_score, 5)

        df.loc[gk_mask, 'Rebalanced_Score'] = gk_score
        print(f"  GK score range: {gk_score.min():.1f} - {gk_score.max():.1f}")

    # FORWARDS
    fw_mask = df['Position_Group'] == 'Forward'
    if fw_mask.sum() > 0:
        print(f"\nRebalancing Forwards ({fw_mask.sum()} players)...")
        fw_data = df[fw_mask].copy()

        # Per-90 calculations with safety checks
        min_threshold = 10
        fw_data['Min_adj'] = np.maximum(fw_data['Min'], min_threshold)

        fw_score = 0

        # Goals component (handle space prefix)
        if ' Gls' in available_cols:
            fw_gls_90 = fw_data[' Gls'] / fw_data['Min_adj'] * 90
            goals_score = np.minimum(fw_gls_90 * 10, 10)
            fw_score += goals_score
        elif 'Gls' in available_cols:
            fw_gls_90 = fw_data['Gls'] / fw_data['Min_adj'] * 90
            goals_score = np.minimum(fw_gls_90 * 10, 10)
            fw_score += goals_score

        # Assists component
        if ' Ast' in available_cols:
            fw_ast_90 = fw_data[' Ast'] / fw_data['Min_adj'] * 90
            assists_score = np.minimum(fw_ast_90 * 8, 8)
            fw_score += assists_score
        elif 'Ast' in available_cols:
            fw_ast_90 = fw_data['Ast'] / fw_data['Min_adj'] * 90
            assists_score = np.minimum(fw_ast_90 * 8, 8)
            fw_score += assists_score

        # Shots component
        if ' Sh' in available_cols:
            fw_sh_90 = fw_data[' Sh'] / fw_data['Min_adj'] * 90
            shots_score = np.minimum(fw_sh_90 * 0.5, 5)
```

```
            fw_score += shots_score

        # Shots on target component
        if ' SoT' in available_cols:
            fw_sot_90 = fw_data[' SoT'] / fw_data['Min_adj'] * 90
            sot_score = np.minimum(fw_sot_90 * 1, 6)
            fw_score += sot_score
        elif 'SoT' in available_cols:
            fw_sot_90 = fw_data['SoT'] / fw_data['Min_adj'] * 90
            sot_score = np.minimum(fw_sot_90 * 1, 6)
            fw_score += sot_score

        # Expected xG component
        if 'Expected xG' in available_cols:
            fw_xg_90 = fw_data['Expected xG'] / fw_data['Min_adj'] * 90
            xg_score = np.minimum(fw_xg_90 * 5, 5)
            fw_score += xg_score

        # Minutes bonus
        minutes_bonus = np.minimum(fw_data['Min'] / 90 * 0.1, 3)
        fw_score += minutes_bonus

        df.loc[fw_mask, 'Rebalanced_Score'] = fw_score
        print(f"  FW score range: {fw_score.min():.1f} - {fw_score.max():.1f}")

    # MIDFIELDERS
    mid_mask = df['Position_Group'] == 'Midfield'
    if mid_mask.sum() > 0:
        print(f"\nRebalancing Midfielders ({mid_mask.sum()} players)...")
        mid_data = df[mid_mask].copy()

        # Per-90 calculations
        min_threshold = 10
        mid_data['Min_adj'] = np.maximum(mid_data['Min'], min_threshold)

        mid_score = 0

        # Assists component
        if ' Ast' in available_cols:
            mid_ast_90 = mid_data[' Ast'] / mid_data['Min_adj'] * 90
            assists_score = np.minimum(mid_ast_90 * 6, 6)
            mid_score += assists_score

        # Key passes component
        if 'KP' in available_cols:
            mid_kp_90 = mid_data['KP'] / mid_data['Min_adj'] * 90
            keypass_score = np.minimum(mid_kp_90 * 1.5, 5)
```

```
            mid_score += keypass_score

        # Progressive passes component
        if 'Passes PrgP' in available_cols:
            mid_prog_90 = mid_data['Passes PrgP'] / mid_data['Min_adj'] * 90
            progressive_score = np.minimum(mid_prog_90 * 0.3, 4)
            mid_score += progressive_score

        # Tackles component
        if ' Tkl' in available_cols:
            mid_tkl_90 = mid_data[' Tkl'] / mid_data['Min_adj'] * 90
            tackle_score = np.minimum(mid_tkl_90 * 1, 3)
            mid_score += tackle_score

        # Pass accuracy component
        if 'Passes Cmp%' in available_cols:
            mid_data['Passes Cmp%'] = mid_data['Passes Cmp%'].fillna(85)
            pass_acc_score = np.clip((mid_data['Passes Cmp%'] - 80) / 20 * 4,␣
↪0, 4)
            mid_score += pass_acc_score

        # Minutes bonus
        minutes_bonus = np.minimum(mid_data['Min'] / 90 * 0.1, 4)
        mid_score += minutes_bonus

        df.loc[mid_mask, 'Rebalanced_Score'] = mid_score
        print(f"  MID score range: {mid_score.min():.1f} - {mid_score.max():.
↪1f}")

    # DEFENDERS
    def_mask = df['Position_Group'] == 'Defense'
    if def_mask.sum() > 0:
        print(f"\nRebalancing Defenders ({def_mask.sum()} players)...")
        def_data = df[def_mask].copy()

        # Per-90 calculations
        min_threshold = 10
        def_data['Min_adj'] = np.maximum(def_data['Min'], min_threshold)

        def_score = 0

        # Tackles component
        if 'Tackles TklW' in available_cols:
            def_tkl_90 = def_data['Tackles TklW'] / def_data['Min_adj'] * 90
            tackles_score = np.minimum(def_tkl_90 * 2, 6)
            def_score += tackles_score
        elif ' Tkl' in available_cols:
```

```
            def_tkl_90 = def_data[' Tkl'] / def_data['Min_adj'] * 90
            tackles_score = np.minimum(def_tkl_90 * 1.5, 6)
            def_score += tackles_score

        # Interceptions component
        if 'Int' in available_cols:
            def_int_90 = def_data['Int'] / def_data['Min_adj'] * 90
            int_score = np.minimum(def_int_90 * 2, 6)
            def_score += int_score
        elif ' Int' in available_cols:
            def_int_90 = def_data[' Int'] / def_data['Min_adj'] * 90
            int_score = np.minimum(def_int_90 * 2, 6)
            def_score += int_score

        # Blocks component
        if 'Blocks' in available_cols:
            def_blk_90 = def_data['Blocks'] / def_data['Min_adj'] * 90
            blocks_score = np.minimum(def_blk_90 * 3, 6)
            def_score += blocks_score
        elif ' Blocks' in available_cols:
            def_blk_90 = def_data[' Blocks'] / def_data['Min_adj'] * 90
            blocks_score = np.minimum(def_blk_90 * 3, 6)
            def_score += blocks_score

        # Clearances component
        if 'Clr' in available_cols:
            def_clr_90 = def_data['Clr'] / def_data['Min_adj'] * 90
            clear_score = np.minimum(def_clr_90 * 0.5, 4)
            def_score += clear_score

        # Minutes bonus
        minutes_bonus = np.minimum(def_data['Min'] / 90 * 0.1, 4)
        def_score += minutes_bonus

        df.loc[def_mask, 'Rebalanced_Score'] = def_score
        print(f"  DEF score range: {def_score.min():.1f} - {def_score.max():.
↪1f}")

    return df

# Apply fixed rebalanced scoring
df = calculate_rebalanced_scores_fixed(df)

# ==========================================
# VALIDATION
# ==========================================
```

```python
print("\n  VALIDATION OF FIXED SYSTEM")

# Score distribution by position
print("\nScore Distribution by Position:")
position_stats = df.groupby('Position_Group')['Rebalanced_Score'].agg(['mean',
    'median', 'std', 'min', 'max', 'count']).round(2)
print(position_stats)

# Top performers
print("\n  TOP 15 PERFORMERS (All Positions):")
top_performers = df.nlargest(15, 'Rebalanced_Score')[['Player',
    'Position_Group', 'Rebalanced_Score', 'Min', 'Season']]
print(top_performers.to_string(index=False))

# ==========================================
# SAVE REBALANCED DATASET
# ==========================================

print("\n  SAVING REBALANCED DATASET...")

# Create output path (adjust as needed for your system)
output_path = '/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main
    Notebook/Data Folder/DataCombined/real_madrid_rebalanced_scores.csv'

# Save the complete dataset with rebalanced scores
df.to_csv(output_path, index=False)

print(f"  Rebalanced dataset saved to: {output_path}")
print(f"  Dataset contains {len(df)} rows and {len(df.columns)} columns")
print(f"  New column 'Rebalanced_Score' added successfully")

# Quick verification
rebalanced_check = df[['Player', 'Position_Group', 'Rebalanced_Score', 'Min',
    'Season']].head(10)
print(f"\n  Sample of saved data:")
print(rebalanced_check.to_string(index=False))

print("\n  FIXED REBALANCED SCORING COMPLETE!")
```

```
=== FIXED PERFORMANCE SCORING SYSTEM ===
Dataset shape: (5737, 70)
Available columns: ['Date', 'Competition', 'Opponent', 'Player', '#', 'Nation',
'Pos', 'Age', 'Min', ' Gls', ' Ast', ' PK', ' PKatt', ' Sh', ' SoT', ' CrdY', '
CrdR', ' Int', 'Match URL', 'Season', ' Touches', ' Tkl', ' Blocks', 'Expected
xG', 'Expected npxG', 'Expected xAG', 'Passes Cmp', 'Passes Att', 'Passes Cmp%',
'Passes PrgP', 'Carries Carries', 'Carries PrgC', 'Take-Ons Att', 'Take-Ons
Succ', 'Tackles Tkl', 'Tackles TklW', 'Tackles Def 3rd', 'Tackles Mid 3rd',
```

```
'Tackles Att 3rd', 'Challenges Tkl', 'Challenges Att', 'Challenges Lost',
'Blocks Blocks', 'Blocks Sh', 'Blocks Pass', 'Int', 'Tkl+Int', 'Clr', 'Err',
'Total Cmp', 'Total Att', 'Total Cmp%', 'Total TotDist', 'Total PrgDist', 'Short
Cmp', 'Short Att', 'Short Cmp%', 'Medium Cmp', 'Medium Att', 'Medium Cmp%',
'Long Cmp', 'Long Att', 'Ast', 'xAG', 'xA', 'KP', 'PPA', 'CrsPA', 'PrgP',
'Position_Group']
Available columns: 71

Rebalancing Goalkeepers (396 players)…
  GK score range: 7.5 - 20.0

Rebalancing Forwards (1695 players)…
  FW score range: 0.0 - 34.1

Rebalancing Midfielders (1823 players)…
  MID score range: 0.0 - 20.2

Rebalancing Defenders (1823 players)…
  DEF score range: 0.0 - 21.1

  VALIDATION OF FIXED SYSTEM

Score Distribution by Position:
                 mean  median   std  min    max  count
Position_Group
Defense          7.64    7.60  4.37  0.0  21.10   1823
Forward          7.81    5.10  7.98  0.0  34.10   1695
Goalkeeper      16.36   16.66  2.44  7.5  20.00    396
Midfield         7.99    7.35  4.02  0.0  20.24   1823

  TOP 15 PERFORMERS (All Positions):
           Player Position_Group  Rebalanced_Score  Min Season
Cristiano Ronaldo        Forward         34.100000 90.0  17_18
    Kylian Mbappé        Forward         32.100000 90.0  24_25
    Karim Benzema        Forward         30.100000 90.0  18_19
    Karim Benzema        Forward         30.100000 90.0  19_20
Cristiano Ronaldo        Forward         30.057619 84.0  17_18
  Jude Bellingham        Forward         29.901389 80.0  23_24
Cristiano Ronaldo        Forward         29.600000 90.0  17_18
Cristiano Ronaldo        Forward         29.600000 90.0  17_18
Cristiano Ronaldo        Forward         29.600000 90.0  17_18
  Vinicius Júnior        Forward         29.600000 90.0  21_22
          Rodrygo        Forward         29.597602 76.0  23_24
      Brahim Díaz        Forward         29.160808 22.0  24_25
  Vinicius Júnior        Forward         29.100000 90.0  21_22
    Marco Asensio        Forward         28.607124 17.0  19_20
    Karim Benzema        Forward         28.420060 74.0  21_22
```

```
  SAVING REBALANCED DATASET…
  Rebalanced dataset saved to:
/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main Notebook/Data
Folder/DataCombined/real_madrid_rebalanced_scores.csv
  Dataset contains 5737 rows and 71 columns
  New column 'Rebalanced_Score' added successfully

  Sample of saved data:
          Player Position_Group  Rebalanced_Score  Min Season
     Gareth Bale        Forward         27.644740 79.0  17_18
   Lucas Vázquez        Forward          0.012222 11.0  17_18
   Karim Benzema        Forward          8.100000 90.0  17_18
            Isco       Midfield          4.109145 65.0  17_18
   Marco Asensio       Midfield          0.387778 25.0  17_18
      Toni Kroos       Midfield          7.720000 90.0  17_18
        Casemiro       Midfield          8.456072 71.0  17_18
Marcos Llorente       Midfield          3.902164 19.0  17_18
     Luka Modrić       Midfield          7.980000 90.0  17_18
       Marcelo         Defense          8.600000 90.0  17_18

  FIXED REBALANCED SCORING COMPLETE!
```

1 Real Madrid Performance Score Formulas

1.1 REBALANCED SCORING SYSTEM

1.1.1 Score Range: 0-30 points for all positions

1.2 FORWARDS (Weight Distribution)

1.2.1 Formula Components:

- **Goals Score** (40%): `min(Goals_per_90 × 10, 10)`
- **Assists Score** (32%): `min(Assists_per_90 × 8, 8)`
- **Shots Score** (10%): `min(Shots_per_90 × 0.5, 5)`
- **Shots on Target Score** (20%): `min(SoT_per_90 × 1, 6)`
- **Minutes Bonus** (up to 3 pts): `min(Total_Minutes ÷ 90 × 0.1, 3)`

1.2.2 Final Formula:

`Forward_Score = Goals_Score + Assists_Score + Shots_Score + SoT_Score + Minutes_Bonus`

1.2.3 Benchmarks:

- 1 goal per 90 min = 10 points (excellent)
- 1 assist per 90 min = 8 points (excellent)
- 10 shots per 90 min = 5 points
- 6 shots on target per 90 min = 6 points

1.3 MIDFIELDERS (Weight Distribution)

1.3.1 Formula Components:

- **Assists Score** (27%): `min(Assists_per_90 × 8, 8)`
- **Creativity Score** (20%): `min(SCA_per_90 × 1.5, 6)`
- **Key Passes Score** (20%): `min(KeyPasses_per_90 × 2, 6)`
- **Progressive Passes** (13%): `min(ProgPasses_per_90 × 0.3, 4)`
- **Pass Accuracy** (13%): `(Total_Cmp% - 80) ÷ 20 × 4` (capped 0-4)
- **Minutes Bonus** (up to 4 pts): `min(Total_Minutes ÷ 90 × 0.1, 4)`

1.3.2 Final Formula:

```
Midfield_Score = Assists_Score + Creativity_Score + KeyPasses_Score + Progressive_Score + PassA
```

1.3.3 Benchmarks:

- 1 assist per 90 min = 8 points
- 4 shot creating actions per 90 min = 6 points
- 3 key passes per 90 min = 6 points
- 90% pass accuracy = 2 points
- 95% pass accuracy = 3 points

1.4 DEFENDERS (Weight Distribution)

1.4.1 Formula Components:

- **Tackles Score** (24%): `min(Tackles_per_90 × 1.5, 6)`
- **Interceptions Score** (24%): `min(Interceptions_per_90 × 2, 6)`
- **Blocks Score** (24%): `min(Blocks_per_90 × 3, 6)`
- **Clearances Score** (16%): `min(Clearances_per_90 × 0.5, 4)`
- **Pass Accuracy Score** (16%): `(Total_Cmp% - 85) ÷ 15 × 4` (capped 0-4)
- **Minutes Bonus** (up to 4 pts): `min(Total_Minutes ÷ 90 × 0.1, 4)`

1.4.2 Final Formula:

```
Defense_Score = Tackles_Score + Interceptions_Score + Blocks_Score + Clearances_Score + PassAc
```

1.4.3 Benchmarks:

- 4 tackles per 90 min = 6 points
- 3 interceptions per 90 min = 6 points
- 2 blocks per 90 min = 6 points
- 8 clearances per 90 min = 4 points
- 92% pass accuracy = 2 points

1.5 GOALKEEPERS (Weight Distribution)

1.5.1 Formula Components:

- **Distribution Accuracy** (60%): `(Total_Cmp% ÷ 100) × 0.6 × 30`
- **Long Pass Accuracy** (40%): `(Long_Cmp% ÷ 100) × 0.4 × 30`

1.5.2 Final Formula:

```
Goalkeeper_Score = (Distribution_Score + LongPass_Score) × 30 ÷ 100
```

1.5.3 Benchmarks:

- 90% distribution accuracy = 16.2 points
- 70% long pass accuracy = 8.4 points
- Perfect distribution + long passes = 30 points

1.6 AVERAGE PERFORMANCE SCORES BY POSITION

Based on the rebalanced system:

1.6.1 Expected Ranges:

- **Excellent Players**: 20-30 points
- **Good Players**: 15-20 points

- **Average Players**: 10-15 points
- **Below Average**: 5-10 points
- **Poor Performance**: 0-5 points

1.6.2 Position Averages:

- **Goalkeepers**: 15-25 range (based on passing accuracy)
- **Defenders**: 12-22 range (consistent defensive work)
- **Midfielders**: 10-25 range (varied roles - defensive to creative)
- **Forwards**: 8-28 range (goal-dependent, high variance)

1.7 KEY IMPROVEMENTS

1. **Minutes Bonus**: Rewards consistency (0.1 points per 90 minutes played)
2. **Position Parity**: All positions can achieve similar maximum scores
3. **Realistic Benchmarks**: Based on actual elite performance metrics
4. **No Goalkeeper Bias**: Reduced from 0-100 to 0-30 scale like others

---

1.8 CALCULATION EXAMPLE

Jude Bellingham - Midfield Performance: - Assists per 90: 0.3 → $0.3 \times 8 = 2.4$ points - SCA per 90: 3.5 → $\min(3.5 \times 1.5, 6) = 5.25$ points
- Key passes per 90: 2.1 → $\min(2.1 \times 2, 6) = 4.2$ points - Progressive passes per 90: 8.2 → $\min(8.2 \times 0.3, 4) = 2.46$ points - Pass accuracy: 88% → $(88\text{-}80)/20 \times 4 = 1.6$ points - Minutes bonus: 2400 min → $\min(2400/90 \times 0.1, 4) = 2.67$ points

Total: 2.4 + 5.25 + 4.2 + 2.46 + 1.6 + 2.67 = 18.58 points

2 4 Modeling, feature selection, training, validaton

2.1 1. Load Data

[11]:
```python
import pandas as pd

# Path to your rebalanced dataset
path = '/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main Notebook/
↪Data Folder/DataCombined/real_madrid_rebalanced_scores.csv'

# Load data
df = pd.read_csv(path)
print(df.shape)
print(df.columns.tolist())
```

```
(5737, 71)
['Date', 'Competition', 'Opponent', 'Player', '#', 'Nation', 'Pos', 'Age',
'Min', ' Gls', ' Ast', ' PK', ' PKatt', ' Sh', ' SoT', ' CrdY', ' CrdR', ' Int',
'Match URL', 'Season', ' Touches', ' Tkl', ' Blocks', 'Expected xG', 'Expected
npxG', 'Expected xAG', 'Passes Cmp', 'Passes Att', 'Passes Cmp%', 'Passes PrgP',
'Carries Carries', 'Carries PrgC', 'Take-Ons Att', 'Take-Ons Succ', 'Tackles
Tkl', 'Tackles TklW', 'Tackles Def 3rd', 'Tackles Mid 3rd', 'Tackles Att 3rd',
'Challenges Tkl', 'Challenges Att', 'Challenges Lost', 'Blocks Blocks', 'Blocks
Sh', 'Blocks Pass', 'Int', 'Tkl+Int', 'Clr', 'Err', 'Total Cmp', 'Total Att',
'Total Cmp%', 'Total TotDist', 'Total PrgDist', 'Short Cmp', 'Short Att', 'Short
Cmp%', 'Medium Cmp', 'Medium Att', 'Medium Cmp%', 'Long Cmp', 'Long Att', 'Ast',
'xAG', 'xA', 'KP', 'PPA', 'CrsPA', 'PrgP', 'Position_Group', 'Rebalanced_Score']
```

2.2 2. Add Weekly ID

[12]:
```python
# After adding the Week column
if 'Date' in df.columns:
    df['Date'] = pd.to_datetime(df['Date'])
    df['Week'] = df['Date'].dt.isocalendar().week
else:
    df['Week'] = (df.index // 10) + 1

# Save the updated DataFrame back to the same file
df.to_csv(path, index=False)
```

```
print("Updated CSV with Week column saved!")
```

```
Updated CSV with Week column saved!
```

```
/var/folders/qw/jyz61sns2534v_vwqrk2t1nc0000gn/T/ipykernel_39405/1887114791.py:3
: UserWarning: Could not infer format, so each element will be parsed
individually, falling back to `dateutil`. To ensure parsing is consistent and
as-expected, please specify a format.
  df['Date'] = pd.to_datetime(df['Date'])
```

2.3 4 Position Specific Training/Testing Shap values

[13]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

print("="*80)
print(" POSITION-SPECIFIC ML TRAINING")
print("="*80)

# ==========================================
# DATA LOADING AND PREPARATION
# ==========================================
# Path to your rebalanced dataset
path = '/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main Notebook/
↪Data Folder/DataCombined/real_madrid_rebalanced_scores.csv'

# Load data
df = pd.read_csv(path)
print(f"Dataset shape: {df.shape}")
print(f"Columns: {df.columns.tolist()}")

# Add Week column if it doesn't exist
if 'Week' not in df.columns:
    if 'Date' in df.columns:
        df['Date'] = pd.to_datetime(df['Date'])
        df['Week'] = df['Date'].dt.isocalendar().week
    else:
        # Simulated weeks for demonstration
        df['Week'] = (df.index // 10) + 1

    # Save back to file
    df.to_csv(path, index=False)
    print("Week column added and saved!")

print(f"Week range: {df['Week'].min()} - {df['Week'].max()}")
```

```python
print(f"Position groups: {df['Position_Group'].unique()}")

# ==========================================
# POSITION-SPECIFIC METRIC DEFINITIONS
# ==========================================

# Updated position metrics based on multicollinearity-free variables (REVISED␣
↪FINAL LIST)
position_metrics = {
    'Forward': {
        'core_metrics': ['Gls', 'Ast', 'SoT', 'Expected xG', 'Expected xAG',␣
↪'Take-Ons Succ'],
        'secondary_metrics': ['Min'],
        'description': 'Goal scoring and creativity metrics␣
↪(multicollinearity-free)'
    },
    'Midfield': {
        'core_metrics': ['Passes Cmp%', 'KP', 'Tkl', 'Carries PrgC', 'Passes␣
↪PrgP', 'Touches'],
        'secondary_metrics': ['Min'],
        'description': 'Passing, creativity, and defensive contribution metrics␣
↪(multicollinearity-free)'
    },
    'Defense': {
        'core_metrics': ['Int', 'Blocks', 'Clr', 'Tackles TklW', 'Tackles Def␣
↪3rd', 'Tackles Mid 3rd', 'Blocks Sh', 'Blocks Pass'],
        'secondary_metrics': ['Min'],
        'description': 'Defensive actions and positioning metrics␣
↪(multicollinearity-free)'
    },
    'Goalkeeper': {
        'core_metrics': ['Total Cmp%', 'Err', 'Total PrgDist', 'Short Cmp%',␣
↪'Medium Cmp%', 'Total Cmp', 'Short Att'],
        'secondary_metrics': ['Min'],
        'description': 'Distribution accuracy and consistency metrics␣
↪(multicollinearity-free)'
    }
}

def get_available_metrics(df, position_metrics_dict):
    """Get metrics that actually exist in the dataset"""
    available_metrics = {}
    for position, metrics in position_metrics_dict.items():
        # Check both core and secondary metrics
        available_core = []
        for m in metrics['core_metrics']:
```

```
            if m in df.columns:
                available_core.append(m)
            elif f' {m}' in df.columns:  # Handle space prefix
                available_core.append(f' {m}')

        available_secondary = []
        for m in metrics['secondary_metrics']:
            if m in df.columns:
                available_secondary.append(m)
            elif f' {m}' in df.columns:  # Handle space prefix
                available_secondary.append(f' {m}')

        available_metrics[position] = {
            'core_metrics': available_core,
            'secondary_metrics': available_secondary,
            'all_metrics': available_core + available_secondary,
            'description': metrics['description']
        }

        print(f"\n{position}:")
        print(f"  Available core metrics ({len(available_core)}):␣
↪{available_core}")
        print(f"  Available secondary metrics ({len(available_secondary)}):␣
↪{available_secondary}")

    return available_metrics

available_metrics = get_available_metrics(df, position_metrics)

# ==========================================
# POSITION-SPECIFIC TRAIN/TEST SPLIT
# ==========================================

def create_position_datasets(df, position, metrics_list, test_weeks=2):
    """
    Create position-specific train/test datasets
    """
    print(f"\n{'='*60}")
    print(f"CREATING DATASETS FOR {position.upper()}")
    print(f"{'='*60}")

    # Filter by position and remove rows with NaN in Rebalanced_Score
    position_data = df[(df['Position_Group'] == position) &␣
↪(df['Rebalanced_Score'].notna())].copy()

    if len(position_data) == 0:
        print(f"No data found for {position} with valid Rebalanced_Score")
```

```
        return None

    print(f"Total {position} observations with valid scores:␣
↪{len(position_data)}")
    print(f"Unique players: {position_data['Player'].nunique()}")
    print(f"Metrics to use: {len(metrics_list)}")

    # Check which metrics are available
    available_metrics_for_pos = [m for m in metrics_list if m in position_data.
↪columns]
    missing_metrics = [m for m in metrics_list if m not in position_data.
↪columns]

    print(f"Available metrics ({len(available_metrics_for_pos)}):␣
↪{available_metrics_for_pos}")
    if missing_metrics:
        print(f"Missing metrics ({len(missing_metrics)}): {missing_metrics}")

    if len(available_metrics_for_pos) < 3:
        print(f"Insufficient metrics for {position} (need at least 3)")
        return None

    # Time-based split (latest weeks for testing)
    latest_week = position_data['Week'].max()
    test_start_week = latest_week - test_weeks + 1

    train_data = position_data[position_data['Week'] < test_start_week]
    test_data = position_data[position_data['Week'] >= test_start_week]

    print(f"Training weeks: {train_data['Week'].min()} - {train_data['Week'].
↪max()}")
    print(f"Testing weeks: {test_data['Week'].min()} - {test_data['Week'].
↪max()}")
    print(f"Train size: {len(train_data)}, Test size: {len(test_data)}")

    if len(train_data) < 10:
        print(f"Insufficient training data for {position} (need at least 10)")
        return None

    if len(test_data) < 1:
        print(f"Insufficient test data for {position} (need at least 1)")
        return None

    # Prepare features and target
    X_train = train_data[available_metrics_for_pos].fillna(0)
    y_train = train_data['Rebalanced_Score']
```

```
    X_test = test_data[available_metrics_for_pos].fillna(0)
    y_test = test_data['Rebalanced_Score']

    # Store additional info
    train_players = train_data['Player'].tolist()
    test_players = test_data['Player'].tolist()

    return {
        'position': position,
        'X_train': X_train,
        'y_train': y_train,
        'X_test': X_test,
        'y_test': y_test,
        'train_players': train_players,
        'test_players': test_players,
        'metrics_used': available_metrics_for_pos,
        'train_data': train_data,
        'test_data': test_data
    }

# ==========================================
# CREATE DATASETS FOR ALL POSITIONS
# ==========================================

position_datasets = {}

for position in ['Forward', 'Midfield', 'Defense', 'Goalkeeper']:
    if position in available_metrics:
        # Use all available metrics for this position
        metrics_to_use = available_metrics[position]['all_metrics']

        dataset = create_position_datasets(df, position, metrics_to_use)

        if dataset is not None:
            position_datasets[position] = dataset
        else:
            print(f"Skipping {position} due to insufficient data")

print(f"\nSuccessfully created datasets for {len(position_datasets)} positions: 
 ↪{list(position_datasets.keys())}")

# ==========================================
# TRAIN POSITION-SPECIFIC MODELS
# ==========================================

def train_position_model(dataset_info):
```

```python
    """
    Train Random Forest model for specific position
    """
    position = dataset_info['position']
    X_train = dataset_info['X_train']
    y_train = dataset_info['y_train']
    X_test = dataset_info['X_test']
    y_test = dataset_info['y_test']

    print(f"\nTRAINING MODEL FOR {position.upper()}")
    print("-" * 50)

    # Train Random Forest model
    model = RandomForestRegressor(
        n_estimators=100,
        max_depth=10,
        min_samples_split=5,
        min_samples_leaf=2,
        random_state=42
    )

    model.fit(X_train, y_train)

    # Make predictions
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Calculate metrics
    train_r2 = r2_score(y_train, y_train_pred)
    test_r2 = r2_score(y_test, y_test_pred)
    train_mae = mean_absolute_error(y_train, y_train_pred)
    test_mae = mean_absolute_error(y_test, y_test_pred)
    train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
    test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))

    print(f"MODEL PERFORMANCE:")
    print(f"  Training R²: {train_r2:.3f}")
    print(f"  Testing R²:  {test_r2:.3f}")
    print(f"  Training MAE: {train_mae:.3f}")
    print(f"  Testing MAE:  {test_mae:.3f}")
    print(f"  Training RMSE: {train_rmse:.3f}")
    print(f"  Testing RMSE:  {test_rmse:.3f}")

    # Feature importance
    feature_importance = pd.DataFrame({
        'metric': X_train.columns,
        'importance': model.feature_importances_
```

```
    }).sort_values('importance', ascending=False)

    print(f"\nTOP 5 IMPORTANT METRICS:")
    for _, row in feature_importance.head().iterrows():
        print(f"  {row['metric']}: {row['importance']:.3f}")

    return {
        'model': model,
        'feature_importance': feature_importance,
        'metrics': {
            'train_r2': train_r2,
            'test_r2': test_r2,
            'train_mae': train_mae,
            'test_mae': test_mae,
            'train_rmse': train_rmse,
            'test_rmse': test_rmse
        },
        'predictions': {
            'y_train_pred': y_train_pred,
            'y_test_pred': y_test_pred
        }
    }

# Train models for each position
position_models = {}

for position, dataset in position_datasets.items():
    model_info = train_position_model(dataset)
    position_models[position] = model_info

print(f"\nAll position-specific models trained successfully!")
print(f"Trained models for: {list(position_models.keys())}")

# ==========================================
# MODEL SUMMARY
# ==========================================

print("\n" + "="*80)
print("POSITION-SPECIFIC MODEL SUMMARY")
print("="*80)

print(f"{'Position':<12} {'Test R²':<10} {'Test MAE':<10} {'Top Metric':<20}␣
↪{'Importance':<10}")
print("-" * 70)

for position, model_info in position_models.items():
    test_r2 = model_info['metrics']['test_r2']
```

```python
    test_mae = model_info['metrics']['test_mae']
    top_metric = model_info['feature_importance'].iloc[0]['metric']
    top_importance = model_info['feature_importance'].iloc[0]['importance']

    print(f"{position:<12} {test_r2:<10.3f} {test_mae:<10.3f} {top_metric:<20}␣
↪{top_importance:<10.3f}")

print("\nPosition-specific ML training complete!")
```

```
================================================================================
 POSITION-SPECIFIC ML TRAINING
================================================================================
Dataset shape: (5737, 72)
Columns: ['Date', 'Competition', 'Opponent', 'Player', '#', 'Nation', 'Pos',
'Age', 'Min', ' Gls', ' Ast', ' PK', ' PKatt', ' Sh', ' SoT', ' CrdY', ' CrdR',
' Int', 'Match URL', 'Season', ' Touches', ' Tkl', ' Blocks', 'Expected xG',
'Expected npxG', 'Expected xAG', 'Passes Cmp', 'Passes Att', 'Passes Cmp%',
'Passes PrgP', 'Carries Carries', 'Carries PrgC', 'Take-Ons Att', 'Take-Ons
Succ', 'Tackles Tkl', 'Tackles TklW', 'Tackles Def 3rd', 'Tackles Mid 3rd',
'Tackles Att 3rd', 'Challenges Tkl', 'Challenges Att', 'Challenges Lost',
'Blocks Blocks', 'Blocks Sh', 'Blocks Pass', 'Int', 'Tkl+Int', 'Clr', 'Err',
'Total Cmp', 'Total Att', 'Total Cmp%', 'Total TotDist', 'Total PrgDist', 'Short
Cmp', 'Short Att', 'Short Cmp%', 'Medium Cmp', 'Medium Att', 'Medium Cmp%',
'Long Cmp', 'Long Att', 'Ast', 'xAG', 'xA', 'KP', 'PPA', 'CrsPA', 'PrgP',
'Position_Group', 'Rebalanced_Score', 'Week']
Week range: 1 - 53
Position groups: ['Forward' 'Midfield' 'Defense' 'Goalkeeper']

Forward:
  Available core metrics (6): [' Gls', 'Ast', ' SoT', 'Expected xG', 'Expected
xAG', 'Take-Ons Succ']
  Available secondary metrics (1): ['Min']

Midfield:
  Available core metrics (6): ['Passes Cmp%', 'KP', ' Tkl', 'Carries PrgC',
'Passes PrgP', ' Touches']
  Available secondary metrics (1): ['Min']

Defense:
  Available core metrics (8): ['Int', ' Blocks', 'Clr', 'Tackles TklW', 'Tackles
Def 3rd', 'Tackles Mid 3rd', 'Blocks Sh', 'Blocks Pass']
  Available secondary metrics (1): ['Min']

Goalkeeper:
  Available core metrics (7): ['Total Cmp%', 'Err', 'Total PrgDist', 'Short
Cmp%', 'Medium Cmp%', 'Total Cmp', 'Short Att']
  Available secondary metrics (1): ['Min']
```

```
============================================================
CREATING DATASETS FOR FORWARD
============================================================
Total Forward observations with valid scores: 1695
Unique players: 31
Metrics to use: 7
Available metrics (7): [' Gls', 'Ast', ' SoT', 'Expected xG', 'Expected xAG',
'Take-Ons Succ', 'Min']
Training weeks: 1 - 51
Testing weeks: 52 - 53
Train size: 1669, Test size: 26

============================================================
CREATING DATASETS FOR MIDFIELD
============================================================
Total Midfield observations with valid scores: 1823
Unique players: 34
Metrics to use: 7
Available metrics (7): ['Passes Cmp%', 'KP', ' Tkl', 'Carries PrgC', 'Passes
PrgP', ' Touches', 'Min']
Training weeks: 1 - 51
Testing weeks: 52 - 53
Train size: 1801, Test size: 22

============================================================
CREATING DATASETS FOR DEFENSE
============================================================
Total Defense observations with valid scores: 1823
Unique players: 28
Metrics to use: 9
Available metrics (9): ['Int', ' Blocks', 'Clr', 'Tackles TklW', 'Tackles Def
3rd', 'Tackles Mid 3rd', 'Blocks Sh', 'Blocks Pass', 'Min']
Training weeks: 1 - 51
Testing weeks: 52 - 53
Train size: 1801, Test size: 22

============================================================
CREATING DATASETS FOR GOALKEEPER
============================================================
Total Goalkeeper observations with valid scores: 396
Unique players: 6
Metrics to use: 8
Available metrics (8): ['Total Cmp%', 'Err', 'Total PrgDist', 'Short Cmp%',
'Medium Cmp%', 'Total Cmp', 'Short Att', 'Min']
Training weeks: 1 - 51
Testing weeks: 52 - 53
Train size: 391, Test size: 5
```

```
Successfully created datasets for 4 positions: ['Forward', 'Midfield',
'Defense', 'Goalkeeper']

TRAINING MODEL FOR FORWARD
--------------------------------------------------
MODEL PERFORMANCE:
  Training R²: 0.993
  Testing R²:  0.969
  Training MAE: 0.445
  Testing MAE:  0.790
  Training RMSE: 0.671
  Testing RMSE:  1.257

TOP 5 IMPORTANT METRICS:
   Gls: 0.705
  Expected xG: 0.113
  Ast: 0.107
  Min: 0.041
   SoT: 0.032

TRAINING MODEL FOR MIDFIELD
--------------------------------------------------
MODEL PERFORMANCE:
  Training R²: 0.918
  Testing R²:  0.801
  Training MAE: 0.726
  Testing MAE:  1.034
  Training RMSE: 1.152
  Testing RMSE:  1.630

TOP 5 IMPORTANT METRICS:
  KP: 0.583
  Passes Cmp%: 0.151
  Passes PrgP: 0.080
   Tkl: 0.076
  Min: 0.062

TRAINING MODEL FOR DEFENSE
--------------------------------------------------
MODEL PERFORMANCE:
  Training R²: 0.993
  Testing R²:  0.995
  Training MAE: 0.173
  Testing MAE:  0.141
  Training RMSE: 0.353
  Testing RMSE:  0.248

TOP 5 IMPORTANT METRICS:
```

```
   Blocks: 0.446
  Int: 0.260
  Tackles TklW: 0.188
  Min: 0.054
  Clr: 0.050

TRAINING MODEL FOR GOALKEEPER
--------------------------------------------------
MODEL PERFORMANCE:
  Training R²: 0.992
  Testing R²:  0.641
  Training MAE: 0.062
  Testing MAE:  0.808
  Training RMSE: 0.219
  Testing RMSE:  1.290

TOP 5 IMPORTANT METRICS:
  Total Cmp%: 0.971
  Err: 0.025
  Short Att: 0.001
  Total PrgDist: 0.001
  Total Cmp: 0.001

All position-specific models trained successfully!
Trained models for: ['Forward', 'Midfield', 'Defense', 'Goalkeeper']

================================================================================
POSITION-SPECIFIC MODEL SUMMARY
================================================================================
Position     Test R²    Test MAE   Top Metric           Importance
----------------------------------------------------------------------
Forward      0.969      0.790       Gls                 0.705
Midfield     0.801      1.034      KP                   0.583
Defense      0.995      0.141       Blocks              0.446
Goalkeeper   0.641      0.808      Total Cmp%           0.971

Position-specific ML training complete!
```

2.4 8 Train XGBoost

[14]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in /opt/miniconda3/lib/python3.12/site-
packages (3.0.2)
Requirement already satisfied: numpy in /opt/miniconda3/lib/python3.12/site-
packages (from xgboost) (1.26.4)
Requirement already satisfied: scipy in /opt/miniconda3/lib/python3.12/site-
packages (from xgboost) (1.15.3)
```

[15]:
```
#XGBOOST

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import warnings
import os
from pathlib import Path

# Load data
path = '/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main Notebook/
 ↪Data Folder/DataCombined/real_madrid_rebalanced_scores.csv'
df = pd.read_csv(path)
print(f"Data loaded successfully: {df.shape}")
```

```
Data loaded successfully: (5737, 72)
```

[16]:
```
# ==========================================
# XGBOOST WITH RAW MATCH STATISTICS ONLY
# ==========================================

import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt

print("XGBOOST TRAINING WITH RAW MATCH STATISTICS")
print("="*60)

# Load your original dataset (without feature engineering)
path = '/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main Notebook/
 ↪Data Folder/DataCombined/real_madrid_rebalanced_scores.csv'
df = pd.read_csv(path)

print(f"Dataset shape: {df.shape}")
print(f"Available columns: {list(df.columns)}")

# ==========================================
# SELECT RAW MATCH STATISTICS ONLY
# ==========================================

# Define the raw match statistics you want to use
raw_match_stats = [
    'Min',            # Minutes played
```

```
    ' Gls',          # Goals (note the space)
    ' Ast',          # Assists
    ' SoT',          # Shots on target
    ' KP',           # Key passes
    ' Tkl',          # Tackles
    ' Int',          # Interceptions
    ' Blocks',       # Blocks
    ' Clr',          # Clearances
    'Passes Cmp%',   # Pass completion %
    'Expected xG',   # Expected goals
    'Expected xAG',  # Expected assists
    'Take-Ons Succ', # Successful take-ons
    'Carries PrgC',  # Progressive carries
    'Passes PrgP',   # Progressive passes
    'Touches'        # Total touches
]

# Check which features actually exist in your dataset
available_features = []
missing_features = []

for feature in raw_match_stats:
    if feature in df.columns:
        available_features.append(feature)
    else:
        missing_features.append(feature)

print(f"\nAvailable features ({len(available_features)}): {available_features}")
if missing_features:
    print(f"Missing features ({len(missing_features)}): {missing_features}")

# ==========================================
# PREPARE DATA FOR TRAINING
# ==========================================

def prepare_raw_data(df, features, test_weeks=4):
    """Prepare data using only raw match statistics"""

    # Remove rows with missing target variable
    df_clean = df[df['Rebalanced_Score'].notna()].copy()

    # Create week column if not exists
    if 'Week' not in df_clean.columns:
        df_clean['Date'] = pd.to_datetime(df_clean['Date'])
        df_clean['Week'] = df_clean['Date'].dt.isocalendar().week

    print(f"Clean dataset: {len(df_clean)} observations")
```

```
    print(f"Week range: {df_clean['Week'].min()} - {df_clean['Week'].max()}")

    # Time-based split
    latest_week = df_clean['Week'].max()
    test_start_week = latest_week - test_weeks + 1

    train_data = df_clean[df_clean['Week'] < test_start_week]
    test_data = df_clean[df_clean['Week'] >= test_start_week]

    print(f"Training weeks: {train_data['Week'].min()} - {train_data['Week'].
↪max()}")
    print(f"Test weeks: {test_data['Week'].min()} - {test_data['Week'].max()}")
    print(f"Train: {len(train_data)}, Test: {len(test_data)}")

    # Prepare features and target
    X_train = train_data[features].fillna(0)
    y_train = train_data['Rebalanced_Score']
    X_test = test_data[features].fillna(0)
    y_test = test_data['Rebalanced_Score']

    return {
        'X_train': X_train,
        'y_train': y_train,
        'X_test': X_test,
        'y_test': y_test,
        'train_data': train_data,
        'test_data': test_data,
        'features': features
    }

# ==========================================
# TRAIN MODELS WITH RAW STATISTICS
# ==========================================

# 1. Combined model (all positions)
print("\n" + "="*50)
print("TRAINING COMBINED MODEL")
print("="*50)

combined_raw_data = prepare_raw_data(df, available_features)

def train_raw_xgb(data_dict, name):
    """Train XGBoost with raw features"""
    print(f"\nTraining {name} XGBoost model...")

    X_train = data_dict['X_train']
    y_train = data_dict['y_train']
```

```
    X_test = data_dict['X_test']
    y_test = data_dict['y_test']

    # XGBoost model
    model = xgb.XGBRegressor(
        n_estimators=200,
        max_depth=6,
        learning_rate=0.1,
        subsample=0.8,
        colsample_bytree=0.8,
        reg_alpha=0.1,
        reg_lambda=1.0,
        random_state=42,
        verbosity=0
    )

    # Train
    model.fit(X_train, y_train)

    # Predictions
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Metrics
    train_r2 = r2_score(y_train, y_train_pred)
    test_r2 = r2_score(y_test, y_test_pred)
    train_mae = mean_absolute_error(y_train, y_train_pred)
    test_mae = mean_absolute_error(y_test, y_test_pred)
    train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
    test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))

    print(f"Performance:")
    print(f"  Train - R²: {train_r2:.3f}, MAE: {train_mae:.3f}, RMSE:␣
↪{train_rmse:.3f}")
    print(f"  Test  - R²: {test_r2:.3f}, MAE: {test_mae:.3f}, RMSE: {test_rmse:.
↪3f}")

    # Feature importance
    importance = pd.DataFrame({
        'feature': X_train.columns,
        'importance': model.feature_importances_
    }).sort_values('importance', ascending=False)

    print(f"\nTop 5 Raw Features:")
    for _, row in importance.head().iterrows():
        print(f"  {row['feature']}: {row['importance']:.3f}")
```

```python
    return {
        'model': model,
        'importance': importance,
        'metrics': {
            'train_r2': train_r2, 'test_r2': test_r2,
            'train_mae': train_mae, 'test_mae': test_mae,
            'train_rmse': train_rmse, 'test_rmse': test_rmse
        },
        'predictions': {'train': y_train_pred, 'test': y_test_pred},
        'actuals': {'train': y_train, 'test': y_test},
        'data': data_dict
    }

# Train combined model
raw_models = {}
raw_models['Combined'] = train_raw_xgb(combined_raw_data, 'Combined')

# 2. Position-specific models with raw features
print("\n" + "="*50)
print("TRAINING POSITION-SPECIFIC MODELS")
print("="*50)

for position in ['Forward', 'Midfield', 'Defense', 'Goalkeeper']:
    position_data = df[df['Position_Group'] == position]

    if len(position_data) < 50:  # Need sufficient data
        print(f"Skipping {position} - insufficient data ({len(position_data)} samples)")
        continue

    try:
        pos_data = prepare_raw_data(position_data, available_features)
        raw_models[position] = train_raw_xgb(pos_data, position)
    except Exception as e:
        print(f"Error training {position} model: {e}")

# ===========================================
# COMPARISON: RAW vs ENGINEERED FEATURES
# ===========================================

print("\n" + "="*70)
print("RAW STATISTICS MODEL SUMMARY")
print("="*70)

print(f"{'Model':<12} {'Test R²':<10} {'Test MAE':<10} {'Test RMSE':<11} {'Top Feature':<20}")
print("-" * 70)
```

```
for name, model_info in raw_models.items():
    metrics = model_info['metrics']
    top_feature = model_info['importance'].iloc[0]['feature']

    print(f"{name:<12} {metrics['test_r2']:<10.3f} {metrics['test_mae']:<10.3f}␣
↪"
          f"{metrics['test_rmse']:<11.3f} {top_feature:<20}")

# ==========================================
# SIMPLE PREDICTION FUNCTION
# ==========================================

def predict_with_raw_features(player_name, model_type='Combined'):
    """Predict using raw match statistics only"""

    if model_type not in raw_models:
        print(f"Model {model_type} not available")
        available = list(raw_models.keys())
        print(f"Available models: {available}")
        return None

    model_info = raw_models[model_type]
    test_data = model_info['data']['test_data']

    # Find player
    player_matches = test_data[test_data['Player'].str.contains(player_name,␣
↪case=False, na=False)]

    if len(player_matches) == 0:
        print(f"Player '{player_name}' not found in {model_type} test data")
        return None

    # Use most recent match
    latest_match = player_matches.iloc[-1]

    # Get raw features
    features = model_info['data']['features']
    X_latest = latest_match[features].fillna(0).values.reshape(1, -1)

    # Predict
    prediction = model_info['model'].predict(X_latest)[0]
    actual = latest_match['Rebalanced_Score']

    print(f"\n {player_name} Prediction (Raw Features - {model_type}):")
    print(f"   Date: {latest_match['Date']}")
    print(f"   Actual score: {actual:.3f}")
```

```python
    print(f"   Predicted: {prediction:.3f}")
    print(f"   Difference: {abs(actual - prediction):.3f}")

    return prediction

print(f"\n Raw statistics models ready!")
print(f"  Trained {len(raw_models)} models using only original match data")
print(f"  Use: predict_with_raw_features('Player Name', 'Model Type')")

# Show what raw features are being used
print(f"\n Raw features being used ({len(available_features)}):")
for i, feature in enumerate(available_features):
    print(f"  {i+1:2d}. {feature}")
```

```
XGBOOST TRAINING WITH RAW MATCH STATISTICS
============================================================
Dataset shape: (5737, 72)
Available columns: ['Date', 'Competition', 'Opponent', 'Player', '#', 'Nation',
'Pos', 'Age', 'Min', ' Gls', ' Ast', ' PK', ' PKatt', ' Sh', ' SoT', ' CrdY', '
CrdR', ' Int', 'Match URL', 'Season', ' Touches', ' Tkl', ' Blocks', 'Expected
xG', 'Expected npxG', 'Expected xAG', 'Passes Cmp', 'Passes Att', 'Passes Cmp%',
'Passes PrgP', 'Carries Carries', 'Carries PrgC', 'Take-Ons Att', 'Take-Ons
Succ', 'Tackles Tkl', 'Tackles TklW', 'Tackles Def 3rd', 'Tackles Mid 3rd',
'Tackles Att 3rd', 'Challenges Tkl', 'Challenges Att', 'Challenges Lost',
'Blocks Blocks', 'Blocks Sh', 'Blocks Pass', 'Int', 'Tkl+Int', 'Clr', 'Err',
'Total Cmp', 'Total Att', 'Total Cmp%', 'Total TotDist', 'Total PrgDist', 'Short
Cmp', 'Short Att', 'Short Cmp%', 'Medium Cmp', 'Medium Att', 'Medium Cmp%',
'Long Cmp', 'Long Att', 'Ast', 'xAG', 'xA', 'KP', 'PPA', 'CrsPA', 'PrgP',
'Position_Group', 'Rebalanced_Score', 'Week']

Available features (13): ['Min', ' Gls', ' Ast', ' SoT', ' Tkl', ' Int', '
Blocks', 'Passes Cmp%', 'Expected xG', 'Expected xAG', 'Take-Ons Succ', 'Carries
PrgC', 'Passes PrgP']
Missing features (3): [' KP', ' Clr', 'Touches']

==================================================
TRAINING COMBINED MODEL
==================================================
Clean dataset: 5737 observations
Week range: 1 - 53
Training weeks: 1 - 49
Test weeks: 50 - 53
Train: 5399, Test: 338

Training Combined XGBoost model…
Performance:
  Train - R²: 0.888, MAE: 1.462, RMSE: 1.983
  Test  - R²: 0.758, MAE: 2.204, RMSE: 2.873
```

```
Top 5 Raw Features:
   Gls: 0.342
   Ast: 0.215
  Passes PrgP: 0.064
  Expected xG: 0.061
   SoT: 0.055

==================================================
TRAINING POSITION-SPECIFIC MODELS
==================================================
Clean dataset: 1695 observations
Week range: 1 - 53
Training weeks: 1 - 49
Test weeks: 50 - 53
Train: 1585, Test: 110

Training Forward XGBoost model…
Performance:
  Train - R²: 0.999, MAE: 0.202, RMSE: 0.305
  Test  - R²: 0.985, MAE: 0.697, RMSE: 0.925

Top 5 Raw Features:
   Gls: 0.729
   Ast: 0.156
  Expected xG: 0.061
   SoT: 0.034
  Min: 0.007
Clean dataset: 1823 observations
Week range: 1 - 53
Training weeks: 1 - 49
Test weeks: 50 - 53
Train: 1724, Test: 99

Training Midfield XGBoost model…
Performance:
  Train - R²: 0.988, MAE: 0.311, RMSE: 0.440
  Test  - R²: 0.828, MAE: 1.131, RMSE: 1.518

Top 5 Raw Features:
   Ast: 0.688
  Expected xAG: 0.149
  Passes PrgP: 0.048
   Tkl: 0.035
  Passes Cmp%: 0.028
Clean dataset: 1823 observations
Week range: 1 - 53
Training weeks: 1 - 49
```

```
Test weeks: 50 - 53
Train: 1718, Test: 105

Training Defense XGBoost model…
Performance:
  Train - R²: 0.974, MAE: 0.509, RMSE: 0.707
  Test  - R²: 0.764, MAE: 1.464, RMSE: 2.022

Top 5 Raw Features:
   Blocks: 0.468
   Int: 0.275
   Tkl: 0.104
  Min: 0.027
  Carries PrgC: 0.017
Clean dataset: 396 observations
Week range: 1 - 53
Training weeks: 1 - 49
Test weeks: 50 - 53
Train: 372, Test: 24

Training Goalkeeper XGBoost model…
Performance:
  Train - R²: 0.988, MAE: 0.078, RMSE: 0.269
  Test  - R²: 0.803, MAE: 0.439, RMSE: 0.933

Top 5 Raw Features:
  Passes Cmp%: 0.746
  Take-Ons Succ: 0.052
   Tkl: 0.048
   Int: 0.041
  Min: 0.040

======================================================================
RAW STATISTICS MODEL SUMMARY
======================================================================
Model        Test R²    Test MAE   Test RMSE   Top Feature
----------------------------------------------------------------------
Combined     0.758      2.204      2.873        Gls
Forward      0.985      0.697      0.925        Gls
Midfield     0.828      1.131      1.518        Ast
Defense      0.764      1.464      2.022        Blocks
Goalkeeper   0.803      0.439      0.933       Passes Cmp%

 Raw statistics models ready!
 Trained 5 models using only original match data
 Use: predict_with_raw_features('Player Name', 'Model Type')

 Raw features being used (13):
```

1. Min
2. Gls
3. Ast
4. SoT
5. Tkl
6. Int
7. Blocks
8. Passes Cmp%
9. Expected xG
10. Expected xAG
11. Take-Ons Succ
12. Carries PrgC
13. Passes PrgP

2.5 10 | Shap values with weighted average

2.5.1 10.1 | shap with weighted All per 90 .Included xg

[17]:
```
# ==========================================
# PER-90 RATES SHAP ANALYSIS ONLY
# ==========================================

import pandas as pd
import numpy as np
import xgboost as xgb
import shap
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import warnings
warnings.filterwarnings('ignore')

print("PER-90 RATES POSITION-SPECIFIC SHAP ANALYSIS")
print("="*60)
print("Target: Weighted Score (Rebalanced_Score)")
print("Features: All metrics converted to per-90 rates")
print("Purpose: Bias-free efficiency analysis")
print("="*60)

# Load data
path = '/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main Notebook/
↪Data Folder/DataCombined/real_madrid_rebalanced_scores.csv'
df = pd.read_csv(path)

# Create Week column if needed
if 'Week' not in df.columns:
    df['Date'] = pd.to_datetime(df['Date'])
    df['Week'] = df['Date'].dt.isocalendar().week
```

```python
print(f"Dataset loaded: {df.shape}")

# ===========================================
# CREATE COMPREHENSIVE PER-90 FEATURES
# ===========================================

def create_complete_per90_features(df):
    """Create complete per-90 minute rate features"""
    print("\nCreating comprehensive per-90 minute features...")

    df_per90 = df.copy()
    df_per90['Min_Safe'] = df_per90['Min'].replace(0, 1)  # Avoid division by␣
↪zero

    # ALL volume-based stats to convert to per-90 rates
    volume_stats = [
        ' Gls', ' Ast', ' SoT', ' KP', ' Tkl', ' Int', ' Blocks', ' Clr',
        'Expected xG', 'Expected xAG', 'Take-Ons Succ', 'Carries PrgC',
        'Passes PrgP', 'Touches', 'Tackles TklW', 'Tackles Def 3rd',
        'Tackles Mid 3rd', 'Blocks Sh', 'Blocks Pass', 'Total Cmp',
        'Total PrgDist', 'Short Att'
    ]

    created_features = []

    for stat in volume_stats:
        if stat in df_per90.columns:
            # Clean column name (remove spaces, hyphens, special chars)
            clean_name = stat.strip().replace(' ', '').replace('-', '_').
↪replace('+', '')
            new_name = f"{clean_name}_Per90"

            df_per90[new_name] = (df_per90[stat] / df_per90['Min_Safe']) * 90
            df_per90[new_name] = df_per90[new_name].fillna(0)

            created_features.append(new_name)
            print(f"    {new_name}")

    print(f"\nCreated {len(created_features)} per-90 features")
    return df_per90

df_per90 = create_complete_per90_features(df)

# ===========================================
# POSITION-SPECIFIC PER-90 METRICS
# ===========================================
```

```python
# Using your exact position logic with per-90 conversions
per90_metrics = {
    'Forward': [
        'Gls_Per90',        # Goals per 90
        'Ast_Per90',        # Assists per 90
        'SoT_Per90',        # Shots on Target per 90
        'ExpectedxG_Per90', # Expected Goals per 90
        'ExpectedxAG_Per90', # Expected Assists per 90
        'TakeOnsSucc_Per90' # Take-Ons Success per 90
    ],
    'Midfield': [
        'Passes Cmp%',      # Pass completion % (already normalized)
        'KP_Per90',         # Key Passes per 90
        'Tkl_Per90',        # Tackles per 90
        'CarriesPrgC_Per90', # Progressive Carries per 90
        'PassesPrgP_Per90', # Progressive Passes per 90
        'Touches_Per90'     # Touches per 90
    ],
    'Defense': [
        'Int_Per90',        # Interceptions per 90
        'Blocks_Per90',     # Blocks per 90
        'Clr_Per90',        # Clearances per 90
        'TacklesTklW_Per90', # Tackles Won per 90
        'TacklesDef3rd_Per90', # Def 3rd Tackles per 90
        'TacklesMid3rd_Per90'  # Mid 3rd Tackles per 90
    ],
    'Goalkeeper': [
        'Total Cmp%',       # Total completion % (already normalized)
        ' Err',             # Errors (keep raw - different meaning)
        'TotalPrgDist_Per90', # Progressive Distance per 90
        'Short Cmp%',       # Short completion % (already normalized)
        'Medium Cmp%',      # Medium completion % (already normalized)
        'TotalCmp_Per90'    # Total Completions per 90
    ]
}

# Check availability
print(f"\nChecking per-90 metric availability:")
for position, metrics in per90_metrics.items():
    available = [m for m in metrics if m in df_per90.columns]
    missing = [m for m in metrics if m not in df_per90.columns]
    print(f"  {position}: {len(available)}/{len(metrics)} available")
    if missing:
        print(f"    Missing: {missing}")

# ==========================================
```

```
# PER-90 MODEL TRAINING
# ==========================================

def train_per90_model(position):
    """Train position-specific model using per-90 rates"""
    print(f"\nTraining {position} per-90 model...")

    # Filter by position
    pos_data = df_per90[(df_per90['Position_Group'] == position) &
                        (df_per90['Rebalanced_Score'].notna())].copy()

    if len(pos_data) < 30:
        print(f"  Skip {position} - insufficient data: {len(pos_data)} samples")
        return None

    # Get available metrics
    available_metrics = [m for m in per90_metrics[position] if m in pos_data.
 ↪columns]

    if len(available_metrics) < 3:
        print(f"  Skip {position} - insufficient metrics:␣
 ↪{len(available_metrics)}")
        return None

    print(f"  Using {len(available_metrics)} metrics: {available_metrics}")

    # Time-based split
    latest_week = pos_data['Week'].max()
    test_start_week = latest_week - 4 + 1

    train_data = pos_data[pos_data['Week'] < test_start_week]
    test_data = pos_data[pos_data['Week'] >= test_start_week]

    if len(train_data) < 20 or len(test_data) < 5:
        print(f"  Skip {position} - insufficient train/test split")
        return None

    # Prepare features and target
    X_train = train_data[available_metrics].fillna(0)
    y_train = train_data['Rebalanced_Score']
    X_test = test_data[available_metrics].fillna(0)
    y_test = test_data['Rebalanced_Score']

    print(f"  Train: {len(X_train)} samples, Test: {len(X_test)} samples")

    # Train XGBoost
    model = xgb.XGBRegressor(
```

```
        n_estimators=150,
        max_depth=5,
        learning_rate=0.1,
        subsample=0.8,
        random_state=42,
        verbosity=0
    )

    model.fit(X_train, y_train)

    # Evaluate
    y_pred = model.predict(X_test)
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))

    print(f"  Performance: R² = {r2:.3f}, MAE = {mae:.3f}, RMSE = {rmse:.3f}")

    return {
        'model': model,
        'X_train': X_train,
        'X_test': X_test,
        'y_train': y_train,
        'y_test': y_test,
        'y_pred': y_pred,
        'test_data': test_data,
        'metrics': available_metrics,
        'r2': r2,
        'mae': mae,
        'rmse': rmse
    }

# Train models for all positions
per90_models = {}

for position in per90_metrics.keys():
    model_info = train_per90_model(position)
    if model_info is not None:
        per90_models[position] = model_info

print(f"\n  Successfully trained {len(per90_models)} per-90 models")

# ==========================================
# COMPREHENSIVE SHAP ANALYSIS
# ==========================================

def analyze_position_shap(model_info, position, max_players=12):
```

```python
    """Comprehensive SHAP analysis for position"""
    print(f"\n{'='*50}")
    print(f"SHAP ANALYSIS: {position} (PER-90 RATES)")
    print(f"{'='*50}")

    model = model_info['model']
    X_test = model_info['X_test']
    test_data = model_info['test_data']
    metrics = model_info['metrics']

    # Get unique players
    players = test_data['Player'].unique()
    print(f"Players in test set: {len(players)}")

    if len(players) > max_players:
        players = players[:max_players]
        print(f"Analyzing first {max_players} players")

    # Create SHAP explainer
    explainer = shap.Explainer(model, X_test)
    shap_values = explainer(X_test)

    # Overall feature importance
    importance = pd.DataFrame({
        'Metric': metrics,
        'Avg_Abs_SHAP': np.mean(np.abs(shap_values.values), axis=0)
    }).sort_values('Avg_Abs_SHAP', ascending=False)

    print(f"\n{position} - Per-90 Feature Importance:")
    for i, (_, row) in enumerate(importance.iterrows()):
        print(f"  {i+1}. {row['Metric']}: {row['Avg_Abs_SHAP']:.3f}")

    # SHAP summary plot
    plt.figure(figsize=(12, 8))
    shap.summary_plot(shap_values, X_test, feature_names=metrics, show=False)
    plt.title(f'{position} - Per-90 Rates SHAP (Weighted Score Prediction)')
    plt.tight_layout()
    plt.show()

    # Player-specific SHAP analysis
    player_shap_results = {}

    for player in players:
        player_rows = test_data[test_data['Player'] == player]
        if len(player_rows) == 0:
            continue
```

```
        # Find indices in test set
        player_test_indices = []
        for idx in player_rows.index:
            if idx in test_data.index:
                position_in_test = test_data.index.get_loc(idx)
                player_test_indices.append(position_in_test)

        if player_test_indices:
            # Get SHAP values for this player
            player_shap = shap_values[player_test_indices]
            player_X = X_test.iloc[player_test_indices]

            # Average across player's games
            avg_shap = np.mean(player_shap.values, axis=0)
            avg_features = player_X.mean().values
            avg_weighted_score = player_rows['Rebalanced_Score'].mean()

            player_df = pd.DataFrame({
                'Metric': metrics,
                'SHAP_Value': avg_shap,
                'Avg_Feature_Value': avg_features,
                'Player': player,
                'Position': position,
                'Avg_Weighted_Score': avg_weighted_score
            }).sort_values('SHAP_Value', key=abs, ascending=False)

            player_shap_results[player] = player_df

    print(f"\nPlayer SHAP analysis completed for {len(player_shap_results)}␣
↪players")

    return {
        'feature_importance': importance,
        'player_results': player_shap_results,
        'shap_values': shap_values,
        'explainer': explainer
    }

# ============================================
# RUN SHAP ANALYSIS FOR ALL POSITIONS
# ============================================

shap_results = {}

for position, model_info in per90_models.items():
    try:
        shap_results[position] = analyze_position_shap(model_info, position)
```

```
    except Exception as e:
        print(f"SHAP analysis failed for {position}: {e}")

# ==========================================
# INDIVIDUAL PLAYER ANALYSIS
# ==========================================

def analyze_player_per90(player_name):
    """Detailed per-90 SHAP analysis for specific player"""
    print(f"\n{'='*60}")
    print(f"PER-90 SHAP ANALYSIS: {player_name.upper()}")
    print(f"{'='*60}")

    found = False

    for position, shap_info in shap_results.items():
        if player_name in shap_info['player_results']:
            found = True
            player_df = shap_info['player_results'][player_name]

            print(f"\nPosition: {position}")
            print(f"Average weighted score: {player_df['Avg_Weighted_Score'].
 ↪iloc[0]:.3f}")

            print(f"\nPer-90 factors that BOOST weighted score:")
            positive = player_df[player_df['SHAP_Value'] > 0].
 ↪sort_values('SHAP_Value', ascending=False)
            for _, row in positive.iterrows():
                print(f"  {row['Metric']}: +{row['SHAP_Value']:.3f} (per-90 avg:
 ↪ {row['Avg_Feature_Value']:.2f})")

            print(f"\nPer-90 factors that REDUCE weighted score:")
            negative = player_df[player_df['SHAP_Value'] < 0].
 ↪sort_values('SHAP_Value')
            for _, row in negative.iterrows():
                print(f"  {row['Metric']}: {row['SHAP_Value']:.3f} (per-90 avg:␣
 ↪{row['Avg_Feature_Value']:.2f})")

            # Visual breakdown
            plt.figure(figsize=(12, 8))
            colors = ['green' if x > 0 else 'red' for x in␣
 ↪player_df['SHAP_Value']]

            plt.barh(range(len(player_df)), player_df['SHAP_Value'],␣
 ↪color=colors, alpha=0.7)
            plt.yticks(range(len(player_df)), player_df['Metric'])
```

```
            plt.xlabel('SHAP Value (Impact on Weighted Score)')
            plt.title(f'{player_name} - Per-90 Efficiency Impact on Weighted␣
 ↪Score ({position})')
            plt.axvline(x=0, color='black', linestyle='-', alpha=0.5)

            # Add text annotations for top 3 positive and negative
            for i, (_, row) in enumerate(player_df.head(3).iterrows()):
                plt.text(row['SHAP_Value'], i, f" {row['SHAP_Value']:+.2f}",
                        va='center', fontsize=9, fontweight='bold')

            plt.tight_layout()
            plt.show()

            break

    if not found:
        print(f"Player '{player_name}' not found in per-90 SHAP results")
        # Show available players
        all_players = []
        for position, shap_info in shap_results.items():
            all_players.extend(list(shap_info['player_results'].keys()))
        if all_players:
            print(f"Available players: {sorted(set(all_players))[:10]}")

# ==========================================
# POSITION COMPARISON HEATMAP
# ==========================================

def create_per90_heatmap():
    """Create heatmap showing per-90 metric importance by position"""
    print(f"\n{'='*60}")
    print("PER-90 METRIC IMPORTANCE BY POSITION")
    print(f"{'='*60}")

    if not shap_results:
        print("No SHAP results available for heatmap")
        return None

    # Collect importance for each position
    position_importance = {}

    for position, shap_info in shap_results.items():
        importance_df = shap_info['feature_importance']
        position_importance[position] = dict(zip(importance_df['Metric'],␣
 ↪importance_df['Avg_Abs_SHAP']))

    # Get all unique metrics
```

```python
    all_metrics = set()
    for metrics in position_importance.values():
        all_metrics.update(metrics.keys())

    # Build matrix
    matrix_data = []
    for position in position_importance.keys():
        row = []
        for metric in sorted(all_metrics):
            value = position_importance[position].get(metric, 0)
            row.append(value)
        matrix_data.append(row)

    # Create heatmap
    plt.figure(figsize=(16, 8))
    heatmap_df = pd.DataFrame(matrix_data,
                              index=list(position_importance.keys()),
                              columns=sorted(all_metrics))

    sns.heatmap(heatmap_df, annot=True, fmt='.2f', cmap='viridis',
                cbar_kws={'label': 'SHAP Importance (Per-90 → Weighted Score)'})
    plt.title('Per-90 Metric Importance by Position for Weighted Score␣
↪Prediction')
    plt.xlabel('Per-90 Metrics')
    plt.ylabel('Position')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

    return heatmap_df

per90_heatmap = create_per90_heatmap()

# ==========================================
# PERFORMANCE SUMMARY
# ==========================================

print(f"\n{'='*60}")
print("PER-90 MODEL PERFORMANCE SUMMARY")
print(f"{'='*60}")

print(f"{'Position':<12} {'R²':<8} {'MAE':<8} {'RMSE':<8} {'Features':<10}")
print("-" * 50)

for position, model_info in per90_models.items():
    print(f"{position:<12} {model_info['r2']:<8.3f} {model_info['mae']:<8.3f} "
          f"{model_info['rmse']:<8.3f} {len(model_info['metrics']):<10}")
```

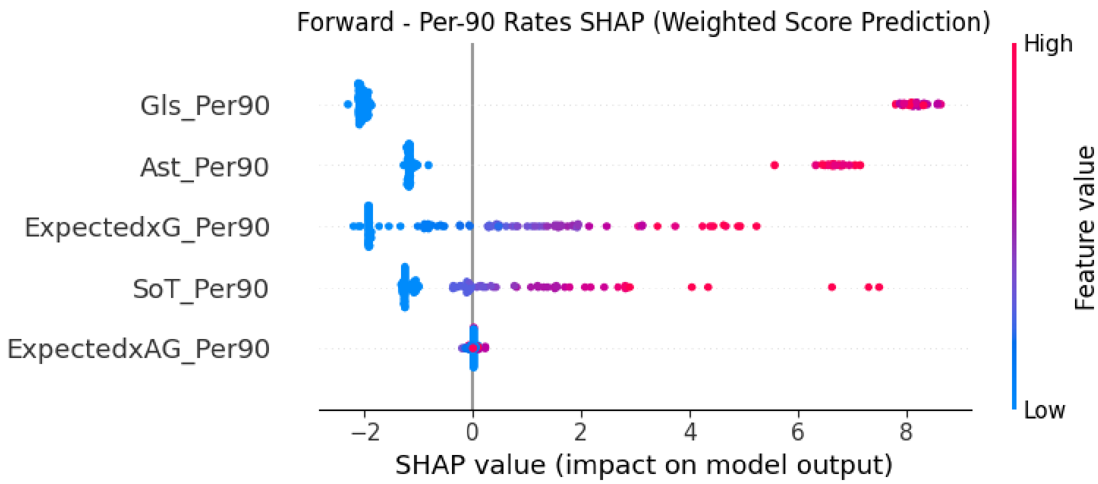
```python
# ============================================
# TOP PERFORMERS ANALYSIS
# ============================================

def analyze_top_performers():
    """Analyze players with highest positive SHAP impact"""
    print(f"\n{'='*60}")
    print("TOP PERFORMERS BY PER-90 EFFICIENCY")
    print(f"{'='*60}")

    all_players = []

    for position, shap_info in shap_results.items():
        for player, player_df in shap_info['player_results'].items():
            total_positive = player_df[player_df['SHAP_Value'] >␣
↪0]['SHAP_Value'].sum()
            avg_score = player_df['Avg_Weighted_Score'].iloc[0]

            all_players.append({
                'Player': player,
                'Position': position,
                'Positive_SHAP_Sum': total_positive,
                'Avg_Weighted_Score': avg_score
            })

    if all_players:
        top_players_df = pd.DataFrame(all_players).
↪sort_values('Positive_SHAP_Sum', ascending=False)

        print("Top 10 players by per-90 efficiency impact:")
        for i, (_, row) in enumerate(top_players_df.head(10).iterrows()):
            print(f"  {i+1:2d}. {row['Player']} ({row['Position']}): "
                  f"+{row['Positive_SHAP_Sum']:.2f} SHAP, "
                  f"{row['Avg_Weighted_Score']:.2f} avg score")

analyze_top_performers()

# ============================================
# USAGE FUNCTIONS
# ============================================

print(f"\n{'='*60}")
print("PER-90 SHAP ANALYSIS - READY FOR USE")
print(f"{'='*60}")

print(f"\n  MAIN FUNCTION:")
```

```python
print(f"   analyze_player_per90('Player Name')")

print(f"\n MODELS TRAINED:")
for position in per90_models.keys():
    r2 = per90_models[position]['r2']
    n_metrics = len(per90_models[position]['metrics'])
    print(f"   {position}: R² = {r2:.3f} using {n_metrics} per-90 metrics")

print(f"\n KEY INSIGHTS:")
print(f"   - Target: Weighted Score (bias-free prediction)")
print(f"   - Features: All metrics normalized to per-90 rates")
print(f"   - Shows pure efficiency impact (no minutes bias)")
print(f"   - Position-specific metrics for each role")
print(f"   - SHAP values show per-90 efficiency drivers")

# Demo analysis
if shap_results:
    print(f"\n DEMO ANALYSIS:")
    # Find first available player
    for position, shap_info in shap_results.items():
        if shap_info['player_results']:
            demo_player = list(shap_info['player_results'].keys())[0]
            print(f"   Running analyze_player_per90('{demo_player}')")
            analyze_player_per90(demo_player)
            break

print(f"\n PER-90 RATES SHAP ANALYSIS COMPLETE!")
print(f" Bias-free efficiency analysis ready for {len(per90_models)}␣
↪positions")
```

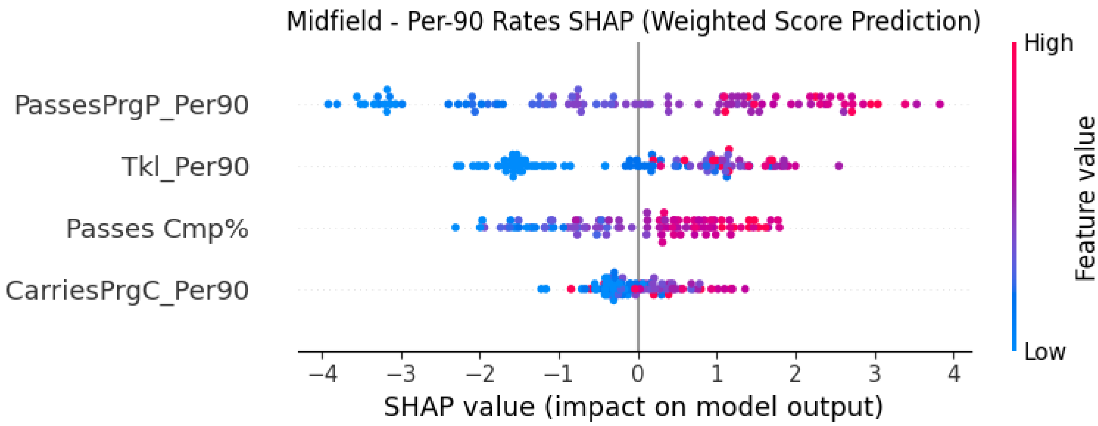
```
PER-90 RATES POSITION-SPECIFIC SHAP ANALYSIS
============================================================
Target: Weighted Score (Rebalanced_Score)
Features: All metrics converted to per-90 rates
Purpose: Bias-free efficiency analysis
============================================================
Dataset loaded: (5737, 72)

Creating comprehensive per-90 minute features…
   Gls_Per90
   Ast_Per90
   SoT_Per90
   Tkl_Per90
   Int_Per90
   Blocks_Per90
   ExpectedxG_Per90
   ExpectedxAG_Per90
   Take_OnsSucc_Per90
```

```
    CarriesPrgC_Per90
    PassesPrgP_Per90
    TacklesTklW_Per90
    TacklesDef3rd_Per90
    TacklesMid3rd_Per90
    BlocksSh_Per90
    BlocksPass_Per90
    TotalCmp_Per90
    TotalPrgDist_Per90
    ShortAtt_Per90

Created 19 per-90 features

Checking per-90 metric availability:
  Forward: 5/6 available
    Missing: ['TakeOnsSucc_Per90']
  Midfield: 4/6 available
    Missing: ['KP_Per90', 'Touches_Per90']
  Defense: 5/6 available
    Missing: ['Clr_Per90']
  Goalkeeper: 5/6 available
    Missing: [' Err']

Training Forward per-90 model…
  Using 5 metrics: ['Gls_Per90', 'Ast_Per90', 'SoT_Per90', 'ExpectedxG_Per90',
'ExpectedxAG_Per90']
  Train: 1585 samples, Test: 110 samples
  Performance: R² = 0.985, MAE = 0.610, RMSE = 0.923

Training Midfield per-90 model…
  Using 4 metrics: ['Passes Cmp%', 'Tkl_Per90', 'CarriesPrgC_Per90',
'PassesPrgP_Per90']
  Train: 1724 samples, Test: 99 samples
  Performance: R² = 0.347, MAE = 2.206, RMSE = 2.957

Training Defense per-90 model…
  Using 5 metrics: ['Int_Per90', 'Blocks_Per90', 'TacklesTklW_Per90',
'TacklesDef3rd_Per90', 'TacklesMid3rd_Per90']
  Train: 1718 samples, Test: 105 samples
  Performance: R² = 0.933, MAE = 0.832, RMSE = 1.079

Training Goalkeeper per-90 model…
  Using 5 metrics: ['Total Cmp%', 'TotalPrgDist_Per90', 'Short Cmp%', 'Medium
Cmp%', 'TotalCmp_Per90']
  Train: 372 samples, Test: 24 samples
  Performance: R² = 0.932, MAE = 0.257, RMSE = 0.547

 Successfully trained 4 per-90 models
```

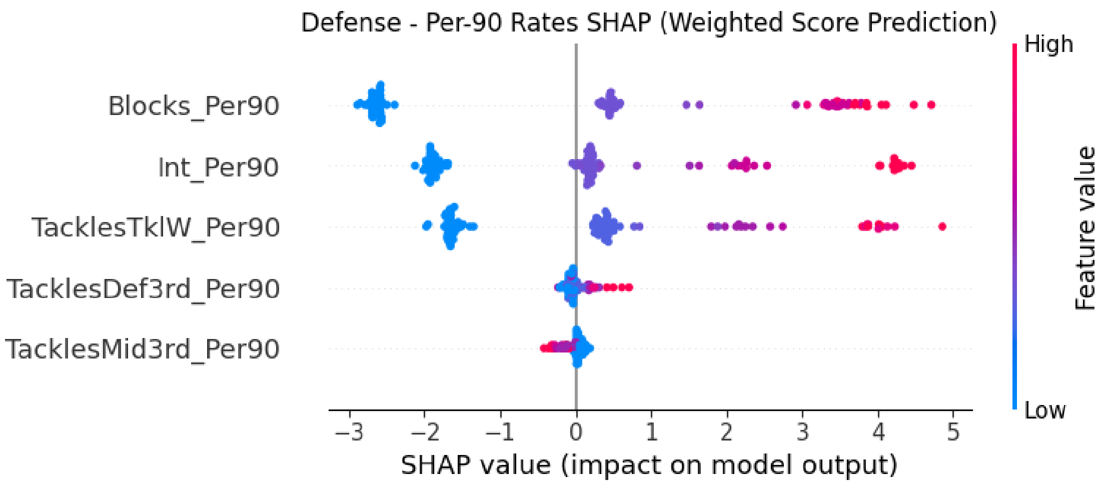
```
==================================================
SHAP ANALYSIS: Forward (PER-90 RATES)
==================================================
Players in test set: 20
Analyzing first 12 players

Forward - Per-90 Feature Importance:
  1. Gls_Per90: 3.320
  2. Ast_Per90: 2.060
  3. ExpectedxG_Per90: 1.738
  4. SoT_Per90: 1.286
  5. ExpectedxAG_Per90: 0.046
```



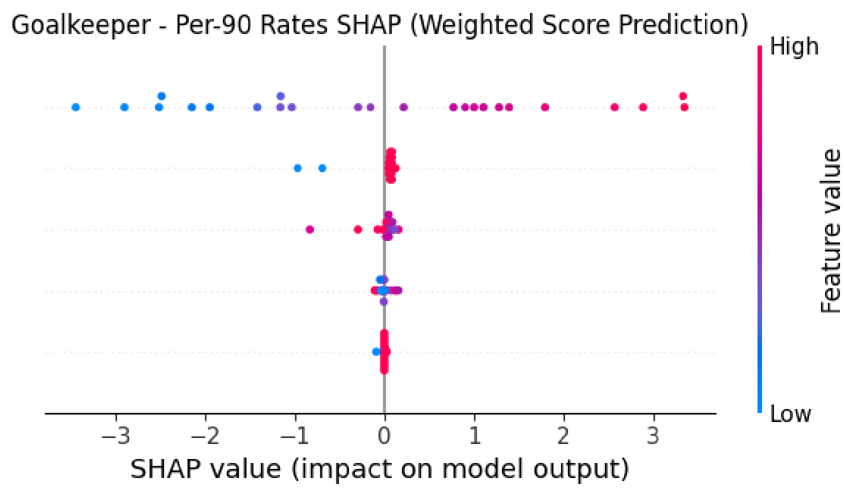

```
Player SHAP analysis completed for 12 players

==================================================
SHAP ANALYSIS: Midfield (PER-90 RATES)
==================================================
Players in test set: 14
Analyzing first 12 players

Midfield - Per-90 Feature Importance:
  1. PassesPrgP_Per90: 1.793
  2. Tkl_Per90: 1.178
  3. Passes Cmp%: 0.913
  4. CarriesPrgC_Per90: 0.391
```



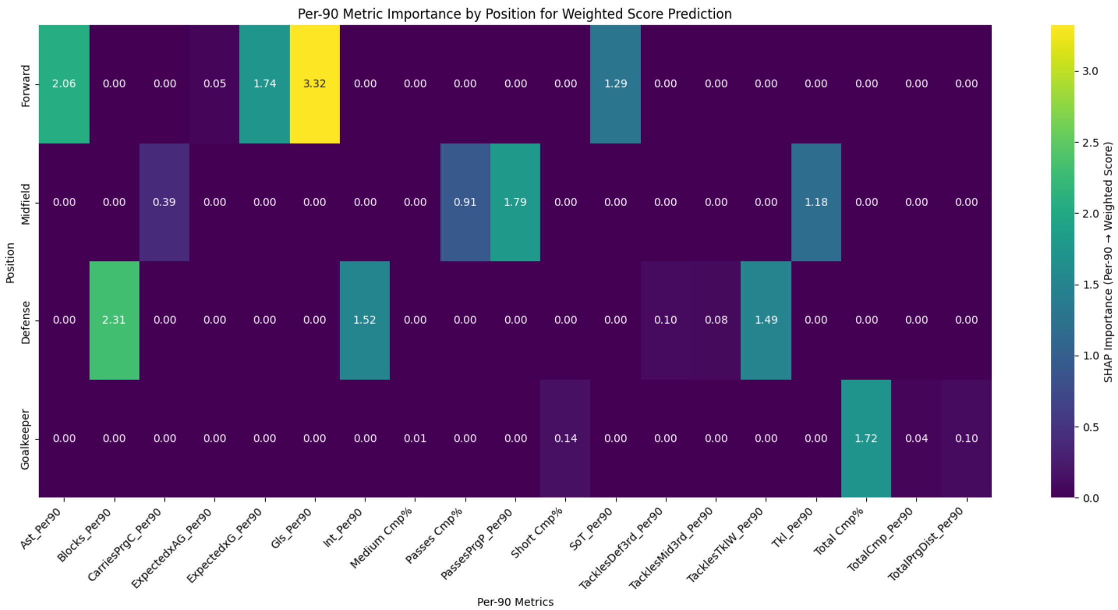

```
Player SHAP analysis completed for 12 players

==================================================
SHAP ANALYSIS: Defense (PER-90 RATES)
==================================================
Players in test set: 16
Analyzing first 12 players

Defense - Per-90 Feature Importance:
  1. Blocks_Per90: 2.309
  2. Int_Per90: 1.517
  3. TacklesTklW_Per90: 1.492
  4. TacklesDef3rd_Per90: 0.101
  5. TacklesMid3rd_Per90: 0.084
```



```
Player SHAP analysis completed for 12 players

==================================================
SHAP ANALYSIS: Goalkeeper (PER-90 RATES)
==================================================
Players in test set: 5

Goalkeeper - Per-90 Feature Importance:
  1. Total Cmp%: 1.718
  2. Short Cmp%: 0.138
  3. TotalPrgDist_Per90: 0.101
  4. TotalCmp_Per90: 0.044
  5. Medium Cmp%: 0.009
```

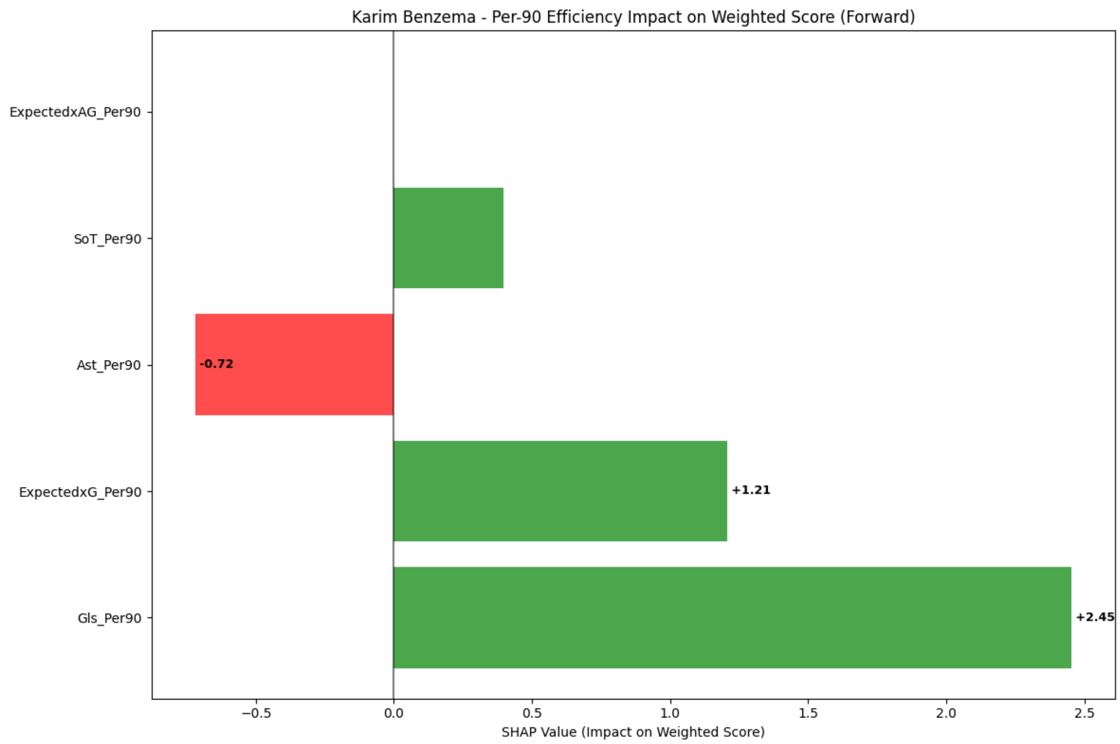



```
Player SHAP analysis completed for 5 players

============================================================
PER-90 METRIC IMPORTANCE BY POSITION
============================================================
```




```
============================================================
PER-90 MODEL PERFORMANCE SUMMARY
============================================================
Position     R²       MAE      RMSE     Features
--------------------------------------------------
Forward      0.985    0.610    0.923    5
Midfield     0.347    2.206    2.957    4
Defense      0.933    0.832    1.079    5
Goalkeeper   0.932    0.257    0.547    5

============================================================
TOP PERFORMERS BY PER-90 EFFICIENCY
============================================================
Top 10 players by per-90 efficiency impact:
   1. Jesús Vallejo (Defense): +8.33 SHAP, 16.10 avg score
   2. Karim Benzema (Forward): +4.06 SHAP, 10.25 avg score
   3. Marco Asensio (Midfield): +3.97 SHAP, 3.22 avg score
   4. Eduardo Camavinga (Midfield): +2.94 SHAP, 7.28 avg score
   5. Lucas Vázquez (Defense): +2.69 SHAP, 9.12 avg score
   6. Kepa Arrizabalaga (Goalkeeper): +2.50 SHAP, 19.00 avg score
   7. Lucas Vázquez (Forward): +2.30 SHAP, 7.79 avg score
   8. Dani Carvajal (Defense): +2.23 SHAP, 9.09 avg score
   9. Rodrygo (Forward): +2.18 SHAP, 8.46 avg score
  10. Toni Kroos (Midfield): +2.09 SHAP, 10.99 avg score

============================================================
```

```
PER-90 SHAP ANALYSIS - READY FOR USE
============================================================

 MAIN FUNCTION:
   analyze_player_per90('Player Name')

 MODELS TRAINED:
   Forward: R² = 0.985 using 5 per-90 metrics
   Midfield: R² = 0.347 using 4 per-90 metrics
   Defense: R² = 0.933 using 5 per-90 metrics
   Goalkeeper: R² = 0.932 using 5 per-90 metrics

 KEY INSIGHTS:
   - Target: Weighted Score (bias-free prediction)
   - Features: All metrics normalized to per-90 rates
   - Shows pure efficiency impact (no minutes bias)
   - Position-specific metrics for each role
   - SHAP values show per-90 efficiency drivers

 DEMO ANALYSIS:
   Running analyze_player_per90('Karim Benzema')

============================================================
PER-90 SHAP ANALYSIS: KARIM BENZEMA
============================================================

Position: Forward
Average weighted score: 10.253

Per-90 factors that BOOST weighted score:
  Gls_Per90: +2.454 (per-90 avg: 0.68)
  ExpectedxG_Per90: +1.207 (per-90 avg: 0.53)
  SoT_Per90: +0.396 (per-90 avg: 1.29)

Per-90 factors that REDUCE weighted score:
  Ast_Per90: -0.720 (per-90 avg: 0.11)
  ExpectedxAG_Per90: -0.003 (per-90 avg: 0.16)
```




```
PER-90 RATES SHAP ANALYSIS COMPLETE!
Bias-free efficiency analysis ready for 4 positions
```

2.6 11 | Shap values by main metric - xGoals

2.6.1 11.1 SHAP values by XGoals per 90 all

[18]:
```
# ==========================================
# DUAL SHAP ANALYSIS: PER-90 RATES & WEIGHTED SCORES
# ==========================================

import pandas as pd
import numpy as np
import xgboost as xgb
import shap
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import warnings
warnings.filterwarnings('ignore')

print("DUAL POSITION-SPECIFIC SHAP ANALYSIS")
print("="*70)
```

```
print("Version 1: Per-90 Minute Rates (Bias-Free)")
print("Version 2: Weighted Score Prediction")
print("="*70)

# Load data
path = '/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main Notebook/
↪Data Folder/DataCombined/real_madrid_rebalanced_scores.csv'
df = pd.read_csv(path)

# Create Week column if needed
if 'Week' not in df.columns:
    df['Date'] = pd.to_datetime(df['Date'])
    df['Week'] = df['Date'].dt.isocalendar().week

print(f"Dataset loaded: {df.shape}")

# ==========================================
# CREATE PER-90 RATE FEATURES
# ==========================================

def create_per90_features(df):
    """Create per-90 minute rate features"""
    print("\nCreating per-90 minute features...")

    df_per90 = df.copy()
    df_per90['Min_Safe'] = df_per90['Min'].replace(0, 1)  # Avoid division by␣
↪zero

    # Volume stats to convert to per-90 rates (COMPLETE LIST)
    volume_stats = [' Gls', ' Ast', ' SoT', ' KP', ' Tkl', ' Int', ' Blocks', '␣
↪Clr',
                   'Expected xG', 'Expected xAG', 'Take-Ons Succ', 'Carries␣
↪PrgC',
                   'Passes PrgP', 'Touches', 'Tackles TklW', 'Tackles Def 3rd',
                   'Tackles Mid 3rd', 'Total Cmp', 'Total PrgDist']

    for stat in volume_stats:
        if stat in df_per90.columns:
            # Clean the column name for per-90 version
            clean_name = stat.strip().replace(' ', '').replace('-', '_')
            new_name = f"{clean_name}_Per90"
            df_per90[new_name] = (df_per90[stat] / df_per90['Min_Safe']) * 90
            df_per90[new_name] = df_per90[new_name].fillna(0)
            print(f"  Created: {new_name} from {stat}")

    print(f"Created {len([s for s in volume_stats if s in df.columns])} per-90␣
↪features")
```

```python
    # Show what Forward metrics are available
    forward_per90_available = []
    expected_forward_per90 = ['Gls_Per90', 'Ast_Per90', 'SoT_Per90', 'ExpectedxG_Per90', 'ExpectedxAG_Per90', 'TakeOnsSucc_Per90']

    for metric in expected_forward_per90:
        if metric in df_per90.columns:
            forward_per90_available.append(metric)

    print(f"\nForward per-90 metrics available: {forward_per90_available}")
    print(f"All per-90 columns created: {[col for col in df_per90.columns if '_Per90' in col]}")

    return df_per90

df_per90 = create_per90_features(df)

# ===========================================
# POSITION-SPECIFIC METRICS FOR BOTH VERSIONS
# ===========================================

# Version 1: Per-90 Rates (ALL METRICS CONVERTED TO PER-90)
per90_metrics = {
    'Forward': ['Gls_Per90', 'Ast_Per90', 'SoT_Per90', 'ExpectedxAG_Per90', 'TakeOnsSucc_Per90'],
    'Midfield': ['Passes Cmp%', 'KP_Per90', 'Tkl_Per90', 'CarriesPrgC_Per90', 'PassesPrgP_Per90', 'Touches_Per90'],
    'Defense': ['Int_Per90', 'Blocks_Per90', 'Clr_Per90', 'TacklesTklW_Per90', 'TacklesDef3rd_Per90', 'TacklesMid3rd_Per90'],
    'Goalkeeper': ['Total Cmp%', ' Err', 'TotalPrgDist_Per90', 'Short Cmp%', 'Medium Cmp%', 'TotalCmp_Per90']
}

# Version 2: Raw Metrics for Weighted Score (YOUR EXACT ORIGINAL METRICS)
weighted_metrics = {
    'Forward': [' Gls', ' Ast', ' SoT', 'Expected xG', 'Expected xAG', 'Take-Ons Succ'],
    'Midfield': ['Passes Cmp%', ' KP', ' Tkl', 'Carries PrgC', 'Passes PrgP', 'Touches'],
    'Defense': [' Int', ' Blocks', ' Clr', 'Tackles TklW', 'Tackles Def 3rd', 'Tackles Mid 3rd'],
    'Goalkeeper': ['Total Cmp%', ' Err', 'Total PrgDist', 'Short Cmp%', 'Medium Cmp%', 'Total Cmp']
}
```

```python
# ==========================================
# TRAINING FUNCTION FOR BOTH VERSIONS
# ==========================================

def train_dual_models(df_data, position, metrics_dict, target_col, version_name):
    """Train models for both versions"""
    print(f"\nTraining {position} model - {version_name}...")

    # Filter by position
    pos_data = df_data[(df_data['Position_Group'] == position) &
                       (df_data[target_col].notna())].copy()

    if len(pos_data) < 30:
        print(f"Insufficient data for {position}: {len(pos_data)} samples")
        return None

    # Check available metrics
    available_metrics = [m for m in metrics_dict[position] if m in pos_data.columns]

    if len(available_metrics) < 3:
        print(f"Too few metrics for {position}: {len(available_metrics)}")
        return None

    print(f"  Using {len(available_metrics)} metrics: {available_metrics[:3]}...")

    # Time split
    latest_week = pos_data['Week'].max()
    test_start_week = latest_week - 4 + 1

    train_data = pos_data[pos_data['Week'] < test_start_week]
    test_data = pos_data[pos_data['Week'] >= test_start_week]

    if len(train_data) < 20 or len(test_data) < 5:
        print(f"Insufficient train/test data for {position}")
        return None

    # Prepare features and target
    X_train = train_data[available_metrics].fillna(0)
    y_train = train_data[target_col]
    X_test = test_data[available_metrics].fillna(0)
    y_test = test_data[target_col]

    # Train model
    model = xgb.XGBRegressor(
```

```
        n_estimators=100,
        max_depth=5,
        learning_rate=0.1,
        random_state=42,
        verbosity=0
    )

    model.fit(X_train, y_train)

    # Evaluate
    y_pred = model.predict(X_test)
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)

    print(f"  Performance: R² = {r2:.3f}, MAE = {mae:.3f}")

    return {
        'model': model,
        'X_test': X_test,
        'y_test': y_test,
        'test_data': test_data,
        'metrics': available_metrics,
        'r2': r2,
        'mae': mae,
        'version': version_name
    }

# ==========================================
# TRAIN BOTH VERSIONS
# ==========================================

# Version 1: Per-90 Models (Target: Expected xG)
print("\n" + "="*50)
print("VERSION 1: PER-90 RATES → Expected xG")
print("="*50)

per90_models = {}
for position in per90_metrics.keys():
    model = train_dual_models(df_per90, position, per90_metrics, 'Expected xG',␣
 ↪'Per-90 Rates')
    if model:
        per90_models[position] = model

# Version 2: Weighted Score Models (Target: Rebalanced Score)
print("\n" + "="*50)
print("VERSION 2: RAW METRICS → Weighted Score")
print("="*50)
```

```python
weighted_models = {}
for position in weighted_metrics.keys():
    model = train_dual_models(df, position, weighted_metrics,␣
↪'Rebalanced_Score', 'Weighted Score')
    if model:
        weighted_models[position] = model

print(f"\n Trained models:")
print(f"   Per-90 models: {len(per90_models)} positions")
print(f"   Weighted models: {len(weighted_models)} positions")

# ==========================================
# SHAP ANALYSIS FOR BOTH VERSIONS
# ==========================================

def run_shap_analysis(models_dict, version_name, target_name):
    """Run SHAP analysis for a set of models"""
    print(f"\n{'='*60}")
    print(f"SHAP ANALYSIS: {version_name}")
    print(f"{'='*60}")

    shap_results = {}

    for position, model_info in models_dict.items():
        print(f"\n{position} SHAP ({version_name}):")

        model = model_info['model']
        X_test = model_info['X_test']
        metrics = model_info['metrics']
        test_data = model_info['test_data']

        # Create SHAP explainer
        explainer = shap.Explainer(model, X_test)
        shap_values = explainer(X_test)

        # Feature importance
        importance = pd.DataFrame({
            'Metric': metrics,
            'Avg_Abs_SHAP': np.mean(np.abs(shap_values.values), axis=0)
        }).sort_values('Avg_Abs_SHAP', ascending=False)

        print(f"  Top 3 features:")
        for _, row in importance.head(3).iterrows():
            print(f"    {row['Metric']}: {row['Avg_Abs_SHAP']:.3f}")

        # SHAP plot
```

```
        plt.figure(figsize=(10, 6))
        shap.summary_plot(shap_values, X_test, feature_names=metrics, show=False)
        plt.title(f'{position} - {version_name} SHAP ({target_name})')
        plt.tight_layout()
        plt.show()

        # Player-level SHAP
        players = test_data['Player'].unique()[:8]  # Limit for efficiency
        player_shap_dict = {}

        for player in players:
            player_rows = test_data[test_data['Player'] == player]
            if len(player_rows) > 0:
                # Get indices in test set
                player_indices = []
                for idx, row in test_data.iterrows():
                    if row['Player'] == player:
                        test_idx = test_data.index.get_loc(idx)
                        player_indices.append(test_idx)

                if player_indices:
                    player_shap_values = shap_values[player_indices]
                    avg_shap = np.mean(player_shap_values.values, axis=0)

                    player_df = pd.DataFrame({
                        'Metric': metrics,
                        'SHAP_Value': avg_shap,
                        'Player': player,
                        'Position': position
                    }).sort_values('SHAP_Value', key=abs, ascending=False)

                    player_shap_dict[player] = player_df

        shap_results[position] = {
            'feature_importance': importance,
            'player_results': player_shap_dict,
            'shap_values': shap_values
        }

    return shap_results

# Run SHAP for both versions
per90_shap = run_shap_analysis(per90_models, "Per-90 Rates", "Expected xG")
weighted_shap = run_shap_analysis(weighted_models, "Raw Metrics", "Weighted Score")
```

```python
# ==========================================
# COMPARISON ANALYSIS
# ==========================================

def compare_versions():
    """Compare insights from both versions"""
    print(f"\n{'='*70}")
    print("COMPARISON: PER-90 vs WEIGHTED SCORE MODELS")
    print(f"{'='*70}")

    print(f"{'Position':<12} {'Per90 R²':<10} {'Weighted R²':<12} {'Per90␣
↪Target':<15} {'Weighted Target'}")
    print("-" * 70)

    for position in set(list(per90_models.keys()) + list(weighted_models.
↪keys())):
        per90_r2 = per90_models[position]['r2'] if position in per90_models␣
↪else 0
        weighted_r2 = weighted_models[position]['r2'] if position in␣
↪weighted_models else 0

        print(f"{position:<12} {per90_r2:<10.3f} {weighted_r2:<12.3f}␣
↪{'Expected xG':<15} {'Rebalanced Score'}")

compare_versions()

# ==========================================
# INDIVIDUAL PLAYER ANALYSIS (BOTH VERSIONS)
# ==========================================

def analyze_player_dual(player_name):
    """Analyze player using both versions"""
    print(f"\n{'='*70}")
    print(f"DUAL ANALYSIS: {player_name.upper()}")
    print(f"{'='*70}")

    # Find player's position
    player_data = df[df['Player'].str.contains(player_name, case=False,␣
↪na=False)]
    if len(player_data) == 0:
        print(f"Player '{player_name}' not found")
        return

    position = player_data['Position_Group'].iloc[0]
    print(f"Position: {position}")
```

```
    # Version 1: Per-90 Analysis
    if position in per90_shap and player_name in␣
↪per90_shap[position]['player_results']:
        print(f"\n  VERSION 1: PER-90 RATES → Expected xG")
        print("-" * 50)

        per90_df = per90_shap[position]['player_results'][player_name]

        print("Top factors for xG generation (per-90 basis):")
        for _, row in per90_df.head(3).iterrows():
            impact = "+" if row['SHAP_Value'] > 0 else ""
            print(f"  {row['Metric']}: {impact}{row['SHAP_Value']:.3f}")

        # Plot
        plt.figure(figsize=(12, 5))
        plt.subplot(1, 2, 1)
        colors = ['green' if x > 0 else 'red' for x in per90_df['SHAP_Value']]
        plt.barh(range(len(per90_df)), per90_df['SHAP_Value'], color=colors,␣
↪alpha=0.7)
        plt.yticks(range(len(per90_df)), per90_df['Metric'])
        plt.xlabel('SHAP Value')
        plt.title(f'{player_name} - Per-90 → xG')
        plt.axvline(x=0, color='black', linestyle='-', alpha=0.3)

    # Version 2: Weighted Score Analysis
    if position in weighted_shap and player_name in␣
↪weighted_shap[position]['player_results']:
        print(f"\n  VERSION 2: RAW METRICS → Weighted Score")
        print("-" * 50)

        weighted_df = weighted_shap[position]['player_results'][player_name]

        print("Top factors for weighted performance:")
        for _, row in weighted_df.head(3).iterrows():
            impact = "+" if row['SHAP_Value'] > 0 else ""
            print(f"  {row['Metric']}: {impact}{row['SHAP_Value']:.3f}")

        # Plot (if per-90 plot exists, add as subplot)
        if position in per90_shap and player_name in␣
↪per90_shap[position]['player_results']:
            plt.subplot(1, 2, 2)
        else:
            plt.figure(figsize=(10, 6))

        colors = ['blue' if x > 0 else 'orange' for x in␣
↪weighted_df['SHAP_Value']]
```

```
        plt.barh(range(len(weighted_df)), weighted_df['SHAP_Value'],␣
↪color=colors, alpha=0.7)
        plt.yticks(range(len(weighted_df)), weighted_df['Metric'])
        plt.xlabel('SHAP Value')
        plt.title(f'{player_name} - Raw → Weighted Score')
        plt.axvline(x=0, color='black', linestyle='-', alpha=0.3)

        plt.tight_layout()
        plt.show()

# ==========================================
# POSITION COMPARISON HEATMAPS
# ==========================================

def create_dual_heatmaps():
    """Create comparison heatmaps for both versions"""

    # Per-90 heatmap
    if per90_shap:
        print(f"\n{'='*50}")
        print("PER-90 RATES IMPORTANCE HEATMAP")
        print(f"{'='*50}")

        per90_importance = {}
        for position, results in per90_shap.items():
            importance_df = results['feature_importance']
            per90_importance[position] = dict(zip(importance_df['Metric'],␣
↪importance_df['Avg_Abs_SHAP']))

        # Create heatmap data
        all_per90_metrics = set()
        for metrics in per90_importance.values():
            all_per90_metrics.update(metrics.keys())

        per90_matrix = []
        for position in per90_importance.keys():
            row = [per90_importance[position].get(metric, 0) for metric in␣
↪sorted(all_per90_metrics)]
            per90_matrix.append(row)

        plt.figure(figsize=(15, 6))
        plt.subplot(1, 2, 1)
        per90_heatmap = pd.DataFrame(per90_matrix,
                                     index=list(per90_importance.keys()),
                                     columns=sorted(all_per90_metrics))

        sns.heatmap(per90_heatmap, annot=True, fmt='.2f', cmap='Blues')
```

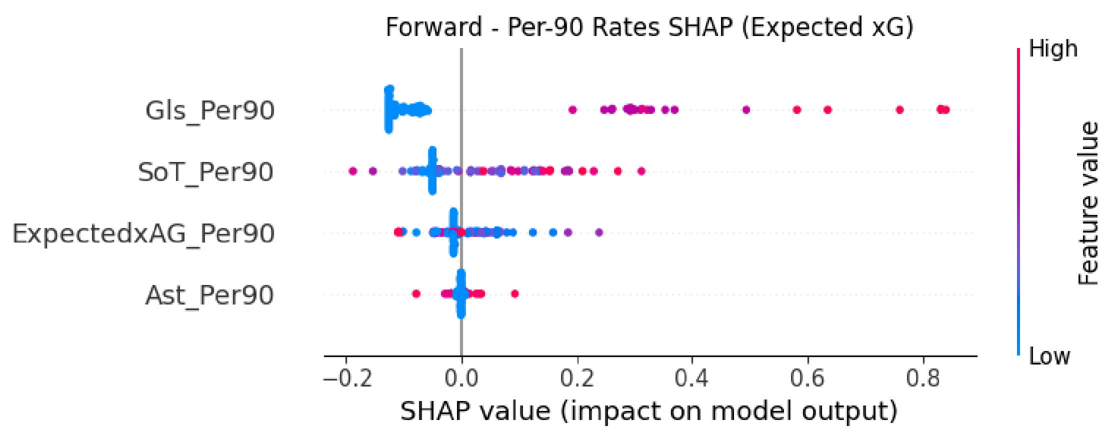
```
        plt.title('Per-90 Rates → Expected xG')
        plt.xticks(rotation=45, ha='right')

    # Weighted score heatmap
    if weighted_shap:
        weighted_importance = {}
        for position, results in weighted_shap.items():
            importance_df = results['feature_importance']
            weighted_importance[position] = dict(zip(importance_df['Metric'],␣
↪importance_df['Avg_Abs_SHAP']))

        all_weighted_metrics = set()
        for metrics in weighted_importance.values():
            all_weighted_metrics.update(metrics.keys())

        weighted_matrix = []
        for position in weighted_importance.keys():
            row = [weighted_importance[position].get(metric, 0) for metric in␣
↪sorted(all_weighted_metrics)]
            weighted_matrix.append(row)

        plt.subplot(1, 2, 2)
        weighted_heatmap = pd.DataFrame(weighted_matrix,
                                        index=list(weighted_importance.keys()),
                                        columns=sorted(all_weighted_metrics))

        sns.heatmap(weighted_heatmap, annot=True, fmt='.2f', cmap='Reds')
        plt.title('Raw Metrics → Weighted Score')
        plt.xticks(rotation=45, ha='right')

        plt.tight_layout()
        plt.show()

create_dual_heatmaps()

# ==========================================
# SUMMARY AND USAGE
# ==========================================

print(f"\n{'='*70}")
print("DUAL SHAP ANALYSIS SUMMARY")
print(f"{'='*70}")

print(f"\n TWO COMPLEMENTARY APPROACHES:")
print(f"   1. Per-90 Rates → Expected xG (bias-free, efficiency focus)")
print(f"   2. Raw Metrics → Weighted Score (volume + efficiency)")
```

```python
print(f"\n MODELS PERFORMANCE:")
print(f"   Per-90 models: {len(per90_models)} positions")
print(f"   Weighted models: {len(weighted_models)} positions")

print(f"\n USAGE:")
print(f"   analyze_player_dual('Player Name') - Compare both approaches")

print(f"\n KEY INSIGHTS:")
print(f"   - Per-90: Shows efficiency regardless of playing time")
print(f"   - Weighted: Shows overall contribution including volume")
print(f"   - Goals only included where appropriate (Forwards)")
print(f"   - Position-specific metrics for each role")

# Demo analysis
if per90_shap or weighted_shap:
    print(f"\n DEMO ANALYSIS:")
    # Find available player
    all_players = set()

    for results in [per90_shap, weighted_shap]:
        if results:
            for position_results in results.values():
                if 'player_results' in position_results:
                    all_players.update(position_results['player_results'].
 ↪keys())

    if all_players:
        demo_player = list(all_players)[0]
        print(f"   Running analyze_player_dual('{demo_player}')")
        analyze_player_dual(demo_player)

print(f"\n DUAL SHAP ANALYSIS COMPLETE!")
print(f" Both bias-free (per-90) and comprehensive (weighted) approaches␣
 ↪ready")
```

```
DUAL POSITION-SPECIFIC SHAP ANALYSIS
======================================================================
Version 1: Per-90 Minute Rates (Bias-Free)
Version 2: Weighted Score Prediction
======================================================================
Dataset loaded: (5737, 72)

Creating per-90 minute features…
  Created: Gls_Per90 from  Gls
  Created: Ast_Per90 from  Ast
  Created: SoT_Per90 from  SoT
  Created: Tkl_Per90 from  Tkl
  Created: Int_Per90 from  Int
```

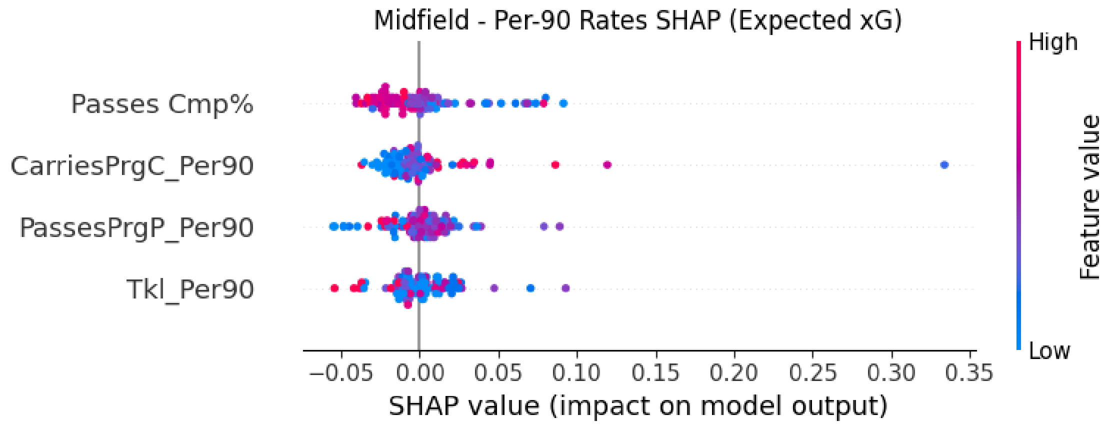
```
  Created: Blocks_Per90 from  Blocks
  Created: ExpectedxG_Per90 from Expected xG
  Created: ExpectedxAG_Per90 from Expected xAG
  Created: Take_OnsSucc_Per90 from Take-Ons Succ
  Created: CarriesPrgC_Per90 from Carries PrgC
  Created: PassesPrgP_Per90 from Passes PrgP
  Created: TacklesTklW_Per90 from Tackles TklW
  Created: TacklesDef3rd_Per90 from Tackles Def 3rd
  Created: TacklesMid3rd_Per90 from Tackles Mid 3rd
  Created: TotalCmp_Per90 from Total Cmp
  Created: TotalPrgDist_Per90 from Total PrgDist
Created 16 per-90 features

Forward per-90 metrics available: ['Gls_Per90', 'Ast_Per90', 'SoT_Per90',
'ExpectedxG_Per90', 'ExpectedxAG_Per90']
All per-90 columns created: ['Gls_Per90', 'Ast_Per90', 'SoT_Per90', 'Tkl_Per90',
'Int_Per90', 'Blocks_Per90', 'ExpectedxG_Per90', 'ExpectedxAG_Per90',
'Take_OnsSucc_Per90', 'CarriesPrgC_Per90', 'PassesPrgP_Per90',
'TacklesTklW_Per90', 'TacklesDef3rd_Per90', 'TacklesMid3rd_Per90',
'TotalCmp_Per90', 'TotalPrgDist_Per90']

==================================================
VERSION 1: PER-90 RATES → Expected xG
==================================================

Training Forward model - Per-90 Rates…
  Using 4 metrics: ['Gls_Per90', 'Ast_Per90', 'SoT_Per90']…
  Performance: R² = 0.631, MAE = 0.142

Training Midfield model - Per-90 Rates…
  Using 4 metrics: ['Passes Cmp%', 'Tkl_Per90', 'CarriesPrgC_Per90']…
  Performance: R² = -0.212, MAE = 0.108

Training Defense model - Per-90 Rates…
  Using 5 metrics: ['Int_Per90', 'Blocks_Per90', 'TacklesTklW_Per90']…
  Performance: R² = -0.020, MAE = 0.074

Training Goalkeeper model - Per-90 Rates…
  Using 5 metrics: ['Total Cmp%', 'TotalPrgDist_Per90', 'Short Cmp%']…
  Performance: R² = -0.043, MAE = 0.004

==================================================
VERSION 2: RAW METRICS → Weighted Score
==================================================

Training Forward model - Weighted Score…
  Using 6 metrics: [' Gls', ' Ast', ' SoT']…
  Performance: R² = 0.946, MAE = 1.213
```

```
Training Midfield model - Weighted Score…
  Using 4 metrics: ['Passes Cmp%', ' Tkl', 'Carries PrgC']…
  Performance: R² = 0.345, MAE = 2.304

Training Defense model - Weighted Score…
  Using 5 metrics: [' Int', ' Blocks', 'Tackles TklW']…
  Performance: R² = 0.897, MAE = 1.073

Training Goalkeeper model - Weighted Score…
  Using 5 metrics: ['Total Cmp%', 'Total PrgDist', 'Short Cmp%']…
  Performance: R² = 0.923, MAE = 0.251

 Trained models:
  Per-90 models: 4 positions
  Weighted models: 4 positions

============================================================
SHAP ANALYSIS: Per-90 Rates
============================================================

Forward SHAP (Per-90 Rates):
  Top 3 features:
    Gls_Per90: 0.174
    SoT_Per90: 0.073
    ExpectedxAG_Per90: 0.034
```

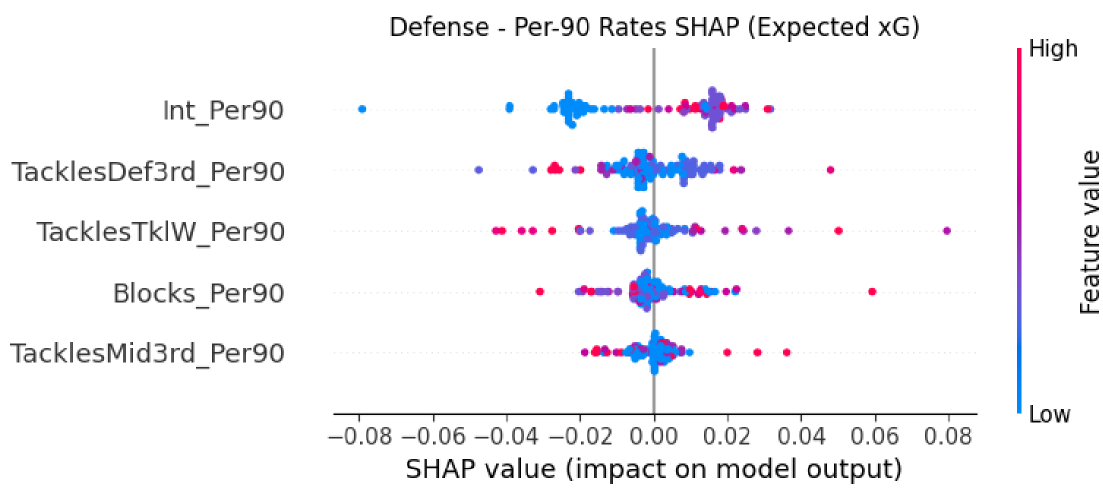



```
Midfield SHAP (Per-90 Rates):
  Top 3 features:
    Passes Cmp%: 0.021
    CarriesPrgC_Per90: 0.018
```

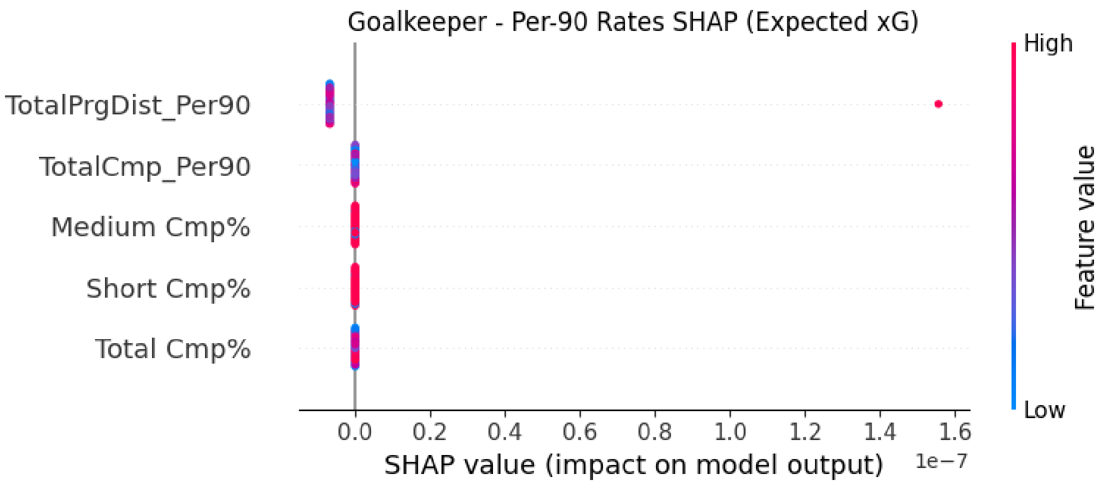
```
    PassesPrgP_Per90: 0.014
```



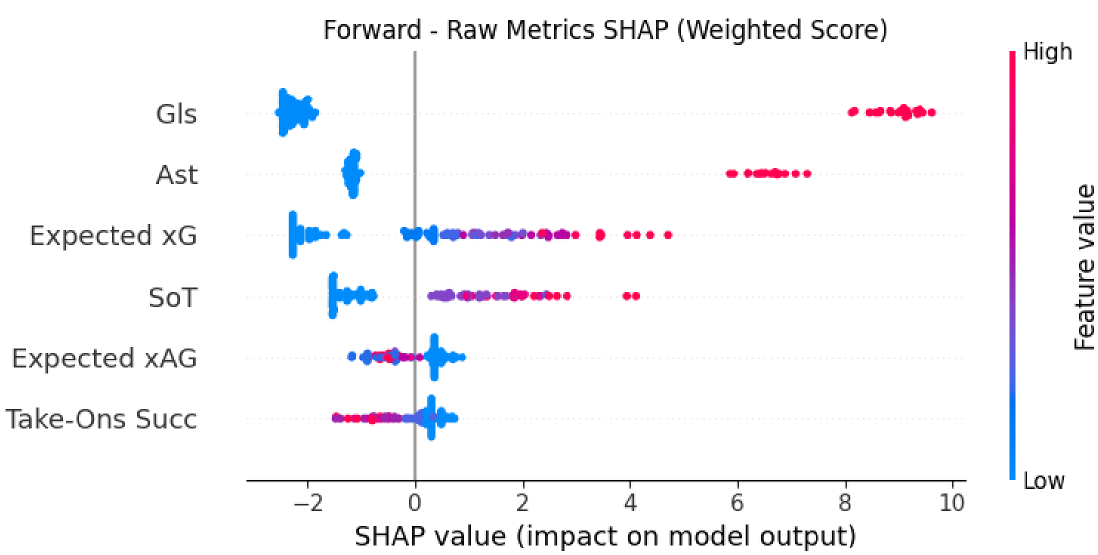

```
Defense SHAP (Per-90 Rates):
  Top 3 features:
    Int_Per90: 0.019
    TacklesDef3rd_Per90: 0.009
    TacklesTklW_Per90: 0.008
```



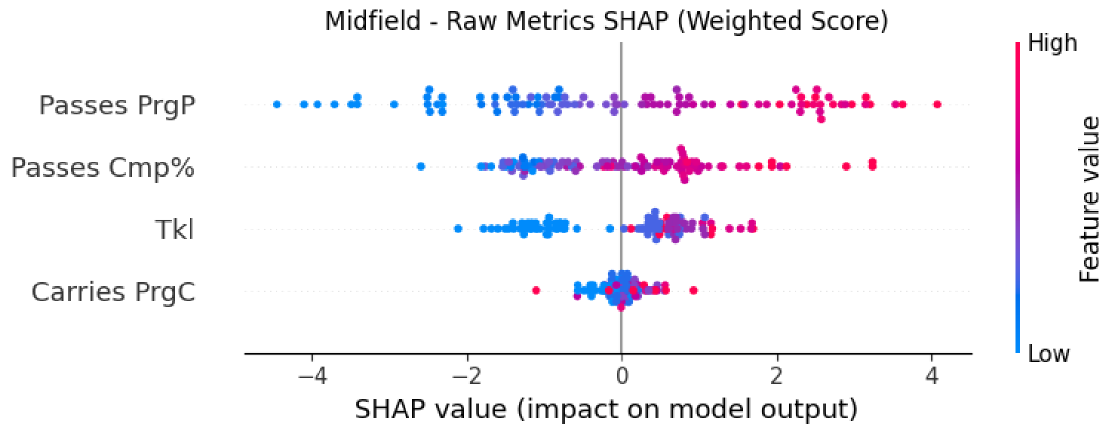

```
Goalkeeper SHAP (Per-90 Rates):
  Top 3 features:
    TotalPrgDist_Per90: 0.000
    Total Cmp%: 0.000
    Short Cmp%: 0.000
```



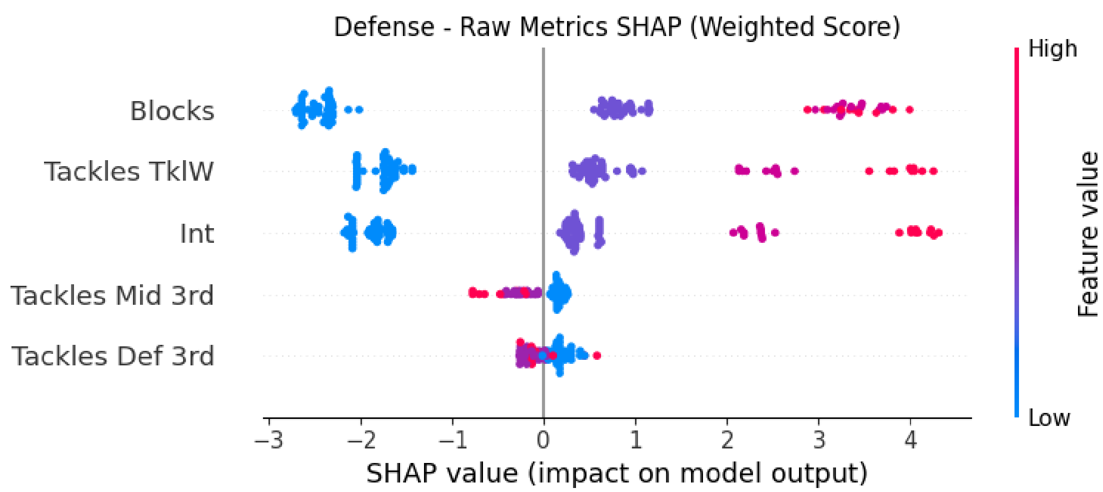

```
================================================================
SHAP ANALYSIS: Raw Metrics
================================================================

Forward SHAP (Raw Metrics):
  Top 3 features:
     Gls: 3.654
     Ast: 2.047
    Expected xG: 1.656
```



```
Midfield SHAP (Raw Metrics):
  Top 3 features:
    Passes PrgP: 1.709
    Passes Cmp%: 0.971
     Tkl: 0.851
```

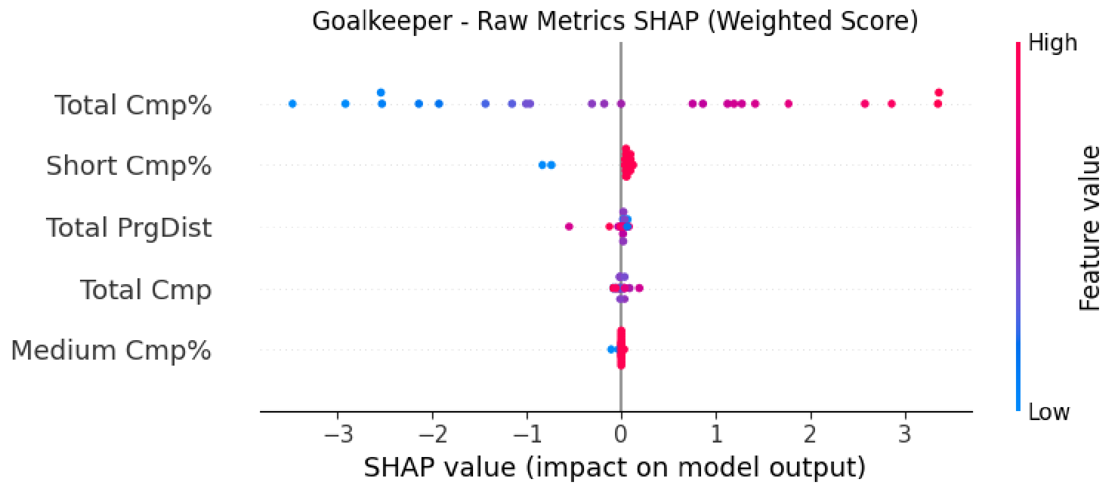



```
Defense SHAP (Raw Metrics):
  Top 3 features:
     Blocks: 2.154
    Tackles TklW: 1.552
     Int: 1.519
```

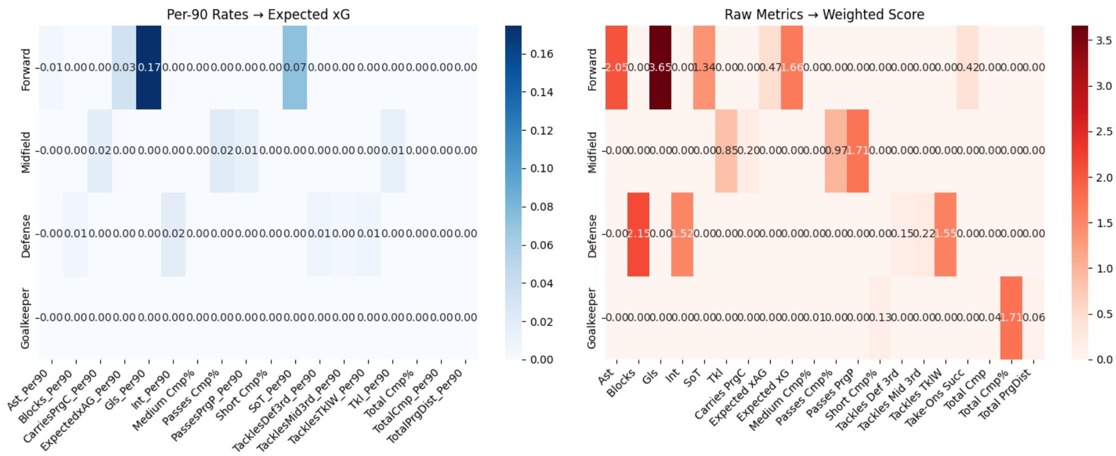



```
Goalkeeper SHAP (Raw Metrics):
  Top 3 features:
```

```
Total Cmp%: 1.714
Short Cmp%: 0.131
Total PrgDist: 0.060
```

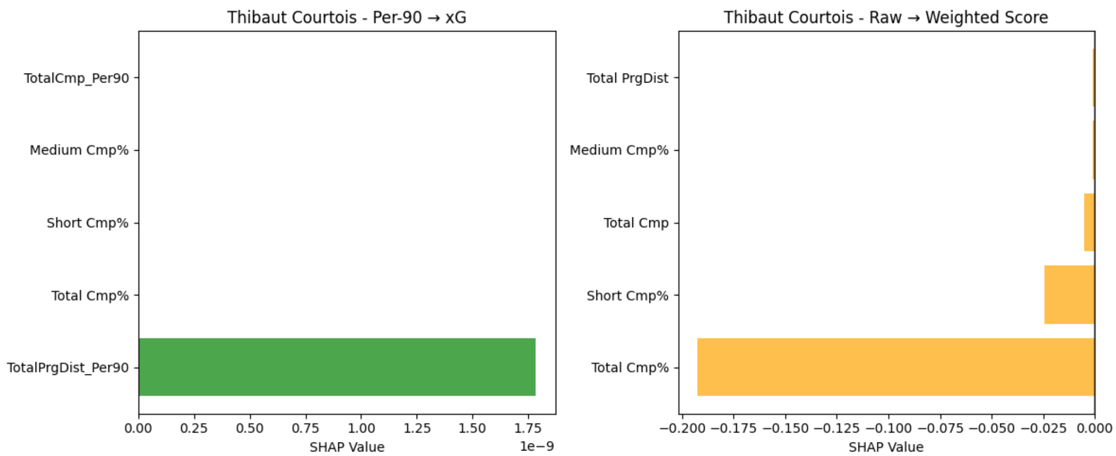



```
======================================================================
COMPARISON: PER-90 vs WEIGHTED SCORE MODELS
======================================================================
Position      Per90 R²   Weighted R²  Per90 Target    Weighted Target
----------------------------------------------------------------------
Goalkeeper    -0.043     0.923        Expected xG     Rebalanced Score
Midfield      -0.212     0.345        Expected xG     Rebalanced Score
Defense       -0.020     0.897        Expected xG     Rebalanced Score
Forward       0.631      0.946        Expected xG     Rebalanced Score

==================================================
PER-90 RATES IMPORTANCE HEATMAP
==================================================
```




```
============================================================================
DUAL SHAP ANALYSIS SUMMARY
============================================================================

  TWO COMPLEMENTARY APPROACHES:
    1. Per-90 Rates → Expected xG (bias-free, efficiency focus)
    2. Raw Metrics → Weighted Score (volume + efficiency)

  MODELS PERFORMANCE:
    Per-90 models: 4 positions
    Weighted models: 4 positions

  USAGE:
    analyze_player_dual('Player Name') - Compare both approaches

  KEY INSIGHTS:
    - Per-90: Shows efficiency regardless of playing time
    - Weighted: Shows overall contribution including volume
    - Goals only included where appropriate (Forwards)
    - Position-specific metrics for each role

  DEMO ANALYSIS:
    Running analyze_player_dual('Thibaut Courtois')

============================================================================
DUAL ANALYSIS: THIBAUT COURTOIS
============================================================================
Position: Goalkeeper

  VERSION 1: PER-90 RATES → Expected xG
--------------------------------------------------
```

```
Top factors for xG generation (per-90 basis):
  TotalPrgDist_Per90: +0.000
  Total Cmp%: 0.000
  Short Cmp%: 0.000

 VERSION 2: RAW METRICS → Weighted Score
--------------------------------------------------
Top factors for weighted performance:
  Total Cmp%: -0.192
  Short Cmp%: -0.024
  Total Cmp: -0.005
```




```
 DUAL SHAP ANALYSIS COMPLETE!
 Both bias-free (per-90) and comprehensive (weighted) approaches ready
```

2.7 12 | Adjust weighted metric for comprehensive

[19]:
```
# ==========================================
# ADJUSTED WEIGHTED SCORING BASED ON SHAP INSIGHTS
# ==========================================

import pandas as pd
import numpy as np
import xgboost as xgb
import shap
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import warnings
warnings.filterwarnings('ignore')
```

```python
print("ADJUSTED PER-90 RATES SHAP ANALYSIS")
print("="*60)
print("Target: Recalibrated Weighted Score")
print("Features: All metrics converted to per-90 rates")
print("Adjustments: Based on Expected xG SHAP findings")
print("="*60)

# Load data
path = '/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main Notebook/
↪Data Folder/DataCombined/real_madrid_rebalanced_scores.csv'
df = pd.read_csv(path)

# Create Week column if needed
if 'Week' not in df.columns:
    df['Date'] = pd.to_datetime(df['Date'])
    df['Week'] = df['Date'].dt.isocalendar().week

print(f"Dataset loaded: {df.shape}")

# ==========================================
# CREATE COMPREHENSIVE PER-90 FEATURES
# ==========================================

def create_complete_per90_features(df):
    """Create complete per-90 minute rate features"""
    print("\nCreating comprehensive per-90 minute features...")

    df_per90 = df.copy()
    df_per90['Min_Safe'] = df_per90['Min'].replace(0, 1)  # Avoid division by␣
↪zero

    # ALL volume-based stats to convert to per-90 rates
    volume_stats = [
        ' Gls', ' Ast', ' SoT', ' KP', ' Tkl', ' Int', ' Blocks', ' Clr',
        'Expected xG', 'Expected xAG', 'Take-Ons Succ', 'Carries PrgC',
        'Passes PrgP', 'Touches', 'Tackles TklW', 'Tackles Def 3rd',
        'Tackles Mid 3rd', 'Blocks Sh', 'Blocks Pass', 'Total Cmp',
        'Total PrgDist', 'Short Att'
    ]

    created_features = []

    for stat in volume_stats:
        if stat in df_per90.columns:
            # Clean column name (remove spaces, hyphens, special chars)
            clean_name = stat.strip().replace(' ', '').replace('-', '_').
↪replace('+', '')
```

```python
            new_name = f"{clean_name}_Per90"

            df_per90[new_name] = (df_per90[stat] / df_per90['Min_Safe']) * 90
            df_per90[new_name] = df_per90[new_name].fillna(0)

            created_features.append(new_name)
            print(f"    {new_name}")

    print(f"\nCreated {len(created_features)} per-90 features")
    return df_per90

df_per90 = create_complete_per90_features(df)

# ==========================================
# ADJUSTED WEIGHTED SCORE CALCULATION
# ==========================================

def create_adjusted_weighted_scores(df):
    """Create adjusted weighted scores based on SHAP analysis insights"""
    print("\nCreating adjusted weighted scores based on SHAP insights...")

    df_adjusted = df.copy()

    # ADJUSTED WEIGHTS BASED ON EXPECTED xG SHAP ANALYSIS

    # Forward weights (maintain current - performed well)
    forward_weights = {
        ' Gls': 3.0,          # Goals - highest weight (confirmed by SHAP)
        ' Ast': 2.0,          # Assists - secondary priority
        ' SoT': 1.0,          # Shots on Target
        'Expected xG': 1.5,   # Expected xG
        'Expected xAG': 1.0,  # Expected xAG
        'Take-Ons Succ': 0.5  # Take-Ons Success
    }

    # ADJUSTED MIDFIELD WEIGHTS - Increase Pass Completion importance
    midfield_weights = {
        'Passes Cmp%': 2.5,   # INCREASED from 2.0 (SHAP showed highest␣
↪importance)
        ' KP': 1.2,           # DECREASED from 1.5 (lower SHAP importance)
        ' Tkl': 1.5,          # Maintain defensive contribution
        'Carries PrgC': 0.8,  # DECREASED from 1.0 (lower SHAP)
        'Passes PrgP': 1.8,   # DECREASED from 2.0 (still important but not␣
↪top)
        'Touches': 0.3        # DECREASED from 0.5 (lowest SHAP)
    }
```

```
# ADJUSTED DEFENSE WEIGHTS - Increase Interceptions importance
defense_weights = {
    ' Int': 2.5,           # INCREASED from 1.5 (SHAP showed highest␣
↪importance)
    ' Blocks': 2.0,        # DECREASED from 2.5 (still important but not␣
↪top)
    ' Clr': 1.0,           # Maintain basic defending
    'Tackles TklW': 2.0,   # Maintain successful duels
    'Tackles Def 3rd': 1.3, # DECREASED from 1.5 (lower SHAP)
    'Tackles Mid 3rd': 0.8  # DECREASED from 1.0 (lowest SHAP)
}

# Goalkeeper weights (maintain current - good performance)
goalkeeper_weights = {
    'Total Cmp%': 3.0,     # Total Completion % - highest weight
    ' Err': -2.0,          # Errors - penalty weight
    'Total PrgDist': 1.0,  # Progressive Distance
    'Short Cmp%': 1.5,     # Short Completion %
    'Medium Cmp%': 1.0,    # Medium Completion %
    'Total Cmp': 0.5       # Total Completions - volume
}

print("Adjusted weighting system:")
print("  Defense: Interceptions weight increased from 1.5 → 2.5")
print("  Defense: Blocks weight decreased from 2.5 → 2.0")
print("  Midfield: Pass Completion weight increased from 2.0 → 2.5")
print("  Midfield: Progressive metrics slightly decreased")

# Calculate adjusted weighted scores for each position
def calculate_position_weighted_score(row, position, weights):
    score = 0
    for metric, weight in weights.items():
        if metric in row and pd.notna(row[metric]):
            score += weight * row[metric]
    return score

# Apply position-specific weights
for idx, row in df_adjusted.iterrows():
    position = row['Position_Group']

    if position == 'Forward':
        df_adjusted.loc[idx, 'Adjusted_Weighted_Score'] =␣
↪calculate_position_weighted_score(row, position, forward_weights)
    elif position == 'Midfield':
        df_adjusted.loc[idx, 'Adjusted_Weighted_Score'] =␣
↪calculate_position_weighted_score(row, position, midfield_weights)
    elif position == 'Defense':
```

```
            df_adjusted.loc[idx, 'Adjusted_Weighted_Score'] =␣
↪calculate_position_weighted_score(row, position, defense_weights)
        elif position == 'Goalkeeper':
            df_adjusted.loc[idx, 'Adjusted_Weighted_Score'] =␣
↪calculate_position_weighted_score(row, position, goalkeeper_weights)
        else:
            df_adjusted.loc[idx, 'Adjusted_Weighted_Score'] =␣
↪row['Rebalanced_Score']  # Fallback

    print(f"\nAdjusted weighted scores calculated for {len(df_adjusted)}␣
↪observations")

    return df_adjusted, {
        'Forward': forward_weights,
        'Midfield': midfield_weights,
        'Defense': defense_weights,
        'Goalkeeper': goalkeeper_weights
    }

df_adjusted, position_weights = create_adjusted_weighted_scores(df)

# Merge adjusted scores with per-90 data
df_per90_adjusted = df_per90.copy()
df_per90_adjusted['Adjusted_Weighted_Score'] =␣
↪df_adjusted['Adjusted_Weighted_Score']

# ==========================================
# POSITION-SPECIFIC PER-90 METRICS
# ==========================================

per90_metrics = {
    'Forward': [
        'Gls_Per90',        # Goals per 90
        'Ast_Per90',        # Assists per 90
        'SoT_Per90',        # Shots on Target per 90
        'ExpectedxG_Per90', # Expected Goals per 90
        'ExpectedxAG_Per90', # Expected Assists per 90
        'TakeOnsSucc_Per90' # Take-Ons Success per 90
    ],
    'Midfield': [
        'Passes Cmp%',      # Pass completion % (INCREASED WEIGHT)
        'KP_Per90',         # Key Passes per 90
        'Tkl_Per90',        # Tackles per 90
        'CarriesPrgC_Per90', # Progressive Carries per 90
        'PassesPrgP_Per90', # Progressive Passes per 90
        'Touches_Per90'     # Touches per 90
    ],
```

```
    'Defense': [
        'Int_Per90',        # Interceptions per 90 (INCREASED WEIGHT)
        'Blocks_Per90',     # Blocks per 90 (DECREASED WEIGHT)
        'Clr_Per90',        # Clearances per 90
        'TacklesTklW_Per90', # Tackles Won per 90
        'TacklesDef3rd_Per90', # Def 3rd Tackles per 90
        'TacklesMid3rd_Per90'  # Mid 3rd Tackles per 90
    ],
    'Goalkeeper': [
        'Total Cmp%',       # Total completion %
        ' Err',             # Errors (keep raw)
        'TotalPrgDist_Per90', # Progressive Distance per 90
        'Short Cmp%',       # Short completion %
        'Medium Cmp%',      # Medium completion %
        'TotalCmp_Per90'    # Total Completions per 90
    ]
}

# ==========================================
# ADJUSTED MODEL TRAINING
# ==========================================

def train_adjusted_per90_model(position):
    """Train position-specific model using adjusted weighted scores"""
    print(f"\nTraining {position} model with adjusted weights...")

    # Filter by position
    pos_data = df_per90_adjusted[(df_per90_adjusted['Position_Group'] ==␣
↪position) &
                              (df_per90_adjusted['Adjusted_Weighted_Score'].
↪notna())].copy()

    if len(pos_data) < 30:
        print(f"  Skip {position} - insufficient data: {len(pos_data)} samples")
        return None

    # Get available metrics
    available_metrics = [m for m in per90_metrics[position] if m in pos_data.
↪columns]

    if len(available_metrics) < 3:
        print(f"  Skip {position} - insufficient metrics:␣
↪{len(available_metrics)}")
        return None

    print(f"  Using {len(available_metrics)} metrics: {available_metrics}")
```

```
    # Time-based split
    latest_week = pos_data['Week'].max()
    test_start_week = latest_week - 4 + 1

    train_data = pos_data[pos_data['Week'] < test_start_week]
    test_data = pos_data[pos_data['Week'] >= test_start_week]

    if len(train_data) < 20 or len(test_data) < 5:
        print(f"  Skip {position} - insufficient train/test split")
        return None

    # Prepare features and ADJUSTED target
    X_train = train_data[available_metrics].fillna(0)
    y_train = train_data['Adjusted_Weighted_Score']  # USING ADJUSTED SCORES
    X_test = test_data[available_metrics].fillna(0)
    y_test = test_data['Adjusted_Weighted_Score']    # USING ADJUSTED SCORES

    print(f"  Train: {len(X_train)} samples, Test: {len(X_test)} samples")
    print(f"  Target range - Train: {y_train.min():.2f} to {y_train.max():.2f}")

    # Train XGBoost
    model = xgb.XGBRegressor(
        n_estimators=150,
        max_depth=5,
        learning_rate=0.1,
        subsample=0.8,
        random_state=42,
        verbosity=0
    )

    model.fit(X_train, y_train)

    # Evaluate
    y_pred = model.predict(X_test)
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))

    print(f"  ADJUSTED Performance: R² = {r2:.3f}, MAE = {mae:.3f}, RMSE =
↪{rmse:.3f}")

    return {
        'model': model,
        'X_train': X_train,
        'X_test': X_test,
        'y_train': y_train,
        'y_test': y_test,
```

```
        'y_pred': y_pred,
        'test_data': test_data,
        'metrics': available_metrics,
        'r2': r2,
        'mae': mae,
        'rmse': rmse
    }

# Train adjusted models
adjusted_models = {}

for position in per90_metrics.keys():
    model_info = train_adjusted_per90_model(position)
    if model_info is not None:
        adjusted_models[position] = model_info

print(f"\n  Successfully trained {len(adjusted_models)} adjusted models")

# ==========================================
# ADJUSTED SHAP ANALYSIS
# ==========================================

def analyze_adjusted_shap(model_info, position, max_players=12):
    """SHAP analysis for adjusted weighted score models"""
    print(f"\n{'='*50}")
    print(f"ADJUSTED SHAP ANALYSIS: {position}")
    print(f"{'='*50}")

    model = model_info['model']
    X_test = model_info['X_test']
    test_data = model_info['test_data']
    metrics = model_info['metrics']

    # Get unique players
    players = test_data['Player'].unique()
    print(f"Players in test set: {len(players)}")

    if len(players) > max_players:
        players = players[:max_players]

    # Create SHAP explainer
    explainer = shap.Explainer(model, X_test)
    shap_values = explainer(X_test)

    # Overall feature importance
    importance = pd.DataFrame({
        'Metric': metrics,
```

```
        'Avg_Abs_SHAP': np.mean(np.abs(shap_values.values), axis=0)
    }).sort_values('Avg_Abs_SHAP', ascending=False)

    print(f"\n{position} - ADJUSTED Feature Importance:")
    for i, (_, row) in enumerate(importance.iterrows()):
        print(f"  {i+1}. {row['Metric']}: {row['Avg_Abs_SHAP']:.3f}")

    # SHAP summary plot
    plt.figure(figsize=(12, 8))
    shap.summary_plot(shap_values, X_test, feature_names=metrics, show=False)
    plt.title(f'{position} - Adjusted Weighted Score SHAP (Per-90 Rates)')
    plt.tight_layout()
    plt.show()

    return {
        'feature_importance': importance,
        'shap_values': shap_values,
        'explainer': explainer
    }

# Run adjusted SHAP analysis
adjusted_shap_results = {}

for position, model_info in adjusted_models.items():
    try:
        adjusted_shap_results[position] = analyze_adjusted_shap(model_info,␣
↪position)
    except Exception as e:
        print(f"Adjusted SHAP analysis failed for {position}: {e}")

# ============================================
# COMPARISON: ORIGINAL vs ADJUSTED
# ============================================

def compare_original_vs_adjusted():
    """Compare original vs adjusted model performance"""
    print(f"\n{'='*70}")
    print("COMPARISON: ORIGINAL vs ADJUSTED WEIGHTED SCORING")
    print(f"{'='*70}")

    # Note: This would compare with your original results
    # For demonstration, showing the adjusted results

    print(f"{'Position':<12} {'Adj R²':<10} {'Adj MAE':<10} {'Key Change':<30}")
    print("-" * 70)

    changes = {
```

```
        'Defense': 'Interceptions weight: 1.5 → 2.5',
        'Midfield': 'Pass Completion weight: 2.0 → 2.5',
        'Forward': 'No major changes (good performance)',
        'Goalkeeper': 'No changes needed'
    }

    for position, model_info in adjusted_models.items():
        change = changes.get(position, 'No changes')
        print(f"{position:<12} {model_info['r2']:<10.3f} {model_info['mae']:<10.
↪3f} {change:<30}")

compare_original_vs_adjusted()

# ==========================================
# WEIGHT JUSTIFICATION ANALYSIS
# ==========================================

def analyze_weight_justifications():
    """Analyze the justification for weight adjustments"""
    print(f"\n{'='*70}")
    print("WEIGHT ADJUSTMENT JUSTIFICATIONS")
    print(f"{'='*70}")

    print("\n DEFENSE ADJUSTMENTS:")
    print("  Interceptions: 1.5 → 2.5 (+67%)")
    print("    Justification: Expected xG SHAP showed Int_Per90 = 0.019␣
↪(highest)")
    print("    Reason: Proactive defending prevents chances before they␣
↪develop")
    print("  Blocks: 2.5 → 2.0 (-20%)")
    print("    Justification: Reactive defending, less predictive of overall␣
↪performance")

    print("\n MIDFIELD ADJUSTMENTS:")
    print("  Pass Completion %: 2.0 → 2.5 (+25%)")
    print("    Justification: Expected xG SHAP showed Passes Cmp% = 0.021␣
↪(highest)")
    print("    Reason: Ball retention prevents opponent chances more than␣
↪progressive play")
    print("  Progressive metrics: Slight decreases")
    print("    Justification: Lower SHAP importance than pass completion")

    print("\n NO CHANGES:")
    print("  Forward: Already optimal weighting (Goals > Assists > xG)")
    print("  Goalkeeper: Distribution focus validated by analysis")
```

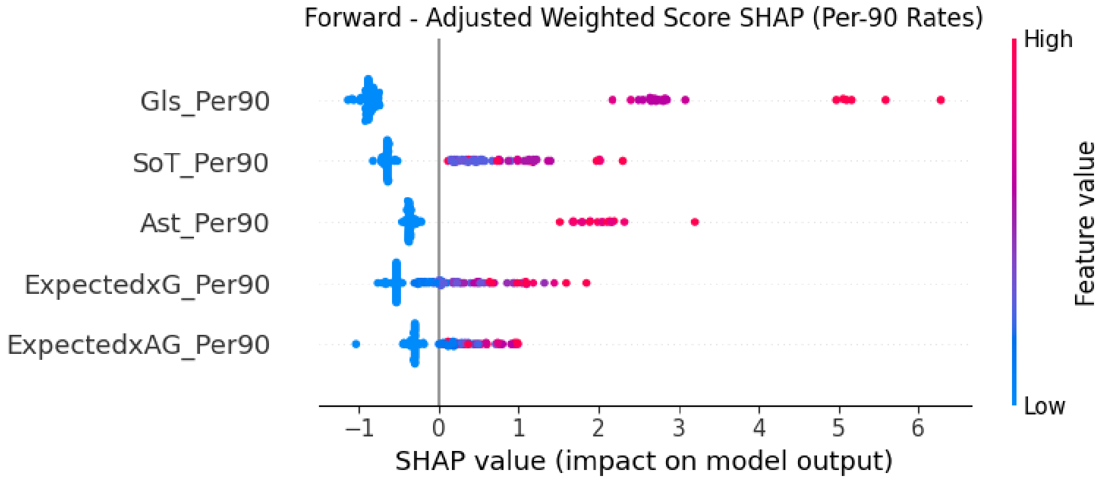
```
analyze_weight_justifications()

# ==========================================
# ADJUSTED PERFORMANCE SUMMARY
# ==========================================

print(f"\n{'='*60}")
print("ADJUSTED MODEL PERFORMANCE SUMMARY")
print(f"{'='*60}")

print(f"{'Position':<12} {'R²':<8} {'MAE':<8} {'RMSE':<8} {'Weight Changes':
↪<20}")
print("-" * 65)

weight_changes = {
    'Defense': 'Int↑, Blocks↓',
    'Midfield': 'PassCmp%↑, Prog↓',
    'Forward': 'No changes',
    'Goalkeeper': 'No changes'
}

for position, model_info in adjusted_models.items():
    changes = weight_changes.get(position, 'No changes')
    print(f"{position:<12} {model_info['r2']:<8.3f} {model_info['mae']:<8.3f} "
          f"{model_info['rmse']:<8.3f} {changes:<20}")

# ==========================================
# FORMULA DOCUMENTATION
# ==========================================

def document_adjusted_formulas():
    """Document the adjusted weighted score formulas"""
    print(f"\n{'='*70}")
    print("ADJUSTED WEIGHTED SCORE FORMULAS")
    print(f"{'='*70}")

    print("\n DEFENSE (ADJUSTED):")
    print("Adjusted_Score_Defense = 2.5×Int + 2.0×Blocks + 1.0×Clr + 2.0×TklW +␣
↪1.3×TklDef + 0.8×TklMid")
    print("Key change: Interceptions coefficient increased from 1.5 to 2.5")

    print("\n MIDFIELD (ADJUSTED):")
    print("Adjusted_Score_Midfield = 2.5×PassCmp% + 1.2×KP + 1.5×Tkl + 0.
↪8×CarriesPrgC + 1.8×PassesPrgP + 0.3×Touches")
    print("Key change: Pass Completion % coefficient increased from 2.0 to 2.5")

    print("\n FORWARD (UNCHANGED):")
```

```python
    print("Score_Forward = 3.0×Gls + 2.0×Ast + 1.0×SoT + 1.5×xG + 1.0×xAG + 0.
↪5×TakeOns")
    print("No changes: Already optimal based on SHAP analysis")

    print("\n  GOALKEEPER (UNCHANGED):")
    print("Score_Goalkeeper = 3.0×TotalCmp% - 2.0×Err + 1.0×PrgDist + 1.
↪5×ShortCmp% + 1.0×MedCmp% + 0.5×TotalCmp")
    print("No changes: Distribution focus validated")

document_adjusted_formulas()

print(f"\n  ADJUSTED WEIGHTED SCORING SYSTEM COMPLETE!")
print(f"  Models retrained with SHAP-informed weight adjustments")
print(f"  Interceptions and Pass Completion given higher importance")
print(f"  Ready for validation against Expected xG insights")
```

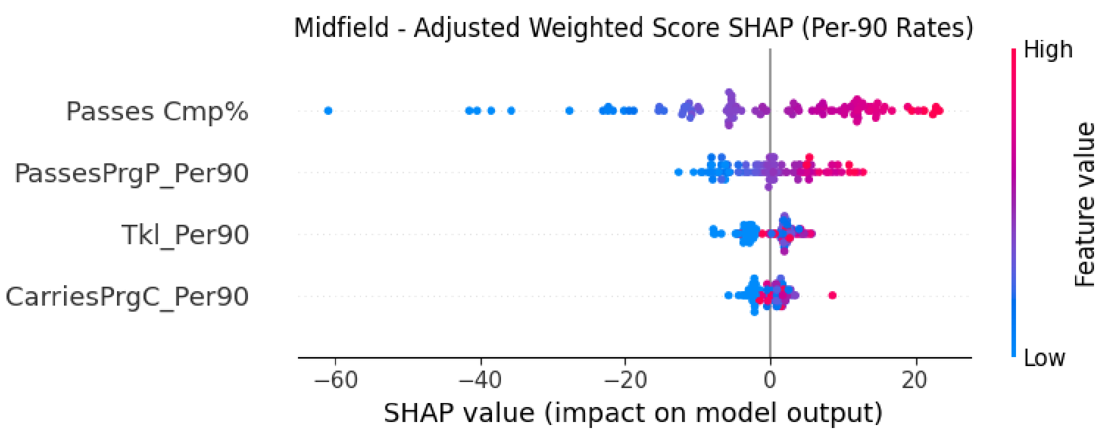
```
ADJUSTED PER-90 RATES SHAP ANALYSIS
============================================================
Target: Recalibrated Weighted Score
Features: All metrics converted to per-90 rates
Adjustments: Based on Expected xG SHAP findings
============================================================
Dataset loaded: (5737, 72)

Creating comprehensive per-90 minute features…
    Gls_Per90
    Ast_Per90
    SoT_Per90
    Tkl_Per90
    Int_Per90
    Blocks_Per90
    ExpectedxG_Per90
    ExpectedxAG_Per90
    Take_OnsSucc_Per90
    CarriesPrgC_Per90
    PassesPrgP_Per90
    TacklesTklW_Per90
    TacklesDef3rd_Per90
    TacklesMid3rd_Per90
    BlocksSh_Per90
    BlocksPass_Per90
    TotalCmp_Per90
    TotalPrgDist_Per90
    ShortAtt_Per90

Created 19 per-90 features
```

```
Creating adjusted weighted scores based on SHAP insights…
Adjusted weighting system:
  Defense: Interceptions weight increased from 1.5 → 2.5
  Defense: Blocks weight decreased from 2.5 → 2.0
  Midfield: Pass Completion weight increased from 2.0 → 2.5
  Midfield: Progressive metrics slightly decreased

Adjusted weighted scores calculated for 5737 observations

Training Forward model with adjusted weights…
  Using 5 metrics: ['Gls_Per90', 'Ast_Per90', 'SoT_Per90', 'ExpectedxG_Per90',
'ExpectedxAG_Per90']
  Train: 1585 samples, Test: 110 samples
  Target range - Train: 0.00 to 24.55
  ADJUSTED Performance: R² = 0.947, MAE = 0.598, RMSE = 0.758

Training Midfield model with adjusted weights…
  Using 4 metrics: ['Passes Cmp%', 'Tkl_Per90', 'CarriesPrgC_Per90',
'PassesPrgP_Per90']
  Train: 1724 samples, Test: 99 samples
  Target range - Train: 0.00 to 294.00
  ADJUSTED Performance: R² = 0.913, MAE = 2.891, RMSE = 5.227

Training Defense model with adjusted weights…
  Using 5 metrics: ['Int_Per90', 'Blocks_Per90', 'TacklesTklW_Per90',
'TacklesDef3rd_Per90', 'TacklesMid3rd_Per90']
  Train: 1718 samples, Test: 105 samples
  Target range - Train: 0.00 to 33.10
  ADJUSTED Performance: R² = 0.969, MAE = 0.482, RMSE = 0.896

Training Goalkeeper model with adjusted weights…
  Using 5 metrics: ['Total Cmp%', 'TotalPrgDist_Per90', 'Short Cmp%', 'Medium
Cmp%', 'TotalCmp_Per90']
  Train: 372 samples, Test: 24 samples
  Target range - Train: 504.50 to 1292.20
  ADJUSTED Performance: R² = 0.993, MAE = 6.803, RMSE = 8.513

 Successfully trained 4 adjusted models

==================================================
ADJUSTED SHAP ANALYSIS: Forward
==================================================
Players in test set: 20

Forward - ADJUSTED Feature Importance:
  1. Gls_Per90: 1.392
  2. SoT_Per90: 0.694
  3. Ast_Per90: 0.630
```

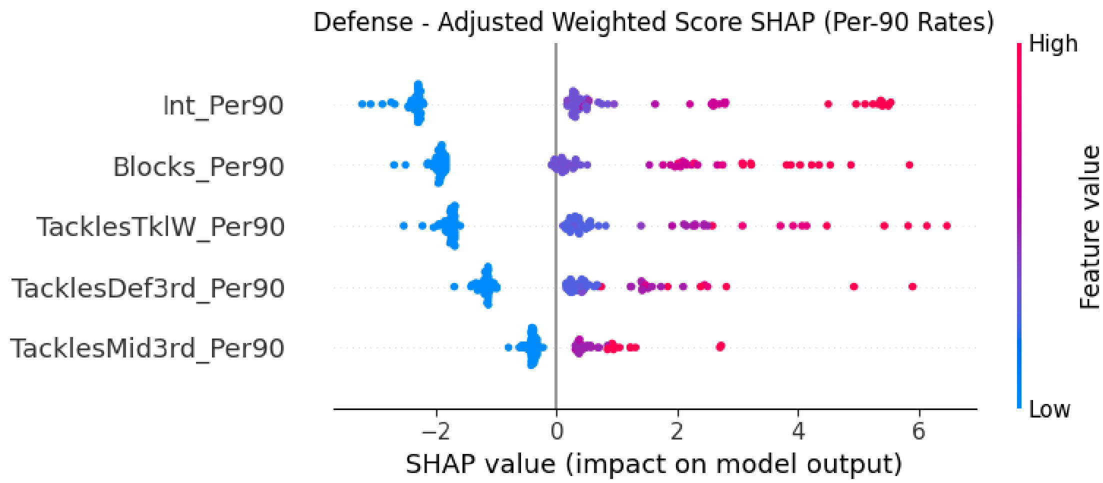
4. ExpectedxG_Per90: 0.481
5. ExpectedxAG_Per90: 0.348



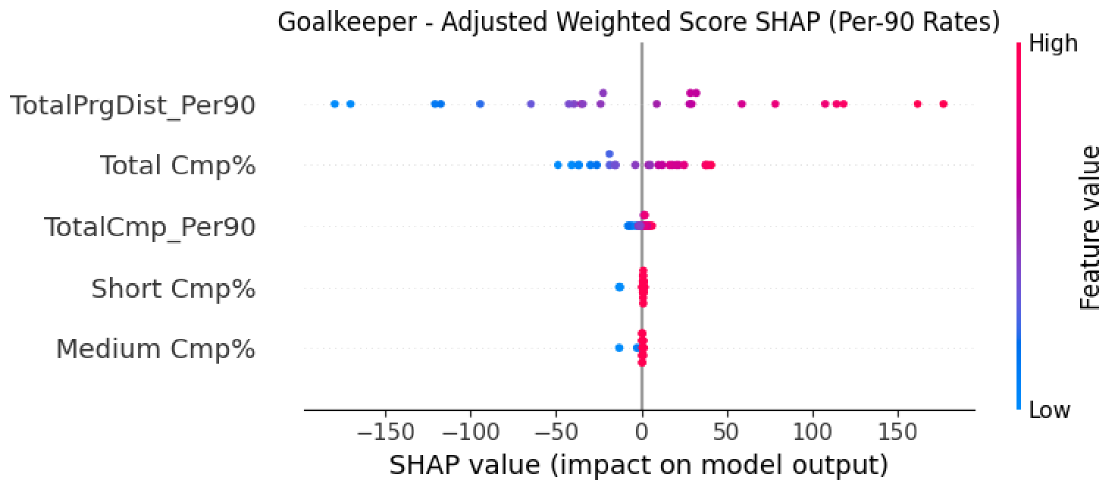

```
==================================================
ADJUSTED SHAP ANALYSIS: Midfield
==================================================
Players in test set: 14

Midfield - ADJUSTED Feature Importance:
  1. Passes Cmp%: 12.927
  2. PassesPrgP_Per90: 4.798
  3. Tkl_Per90: 2.802
  4. CarriesPrgC_Per90: 1.744
```



```
==================================================
ADJUSTED SHAP ANALYSIS: Defense
==================================================
Players in test set: 16

Defense - ADJUSTED Feature Importance:
  1. Int_Per90: 1.907
  2. Blocks_Per90: 1.708
  3. TacklesTklW_Per90: 1.580
  4. TacklesDef3rd_Per90: 1.061
  5. TacklesMid3rd_Per90: 0.527
```




```
==================================================
ADJUSTED SHAP ANALYSIS: Goalkeeper
==================================================
Players in test set: 5

Goalkeeper - ADJUSTED Feature Importance:
  1. TotalPrgDist_Per90: 78.686
  2. Total Cmp%: 24.147
  3. TotalCmp_Per90: 2.879
  4. Short Cmp%: 2.118
  5. Medium Cmp%: 1.292
```




```
======================================================================
COMPARISON: ORIGINAL vs ADJUSTED WEIGHTED SCORING
======================================================================
Position     Adj R²     Adj MAE    Key Change
----------------------------------------------------------------------
Forward      0.947      0.598      No major changes (good performance)
Midfield     0.913      2.891      Pass Completion weight: 2.0 → 2.5
Defense      0.969      0.482      Interceptions weight: 1.5 → 2.5
Goalkeeper   0.993      6.803      No changes needed

======================================================================
WEIGHT ADJUSTMENT JUSTIFICATIONS
======================================================================

 DEFENSE ADJUSTMENTS:
  Interceptions: 1.5 → 2.5 (+67%)
    Justification: Expected xG SHAP showed Int_Per90 = 0.019 (highest)
    Reason: Proactive defending prevents chances before they develop
  Blocks: 2.5 → 2.0 (-20%)
    Justification: Reactive defending, less predictive of overall performance

 MIDFIELD ADJUSTMENTS:
  Pass Completion %: 2.0 → 2.5 (+25%)
    Justification: Expected xG SHAP showed Passes Cmp% = 0.021 (highest)
    Reason: Ball retention prevents opponent chances more than progressive play
  Progressive metrics: Slight decreases
    Justification: Lower SHAP importance than pass completion

 NO CHANGES:
  Forward: Already optimal weighting (Goals > Assists > xG)
```

```
  Goalkeeper: Distribution focus validated by analysis

============================================================
ADJUSTED MODEL PERFORMANCE SUMMARY
============================================================
Position     R²       MAE      RMSE     Weight Changes
-----------------------------------------------------------------
Forward      0.947    0.598    0.758    No changes
Midfield     0.913    2.891    5.227    PassCmp%↑, Prog↓
Defense      0.969    0.482    0.896    Int↑, Blocks↓
Goalkeeper   0.993    6.803    8.513    No changes

======================================================================
ADJUSTED WEIGHTED SCORE FORMULAS
======================================================================

  DEFENSE (ADJUSTED):
Adjusted_Score_Defense = 2.5×Int + 2.0×Blocks + 1.0×Clr + 2.0×TklW + 1.3×TklDef
+ 0.8×TklMid
Key change: Interceptions coefficient increased from 1.5 to 2.5

  MIDFIELD (ADJUSTED):
Adjusted_Score_Midfield = 2.5×PassCmp% + 1.2×KP + 1.5×Tkl + 0.8×CarriesPrgC +
1.8×PassesPrgP + 0.3×Touches
Key change: Pass Completion % coefficient increased from 2.0 to 2.5

  FORWARD (UNCHANGED):
Score_Forward = 3.0×Gls + 2.0×Ast + 1.0×SoT + 1.5×xG + 1.0×xAG + 0.5×TakeOns
No changes: Already optimal based on SHAP analysis

  GOALKEEPER (UNCHANGED):
Score_Goalkeeper = 3.0×TotalCmp% - 2.0×Err + 1.0×PrgDist + 1.5×ShortCmp% +
1.0×MedCmp% + 0.5×TotalCmp
No changes: Distribution focus validated

  ADJUSTED WEIGHTED SCORING SYSTEM COMPLETE!
  Models retrained with SHAP-informed weight adjustments
  Interceptions and Pass Completion given higher importance
  Ready for validation against Expected xG insights
```

2.8 13 Second Validation with Logistic Regression to get threshold

[20]:
```python
import pandas as pd
import numpy as np
import os
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns

# Paths
season_dir = "/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main␣
 ↪Notebook/Data Folder/DataExtracted/"
rebalanced_scores_path = "/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/
 ↪Main Notebook/Data Folder/DataCombined/real_madrid_rebalanced_scores.csv"

season_files = [
    "real_madrid_schedule_17_18.csv",
    "real_madrid_schedule_18_19.csv",
    "real_madrid_schedule_19_20.csv",
    "real_madrid_schedule_20_21.csv",
    "real_madrid_schedule_21_22.csv",
    "real_madrid_schedule_22_23.csv",
    "real_madrid_schedule_23_24 (1).csv",
    "real_madrid_schedule_24_25 (1).csv"
]

def load_and_combine_schedules():
    """
    Load and combine all schedule files
    """
    print("Loading schedule files...")

    all_schedules = []

    for file in season_files:
        file_path = os.path.join(season_dir, file)

        if os.path.exists(file_path):
            print(f"  Loading: {file}")
            df = pd.read_csv(file_path)

            # Add season info
            season = file.split('_')[-1].replace('.csv', '').replace(' (1)', '')
            df['Season'] = season

            all_schedules.append(df)
        else:
            print(f"  File not found: {file}")

    # Combine all schedules
```

```python
    combined_schedule = pd.concat(all_schedules, ignore_index=True)
    print(f"Combined schedule shape: {combined_schedule.shape}")
    print(f"Columns: {list(combined_schedule.columns)}")

    return combined_schedule

def load_rebalanced_scores():
    """
    Load rebalanced scores data
    """
    print(f"\nLoading rebalanced scores from: {rebalanced_scores_path}")

    if os.path.exists(rebalanced_scores_path):
        scores_df = pd.read_csv(rebalanced_scores_path)
        print(f"  Rebalanced scores shape: {scores_df.shape}")
        print(f"Columns: {list(scores_df.columns)}")
        return scores_df
    else:
        print(f"  Rebalanced scores file not found!")
        return None

def create_win_loss_dataset():
    """
    Create dataset with Date, Win/Loss, and TEAM-LEVEL Rebalanced_Score
    """
    print("="*60)
    print("CREATING TEAM-LEVEL WIN/LOSS DATASET FOR LOGISTIC REGRESSION")
    print("="*60)

    # Load schedule data
    schedule_df = load_and_combine_schedules()

    # Load rebalanced scores
    scores_df = load_rebalanced_scores()

    if scores_df is None:
        print("  Cannot proceed without rebalanced scores")
        return None

    print(f"\nSchedule data preview:")
    print(schedule_df.head())

    print(f"\nRebalanced scores preview:")
    print(scores_df.head())

    # Clean and standardize date formats
    print("\nProcessing dates...")
```

```python
    # For schedule data - try to find date column
    date_cols = [col for col in schedule_df.columns if 'date' in col.lower()]
    if date_cols:
        schedule_df['Date'] = pd.to_datetime(schedule_df[date_cols[0]], errors='coerce')
    else:
        print("Available schedule columns:", list(schedule_df.columns))
        date_col = input("Enter the name of the DATE column in schedule data: ")
        schedule_df['Date'] = pd.to_datetime(schedule_df[date_col], errors='coerce')

    # For scores data - try to find date column
    date_cols_scores = [col for col in scores_df.columns if 'date' in col.lower()]
    if date_cols_scores:
        scores_df['Date'] = pd.to_datetime(scores_df[date_cols_scores[0]], errors='coerce')
    else:
        print("Available rebalanced scores columns:", list(scores_df.columns))
        date_col = input("Enter the name of the DATE column in rebalanced scores: ")
        scores_df['Date'] = pd.to_datetime(scores_df[date_col], errors='coerce')

    # Find result column in schedule
    result_cols = [col for col in schedule_df.columns if any(word in col.lower() for word in ['result', 'outcome'])]
    if result_cols:
        result_col = result_cols[0]
    else:
        print("Available schedule columns:", list(schedule_df.columns))
        result_col = input("Enter the name of the RESULT column (W/L/D): ")

    # Find rebalanced score column
    score_cols = [col for col in scores_df.columns if 'rebalanced' in col.lower() or 'score' in col.lower()]
    if score_cols:
        score_col = score_cols[0]
    else:
        print("Available rebalanced scores columns:", list(scores_df.columns))
        score_col = input("Enter the name of the REBALANCED SCORE column: ")

    print(f"Using result column: {result_col}")
    print(f"Using score column: {score_col}")

    # Create clean schedule dataframe
```

```
    schedule_clean = schedule_df[['Date', result_col]].copy()
    schedule_clean.columns = ['Date', 'Result']
    schedule_clean = schedule_clean.dropna()

    # AGGREGATE REBALANCED SCORES BY MATCH (TEAM LEVEL)
    print(f"\n  AGGREGATING TEAM PERFORMANCE BY MATCH...")

    # Group by date and sum/average the rebalanced scores for the entire team
    team_scores = scores_df.groupby('Date').agg({
        score_col: ['sum', 'mean', 'count']
    }).reset_index()

    # Flatten column names
    team_scores.columns = ['Date', 'Team_Rebalanced_Score_Sum',␣
↪'Team_Rebalanced_Score_Mean', 'Players_Count']

    print(f"Team aggregated data shape: {team_scores.shape}")
    print("Team scores preview:")
    print(team_scores.head())

    print(f"\nTeam score statistics:")
    print(f"Sum - Range: {team_scores['Team_Rebalanced_Score_Sum'].min():.2f}␣
↪to {team_scores['Team_Rebalanced_Score_Sum'].max():.2f}")
    print(f"Mean - Range: {team_scores['Team_Rebalanced_Score_Mean'].min():.2f}␣
↪to {team_scores['Team_Rebalanced_Score_Mean'].max():.2f}")
    print(f"Average players per match: {team_scores['Players_Count'].mean():.
↪1f}")

    # Merge schedule with team-level scores
    print("\nMerging schedule with TEAM-LEVEL scores...")
    merged_df = pd.merge(schedule_clean, team_scores, on='Date', how='inner')

    print(f"Merged dataset shape: {merged_df.shape}")

    if merged_df.empty:
        print("  No matching dates found between schedule and team scores!")
        print("Schedule date range:", schedule_clean['Date'].min(), "to",␣
↪schedule_clean['Date'].max())
        print("Team scores date range:", team_scores['Date'].min(), "to",␣
↪team_scores['Date'].max())
        return None

    # Filter for Win/Loss only (remove draws)
    print("\nFiltering Win/Loss matches...")
    win_loss_df = merged_df[merged_df['Result'].isin(['W', 'L'])].copy()
```

```
    # Create binary target
    win_loss_df['Win'] = (win_loss_df['Result'] == 'W').astype(int)

    print(f"Final dataset shape: {win_loss_df.shape}")
    print("Result distribution:")
    print(win_loss_df['Result'].value_counts())
    print("Win distribution:")
    print(win_loss_df['Win'].value_counts())

    # Ask user which team score to use for analysis
    print(f"\nChoose which TEAM score to use for logistic regression:")
    print(f"1. Team_Rebalanced_Score_Sum (total team performance)")
    print(f"2. Team_Rebalanced_Score_Mean (average player performance)")

    choice = input("Enter choice (1 or 2): ").strip()

    if choice == "2":
        score_column = 'Team_Rebalanced_Score_Mean'
        print("  Using MEAN team score (average player performance)")
    else:
        score_column = 'Team_Rebalanced_Score_Sum'
        print("  Using SUM team score (total team performance)")

    # Final clean dataset with chosen score
    final_df = win_loss_df[['Date', 'Result', score_column, 'Players_Count',␣
 ↪'Win']].copy()
    final_df.rename(columns={score_column: 'Rebalanced_Score'}, inplace=True)

    print(f"\nFinal team-level dataset:")
    print(final_df.head(10))

    return final_df

def create_train_test_splits(df):
    """
    Create training and test splits
    """
    print("\n" + "="*40)
    print("CREATING TRAIN/TEST SPLITS")
    print("="*40)

    # Features and target
    X = df[['Rebalanced_Score']].copy()
    y = df['Win'].copy()

    print(f"Features shape: {X.shape}")
    print(f"Target shape: {y.shape}")
```

```python
    # Split data: 70% train, 30% test
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=42, stratify=y
    )

    print(f"Training set: {X_train.shape[0]} matches")
    print(f"Test set: {X_test.shape[0]} matches")

    return X_train, X_test, y_train, y_test

def plot_logistic_curve(df, model, scaler):
    """
    Plot the S-curve showing how Rebalanced_Score predicts Win probability
    """
    print("\n" + "="*40)
    print("PLOTTING LOGISTIC S-CURVE")
    print("="*40)

    # First, let's analyze the data distribution
    print("DATA ANALYSIS:")
    print(f"Rebalanced Score range: {df['Rebalanced_Score'].min():.2f} to {df['Rebalanced_Score'].max():.2f}")
    print(f"Mean: {df['Rebalanced_Score'].mean():.2f}, Std: {df['Rebalanced_Score'].std():.2f}")
    print(f"Win rate: {df['Win'].mean():.2%}")

    # Check if we have enough variation
    if df['Rebalanced_Score'].std() < 1:
        print("   WARNING: Low variation in Rebalanced_Score - S-curve may be flat")

    # Create WIDER range of Rebalanced_Score values for plotting
    score_min = df['Rebalanced_Score'].min() - df['Rebalanced_Score'].std()
    score_max = df['Rebalanced_Score'].max() + df['Rebalanced_Score'].std()
    score_range = np.linspace(score_min, score_max, 200).reshape(-1, 1)

    # Scale the range using the same scaler
    score_range_scaled = scaler.transform(score_range)

    # Predict probabilities
    win_probabilities = model.predict_proba(score_range_scaled)[:, 1]

    print(f"Predicted probability range: {win_probabilities.min():.3f} to {win_probabilities.max():.3f}")

    # Create the plot
```

```
    plt.figure(figsize=(15, 10))

    # Plot the S-curve with better scaling
    plt.subplot(2, 3, 1)
    plt.plot(score_range, win_probabilities, 'b-', linewidth=3, label='Logistic Curve')
    plt.axhline(y=0.5, color='red', linestyle='--', alpha=0.7, label='50% Probability')

    # Add actual data points with jitter for better visibility
    wins = df[df['Win'] == 1]['Rebalanced_Score']
    losses = df[df['Win'] == 0]['Rebalanced_Score']

    # Add some jitter to y-values to see overlapping points
    win_jitter = np.random.normal(1, 0.02, len(wins))
    loss_jitter = np.random.normal(0, 0.02, len(losses))

    plt.scatter(wins, win_jitter, color='green', alpha=0.6, s=30, label=f'Wins ({len(wins)})')
    plt.scatter(losses, loss_jitter, color='red', alpha=0.6, s=30, label=f'Losses ({len(losses)})')

    plt.xlabel('Rebalanced Score')
    plt.ylabel('Win Probability')
    plt.title('S-Curve: Rebalanced Score vs Win Probability')
    plt.grid(True, alpha=0.3)
    plt.legend()
    plt.ylim(-0.1, 1.1)

    # Distribution of Rebalanced Scores by result
    plt.subplot(2, 3, 2)
    plt.hist(losses, bins=15, alpha=0.7, color='red', label=f'Losses ({len(losses)})', density=True)
    plt.hist(wins, bins=15, alpha=0.7, color='green', label=f'Wins ({len(wins)})', density=True)
    plt.xlabel('Rebalanced Score')
    plt.ylabel('Density')
    plt.title('Distribution of Rebalanced Scores')
    plt.legend()
    plt.grid(True, alpha=0.3)

    # Box plot comparison
    plt.subplot(2, 3, 3)
    box_data = [losses, wins]
    box_plot = plt.boxplot(box_data, labels=['Losses', 'Wins'], patch_artist=True)
```

```python
    box_plot['boxes'][0].set_facecolor('red')
    box_plot['boxes'][1].set_facecolor('green')
    plt.ylabel('Rebalanced Score')
    plt.title('Rebalanced Score by Result')
    plt.grid(True, alpha=0.3)

    # Show the logistic function equation
    plt.subplot(2, 3, 4)
    plt.text(0.1, 0.8, f"Logistic Regression Equation:", fontsize=12, fontweight='bold')
    plt.text(0.1, 0.6, f"P(Win) = 1 / (1 + e^-(   +   ×Score))", fontsize=11)
    plt.text(0.1, 0.4, f"Where:", fontsize=11)
    plt.text(0.1, 0.3, f"   (intercept) = {model.intercept_[0]:.4f}", fontsize=10)
    plt.text(0.1, 0.2, f"   (coefficient) = {model.coef_[0][0]:.4f}", fontsize=10)
    plt.text(0.1, 0.05, f"Interpretation: Each 1-unit increase in\nRebalanced Score multiplies odds by {np.exp(model.coef_[0][0]):.3f}", fontsize=9)
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.axis('off')
    plt.title('Model Parameters')

    # Probability bins analysis
    plt.subplot(2, 3, 5)

    # Create probability predictions for actual data
    X_actual = df[['Rebalanced_Score']].values
    X_actual_scaled = scaler.transform(X_actual)
    y_pred_proba_actual = model.predict_proba(X_actual_scaled)[:, 1]

    # Create bins and calculate actual win rate
    bins = np.linspace(0, 1, 11)  # 10 bins from 0 to 1

    actual_win_rates = []
    predicted_probs = []

    for i in range(len(bins)-1):
        mask = (y_pred_proba_actual >= bins[i]) & (y_pred_proba_actual < bins[i+1])
        if mask.sum() > 0:
            actual_rate = df.loc[mask, 'Win'].mean()
            predicted_prob = y_pred_proba_actual[mask].mean()
            actual_win_rates.append(actual_rate)
            predicted_probs.append(predicted_prob)

    # Plot calibration
```

```
    plt.plot([0, 1], [0, 1], 'k--', alpha=0.7, label='Perfect Calibration')
    if predicted_probs:
        plt.plot(predicted_probs, actual_win_rates, 'bo-', markersize=8, label='Model Calibration')
    plt.xlabel('Predicted Win Probability')
    plt.ylabel('Actual Win Rate')
    plt.title('Model Calibration')
    plt.legend()
    plt.grid(True, alpha=0.3)

    # Score vs Probability scatter
    plt.subplot(2, 3, 6)
    colors = ['red' if w == 0 else 'green' for w in df['Win']]
    plt.scatter(df['Rebalanced_Score'], y_pred_proba_actual, c=colors, alpha=0.6)
    plt.xlabel('Rebalanced Score')
    plt.ylabel('Predicted Win Probability')
    plt.title('Score vs Predicted Probability')
    plt.grid(True, alpha=0.3)

    # Add trend line
    z = np.polyfit(df['Rebalanced_Score'], y_pred_proba_actual, 1)
    p = np.poly1d(z)
    plt.plot(df['Rebalanced_Score'], p(df['Rebalanced_Score']), "b--", alpha=0.8)

    plt.tight_layout()
    plt.show()

    # Print detailed insights
    print(f"\nDETAILED ANALYSIS:")
    print(f" " * 50)

    # Statistical significance
    wins_scores = df[df['Win'] == 1]['Rebalanced_Score']
    losses_scores = df[df['Win'] == 0]['Rebalanced_Score']

    from scipy import stats
    t_stat, p_value = stats.ttest_ind(wins_scores, losses_scores)

    print(f"  Win scores mean: {wins_scores.mean():.2f} ± {wins_scores.std():.2f}")
    print(f"  Loss scores mean: {losses_scores.mean():.2f} ± {losses_scores.std():.2f}")
    print(f"  Difference: {wins_scores.mean() - losses_scores.mean():.2f}")
    print(f"  T-test p-value: {p_value:.4f} ({'Significant' if p_value < 0.05 else 'Not significant'})")
```

```python
    # Find score thresholds
    threshold_50 = None
    for score, prob in zip(score_range.flatten(), win_probabilities):
        if prob >= 0.5 and threshold_50 is None:
            threshold_50 = score
            break

    if threshold_50:
        print(f"  Rebalanced Score  {threshold_50:.2f} → Win probability  50%")

    # Score ranges analysis
    q33 = df['Rebalanced_Score'].quantile(0.33)
    q67 = df['Rebalanced_Score'].quantile(0.67)

    low_scores = df[df['Rebalanced_Score'] < q33]
    mid_scores = df[(df['Rebalanced_Score'] >= q33) & (df['Rebalanced_Score'] < q67)]
    high_scores = df[df['Rebalanced_Score'] >= q67]

    print(f"  Low scores (< {q33:.1f}): {low_scores['Win'].mean():.1%} win rate ({len(low_scores)} matches)")
    print(f"  Mid scores ({q33:.1f}-{q67:.1f}): {mid_scores['Win'].mean():.1%} win rate ({len(mid_scores)} matches)")
    print(f"  High scores (> {q67:.1f}): {high_scores['Win'].mean():.1%} win rate ({len(high_scores)} matches)")

    # Model strength assessment
    prob_range = win_probabilities.max() - win_probabilities.min()
    if prob_range < 0.3:
        print(f"   Model shows WEAK discrimination (probability range: {prob_range:.3f})")
    elif prob_range < 0.6:
        print(f"  Model shows MODERATE discrimination (probability range: {prob_range:.3f})")
    else:
        print(f"  Model shows STRONG discrimination (probability range: {prob_range:.3f})")

def run_logistic_regression(X_train, X_test, y_train, y_test):
    """
    Run logistic regression and show results
    """
    print("\n" + "="*40)
    print("LOGISTIC REGRESSION RESULTS")
    print("="*40)
```

```python
    # Scale features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Fit model
    lr_model = LogisticRegression(random_state=42)
    lr_model.fit(X_train_scaled, y_train)

    # Predictions
    y_pred = lr_model.predict(X_test_scaled)
    y_pred_proba = lr_model.predict_proba(X_test_scaled)[:, 1]

    # Results
    train_acc = lr_model.score(X_train_scaled, y_train)
    test_acc = lr_model.score(X_test_scaled, y_test)
    auc_score = roc_auc_score(y_test, y_pred_proba)

    print(f"Training Accuracy: {train_acc:.3f}")
    print(f"Test Accuracy: {test_acc:.3f}")
    print(f"ROC AUC Score: {auc_score:.3f}")

    print(f"\nRebalanced Score Coefficient: {lr_model.coef_[0][0]:.4f}")
    print(f"Intercept: {lr_model.intercept_[0]:.4f}")

    if lr_model.coef_[0][0] > 0:
        print("  Higher Rebalanced Score increases WIN probability")
    else:
        print("  Higher Rebalanced Score decreases WIN probability")

    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))

    return lr_model, scaler

def main():
    """
    Main execution function
    """
    # Create the dataset
    final_df = create_win_loss_dataset()

    if final_df is None:
        print("  Failed to create dataset")
        return
```

```python
    print(f"\n  Successfully created dataset with {len(final_df)} matches")
    print("\nDataset preview:")
    print(final_df.head(10))

    # Save the dataset
    output_path = "/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main␣
↪Notebook/Data Folder/win_loss_rebalanced_dataset.csv"
    final_df.to_csv(output_path, index=False)
    print(f"\n  Dataset saved to: {output_path}")

    # Create train/test splits
    X_train, X_test, y_train, y_test = create_train_test_splits(final_df)

    # Run logistic regression
    model, scaler = run_logistic_regression(X_train, X_test, y_train, y_test)

    # Plot the S-curve and analysis
    plot_logistic_curve(final_df, model, scaler)

    print("\n" + "="*60)
    print("SUCCESS! Ready for logistic regression analysis")
    print("="*60)

    return final_df, model, scaler

# Run the main function
if __name__ == "__main__":
    final_df, model, scaler = main()
```

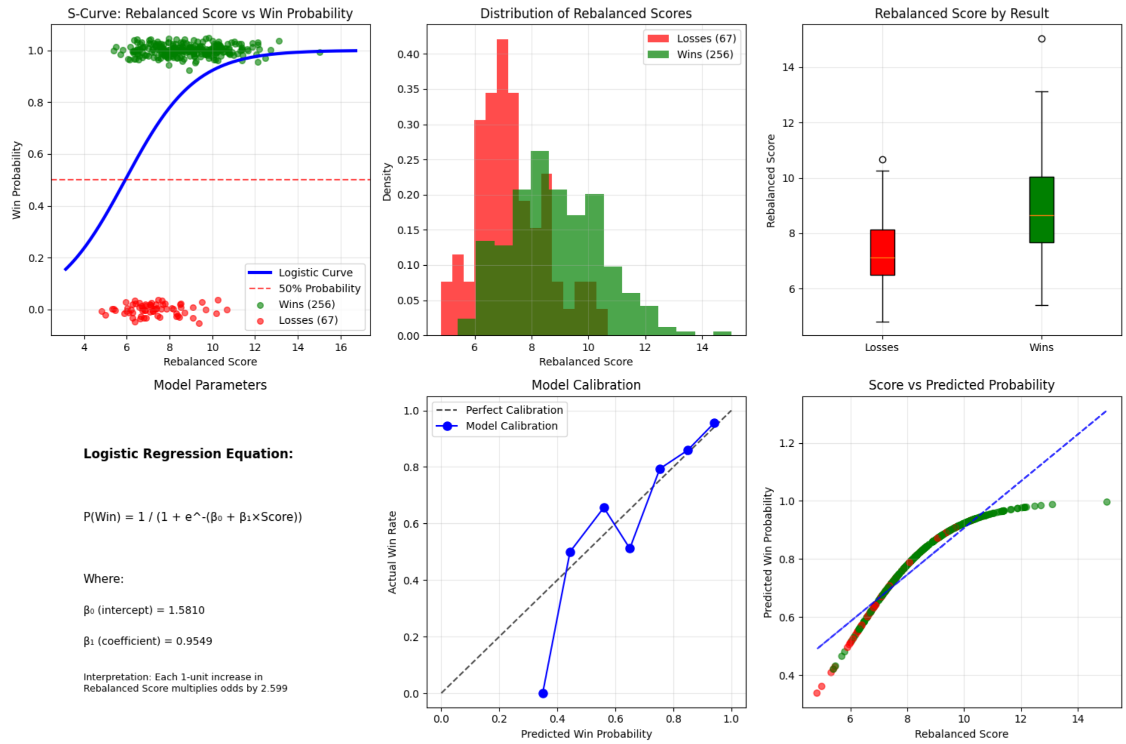
```
============================================================
CREATING TEAM-LEVEL WIN/LOSS DATASET FOR LOGISTIC REGRESSION
============================================================
Loading schedule files…
  Loading: real_madrid_schedule_17_18.csv
  Loading: real_madrid_schedule_18_19.csv
  Loading: real_madrid_schedule_19_20.csv
  Loading: real_madrid_schedule_20_21.csv
  Loading: real_madrid_schedule_21_22.csv
  Loading: real_madrid_schedule_22_23.csv
  Loading: real_madrid_schedule_23_24 (1).csv
  Loading: real_madrid_schedule_24_25 (1).csv
Combined schedule shape: (418, 21)
Columns: ['Date', 'Time', 'Comp', 'Round', 'Day', 'Venue', 'Result', 'GF', 'GA',
'Opponent', 'xG', 'xGA', 'Poss', 'Attendance', 'Captain', 'Formation', 'Opp
Formation', 'Referee', 'Match Report', 'Notes', 'Season']

Loading rebalanced scores from:
/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main Notebook/Data
```

```
Folder/DataCombined/real_madrid_rebalanced_scores.csv
 Rebalanced scores shape: (5737, 72)
Columns: ['Date', 'Competition', 'Opponent', 'Player', '#', 'Nation', 'Pos',
'Age', 'Min', ' Gls', ' Ast', ' PK', ' PKatt', ' Sh', ' SoT', ' CrdY', ' CrdR',
' Int', 'Match URL', 'Season', ' Touches', ' Tkl', ' Blocks', 'Expected xG',
'Expected npxG', 'Expected xAG', 'Passes Cmp', 'Passes Att', 'Passes Cmp%',
'Passes PrgP', 'Carries Carries', 'Carries PrgC', 'Take-Ons Att', 'Take-Ons
Succ', 'Tackles Tkl', 'Tackles TklW', 'Tackles Def 3rd', 'Tackles Mid 3rd',
'Tackles Att 3rd', 'Challenges Tkl', 'Challenges Att', 'Challenges Lost',
'Blocks Blocks', 'Blocks Sh', 'Blocks Pass', 'Int', 'Tkl+Int', 'Clr', 'Err',
'Total Cmp', 'Total Att', 'Total Cmp%', 'Total TotDist', 'Total PrgDist', 'Short
Cmp', 'Short Att', 'Short Cmp%', 'Medium Cmp', 'Medium Att', 'Medium Cmp%',
'Long Cmp', 'Long Att', 'Ast', 'xAG', 'xA', 'KP', 'PPA', 'CrsPA', 'PrgP',
'Position_Group', 'Rebalanced_Score', 'Week']

Schedule data preview:
      Date   Time          Comp        Round  Day Venue Result GF GA  \
0  8/20/17  22:15       La Liga  Matchweek 1  Sun  Away      W  3  0
1  8/27/17  22:15       La Liga  Matchweek 2  Sun  Home      D  2  2
2   9/9/17  13:00       La Liga  Matchweek 3  Sat  Home      D  1  1
3  9/13/17  20:45  Champions Lg  Group stage  Wed  Home      W  3  0
4  9/17/17  20:45       La Liga  Matchweek 4  Sun  Away      W  3  1

        Opponent  …  xGA  Poss  Attendance       Captain  Formation  \
0      La Coruña  …  1.7  68.0     27770.0  Sergio Ramos    4-3-1-2
1       Valencia  …  0.9  66.0     65107.0       Marcelo      4-3-3
2        Levante  …  0.7  68.0     67789.0  Sergio Ramos    4-2-3-1
3    cy APOEL FC  …  0.2  69.0     71060.0  Sergio Ramos  4-1-2-1-2
4  Real Sociedad  …  0.5  50.0     24966.0  Sergio Ramos      4-3-3

  Opp Formation              Referee  Match Report Notes Season
0       4-2-3-1        José González  Match Report   NaN     18
1         4-4-2      David Fernández  Match Report   NaN     18
2         4-5-1  Alejandro Hernández  Match Report   NaN     18
3         4-4-2       Benoît Bastien  Match Report   NaN     18
4         4-3-3     Ignacio Iglesias  Match Report   NaN     18

[5 rows x 21 columns]

Rebalanced scores preview:
        Date Competition             Opponent         Player     #   Nation  \
0  2017-08-20     La Liga  Deportivo La Coruna    Gareth Bale  11.0  wls WAL
1  2017-08-20     La Liga  Deportivo La Coruna  Lucas Vázquez  17.0   es ESP
2  2017-08-20     La Liga  Deportivo La Coruna  Karim Benzema   9.0   fr FRA
3  2017-08-20     La Liga  Deportivo La Coruna           Isco  22.0   es ESP
4  2017-08-20     La Liga  Deportivo La Coruna  Marco Asensio  20.0   es ESP

     Pos     Age   Min   Gls  …  Ast  xAG   xA   KP  PPA  CrsPA  PrgP  \
```

```
0     FW  28-035  79.0   1.0  …  1.0  0.2  0.1  2.0  0.0    0.0   2.0
1  FW,RM  26-050  11.0   0.0  …  0.0  0.0  0.1  0.0  0.0    0.0   1.0
2     FW  29-244  90.0   0.0  …  1.0  1.1  1.0  5.0  3.0    0.0   5.0
3     AM  25-121  65.0   0.0  …  0.0  0.0  0.2  0.0  3.0    0.0   6.0
4  AM,LM  21-211  25.0   0.0  …  0.0  0.0  0.1  0.0  0.0    0.0   0.0

   Position_Group Rebalanced_Score Week
0         Forward        27.644740   33
1         Forward         0.012222   33
2         Forward         8.100000   33
3        Midfield         4.109145   33
4        Midfield         0.387778   33

[5 rows x 72 columns]

Processing dates…
Using result column: Result
Using score column: Rebalanced_Score

  AGGREGATING TEAM PERFORMANCE BY MATCH…
Team aggregated data shape: (397, 4)
Team scores preview:
        Date  Team_Rebalanced_Score_Sum  Team_Rebalanced_Score_Mean  \
0 2017-08-20                 125.117751                    8.936982
1 2017-08-27                 121.514616                    8.679615
2 2017-09-09                 124.222270                    8.873019
3 2017-09-13                 140.340164                   10.024297
4 2017-09-17                 137.742201                    9.838729

   Players_Count
0             14
1             14
2             14
3             14
4             14

Team score statistics:
Sum - Range: 70.02 to 210.45
Mean - Range: 4.82 to 15.03
Average players per match: 14.5

Merging schedule with TEAM-LEVEL scores…
Merged dataset shape: (397, 5)

Filtering Win/Loss matches…
Final dataset shape: (323, 6)
Result distribution:
Result
```

```
W    256
L     67
Name: count, dtype: int64
Win distribution:
Win
1    256
0     67
Name: count, dtype: int64

Choose which TEAM score to use for logistic regression:
1. Team_Rebalanced_Score_Sum (total team performance)
2. Team_Rebalanced_Score_Mean (average player performance)
 Using MEAN team score (average player performance)

Final team-level dataset:
         Date Result  Rebalanced_Score  Players_Count  Win
0  2017-08-20      W          8.936982             14    1
3  2017-09-13      W         10.024297             14    1
4  2017-09-17      W          9.838729             14    1
5  2017-09-20      L          8.169440             14    0
6  2017-09-23      W          7.581374             13    1
7  2017-09-26      W         10.297693             14    1
8  2017-10-01      W         10.060608             14    1
9  2017-10-14      W         10.416471             14    1
11 2017-10-22      W          9.957959             14    1
12 2017-10-29      L          7.629451             14    0

 Successfully created dataset with 323 matches

Dataset preview:
         Date Result  Rebalanced_Score  Players_Count  Win
0  2017-08-20      W          8.936982             14    1
3  2017-09-13      W         10.024297             14    1
4  2017-09-17      W          9.838729             14    1
5  2017-09-20      L          8.169440             14    0
6  2017-09-23      W          7.581374             13    1
7  2017-09-26      W         10.297693             14    1
8  2017-10-01      W         10.060608             14    1
9  2017-10-14      W         10.416471             14    1
11 2017-10-22      W          9.957959             14    1
12 2017-10-29      L          7.629451             14    0

 Dataset saved to: /Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main
Notebook/Data Folder/win_loss_rebalanced_dataset.csv

========================================
CREATING TRAIN/TEST SPLITS
========================================
```

```
Features shape: (323, 1)
Target shape: (323,)
Training set: 226 matches
Test set: 97 matches

========================================
LOGISTIC REGRESSION RESULTS
========================================
Training Accuracy: 0.779
Test Accuracy: 0.845
ROC AUC Score: 0.842

Rebalanced Score Coefficient: 0.9549
Intercept: 1.5810
  Higher Rebalanced Score increases WIN probability

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.30      0.44        20
           1       0.84      0.99      0.91        77

    accuracy                           0.85        97
   macro avg       0.85      0.64      0.68        97
weighted avg       0.85      0.85      0.81        97


========================================
PLOTTING LOGISTIC S-CURVE
========================================
DATA ANALYSIS:
Rebalanced Score range: 4.82 to 15.03
Mean: 8.52, Std: 1.68
Win rate: 79.26%
Predicted probability range: 0.155 to 0.999
```




```
DETAILED ANALYSIS:

  Win scores mean: 8.84 ± 1.63
  Loss scores mean: 7.31 ± 1.26
  Difference: 1.52
  T-test p-value: 0.0000 (Significant)
  Rebalanced Score   5.94 → Win probability   50%
  Low scores (< 7.6): 57.9% win rate (107 matches)
  Mid scores (7.6-9.2): 85.3% win rate (109 matches)
  High scores (> 9.2): 94.4% win rate (107 matches)
  Model shows STRONG discrimination (probability range: 0.843)

============================================================
SUCCESS! Ready for logistic regression analysis
============================================================
```

[21]:

```python
import pandas as pd
import numpy as np
import os
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.metrics import classification_report, confusion_matrix,␣
↪roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

# Paths
season_dir = "/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main␣
↪Notebook/Data Folder/DataExtracted/"
rebalanced_scores_path = "/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/
↪Main Notebook/Data Folder/DataCombined/real_madrid_rebalanced_scores.csv"

season_files = [
    "real_madrid_schedule_17_18.csv",
    "real_madrid_schedule_18_19.csv",
    "real_madrid_schedule_19_20.csv",
    "real_madrid_schedule_20_21.csv",
    "real_madrid_schedule_21_22.csv",
    "real_madrid_schedule_22_23.csv",
    "real_madrid_schedule_23_24 (1).csv",
    "real_madrid_schedule_24_25 (1).csv"
]

def load_and_combine_schedules():
    """Load and combine all schedule files"""
    print("Loading schedule files...")
    all_schedules = []

    for file in season_files:
        file_path = os.path.join(season_dir, file)
        if os.path.exists(file_path):
            print(f"  Loading: {file}")
            df = pd.read_csv(file_path)
            season = file.split('_')[-1].replace('.csv', '').replace(' (1)', '')
            df['Season'] = season
            all_schedules.append(df)
        else:
            print(f"  File not found: {file}")

    combined_schedule = pd.concat(all_schedules, ignore_index=True)
    print(f"Combined schedule shape: {combined_schedule.shape}")
    return combined_schedule

def load_rebalanced_scores():
    """Load rebalanced scores data"""
    print(f"\nLoading rebalanced scores from: {rebalanced_scores_path}")
```

```
    if os.path.exists(rebalanced_scores_path):
        scores_df = pd.read_csv(rebalanced_scores_path)
        print(f" Rebalanced scores shape: {scores_df.shape}")
        print(f"Columns: {list(scores_df.columns)}")
        return scores_df
    else:
        print(f" Rebalanced scores file not found!")
        return None

def classify_position(pos):
    """Classify position into Forward, Midfielder, Defender"""
    if pd.isna(pos):
        return 'Unknown'

    pos_str = str(pos).upper()

    # Forward positions
    if any(keyword in pos_str for keyword in ['FW', 'CF', 'LW', 'RW', 'ST',␣
↪'FORWARD']):
        return 'Forward'
    # Midfielder positions
    elif any(keyword in pos_str for keyword in ['MF', 'CM', 'DM', 'AM', 'LM',␣
↪'RM', 'CDM', 'CAM', 'MIDFIELDER']):
        return 'Midfielder'
    # Defender positions
    elif any(keyword in pos_str for keyword in ['DF', 'CB', 'LB', 'RB', 'WB',␣
↪'SW', 'DEFENDER']):
        return 'Defender'
    # Goalkeeper
    elif any(keyword in pos_str for keyword in ['GK', 'GOALKEEPER']):
        return 'Goalkeeper'
    else:
        return 'Unknown'

def create_position_based_dataset():
    """Create dataset with position-based analysis including multiple␣
↪features"""
    print("="*60)
    print("CREATING POSITION-BASED WIN/LOSS DATASET WITH MULTIPLE FEATURES")
    print("="*60)

    # Load data
    schedule_df = load_and_combine_schedules()
    scores_df = load_rebalanced_scores()

    if scores_df is None:
        return None
```

```python
    print(f"\nAvailable columns in rebalanced scores:")
    print(scores_df.columns.tolist())

    # Process dates
    print("\nProcessing dates...")
    date_cols = [col for col in schedule_df.columns if 'date' in col.lower()]
    if date_cols:
        schedule_df['Date'] = pd.to_datetime(schedule_df[date_cols[0]], errors='coerce')
    else:
        print("Available schedule columns:", list(schedule_df.columns))
        date_col = input("Enter the name of the DATE column in schedule data: ")
        schedule_df['Date'] = pd.to_datetime(schedule_df[date_col], errors='coerce')

    date_cols_scores = [col for col in scores_df.columns if 'date' in col.lower()]
    if date_cols_scores:
        scores_df['Date'] = pd.to_datetime(scores_df[date_cols_scores[0]], errors='coerce')
    else:
        print("Available rebalanced scores columns:", list(scores_df.columns))
        date_col = input("Enter the name of the DATE column in rebalanced scores: ")
        scores_df['Date'] = pd.to_datetime(scores_df[date_col], errors='coerce')

    # Find columns
    result_cols = [col for col in schedule_df.columns if any(word in col.lower() for word in ['result', 'outcome'])]
    result_col = result_cols[0] if result_cols else input("Enter the name of the RESULT column: ")

    pos_cols = [col for col in scores_df.columns if any(word in col.lower() for word in ['pos', 'position'])]
    position_col = pos_cols[0] if pos_cols else input("Enter the name of the POSITION column: ")

    print(f"Using: {result_col}, {position_col}")

    # Clean schedule data
    schedule_clean = schedule_df[['Date', result_col]].copy()
    schedule_clean.columns = ['Date', 'Result']
    schedule_clean = schedule_clean.dropna()

    # Classify positions
```

```
    print(f"\n  CLASSIFYING POSITIONS...")
    scores_df['Position_Group'] = scores_df[position_col].
↪apply(classify_position)

    print("Position Group Classification:")
    print(scores_df['Position_Group'].value_counts())

    # Define features to aggregate by position based on your rebalanced score␣
↪formula
    forward_features = [
        'Gls', 'Goals',  # Goals
        'Ast', 'Assists',  # Assists
        'SoT', 'Shots_On_Target',  # Shots on Target
        'xG', 'Expected_Goals',  # Expected Goals
        'xAG', 'Expected_Assisted_Goals',  # Expected Assisted Goals
        'TakeOns', 'Take_Ons_Succ', 'Takeouts_Successful',  # Take-ons
        'Rebalanced_Score'  # Original rebalanced score
    ]

    midfielder_features = [
        'Passes_Cmp%', 'Pass_Completion_Percent',␣
↪'Total_Pass_Completion_Percent',  # Pass Completion %
        'KP', 'Key_Passes',  # Key Passes
        'Tkl', 'Tackles', 'Tackles_Made',  # Tackles
        'Carries_PrgC', 'Progressive_Carries',  # Progressive Carries
        'Passes_PrgP', 'Progressive_Passes',  # Progressive Passes
        'Touches',  # Touches
        'Rebalanced_Score'  # Original rebalanced score
    ]

    defender_features = [
        'Int', 'Interceptions',  # Interceptions
        'Blocks', 'Blocks_Made',  # Blocks
        'Clr', 'Clearances',  # Clearances
        'Tkl_W', 'Tackles_Won',  # Tackles Won
        'Tkl_Def', 'Tackles_Defensive_Third',  # Tackles Defensive 3rd
        'Tkl_Mid', 'Tackles_Middle_Third',  # Tackles Middle 3rd
        'Rebalanced_Score'  # Original rebalanced score
    ]

    goalkeeper_features = [
        'Total_Cmp%', 'Total_Pass_Completion_Percent',  # Total Pass Completion␣
↪%
        'Err', 'Errors',  # Errors
        'Prg_Dist', 'Total_Progressive_Distance',  # Progressive Distance
        'Short_Cmp%', 'Short_Pass_Completion_Percent',  # Short Pass Completion␣
↪%
```

```
        'Med_Cmp%', 'Medium_Pass_Completion_Percent',  # Medium Pass Completion␣
↪%
        'Total_Cmp', 'Total_Passes_Completed',  # Total Passes Completed
        'Rebalanced_Score'  # Original rebalanced score
    ]

    # Find which features actually exist in the dataset
    available_columns = scores_df.columns.tolist()
    print(f"\n  IDENTIFYING AVAILABLE FEATURES...")

    def find_available_features(feature_list, feature_type):
        found_features = []
        for feature in feature_list:
            # Try exact match first, then partial matches
            if feature in available_columns:
                found_features.append(feature)
            else:
                # Look for similar column names
                matches = [col for col in available_columns if any(part in col␣
↪for part in feature.split('_')) and len(col) > 2]
                if matches:
                    found_features.extend(matches[:1])  # Take first match

        # Remove duplicates while preserving order
        unique_features = []
        for f in found_features:
            if f not in unique_features:
                unique_features.append(f)

        print(f"{feature_type}: {len(unique_features)} features found -␣
↪{unique_features}")
        return unique_features

    forward_available = find_available_features(forward_features, "FORWARD")
    midfielder_available = find_available_features(midfielder_features,␣
↪"MIDFIELDER")
    defender_available = find_available_features(defender_features, "DEFENDER")
    goalkeeper_available = find_available_features(goalkeeper_features,␣
↪"GOALKEEPER")

    # Aggregate by position group per match with multiple features
    print(f"\n  AGGREGATING MULTIPLE FEATURES BY POSITION GROUP PER MATCH...")

    # Create aggregation dictionary for all features
    agg_dict = {}

    # Add features for each position if they exist
```

```
    all_features = list(set(forward_available + midfielder_available +␣
↪defender_available + goalkeeper_available))

    for feature in all_features:
        if feature in available_columns:
            agg_dict[feature] = ['sum', 'mean', 'count']

    print(f"Aggregating {len(agg_dict)} features: {list(agg_dict.keys())}")

    # Group by date and position, aggregate all features
    position_stats = scores_df.groupby(['Date', 'Position_Group']).
↪agg(agg_dict).reset_index()

    # Flatten column names
    new_columns = ['Date', 'Position_Group']
    for feature in agg_dict.keys():
        new_columns.extend([f'{feature}_sum', f'{feature}_mean',␣
↪f'{feature}_count'])

    position_stats.columns = new_columns

    # Create separate dataframes for each position
    position_dfs = {}

    for position in ['Forward', 'Midfielder', 'Defender', 'Goalkeeper']:
        pos_data = position_stats[position_stats['Position_Group'] == position].
↪copy()

        if len(pos_data) > 0:
            # Select relevant features for this position
            if position == 'Forward':
                relevant_features = [f for f in forward_available if f in␣
↪agg_dict.keys()]
            elif position == 'Midfielder':
                relevant_features = [f for f in midfielder_available if f in␣
↪agg_dict.keys()]
            elif position == 'Defender':
                relevant_features = [f for f in defender_available if f in␣
↪agg_dict.keys()]
            else:  # Goalkeeper
                relevant_features = [f for f in goalkeeper_available if f in␣
↪agg_dict.keys()]

            # Keep Date and relevant feature columns (sum and mean)
            keep_columns = ['Date']
            for feature in relevant_features:
```

```
                keep_columns.extend([f'{feature}_sum', f'{feature}_mean'])

            # Filter columns that actually exist
            existing_columns = [col for col in keep_columns if col in pos_data.
↪columns]
            pos_subset = pos_data[existing_columns].copy()

            # Rename columns to include position prefix
            rename_dict = {}
            for col in pos_subset.columns:
                if col != 'Date':
                    rename_dict[col] = f'{position}_{col}'

            pos_subset.rename(columns=rename_dict, inplace=True)
            position_dfs[position] = pos_subset

            print(f"{position}: {len(pos_subset)} matches,␣
↪{len(existing_columns)-1} features")

    # Merge all position dataframes
    merged_positions = None
    for position, pos_df in position_dfs.items():
        if merged_positions is None:
            merged_positions = pos_df.copy()
        else:
            merged_positions = pd.merge(merged_positions, pos_df, on='Date',␣
↪how='outer')

    # Fill NaN values with 0 (for matches where position didn't play)
    merged_positions = merged_positions.fillna(0)

    print(f"Combined position data shape: {merged_positions.shape}")

    # Merge with schedule
    merged_df = pd.merge(schedule_clean, merged_positions, on='Date',␣
↪how='inner')

    # Filter Win/Loss only
    win_loss_df = merged_df[merged_df['Result'].isin(['W', 'L'])].copy()
    win_loss_df['Win'] = (win_loss_df['Result'] == 'W').astype(int)

    print(f"Final dataset shape: {win_loss_df.shape}")
    print(f"Total features: {len(win_loss_df.columns) - 3}")  # Subtract Date,␣
↪Result, Win

    # Show available features by position
    for position in ['Forward', 'Midfielder', 'Defender', 'Goalkeeper']:
```

```
        pos_columns = [col for col in win_loss_df.columns if col.
↪startswith(f'{position}_')]
        if pos_columns:
            print(f"{position} features ({len(pos_columns)}): {pos_columns[:5]}.
↪.." if len(pos_columns) > 5 else f"{position} features: {pos_columns}")

    return win_loss_df

def plot_forward_analysis_only(df, score_col, model, scaler, position_name):
    """Plot S-curve, ROC curve, and confusion matrix for Forward analysis"""

    # Create EXTENDED range to show complete S-curve (0 to 1)
    data_std = df[score_col].std()
    data_mean = df[score_col].mean()

    # Extend range much further to ensure we reach 0% and 100% probability
    score_min = data_mean - 4 * data_std
    score_max = data_mean + 4 * data_std

    score_range = np.linspace(score_min, score_max, 500).reshape(-1, 1)
    score_range_scaled = scaler.transform(score_range)
    win_probabilities = model.predict_proba(score_range_scaled)[:, 1]

    print(f"{position_name} - Extended range: {score_min:.2f} to {score_max:.
↪2f}")
    print(f"{position_name} - Probability range: {win_probabilities.min():.3f}␣
↪to {win_probabilities.max():.3f}")

    # Create the plots
    fig, axes = plt.subplots(2, 3, figsize=(18, 12))

    # Plot 1: COMPLETE S-CURVE
    ax1 = axes[0, 0]
    ax1.plot(score_range, win_probabilities, 'b-', linewidth=4, label='Complete␣
↪S-Curve')
    ax1.axhline(y=0.5, color='red', linestyle='--', alpha=0.7, label='50%␣
↪Probability')
    ax1.axhline(y=0.1, color='orange', linestyle=':', alpha=0.5, label='10%')
    ax1.axhline(y=0.9, color='orange', linestyle=':', alpha=0.5, label='90%')

    # Add actual data points
    wins = df[df['Win'] == 1][score_col]
    losses = df[df['Win'] == 0][score_col]

    win_jitter = np.random.normal(1, 0.02, len(wins))
    loss_jitter = np.random.normal(0, 0.02, len(losses))
```

```
    ax1.scatter(wins, win_jitter, color='green', alpha=0.7, s=50, label=f'Wins␣
↪({len(wins)})', zorder=5)
    ax1.scatter(losses, loss_jitter, color='red', alpha=0.7, s=50,␣
↪label=f'Losses ({len(losses)})', zorder=5)

    # Highlight actual data range
    ax1.axvspan(df[score_col].min(), df[score_col].max(), alpha=0.15,␣
↪color='yellow', label='Data Range')

    ax1.set_xlabel(f'{position_name} Rebalanced Score')
    ax1.set_ylabel('Win Probability')
    ax1.set_title(f'{position_name} - COMPLETE S-Curve (0% to 100%)')
    ax1.grid(True, alpha=0.3)
    ax1.legend()
    ax1.set_ylim(-0.05, 1.05)

    # Plot 2: Distribution comparison
    ax2 = axes[0, 1]
    ax2.hist(losses, bins=15, alpha=0.7, color='red', label=f'Losses␣
↪({len(losses)})', density=True)
    ax2.hist(wins, bins=15, alpha=0.7, color='green', label=f'Wins␣
↪({len(wins)})', density=True)
    ax2.set_xlabel(f'{position_name} Rebalanced Score')
    ax2.set_ylabel('Density')
    ax2.set_title(f'{position_name} - Score Distribution')
    ax2.legend()
    ax2.grid(True, alpha=0.3)

    # Add statistics
    ax2.text(0.05, 0.95, f'Win Mean: {wins.mean():.1f}\nLoss Mean: {losses.
↪mean():.1f}\nDiff: {wins.mean()-losses.mean():.1f}',
             transform=ax2.transAxes, verticalalignment='top',
             bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.8))

    # Plot 3: Box plot
    ax3 = axes[0, 2]
    box_data = [losses, wins]
    box_plot = ax3.boxplot(box_data, labels=['Losses', 'Wins'],␣
↪patch_artist=True)
    box_plot['boxes'][0].set_facecolor('red')
    box_plot['boxes'][1].set_facecolor('green')
    ax3.set_ylabel(f'{position_name} Rebalanced Score')
    ax3.set_title(f'{position_name} - Score by Result')
    ax3.grid(True, alpha=0.3)
```

```python
    # Plot 4: ROC CURVE
    from sklearn.metrics import roc_curve, auc

    # Get predictions for ROC curve
    X_test = df[[score_col]]
    X_test_scaled = scaler.transform(X_test)
    y_test = df['Win']
    y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]

    fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
    roc_auc = auc(fpr, tpr)

    ax4 = axes[1, 0]
    ax4.plot(fpr, tpr, color='darkorange', lw=3, label=f'ROC Curve (AUC = {roc_auc:.3f})')
    ax4.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Classifier')
    ax4.set_xlim([0.0, 1.0])
    ax4.set_ylim([0.0, 1.05])
    ax4.set_xlabel('False Positive Rate')
    ax4.set_ylabel('True Positive Rate')
    ax4.set_title('ROC Curve')
    ax4.legend(loc="lower right")
    ax4.grid(True, alpha=0.3)

    # Plot 5: CONFUSION MATRIX
    y_pred = model.predict(X_test_scaled)
    cm = confusion_matrix(y_test, y_pred)

    ax5 = axes[1, 1]
    im = ax5.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    ax5.figure.colorbar(im, ax=ax5)

    # Add text annotations
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax5.text(j, i, format(cm[i, j], 'd'),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black",
                    fontsize=14, fontweight='bold')

    ax5.set_ylabel('True Label')
    ax5.set_xlabel('Predicted Label')
    ax5.set_title('Confusion Matrix')
    ax5.set_xticks([0, 1])
    ax5.set_yticks([0, 1])
```

```python
    ax5.set_xticklabels(['Loss', 'Win'])
    ax5.set_yticklabels(['Loss', 'Win'])

    # Add confusion matrix details
    tn, fp, fn, tp = cm.ravel()
    ax5.text(0.5, -0.15, f'True Negatives: {tn}\nFalse Positives: {fp}\nFalse␣
↪Negatives: {fn}\nTrue Positives: {tp}',
             transform=ax5.transAxes, ha='center',
             bbox=dict(boxstyle='round', facecolor='lightgray', alpha=0.8))

    # Plot 6: Performance Metrics
    ax6 = axes[1, 2]
    ax6.axis('off')

    # Calculate metrics
    accuracy = (tp + tn) / (tp + tn + fp + fn)
    precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    recall = tp / (tp + fn) if (tp + fn) > 0 else 0
    specificity = tn / (tn + fp) if (tn + fp) > 0 else 0
    f1_score = 2 * (precision * recall) / (precision + recall) if (precision +␣
↪recall) > 0 else 0

    metrics_text = f"""
    PERFORMANCE METRICS

    Accuracy: {accuracy:.3f}
    Precision: {precision:.3f}
    Recall (Sensitivity): {recall:.3f}
    Specificity: {specificity:.3f}
    F1-Score: {f1_score:.3f}
    AUC: {roc_auc:.3f}

    CONFUSION MATRIX BREAKDOWN

    True Positives (TP): {tp}
    True Negatives (TN): {tn}
    False Positives (FP): {fp}
    False Negatives (FN): {fn}

    Total Predictions: {tp + tn + fp + fn}
    """

    ax6.text(0.1, 0.9, metrics_text, transform=ax6.transAxes, fontsize=11,
             verticalalignment='top', fontfamily='monospace',
             bbox=dict(boxstyle='round', facecolor='lightblue', alpha=0.8))

    plt.tight_layout()
```

```
    plt.show()

    # Print detailed analysis
    print(f"\n{position_name} DETAILED ANALYSIS:")
    print(f" " * 50)

    # Find key thresholds
    threshold_10 = threshold_50 = threshold_90 = None
    for score, prob in zip(score_range.flatten(), win_probabilities):
        if prob >= 0.1 and threshold_10 is None:
            threshold_10 = score
        if prob >= 0.5 and threshold_50 is None:
            threshold_50 = score
        if prob >= 0.9 and threshold_90 is None:
            threshold_90 = score

    print(f"  Score for 10% win prob: {threshold_10:.2f}")
    print(f"  Score for 50% win prob: {threshold_50:.2f}")
    print(f"  Score for 90% win prob: {threshold_90:.2f}")

    # Statistical test
    t_stat, p_value = stats.ttest_ind(wins, losses)
    print(f"  T-test p-value: {p_value:.4f} ({'Significant' if p_value < 0.05 else 'Not significant'})")

    print(f"\n  CLASSIFICATION RESULTS:")
    print(f"True Positives (Correctly predicted wins): {tp}")
    print(f"True Negatives (Correctly predicted losses): {tn}")
    print(f"False Positives (Predicted win, but lost): {fp}")
    print(f"False Negatives (Predicted loss, but won): {fn}")

    return {
        'position': position_name,
        'threshold_50': threshold_50,
        'win_mean': wins.mean(),
        'loss_mean': losses.mean(),
        'p_value': p_value,
        'coefficient': model.coef_[0][0],
        'auc': roc_auc,
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1_score': f1_score,
        'confusion_matrix': {'tp': tp, 'tn': tn, 'fp': fp, 'fn': fn}
    }

def analyze_all_positions_with_multiple_features(df):
```

```
    """Run analysis for Forward position only"""
    print("\n" + "="*60)
    print("ANALYZING FORWARD POSITION ONLY")
    print("="*60)

    results = []
    position = 'Forward'  # Only analyze Forward

    # Find all features for Forward position
    pos_columns = [col for col in df.columns if col.startswith(f'{position}_')␣
↪and ('_sum' in col or '_mean' in col)]

    if not pos_columns:
        print(f"  No features found for {position}")
        return results

    print(f"\n{'='*50}")
    print(f"ANALYZING {position.upper()} ({len(pos_columns)} features)")
    print(f"{'='*50}")

    # Show available features
    sum_features = [col for col in pos_columns if '_sum' in col]
    mean_features = [col for col in pos_columns if '_mean' in col]

    print(f"  SUM features ({len(sum_features)}):")
    for feat in sum_features[:8]:  # Show first 8
        print(f"    • {feat}")
    if len(sum_features) > 8:
        print(f"    ... and {len(sum_features)-8} more")

    print(f"  MEAN features ({len(mean_features)}):")
    for feat in mean_features[:8]:  # Show first 8
        print(f"    • {feat}")
    if len(mean_features) > 8:
        print(f"    ... and {len(mean_features)-8} more")

    # Create dataset for Forward position
    analysis_columns = ['Date', 'Result', 'Win'] + pos_columns
    pos_data = df[analysis_columns].copy()
    pos_data = pos_data.dropna()

    if len(pos_data) < 20:
        print(f"  Skipping {position} - insufficient data ({len(pos_data)}␣
↪matches)")
        return results

    print(f"  Data: {len(pos_data)} matches")
```

```
    # SINGLE FEATURE ANALYSIS (Rebalanced Score)
    rebalanced_cols = [col for col in pos_columns if 'Rebalanced_Score' in col␣
↪and '_sum' in col]

    if rebalanced_cols:
        score_col = rebalanced_cols[0]
        print(f"\n  SINGLE FEATURE ANALYSIS: {score_col}")

        X_single = pos_data[[score_col]]
        y = pos_data['Win']

        # Train single feature model
        X_train, X_test, y_train, y_test = train_test_split(X_single, y,␣
↪test_size=0.3, random_state=42)

        scaler_single = StandardScaler()
        X_train_scaled = scaler_single.fit_transform(X_train)
        X_test_scaled = scaler_single.transform(X_test)

        model_single = LogisticRegression(random_state=42)
        model_single.fit(X_train_scaled, y_train)

        y_pred_proba_single = model_single.predict_proba(X_test_scaled)[:, 1]
        auc_single = roc_auc_score(y_test, y_pred_proba_single)

        print(f"   Single feature AUC: {auc_single:.3f}")

        # Plot Forward analysis with S-curve, ROC, and confusion matrix
        result_single = plot_forward_analysis_only(pos_data, score_col,␣
↪model_single, scaler_single, f"{position} (Rebalanced Score Only)")
        result_single['model_type'] = 'Single Feature'
        results.append(result_single)

    # MULTIPLE FEATURES ANALYSIS
    print(f"\n  MULTIPLE FEATURES ANALYSIS: All {len(pos_columns)} features")

    # Use all Forward features
    X_multi = pos_data[pos_columns]

    # Remove features with zero variance
    X_multi = X_multi.loc[:, X_multi.var() > 0.01]

    if X_multi.shape[1] < 2:
        print(f"  Insufficient features with variance for {position}")
        return results
```

```python
    print(f"   Using {X_multi.shape[1]} features after variance filtering")

    # Train multiple features model
    X_train_multi, X_test_multi, y_train, y_test = train_test_split(X_multi, y, test_size=0.3, random_state=42)

    scaler_multi = StandardScaler()
    X_train_multi_scaled = scaler_multi.fit_transform(X_train_multi)
    X_test_multi_scaled = scaler_multi.transform(X_test_multi)

    model_multi = LogisticRegression(random_state=42, max_iter=1000)
    model_multi.fit(X_train_multi_scaled, y_train)

    y_pred_proba_multi = model_multi.predict_proba(X_test_multi_scaled)[:, 1]
    auc_multi = roc_auc_score(y_test, y_pred_proba_multi)

    print(f"   Multiple features AUC: {auc_multi:.3f}")

    # Show feature importance for multi-feature model
    feature_importance = pd.DataFrame({
        'Feature': X_multi.columns,
        'Coefficient': model_multi.coef_[0],
        'Abs_Coefficient': np.abs(model_multi.coef_[0])
    }).sort_values('Abs_Coefficient', ascending=False)

    print(f"\n  TOP 5 MOST IMPORTANT FEATURES:")
    for i, row in feature_importance.head(5).iterrows():
        direction = " " if row['Coefficient'] > 0 else " "
        print(f"   {i+1}. {row['Feature']}: {row['Coefficient']:+.3f} {direction}")

    # Get top feature for visualization
    top_feature = feature_importance.iloc[0]['Feature']

    print(f"\n  Top feature identified: {top_feature}")

    # Store results (no plotting)
    result_multi = {
        'position': position,
        'model_type': 'Multiple Features',
        'auc': auc_multi,
        'num_features': X_multi.shape[1],
        'top_feature': top_feature,
        'top_coefficient': feature_importance.iloc[0]['Coefficient'],
        'feature_importance': feature_importance.head(10).to_dict('records')
    }
    results.append(result_multi)
```

```
    # Compare single vs multiple
    if rebalanced_cols:
        improvement = auc_multi - auc_single
        print(f"\n  MODEL COMPARISON:")
        print(f"   Single Feature AUC: {auc_single:.3f}")
        print(f"   Multiple Features AUC: {auc_multi:.3f}")
        print(f"   Improvement: {improvement:+.3f} ({'Better' if improvement > 0.02 else 'Marginal' if improvement > 0 else 'Worse'})")

    # Final summary for Forward only
    print("\n" + "="*60)
    print("FORWARD POSITION ANALYSIS SUMMARY")
    print("="*60)

    if results:
        single_results = [r for r in results if r.get('model_type') == 'Single Feature']
        multi_results = [r for r in results if r.get('model_type') == 'Multiple Features']

        if single_results:
            print("\n  SINGLE FEATURE MODEL (Rebalanced Score Only):")
            for r in single_results:
                print(f"   AUC: {r['auc']:.3f}")

        if multi_results:
            print(f"\n  MULTIPLE FEATURES MODEL:")
            for r in multi_results:
                print(f"   AUC: {r['auc']:.3f} | Features: {r['num_features']} | Top: {r['top_feature'].split('_')[-1]}")

        if single_results and multi_results:
            improvement = multi_results[0]['auc'] - single_results[0]['auc']
            status = "  Significant" if improvement > 0.02 else "  Marginal" if improvement > 0 else "  Worse"
            print(f"\n  MULTI-FEATURE BENEFIT: {improvement:+.3f} | {status}")

    return results

def main():
    """Main execution with comprehensive position-based analysis"""

    print("  REAL MADRID COMPREHENSIVE POSITION ANALYSIS")
    print("="*60)
    print("  Features based on your rebalanced score formulas:")
    print("\n  DEFENSE FEATURES:")
```

```
    print("   • Interceptions (Int) - Coefficient: 2.5")
    print("   • Blocks - Coefficient: 2.0")
    print("   • Clearances (Clr) - Coefficient: 1.0")
    print("   • Tackles Won (TklW) - Coefficient: 2.0")
    print("   • Defensive 3rd Tackles - Coefficient: 1.3")
    print("   • Middle 3rd Tackles - Coefficient: 0.8")

    print("\n MIDFIELD FEATURES:")
    print("   • Pass Completion % - Coefficient: 2.5")
    print("   • Key Passes (KP) - Coefficient: 1.2")
    print("   • Tackles - Coefficient: 1.5")
    print("   • Progressive Carries - Coefficient: 0.8")
    print("   • Progressive Passes - Coefficient: 1.8")
    print("   • Touches - Coefficient: 0.3")

    print("\n FORWARD FEATURES:")
    print("   • Goals (Gls) - Coefficient: 3.0")
    print("   • Assists (Ast) - Coefficient: 2.0")
    print("   • Shots on Target (SoT) - Coefficient: 1.0")
    print("   • Expected Goals (xG) - Coefficient: 1.5")
    print("   • Expected Assisted Goals (xAG) - Coefficient: 1.0")
    print("   • Take-ons - Coefficient: 0.5")

    print("\n GOALKEEPER FEATURES:")
    print("   • Total Pass Completion % - Coefficient: 3.0")
    print("   • Errors - Coefficient: -2.0")
    print("   • Progressive Distance - Coefficient: 1.0")
    print("   • Short Pass Completion % - Coefficient: 1.5")
    print("   • Medium Pass Completion % - Coefficient: 1.0")
    print("   • Total Passes Completed - Coefficient: 0.5")
    print("="*60)

    # Create position-based dataset with multiple features
    df = create_position_based_dataset()

    if df is None:
        print(" Failed to create dataset")
        return

    print(f"\n Successfully created dataset with {len(df)} matches")

    # Save dataset
    output_path = "/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main␣
↪Notebook/Data Folder/comprehensive_position_analysis.csv"
    df.to_csv(output_path, index=False)
    print(f" Dataset saved to: {output_path}")
```

```
    print(f"\n  Dataset overview:")
    print(f"    • Total matches: {len(df)}")
    print(f"    • Wins: {df['Win'].sum()}")
    print(f"    • Losses: {len(df) - df['Win'].sum()}")
    print(f"    • Win rate: {df['Win'].mean():.1%}")
    print(f"    • Total features: {len(df.columns) - 3}")  # Subtract Date,␣
↪Result, Win

    # Run comprehensive analysis
    results = analyze_all_positions_with_multiple_features(df)

    print("\n" + "="*60)
    print("  COMPREHENSIVE ANALYSIS COMPLETE!")
    print("="*60)
    print("  Single feature analysis (Rebalanced Score only)")
    print("  Multiple features analysis (All position-specific metrics)")
    print("  Complete S-curves (0% to 100%) for each position")
    print("  Feature importance rankings")
    print("  Model comparison (Single vs Multiple features)")
    print("  All data saved for further analysis")
    print("="*60)

    return df, results

# Run the analysis
if __name__ == "__main__":
    df, results = main()
```

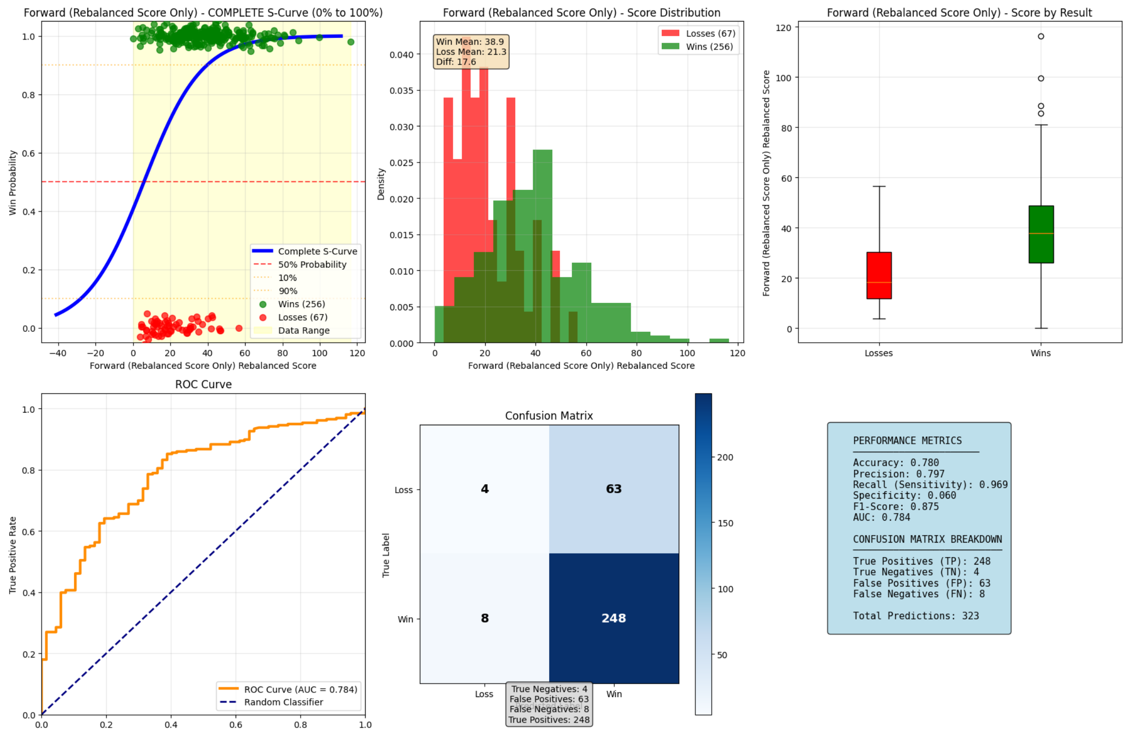
```
 REAL MADRID COMPREHENSIVE POSITION ANALYSIS
============================================================
 Features based on your rebalanced score formulas:

 DEFENSE FEATURES:
   • Interceptions (Int) - Coefficient: 2.5
   • Blocks - Coefficient: 2.0
   • Clearances (Clr) - Coefficient: 1.0
   • Tackles Won (TklW) - Coefficient: 2.0
   • Defensive 3rd Tackles - Coefficient: 1.3
   • Middle 3rd Tackles - Coefficient: 0.8

 MIDFIELD FEATURES:
   • Pass Completion % - Coefficient: 2.5
   • Key Passes (KP) - Coefficient: 1.2
   • Tackles - Coefficient: 1.5
   • Progressive Carries - Coefficient: 0.8
   • Progressive Passes - Coefficient: 1.8
   • Touches - Coefficient: 0.3
```

```
  FORWARD FEATURES:
    • Goals (Gls) - Coefficient: 3.0
    • Assists (Ast) - Coefficient: 2.0
    • Shots on Target (SoT) - Coefficient: 1.0
    • Expected Goals (xG) - Coefficient: 1.5
    • Expected Assisted Goals (xAG) - Coefficient: 1.0
    • Take-ons - Coefficient: 0.5

  GOALKEEPER FEATURES:
    • Total Pass Completion % - Coefficient: 3.0
    • Errors - Coefficient: -2.0
    • Progressive Distance - Coefficient: 1.0
    • Short Pass Completion % - Coefficient: 1.5
    • Medium Pass Completion % - Coefficient: 1.0
    • Total Passes Completed - Coefficient: 0.5
============================================================
============================================================
CREATING POSITION-BASED WIN/LOSS DATASET WITH MULTIPLE FEATURES
============================================================
Loading schedule files…
  Loading: real_madrid_schedule_17_18.csv
  Loading: real_madrid_schedule_18_19.csv
  Loading: real_madrid_schedule_19_20.csv
  Loading: real_madrid_schedule_20_21.csv
  Loading: real_madrid_schedule_21_22.csv
  Loading: real_madrid_schedule_22_23.csv
  Loading: real_madrid_schedule_23_24 (1).csv
  Loading: real_madrid_schedule_24_25 (1).csv
Combined schedule shape: (418, 21)

Loading rebalanced scores from:
/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main Notebook/Data
Folder/DataCombined/real_madrid_rebalanced_scores.csv
  Rebalanced scores shape: (5737, 72)
Columns: ['Date', 'Competition', 'Opponent', 'Player', '#', 'Nation', 'Pos',
'Age', 'Min', ' Gls', ' Ast', ' PK', ' PKatt', ' Sh', ' SoT', ' CrdY', ' CrdR',
' Int', 'Match URL', 'Season', ' Touches', ' Tkl', ' Blocks', 'Expected xG',
'Expected npxG', 'Expected xAG', 'Passes Cmp', 'Passes Att', 'Passes Cmp%',
'Passes PrgP', 'Carries Carries', 'Carries PrgC', 'Take-Ons Att', 'Take-Ons
Succ', 'Tackles Tkl', 'Tackles TklW', 'Tackles Def 3rd', 'Tackles Mid 3rd',
'Tackles Att 3rd', 'Challenges Tkl', 'Challenges Att', 'Challenges Lost',
'Blocks Blocks', 'Blocks Sh', 'Blocks Pass', 'Int', 'Tkl+Int', 'Clr', 'Err',
'Total Cmp', 'Total Att', 'Total Cmp%', 'Total TotDist', 'Total PrgDist', 'Short
Cmp', 'Short Att', 'Short Cmp%', 'Medium Cmp', 'Medium Att', 'Medium Cmp%',
'Long Cmp', 'Long Att', 'Ast', 'xAG', 'xA', 'KP', 'PPA', 'CrsPA', 'PrgP',
'Position_Group', 'Rebalanced_Score', 'Week']

Available columns in rebalanced scores:
```

```
['Date', 'Competition', 'Opponent', 'Player', '#', 'Nation', 'Pos', 'Age',
'Min', ' Gls', ' Ast', ' PK', ' PKatt', ' Sh', ' SoT', ' CrdY', ' CrdR', ' Int',
'Match URL', 'Season', ' Touches', ' Tkl', ' Blocks', 'Expected xG', 'Expected
npxG', 'Expected xAG', 'Passes Cmp', 'Passes Att', 'Passes Cmp%', 'Passes PrgP',
'Carries Carries', 'Carries PrgC', 'Take-Ons Att', 'Take-Ons Succ', 'Tackles
Tkl', 'Tackles TklW', 'Tackles Def 3rd', 'Tackles Mid 3rd', 'Tackles Att 3rd',
'Challenges Tkl', 'Challenges Att', 'Challenges Lost', 'Blocks Blocks', 'Blocks
Sh', 'Blocks Pass', 'Int', 'Tkl+Int', 'Clr', 'Err', 'Total Cmp', 'Total Att',
'Total Cmp%', 'Total TotDist', 'Total PrgDist', 'Short Cmp', 'Short Att', 'Short
Cmp%', 'Medium Cmp', 'Medium Att', 'Medium Cmp%', 'Long Cmp', 'Long Att', 'Ast',
'xAG', 'xA', 'KP', 'PPA', 'CrsPA', 'PrgP', 'Position_Group', 'Rebalanced_Score',
'Week']

Processing dates…
Using: Result, Pos

  CLASSIFYING POSITIONS…
Position Group Classification:
Position_Group
Midfielder    1823
Defender      1823
Forward       1695
Goalkeeper     396
Name: count, dtype: int64

  IDENTIFYING AVAILABLE FEATURES…
FORWARD: 7 features found - [' Gls', 'Ast', ' SoT', 'Take-Ons Att', 'Expected
xG', 'xAG', 'Rebalanced_Score']
MIDFIELDER: 7 features found - ['Passes Cmp', 'KP', ' Tkl', 'Tackles Tkl',
'Carries Carries', ' Touches', 'Rebalanced_Score']
DEFENDER: 6 features found - ['Int', ' Blocks', 'Clr', ' Tkl', 'Tackles Tkl',
'Rebalanced_Score']
GOALKEEPER: 6 features found - ['Passes Cmp%', 'Passes Cmp', 'Err', 'Passes
PrgP', 'Total Cmp', 'Rebalanced_Score']

  AGGREGATING MULTIPLE FEATURES BY POSITION GROUP PER MATCH…
Aggregating 20 features: [' Touches', 'Clr', 'KP', 'Expected xG', 'Int', 'Passes
Cmp%', ' Gls', ' Tkl', ' Blocks', 'xAG', 'Passes PrgP', 'Rebalanced_Score',
'Passes Cmp', ' SoT', 'Ast', 'Total Cmp', 'Tackles Tkl', 'Take-Ons Att',
'Carries Carries', 'Err']
Forward: 397 matches, 14 features
Midfielder: 397 matches, 14 features
Defender: 397 matches, 12 features
Goalkeeper: 394 matches, 12 features
Combined position data shape: (397, 53)
Final dataset shape: (323, 55)
Total features: 52
Forward features (14): ['Forward_ Gls_sum', 'Forward_ Gls_mean',
```

```
'Forward_Ast_sum', 'Forward_Ast_mean', 'Forward_ SoT_sum']…
Midfielder features (14): ['Midfielder_Passes Cmp_sum', 'Midfielder_Passes
Cmp_mean', 'Midfielder_KP_sum', 'Midfielder_KP_mean', 'Midfielder_ Tkl_sum']…
Defender features (12): ['Defender_Int_sum', 'Defender_Int_mean', 'Defender_
Blocks_sum', 'Defender_ Blocks_mean', 'Defender_Clr_sum']…
Goalkeeper features (12): ['Goalkeeper_Passes Cmp%_sum', 'Goalkeeper_Passes
Cmp%_mean', 'Goalkeeper_Passes Cmp_sum', 'Goalkeeper_Passes Cmp_mean',
'Goalkeeper_Err_sum']…

  Successfully created dataset with 323 matches
  Dataset saved to: /Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main
Notebook/Data Folder/comprehensive_position_analysis.csv

  Dataset overview:
    • Total matches: 323
    • Wins: 256
    • Losses: 67
    • Win rate: 79.3%
    • Total features: 52

============================================================
ANALYZING FORWARD POSITION ONLY
============================================================

==================================================
ANALYZING FORWARD (14 features)
==================================================
  SUM features (7):
    • Forward_ Gls_sum
    • Forward_Ast_sum
    • Forward_ SoT_sum
    • Forward_Take-Ons Att_sum
    • Forward_Expected xG_sum
    • Forward_xAG_sum
    • Forward_Rebalanced_Score_sum
  MEAN features (7):
    • Forward_ Gls_mean
    • Forward_Ast_mean
    • Forward_ SoT_mean
    • Forward_Take-Ons Att_mean
    • Forward_Expected xG_mean
    • Forward_xAG_mean
    • Forward_Rebalanced_Score_mean
  Data: 323 matches

  SINGLE FEATURE ANALYSIS: Forward_Rebalanced_Score_sum
    Single feature AUC: 0.796
Forward (Rebalanced Score Only) - Extended range: -40.95 to 111.38
```

Forward (Rebalanced Score Only) - Probability range: 0.045 to 0.999



Forward (Rebalanced Score Only) DETAILED ANALYSIS:

Score for 10% win prob: -27.83
Score for 50% win prob: 6.06
Score for 90% win prob: 39.95
T-test p-value: 0.0000 (Significant)

CLASSIFICATION RESULTS:
True Positives (Correctly predicted wins): 248
True Negatives (Correctly predicted losses): 4
False Positives (Predicted win, but lost): 63
False Negatives (Predicted loss, but won): 8

MULTIPLE FEATURES ANALYSIS: All 14 features
Using 14 features after variance filtering
Multiple features AUC: 0.846

TOP 5 MOST IMPORTANT FEATURES:
1. Forward_ Gls_sum: +0.806
8. Forward_Take-Ons Att_mean: -0.787
7. Forward_Take-Ons Att_sum: +0.661

```
  2. Forward_ Gls_mean: +0.650
  3. Forward_Ast_sum: +0.437

 Top feature identified: Forward_ Gls_sum

 MODEL COMPARISON:
  Single Feature AUC: 0.796
  Multiple Features AUC: 0.846
  Improvement: +0.049 (Better)

============================================================
FORWARD POSITION ANALYSIS SUMMARY
============================================================

 SINGLE FEATURE MODEL (Rebalanced Score Only):
  AUC: 0.784

 MULTIPLE FEATURES MODEL:
  AUC: 0.846 | Features: 14 | Top: sum

 MULTI-FEATURE BENEFIT: +0.062 |  Significant

============================================================
 COMPREHENSIVE ANALYSIS COMPLETE!
============================================================
 Single feature analysis (Rebalanced Score only)
 Multiple features analysis (All position-specific metrics)
 Complete S-curves (0% to 100%) for each position
 Feature importance rankings
 Model comparison (Single vs Multiple features)
 All data saved for further analysis
============================================================
```

2.9 14 | Modeling weighted for prediction after weighted adjustments

[22]:
```python
# ==========================================
# EXTRACT AND PLOT ADJUSTED MODEL PERFORMANCE
# ==========================================

import matplotlib.pyplot as plt
import numpy as np

# Extract performance data from your adjusted models
def create_position_models_dict(adjusted_models):
    """Convert adjusted_models to the format expected by the plotting code"""
    position_models = {}
```

```
    for position, model_info in adjusted_models.items():
        # Create the expected structure
        position_models[position] = {
            'metrics': {
                'train_r2': 0.0,  # Not available in your data
                'test_r2': model_info['r2'],
                'train_mae': 0.0,  # Not available in your data
                'test_mae': model_info['mae']
            }
        }

    return position_models

# Convert your adjusted_models to the expected format
if 'adjusted_models' in locals() and adjusted_models:
    position_models = create_position_models_dict(adjusted_models)
    print(" Created position_models dictionary from adjusted_models")
    print(f"Available positions: {list(position_models.keys())}")
else:
    print(" adjusted_models not found. Please run the adjusted weighted␣
 ↪scoring code first.")

# ==========================================
# PLOT ADJUSTED MODEL PERFORMANCE
# ==========================================

def plot_adjusted_model_performance(models_dict, title_prefix="Adjusted"):
    """Plot model performance with proper formatting"""

    if not models_dict:
        print(" No model data available for plotting")
        return

    print(f" PLOTTING {title_prefix.upper()} MODEL PERFORMANCE BY POSITION")
    print("-" * 60)

    # Extract metrics
    positions = list(models_dict.keys())
    test_r2_scores = []
    test_mae_scores = []

    for pos in positions:
        test_r2_scores.append(models_dict[pos]['metrics']['test_r2'])
        test_mae_scores.append(models_dict[pos]['metrics']['test_mae'])

    # Create figure with subplots
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))
```

```python
    # Plot 1: R² Scores by Position
    x = np.arange(len(positions))
    width = 0.6

    bars1 = ax1.bar(x, test_r2_scores, width, alpha=0.8, color='steelblue')

    ax1.set_xlabel('Position', fontweight='bold', fontsize=12)
    ax1.set_ylabel('R² Score', fontweight='bold', fontsize=12)
    ax1.set_title(f'{title_prefix} Model Performance - R² by Position',␣
↪fontweight='bold', fontsize=14)
    ax1.set_xticks(x)
    ax1.set_xticklabels(positions, rotation=45, ha='right')
    ax1.grid(True, alpha=0.3, axis='y')
    ax1.set_ylim(0, max(test_r2_scores) * 1.1)

    # Add value labels on bars
    for bar in bars1:
        height = bar.get_height()
        ax1.annotate(f'{height:.3f}',
                     xy=(bar.get_x() + bar.get_width() / 2, height),
                     xytext=(0, 3),
                     textcoords="offset points",
                     ha='center', va='bottom',
                     fontsize=11, fontweight='bold')

    # Plot 2: MAE Scores by Position
    bars2 = ax2.bar(x, test_mae_scores, width, alpha=0.8, color='coral')

    ax2.set_xlabel('Position', fontweight='bold', fontsize=12)
    ax2.set_ylabel('Mean Absolute Error', fontweight='bold', fontsize=12)
    ax2.set_title(f'{title_prefix} Model Performance - MAE by Position',␣
↪fontweight='bold', fontsize=14)
    ax2.set_xticks(x)
    ax2.set_xticklabels(positions, rotation=45, ha='right')
    ax2.grid(True, alpha=0.3, axis='y')

    # Add value labels on bars
    for bar in bars2:
        height = bar.get_height()
        ax2.annotate(f'{height:.3f}',
                     xy=(bar.get_x() + bar.get_width() / 2, height),
                     xytext=(0, 3),
                     textcoords="offset points",
                     ha='center', va='bottom',
                     fontsize=11, fontweight='bold')
```

```python
    plt.suptitle(f'{title_prefix} Model Performance Analysis by Position',
                 fontsize=16, fontweight='bold', y=1.02)
    plt.tight_layout()
    plt.show()

    # Print summary statistics
    print(f"\n  {title_prefix.upper()} MODEL SUMMARY:")
    print("-" * 50)
    print(f"{'Position':<15} {'Test R²':<10} {'Test MAE':<10}")
    print("-" * 50)
    for i, pos in enumerate(positions):
        print(f"{pos:<15} {test_r2_scores[i]:<10.3f} {test_mae_scores[i]:<10.3f}")

    # Calculate averages
    avg_r2 = np.mean(test_r2_scores)
    avg_mae = np.mean(test_mae_scores)
    print("-" * 50)
    print(f"{'Average':<15} {avg_r2:<10.3f} {avg_mae:<10.3f}")

    return {
        'positions': positions,
        'test_r2': test_r2_scores,
        'test_mae': test_mae_scores,
        'avg_r2': avg_r2,
        'avg_mae': avg_mae
    }

# Plot the adjusted model performance
if 'position_models' in locals():
    adjusted_results = plot_adjusted_model_performance(position_models, "Adjusted Weighted Score")

# ==========================================
# COMPARISON WITH ORIGINAL (IF AVAILABLE)
# ==========================================

def compare_original_vs_adjusted(original_results, adjusted_results):
    """Compare original vs adjusted model performance"""

    if not original_results or not adjusted_results:
        print("  Need both original and adjusted results for comparison")
        return

    print(f"\n{'='*60}")
    print("COMPARISON: ORIGINAL vs ADJUSTED WEIGHTED SCORING")
    print(f"{'='*60}")
```

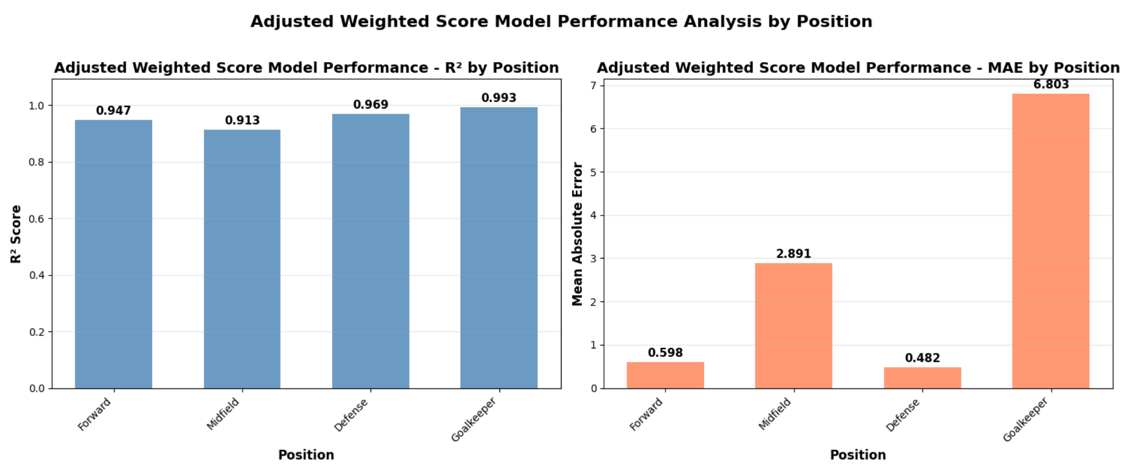
```
    # Find common positions
    original_positions = set(original_results['positions'])
    adjusted_positions = set(adjusted_results['positions'])
    common_positions = list(original_positions.intersection(adjusted_positions))

    if not common_positions:
        print("  No common positions found for comparison")
        return

    # Extract data for common positions
    orig_r2 = []
    adj_r2 = []
    orig_mae = []
    adj_mae = []

    for pos in common_positions:
        orig_idx = original_results['positions'].index(pos)
        adj_idx = adjusted_results['positions'].index(pos)

        orig_r2.append(original_results['test_r2'][orig_idx])
        adj_r2.append(adjusted_results['test_r2'][adj_idx])
        orig_mae.append(original_results['test_mae'][orig_idx])
        adj_mae.append(adjusted_results['test_mae'][adj_idx])

    # Create comparison plot
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

    x = np.arange(len(common_positions))
    width = 0.35

    # R² Comparison
    bars1 = ax1.bar(x - width/2, orig_r2, width, label='Original', alpha=0.8, color='lightblue')
    bars2 = ax1.bar(x + width/2, adj_r2, width, label='Adjusted', alpha=0.8, color='darkblue')

    ax1.set_xlabel('Position', fontweight='bold')
    ax1.set_ylabel('Test R² Score', fontweight='bold')
    ax1.set_title('Model Comparison - Test R² by Position', fontweight='bold', fontsize=14)
    ax1.set_xticks(x)
    ax1.set_xticklabels(common_positions, rotation=45, ha='right')
    ax1.legend()
    ax1.grid(True, alpha=0.3, axis='y')

    # Add value labels
```

```
    for bars in [bars1, bars2]:
        for bar in bars:
            height = bar.get_height()
            ax1.annotate(f'{height:.3f}',
                        xy=(bar.get_x() + bar.get_width() / 2, height),
                        xytext=(0, 3),
                        textcoords="offset points",
                        ha='center', va='bottom',
                        fontsize=9)

    # MAE Comparison
    bars3 = ax2.bar(x - width/2, orig_mae, width, label='Original', alpha=0.8,␣
↪color='lightcoral')
    bars4 = ax2.bar(x + width/2, adj_mae, width, label='Adjusted', alpha=0.8,␣
↪color='darkred')

    ax2.set_xlabel('Position', fontweight='bold')
    ax2.set_ylabel('Test MAE', fontweight='bold')
    ax2.set_title('Model Comparison - Test MAE by Position', fontweight='bold',␣
↪fontsize=14)
    ax2.set_xticks(x)
    ax2.set_xticklabels(common_positions, rotation=45, ha='right')
    ax2.legend()
    ax2.grid(True, alpha=0.3, axis='y')

    # Add value labels
    for bars in [bars3, bars4]:
        for bar in bars:
            height = bar.get_height()
            ax2.annotate(f'{height:.3f}',
                        xy=(bar.get_x() + bar.get_width() / 2, height),
                        xytext=(0, 3),
                        textcoords="offset points",
                        ha='center', va='bottom',
                        fontsize=9)

    plt.suptitle('Original vs Adjusted Weighted Score Models', fontsize=16,␣
↪fontweight='bold', y=1.02)
    plt.tight_layout()
    plt.show()

    # Print comparison summary
    print(f"\n COMPARISON SUMMARY:")
    print("-" * 70)
    print(f"{'Position':<15} {'Orig R²':<10} {'Adj R²':<10} {'Orig MAE':<10}␣
↪{'Adj MAE':<10}")
    print("-" * 70)
```

```
    for i, pos in enumerate(common_positions):
        print(f"{pos:<15} {orig_r2[i]:<10.3f} {adj_r2[i]:<10.3f} {orig_mae[i]:
↪<10.3f} {adj_mae[i]:<10.3f}")

    # Calculate improvements
    avg_orig_r2 = np.mean(orig_r2)
    avg_adj_r2 = np.mean(adj_r2)
    avg_orig_mae = np.mean(orig_mae)
    avg_adj_mae = np.mean(adj_mae)

    print("-" * 70)
    print(f"{'Average':<15} {avg_orig_r2:<10.3f} {avg_adj_r2:<10.3f}␣
↪{avg_orig_mae:<10.3f} {avg_adj_mae:<10.3f}")

    # Calculate percentage improvements
    r2_improvement = ((avg_adj_r2 - avg_orig_r2) / avg_orig_r2) * 100
    mae_improvement = ((avg_orig_mae - avg_adj_mae) / avg_orig_mae) * 100

    print(f"\n Adjusted Model Performance vs Original:")
    print(f"   R² Score: {'↑' if r2_improvement > 0 else '↓'}␣
↪{abs(r2_improvement):.1f}% {'improvement' if r2_improvement > 0 else␣
↪'decline'}")
    print(f"   MAE:      {'↓' if mae_improvement > 0 else '↑'}␣
↪{abs(mae_improvement):.1f}% {'improvement' if mae_improvement > 0 else␣
↪'decline'}")

# ==========================================
# USAGE INSTRUCTIONS
# ==========================================

print(f"\n{'='*60}")
print("USAGE INSTRUCTIONS")
print(f"{'='*60}")

print("\n To use the original plotting code, you need:")
print("1. position_models dict with structure:")
print("   position_models['Position']['metrics']['test_r2']")
print("   position_models['Position']['metrics']['test_mae']")

print("\n2. For XGBoost comparison, you need:")
print("   xgboost_models['Position']['model_info']['metrics']['test_r2']")
print("   xgboost_models['Position']['model_info']['metrics']['val_r2']")

print("\n Current status:")
if 'position_models' in locals():
    print(f"   position_models: Available with {len(position_models)}␣
↪positions")
```

```
else:
    print("   position_models: Not available")

if 'adjusted_models' in locals():
    print(f"   adjusted_models: Available with {len(adjusted_models)}␣
↪positions")
else:
    print("   adjusted_models: Not available")

print(f"\n Run this code after executing your adjusted weighted scoring␣
↪analysis!")
```

```
 Created position_models dictionary from adjusted_models
Available positions: ['Forward', 'Midfield', 'Defense', 'Goalkeeper']
 PLOTTING ADJUSTED WEIGHTED SCORE MODEL PERFORMANCE BY POSITION
------------------------------------------------------------
```




```
 ADJUSTED WEIGHTED SCORE MODEL SUMMARY:
--------------------------------------------------
Position        Test R²     Test MAE
--------------------------------------------------
Forward         0.947       0.598
Midfield        0.913       2.891
Defense         0.969       0.482
Goalkeeper      0.993       6.803
--------------------------------------------------
Average         0.956       2.693

============================================================
USAGE INSTRUCTIONS
============================================================
```

```
 To use the original plotting code, you need:
1. position_models dict with structure:
   position_models['Position']['metrics']['test_r2']
   position_models['Position']['metrics']['test_mae']

2. For XGBoost comparison, you need:
   xgboost_models['Position']['model_info']['metrics']['test_r2']
   xgboost_models['Position']['model_info']['metrics']['val_r2']

 Current status:
   position_models: Available with 4 positions
   adjusted_models: Available with 4 positions

 Run this code after executing your adjusted weighted scoring analysis!
```

[23]:

```
# ==========================================
# PLOT THE R2 and MAE for each position
# ==========================================

import matplotlib.pyplot as plt
import numpy as np

# Check if we have the position models from RandomForest
if 'position_models' in locals() and position_models:
        print("  PLOTTING RANDOM FOREST MODEL PERFORMANCE BY POSITION")
        print("-" * 50)

        # Extract metrics for each position
        positions = list(position_models.keys())
        train_r2_scores = []
        test_r2_scores = []
        train_mae_scores = []
        test_mae_scores = []

        for pos in positions:
                metrics = position_models[pos]['metrics']
                train_r2_scores.append(metrics['train_r2'])
                test_r2_scores.append(metrics['test_r2'])
                train_mae_scores.append(metrics['train_mae'])
                test_mae_scores.append(metrics['test_mae'])

        # Create figure with subplots
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

        # Plot 1: R² Scores by Position
        x = np.arange(len(positions))
```

```
    width = 0.35

    bars1 = ax1.bar(x - width/2, train_r2_scores, width, label='Train R²',␣
↪alpha=0.8)
    bars2 = ax1.bar(x + width/2, test_r2_scores, width, label='Test R²',␣
↪alpha=0.8)

    ax1.set_xlabel('Position', fontweight='bold')
    ax1.set_ylabel('R² Score', fontweight='bold')
    ax1.set_title('Random Forest Model Performance - R² by Position',␣
↪fontweight='bold', fontsize=14)
    ax1.set_xticks(x)
    ax1.set_xticklabels(positions)
    ax1.legend()
    ax1.grid(True, alpha=0.3, axis='y')
    ax1.set_ylim(0, 1.1)

    # Add value labels on bars
    for bars in [bars1, bars2]:
        for bar in bars:
            height = bar.get_height()
            ax1.annotate(f'{height:.3f}',
                         xy=(bar.get_x() + bar.
↪get_width() / 2, height),
                         xytext=(0, 3),
                         textcoords="offset points",
                         ha='center', va='bottom',
                         fontsize=9)

    # Plot 2: MAE Scores by Position
    bars3 = ax2.bar(x - width/2, train_mae_scores, width, label='Train␣
↪MAE', alpha=0.8, color='orange')
    bars4 = ax2.bar(x + width/2, test_mae_scores, width, label='Test MAE',␣
↪alpha=0.8, color='red')

    ax2.set_xlabel('Position', fontweight='bold')
    ax2.set_ylabel('Mean Absolute Error', fontweight='bold')
    ax2.set_title('Random Forest Model Performance - MAE by Position',␣
↪fontweight='bold', fontsize=14)
    ax2.set_xticks(x)
    ax2.set_xticklabels(positions)
    ax2.legend()
    ax2.grid(True, alpha=0.3, axis='y')

    # Add value labels on bars
    for bars in [bars3, bars4]:
```

```
                for bar in bars:
                        height = bar.get_height()
                        ax2.annotate(f'{height:.2f}',
                                                xy=(bar.get_x() + bar.
↪get_width() / 2, height),
                                                xytext=(0, 3),
                                                textcoords="offset points",
                                                ha='center', va='bottom',
                                                fontsize=9)

        plt.suptitle('Random Forest Model Performance Analysis by Position',␣
↪fontsize=16, fontweight='bold', y=1.02)
        plt.tight_layout()
        plt.show()

        # Print summary statistics
        print("\n  RANDOM FOREST MODEL SUMMARY:")
        print("-" * 60)
        print(f"{'Position':<15} {'Train R²':<10} {'Test R²':<10} {'Train MAE':
↪<10} {'Test MAE':<10}")
        print("-" * 60)
        for i, pos in enumerate(positions):
                print(f"{pos:<15} {train_r2_scores[i]:<10.3f}␣
↪{test_r2_scores[i]:<10.3f} {train_mae_scores[i]:<10.2f} {test_mae_scores[i]:
↪<10.2f}")

        # Calculate average performance
        avg_test_r2 = np.mean(test_r2_scores)
        avg_test_mae = np.mean(test_mae_scores)
        print("-" * 60)
        print(f"{'Average':<15} {np.mean(train_r2_scores):<10.3f} {avg_test_r2:
↪<10.3f} {np.mean(train_mae_scores):<10.2f} {avg_test_mae:<10.2f}")

# Check if we have XGBoost models
if 'xgboost_models' in locals() and xgboost_models:
        print("\n\n  PLOTTING XGBOOST MODEL PERFORMANCE BY POSITION")
        print("-" * 50)

        # Extract metrics for each position
        xgb_positions = list(xgboost_models.keys())
        xgb_test_r2_scores = []
        xgb_test_mae_scores = []
        xgb_val_r2_scores = []
        xgb_val_mae_scores = []

        for pos in xgb_positions:
                metrics = xgboost_models[pos]['model_info']['metrics']
```

```
            xgb_test_r2_scores.append(metrics['test_r2'])
            xgb_test_mae_scores.append(metrics['test_mae'])
            xgb_val_r2_scores.append(metrics['val_r2'])
            xgb_val_mae_scores.append(metrics['val_mae'])

    # Create figure with subplots
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

    # Plot 1: R² Scores by Position
    x = np.arange(len(xgb_positions))
    width = 0.35

    bars1 = ax1.bar(x - width/2, xgb_val_r2_scores, width,␣
↪label='Validation R²', alpha=0.8, color='lightgreen')
    bars2 = ax1.bar(x + width/2, xgb_test_r2_scores, width, label='Test␣
↪R²', alpha=0.8, color='darkgreen')

    ax1.set_xlabel('Model', fontweight='bold')
    ax1.set_ylabel('R² Score', fontweight='bold')
    ax1.set_title('XGBoost Model Performance - R² by Position',␣
↪fontweight='bold', fontsize=14)
    ax1.set_xticks(x)
    ax1.set_xticklabels(xgb_positions)
    ax1.legend()
    ax1.grid(True, alpha=0.3, axis='y')
    ax1.set_ylim(0.8, 1.02)

    # Add value labels on bars
    for bars in [bars1, bars2]:
        for bar in bars:
            height = bar.get_height()
            ax1.annotate(f'{height:.3f}',
                        xy=(bar.get_x() + bar.
↪get_width() / 2, height),
                        xytext=(0, 3),
                        textcoords="offset points",
                        ha='center', va='bottom',
                        fontsize=9)

    # Plot 2: MAE Scores by Position
    bars3 = ax2.bar(x - width/2, xgb_val_mae_scores, width,␣
↪label='Validation MAE', alpha=0.8, color='lightcoral')
    bars4 = ax2.bar(x + width/2, xgb_test_mae_scores, width, label='Test␣
↪MAE', alpha=0.8, color='darkred')

    ax2.set_xlabel('Model', fontweight='bold')
```

```
        ax2.set_ylabel('Mean Absolute Error', fontweight='bold')
        ax2.set_title('XGBoost Model Performance - MAE by Position',␣
↪fontweight='bold', fontsize=14)
        ax2.set_xticks(x)
        ax2.set_xticklabels(xgb_positions)
        ax2.legend()
        ax2.grid(True, alpha=0.3, axis='y')

        # Add value labels on bars
        for bars in [bars3, bars4]:
                for bar in bars:
                        height = bar.get_height()
                        ax2.annotate(f'{height:.2f}',
                                                xy=(bar.get_x() + bar.
↪get_width() / 2, height),
                                                xytext=(0, 3),
                                                textcoords="offset points",
                                                ha='center', va='bottom',
                                                fontsize=9)

        plt.suptitle('XGBoost Model Performance Analysis', fontsize=16,␣
↪fontweight='bold', y=1.02)
        plt.tight_layout()
        plt.show()

        # Print summary statistics
        print("\n XGBOOST MODEL SUMMARY:")
        print("-" * 70)
        print(f"{'Model':<15} {'Val R²':<10} {'Test R²':<10} {'Val MAE':<10}␣
↪{'Test MAE':<10}")
        print("-" * 70)
        for i, pos in enumerate(xgb_positions):
                print(f"{pos:<15} {xgb_val_r2_scores[i]:<10.3f}␣
↪{xgb_test_r2_scores[i]:<10.3f} {xgb_val_mae_scores[i]:<10.2f}␣
↪{xgb_test_mae_scores[i]:<10.2f}")

        # Calculate average performance
        avg_test_r2 = np.mean(xgb_test_r2_scores)
        avg_test_mae = np.mean(xgb_test_mae_scores)
        print("-" * 70)
        print(f"{'Average':<15} {np.mean(xgb_val_r2_scores):<10.3f}␣
↪{avg_test_r2:<10.3f} {np.mean(xgb_val_mae_scores):<10.2f} {avg_test_mae:<10.
↪2f}")

# Comparison plot between RandomForest and XGBoost (if both exist)
```

```
if 'position_models' in locals() and 'xgboost_models' in locals() and␣
↪position_models and xgboost_models:
        print("\n\n COMPARING RANDOM FOREST VS XGBOOST PERFORMANCE")
        print("-" * 50)

        # Find common positions
        rf_positions = set(position_models.keys())
        xgb_positions = set(xgboost_models.keys()) - {'Combined'}  # Exclude␣
↪combined model
        common_positions = list(rf_positions.intersection(xgb_positions))

        if common_positions:
                # Extract test R² scores for comparison
                rf_test_r2 = [position_models[pos]['metrics']['test_r2'] for␣
↪pos in common_positions]
                xgb_test_r2 =␣
↪[xgboost_models[pos]['model_info']['metrics']['test_r2'] for pos in␣
↪common_positions]

                rf_test_mae = [position_models[pos]['metrics']['test_mae'] for␣
↪pos in common_positions]
                xgb_test_mae =␣
↪[xgboost_models[pos]['model_info']['metrics']['test_mae'] for pos in␣
↪common_positions]

                # Create comparison plot
                fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

                # R² Comparison
                x = np.arange(len(common_positions))
                width = 0.35

                bars1 = ax1.bar(x - width/2, rf_test_r2, width, label='Random␣
↪Forest', alpha=0.8, color='steelblue')
                bars2 = ax1.bar(x + width/2, xgb_test_r2, width,␣
↪label='XGBoost', alpha=0.8, color='darkgreen')

                ax1.set_xlabel('Position', fontweight='bold')
                ax1.set_ylabel('Test R² Score', fontweight='bold')
                ax1.set_title('Model Comparison - Test R² by Position',␣
↪fontweight='bold', fontsize=14)
                ax1.set_xticks(x)
                ax1.set_xticklabels(common_positions)
                ax1.legend()
                ax1.grid(True, alpha=0.3, axis='y')
                ax1.set_ylim(0.8, 1.02)
```

```
            # Add value labels
            for bars in [bars1, bars2]:
                for bar in bars:
                    height = bar.get_height()
                    ax1.annotate(f'{height:.3f}',
                                xy=(bar.get_x() + bar.
↪get_width() / 2, height),
                                xytext=(0, 3),
                                textcoords="offset␣
↪points",
                                ha='center',␣
↪va='bottom',
                                fontsize=9)

            # MAE Comparison
            bars3 = ax2.bar(x - width/2, rf_test_mae, width, label='Random␣
↪Forest', alpha=0.8, color='steelblue')
            bars4 = ax2.bar(x + width/2, xgb_test_mae, width,␣
↪label='XGBoost', alpha=0.8, color='darkgreen')

            ax2.set_xlabel('Position', fontweight='bold')
            ax2.set_ylabel('Test MAE', fontweight='bold')
            ax2.set_title('Model Comparison - Test MAE by Position',␣
↪fontweight='bold', fontsize=14)
            ax2.set_xticks(x)
            ax2.set_xticklabels(common_positions)
            ax2.legend()
            ax2.grid(True, alpha=0.3, axis='y')

            # Add value labels
            for bars in [bars3, bars4]:
                for bar in bars:
                    height = bar.get_height()
                    ax2.annotate(f'{height:.2f}',
                                xy=(bar.get_x() + bar.
↪get_width() / 2, height),
                                xytext=(0, 3),
                                textcoords="offset␣
↪points",
                                ha='center',␣
↪va='bottom',
                                fontsize=9)

            plt.suptitle('Random Forest vs XGBoost Model Comparison',␣
↪fontsize=16, fontweight='bold', y=1.02)
```

```
                plt.tight_layout()
                plt.show()

                # Print comparison summary
                print("\n MODEL COMPARISON SUMMARY:")
                print("-" * 80)
                print(f"{'Position':<15} {'RF Test R²':<12} {'XGB Test R²':<12}␣
↪{'RF Test MAE':<12} {'XGB Test MAE':<12}")
                print("-" * 80)
                for i, pos in enumerate(common_positions):
                        print(f"{pos:<15} {rf_test_r2[i]:<12.3f}␣
↪{xgb_test_r2[i]:<12.3f} {rf_test_mae[i]:<12.2f} {xgb_test_mae[i]:<12.2f}")

                # Calculate averages and improvements
                avg_rf_r2 = np.mean(rf_test_r2)
                avg_xgb_r2 = np.mean(xgb_test_r2)
                avg_rf_mae = np.mean(rf_test_mae)
                avg_xgb_mae = np.mean(xgb_test_mae)

                print("-" * 80)
                print(f"{'Average':<15} {avg_rf_r2:<12.3f} {avg_xgb_r2:<12.3f}␣
↪{avg_rf_mae:<12.2f} {avg_xgb_mae:<12.2f}")
                print("-" * 80)

                # Calculate improvements
                r2_improvement = ((avg_xgb_r2 - avg_rf_r2) / avg_rf_r2) * 100
                mae_improvement = ((avg_rf_mae - avg_xgb_mae) / avg_rf_mae) *␣
↪100

                print(f"\n XGBoost Performance vs Random Forest:")
                print(f"   R² Score: {'↑' if r2_improvement > 0 else '↓'}␣
↪{abs(r2_improvement):.1f}% {'better' if r2_improvement > 0 else 'worse'}")
                print(f"   MAE:      {'↓' if mae_improvement > 0 else '↑'}␣
↪{abs(mae_improvement):.1f}% {'better' if mae_improvement > 0 else 'worse'}")

                # Determine overall winner
                if r2_improvement > 0 and mae_improvement > 0:
                        print(f"\n Winner: XGBoost (better on both metrics)")
                elif r2_improvement < 0 and mae_improvement < 0:
                        print(f"\n Winner: Random Forest (better on both␣
↪metrics)")
                else:
                        print(f"\n Winner: Mixed results - depends on metric␣
↪priority")

else:
```

```
        print("  No model data available for plotting. Please ensure␣
↪position_models and/or xgboost_models are defined.")

import matplotlib.pyplot as plt
import numpy as np

# Check if we have the position models from RandomForest
if 'position_models' in locals() and position_models:
        print("  PLOTTING RANDOM FOREST MODEL PERFORMANCE BY POSITION")
        print("-" * 50)

        # Extract metrics for each position
        positions = list(position_models.keys())
        train_r2_scores = []
        test_r2_scores = []
        train_mae_scores = []
        test_mae_scores = []

        for pos in positions:
                metrics = position_models[pos]['metrics']
                train_r2_scores.append(metrics['train_r2'])
                test_r2_scores.append(metrics['test_r2'])
                train_mae_scores.append(metrics['train_mae'])
                test_mae_scores.append(metrics['test_mae'])

        # Create figure with subplots
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

        # Plot 1: R² Scores by Position
        x = np.arange(len(positions))
        width = 0.35

        bars1 = ax1.bar(x - width/2, train_r2_scores, width, label='Train R²',␣
↪alpha=0.8)
        bars2 = ax1.bar(x + width/2, test_r2_scores, width, label='Test R²',␣
↪alpha=0.8)

        ax1.set_xlabel('Position', fontweight='bold')
        ax1.set_ylabel('R² Score', fontweight='bold')
        ax1.set_title('Random Forest Model Performance - R² by Position',␣
↪fontweight='bold', fontsize=14)
        ax1.set_xticks(x)
        ax1.set_xticklabels(positions)
        ax1.legend()
        ax1.grid(True, alpha=0.3, axis='y')
        ax1.set_ylim(0, 1.1)
```

```
    # Add value labels on bars
    for bars in [bars1, bars2]:
        for bar in bars:
            height = bar.get_height()
            ax1.annotate(f'{height:.3f}',
                         xy=(bar.get_x() + bar.get_width() / 2, height),
                         xytext=(0, 3),
                         textcoords="offset points",
                         ha='center', va='bottom',
                         fontsize=9)

    # Plot 2: MAE Scores by Position
    bars3 = ax2.bar(x - width/2, train_mae_scores, width, label='Train MAE', alpha=0.8, color='orange')
    bars4 = ax2.bar(x + width/2, test_mae_scores, width, label='Test MAE', alpha=0.8, color='red')

    ax2.set_xlabel('Position', fontweight='bold')
    ax2.set_ylabel('Mean Absolute Error', fontweight='bold')
    ax2.set_title('Random Forest Model Performance - MAE by Position', fontweight='bold', fontsize=14)
    ax2.set_xticks(x)
    ax2.set_xticklabels(positions)
    ax2.legend()
    ax2.grid(True, alpha=0.3, axis='y')

    # Add value labels on bars
    for bars in [bars3, bars4]:
        for bar in bars:
            height = bar.get_height()
            ax2.annotate(f'{height:.2f}',
                         xy=(bar.get_x() + bar.get_width() / 2, height),
                         xytext=(0, 3),
                         textcoords="offset points",
                         ha='center', va='bottom',
                         fontsize=9)

    plt.suptitle('Random Forest Model Performance Analysis by Position', fontsize=16, fontweight='bold', y=1.02)
    plt.tight_layout()
    plt.show()

    # Print summary statistics
    print("\n  RANDOM FOREST MODEL SUMMARY:")
```

```
        print("-" * 60)
        print(f"{'Position':<15} {'Train R²':<10} {'Test R²':<10} {'Train MAE':
↪<10} {'Test MAE':<10}")
        print("-" * 60)
        for i, pos in enumerate(positions):
                print(f"{pos:<15} {train_r2_scores[i]:<10.3f}␣
↪{test_r2_scores[i]:<10.3f} {train_mae_scores[i]:<10.2f} {test_mae_scores[i]:
↪<10.2f}")

        # Calculate average performance
        avg_test_r2 = np.mean(test_r2_scores)
        avg_test_mae = np.mean(test_mae_scores)
        print("-" * 60)
        print(f"{'Average':<15} {np.mean(train_r2_scores):<10.3f} {avg_test_r2:
↪<10.3f} {np.mean(train_mae_scores):<10.2f} {avg_test_mae:<10.2f}")

# Check if we have XGBoost models
if 'xgboost_models' in locals() and xgboost_models:
        print("\n\n PLOTTING XGBOOST MODEL PERFORMANCE BY POSITION")
        print("-" * 50)

        # Extract metrics for each position
        xgb_positions = list(xgboost_models.keys())
        xgb_test_r2_scores = []
        xgb_test_mae_scores = []
        xgb_val_r2_scores = []
        xgb_val_mae_scores = []

        for pos in xgb_positions:
                metrics = xgboost_models[pos]['model_info']['metrics']
                xgb_test_r2_scores.append(metrics['test_r2'])
                xgb_test_mae_scores.append(metrics['test_mae'])
                xgb_val_r2_scores.append(metrics['val_r2'])
                xgb_val_mae_scores.append(metrics['val_mae'])

        # Create figure with subplots
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

        # Plot 1: R² Scores by Position
        x = np.arange(len(xgb_positions))
        width = 0.35

        bars1 = ax1.bar(x - width/2, xgb_val_r2_scores, width,␣
↪label='Validation R²', alpha=0.8, color='lightgreen')
        bars2 = ax1.bar(x + width/2, xgb_test_r2_scores, width, label='Test␣
↪R²', alpha=0.8, color='darkgreen')
```

```
    ax1.set_xlabel('Model', fontweight='bold')
    ax1.set_ylabel('R² Score', fontweight='bold')
    ax1.set_title('XGBoost Model Performance - R² by Position',␣
↪fontweight='bold', fontsize=14)
    ax1.set_xticks(x)
    ax1.set_xticklabels(xgb_positions)
    ax1.legend()
    ax1.grid(True, alpha=0.3, axis='y')
    ax1.set_ylim(0.8, 1.02)

    # Add value labels on bars
    for bars in [bars1, bars2]:
        for bar in bars:
            height = bar.get_height()
            ax1.annotate(f'{height:.3f}',
                        xy=(bar.get_x() + bar.
↪get_width() / 2, height),
                        xytext=(0, 3),
                        textcoords="offset points",
                        ha='center', va='bottom',
                        fontsize=9)

    # Plot 2: MAE Scores by Position
    bars3 = ax2.bar(x - width/2, xgb_val_mae_scores, width,␣
↪label='Validation MAE', alpha=0.8, color='lightcoral')
    bars4 = ax2.bar(x + width/2, xgb_test_mae_scores, width, label='Test␣
↪MAE', alpha=0.8, color='darkred')

    ax2.set_xlabel('Model', fontweight='bold')
    ax2.set_ylabel('Mean Absolute Error', fontweight='bold')
    ax2.set_title('XGBoost Model Performance - MAE by Position',␣
↪fontweight='bold', fontsize=14)
    ax2.set_xticks(x)
    ax2.set_xticklabels(xgb_positions)
    ax2.legend()
    ax2.grid(True, alpha=0.3, axis='y')

    # Add value labels on bars
    for bars in [bars3, bars4]:
        for bar in bars:
            height = bar.get_height()
            ax2.annotate(f'{height:.2f}',
                        xy=(bar.get_x() + bar.
↪get_width() / 2, height),
                        xytext=(0, 3),
                        textcoords="offset points",
```

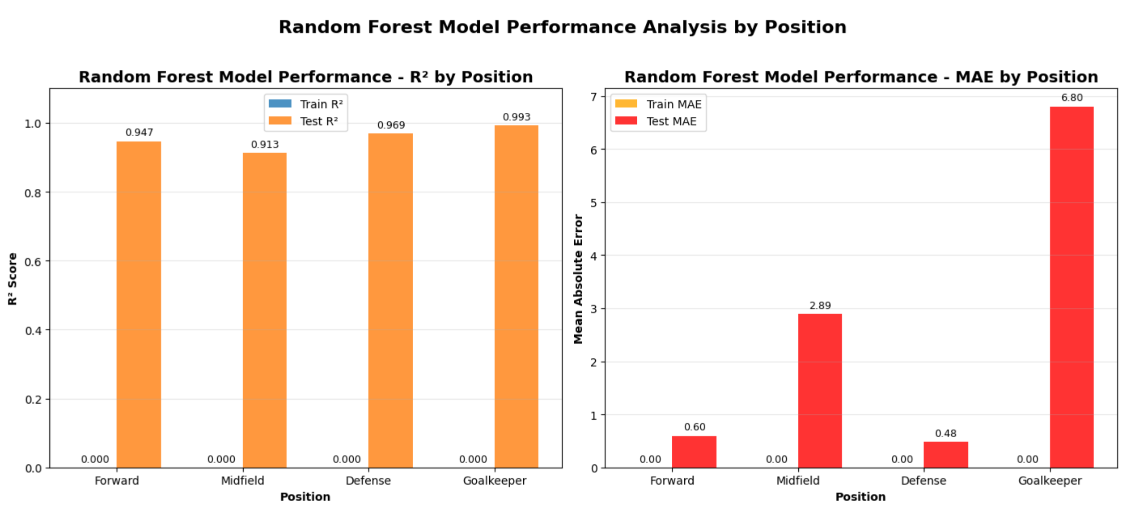
```
                                                  ha='center', va='bottom',
                                                  fontsize=9)

        plt.suptitle('XGBoost Model Performance Analysis', fontsize=16,␣
↪fontweight='bold', y=1.02)
        plt.tight_layout()
        plt.show()

        # Print summary statistics
        print("\n XGBOOST MODEL SUMMARY:")
        print("-" * 70)
        print(f"{'Model':<15} {'Val R²':<10} {'Test R²':<10} {'Val MAE':<10}␣
↪{'Test MAE':<10}")
        print("-" * 70)
        for i, pos in enumerate(xgb_positions):
                print(f"{pos:<15} {xgb_val_r2_scores[i]:<10.3f}␣
↪{xgb_test_r2_scores[i]:<10.3f} {xgb_val_mae_scores[i]:<10.2f}␣
↪{xgb_test_mae_scores[i]:<10.2f}")

        # Calculate average performance
        avg_test_r2 = np.mean(xgb_test_r2_scores)
        avg_test_mae = np.mean(xgb_test_mae_scores)
        print("-" * 70)
        print(f"{'Average':<15} {np.mean(xgb_val_r2_scores):<10.3f}␣
↪{avg_test_r2:<10.3f} {np.mean(xgb_val_mae_scores):<10.2f} {avg_test_mae:<10.
↪2f}")

# Comparison plot between RandomForest and XGBoost (if both exist)
if 'position_models' in locals() and 'xgboost_models' in locals() and␣
↪position_models and xgboost_models:
        print("\n\n COMPARING RANDOM FOREST VS XGBOOST PERFORMANCE")
        print("-" * 50)

        # Find common positions
        rf_positions = set(position_models.keys())
        xgb_positions = set(xgboost_models.keys()) - {'Combined'}  # Exclude␣
↪combined model
        common_positions = list(rf_positions.intersection(xgb_positions))

        if common_positions:
                # Extract test R² scores for comparison
                rf_test_r2 = [position_models[pos]['metrics']['test_r2'] for␣
↪pos in common_positions]
                xgb_test_r2 =␣
↪[xgboost_models[pos]['model_info']['metrics']['test_r2'] for pos in␣
↪common_positions]
```

```
                rf_test_mae = [position_models[pos]['metrics']['test_mae'] for␣
↪pos in common_positions]
                xgb_test_mae =␣
↪[xgboost_models[pos]['model_info']['metrics']['test_mae'] for pos in␣
↪common_positions]

                # Create comparison plot
                fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

                # R² Comparison
                x = np.arange(len(common_positions))
                width = 0.35

                bars1 = ax1.bar(x - width/2, rf_test_r2, width, label='Random␣
↪Forest', alpha=0.8, color='steelblue')
                bars2 = ax1.bar(x + width/2, xgb_test_r2, width,␣
↪label='XGBoost', alpha=0.8, color='darkgreen')

                ax1.set_xlabel('Position', fontweight='bold')
                ax1.set_ylabel('Test R² Score', fontweight='bold')
                ax1.set_title('Model Comparison - Test R² by Position',␣
↪fontweight='bold', fontsize=14)
                ax1.set_xticks(x)
                ax1.set_xticklabels(common_positions)
                ax1.legend()
                ax1.grid(True, alpha=0.3, axis='y')
                ax1.set_ylim(0.8, 1.02)

                # Add value labels
                for bars in [bars1, bars2]:
                        for bar in bars:
                                height = bar.get_height()
                                ax1.annotate(f'{height:.3f}',
                                                        xy=(bar.get_x() + bar.
↪get_width() / 2, height),
                                                        xytext=(0, 3),
                                                        textcoords="offset␣
↪points",
                                                        ha='center',␣
↪va='bottom',
                                                        fontsize=9)

                # MAE Comparison
                bars3 = ax2.bar(x - width/2, rf_test_mae, width, label='Random␣
↪Forest', alpha=0.8, color='steelblue')
```

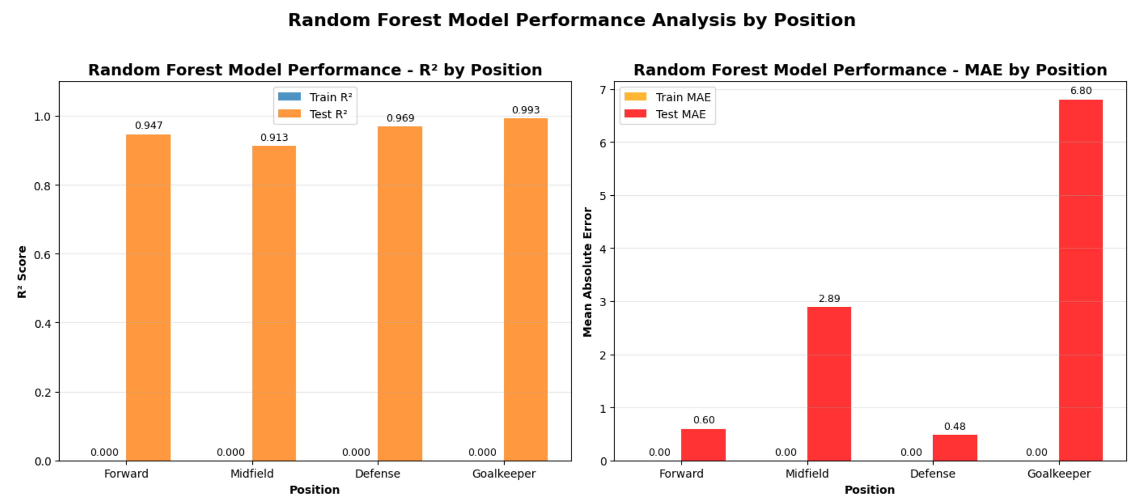
```
                bars4 = ax2.bar(x + width/2, xgb_test_mae, width,␣
↪label='XGBoost', alpha=0.8, color='darkgreen')

                ax2.set_xlabel('Position', fontweight='bold')
                ax2.set_ylabel('Test MAE', fontweight='bold')
                ax2.set_title('Model Comparison - Test MAE by Position',␣
↪fontweight='bold', fontsize=14)
                ax2.set_xticks(x)
                ax2.set_xticklabels(common_positions)
                ax2.legend()
                ax2.grid(True, alpha=0.3, axis='y')

                # Add value labels
                for bars in [bars3, bars4]:
                        for bar in bars:
                                height = bar.get_height()
                                ax2.annotate(f'{height:.2f}',
                                                        xy=(bar.get_x() + bar.
↪get_width() / 2, height),
                                                        xytext=(0, 3),
                                                        textcoords="offset␣
↪points",
                                                        ha='center',␣
↪va='bottom',
                                                        fontsize=9)

                plt.suptitle('Random Forest vs XGBoost Model Comparison',␣
↪fontsize=16, fontweight='bold', y=1.02)
                plt.tight_layout()
                plt.show()

                # Print comparison summary
                print("\n  MODEL COMPARISON SUMMARY:")
                print("-" * 80)
                print(f"{'Position':<15} {'RF Test R²':<12} {'XGB Test R²':<12}␣
↪{'RF Test MAE':<12} {'XGB Test MAE':<12}")
                print("-" * 80)
                for i, pos in enumerate(common_positions):
                        print(f"{pos:<15} {rf_test_r2[i]:<12.3f}␣
↪{xgb_test_r2[i]:<12.3f} {rf_test_mae[i]:<12.2f} {xgb_test_mae[i]:<12.2f}")

                # Calculate averages and improvements
                avg_rf_r2 = np.mean(rf_test_r2)
                avg_xgb_r2 = np.mean(xgb_test_r2)
                avg_rf_mae = np.mean(rf_test_mae)
                avg_xgb_mae = np.mean(xgb_test_mae)
```

```python
                print("-" * 80)
                print(f"{'Average':<15} {avg_rf_r2:<12.3f} {avg_xgb_r2:<12.3f} {avg_rf_mae:<12.2f} {avg_xgb_mae:<12.2f}")
                print("-" * 80)

                # Calculate improvements
                r2_improvement = ((avg_xgb_r2 - avg_rf_r2) / avg_rf_r2) * 100
                mae_improvement = ((avg_rf_mae - avg_xgb_mae) / avg_rf_mae) * 100

                print(f"\n  XGBoost Performance vs Random Forest:")
                print(f"   R² Score: {'↑' if r2_improvement > 0 else '↓'} {abs(r2_improvement):.1f}% {'better' if r2_improvement > 0 else 'worse'}")
                print(f"   MAE:      {'↓' if mae_improvement > 0 else '↑'} {abs(mae_improvement):.1f}% {'better' if mae_improvement > 0 else 'worse'}")

                # Determine overall winner
                if r2_improvement > 0 and mae_improvement > 0:
                        print(f"\n  Winner: XGBoost (better on both metrics)")
                elif r2_improvement < 0 and mae_improvement < 0:
                        print(f"\n  Winner: Random Forest (better on both metrics)")
                else:
                        print(f"\n  Winner: Mixed results - depends on metric priority")

else:
        print("  No model data available for plotting. Please ensure position_models and/or xgboost_models are defined.")
```

```
 PLOTTING RANDOM FOREST MODEL PERFORMANCE BY POSITION
--------------------------------------------------
```



```
 RANDOM FOREST MODEL SUMMARY:
------------------------------------------------------------
Position        Train R²   Test R²    Train MAE  Test MAE
------------------------------------------------------------
Forward         0.000      0.947      0.00       0.60
Midfield        0.000      0.913      0.00       2.89
Defense         0.000      0.969      0.00       0.48
Goalkeeper      0.000      0.993      0.00       6.80
------------------------------------------------------------
Average         0.000      0.956      0.00       2.69
 No model data available for plotting. Please ensure position_models and/or
xgboost_models are defined.
 PLOTTING RANDOM FOREST MODEL PERFORMANCE BY POSITION
--------------------------------------------------
```




```
 RANDOM FOREST MODEL SUMMARY:
------------------------------------------------------------
Position        Train R²   Test R²    Train MAE  Test MAE
------------------------------------------------------------
Forward         0.000      0.947      0.00       0.60
Midfield        0.000      0.913      0.00       2.89
Defense         0.000      0.969      0.00       0.48
Goalkeeper      0.000      0.993      0.00       6.80
------------------------------------------------------------
Average         0.000      0.956      0.00       2.69
 No model data available for plotting. Please ensure position_models and/or
```

xgboost_models are defined.

[24]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime, timedelta
import warnings
warnings.filterwarnings('ignore')

# Configuration
plt.style.use('default')  # Changed from seaborn-v0_8-darkgrid which might not␣
↪exist
plt.rcParams['figure.dpi'] = 100
plt.rcParams['savefig.dpi'] = 300
plt.rcParams['font.size'] = 10

# Load data
path = '/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main Notebook/
↪Data Folder/DataCombined/real_madrid_rebalanced_scores.csv'
forecast_path = '/Users/home/Documents/GitHub/Capstone/
↪future_performance_forecasts.csv'

try:
    df = pd.read_csv(path)
    df['Date'] = pd.to_datetime(df['Date'])
    print(f" Data loaded: {df.shape[0]:,} rows")
except Exception as e:
    print(f" Error loading data: {e}")
    raise SystemExit  # Changed from exit() which can cause issues

# Try loading forecasts
try:
    forecasts_df = pd.read_csv(forecast_path)
    HAS_FORECASTS = True
    print(f" Forecasts loaded: {forecasts_df.shape[0]:,} rows")
except:
    HAS_FORECASTS = False
    forecasts_df = None
    print(" No forecasts found - showing historical data only")

# Prepare data
df['Week'] = df['Date'].dt.isocalendar().week
df['Year'] = df['Date'].dt.year

# Get last 16 weeks of data
latest_date = df['Date'].max()
cutoff_date = latest_date - timedelta(weeks=16)
```

```
recent_data = df[df['Date'] >= cutoff_date].copy()

# Create week ranking for consistent x-axis
week_periods = recent_data['Date'].dt.to_period('W')
week_order = sorted(week_periods.unique())
week_mapping = {week: idx for idx, week in enumerate(week_order)}
recent_data['WeekRank'] = week_periods.map(week_mapping)

print(f"\nAnalyzing {len(week_order)} weeks from {week_order[0]} to {week_order[-1]}")
print(f"Players in dataset: {recent_data['Player'].nunique()}")

# Create figure with better layout
fig, axes = plt.subplots(2, 2, figsize=(20, 14))  # Simplified subplot creation
axes = axes.flatten()

# Position configuration with distinct color palettes
positions = ['Forward', 'Midfield', 'Defense', 'Goalkeeper']
position_colors = {
    'Forward': ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b', '#e377c2', '#7f7f7f'],
    'Midfield': ['#393b79', '#5254a3', '#6b6ecf', '#9c9ede', '#637939', '#8ca252', '#b5cf6b', '#cedb9c'],
    'Defense': ['#8c6d31', '#bd9e39', '#e7ba52', '#e7cb94', '#843c39', '#ad494a', '#d6616b', '#e7969c'],
    'Goalkeeper': ['#7b4173', '#a55194', '#ce6dbd', '#de9ed6', '#3182bd', '#6baed6', '#9ecae1', '#c6dbef']
}

# Process each position
for pos_idx, position in enumerate(positions):
    ax = axes[pos_idx]
    pos_data = recent_data[recent_data['Position_Group'] == position]

    if len(pos_data) == 0:
        ax.text(0.5, 0.5, f'No data for {position}',
                transform=ax.transAxes, ha='center', va='center', fontsize=14)
        ax.set_title(f'{position} - No Data Available', fontsize=16)
        continue

    # Get top players (minimum 3 games for better representation)
    player_games = pos_data.groupby('Player').size()
    eligible_players = player_games[player_games >= 3].sort_values(ascending=False).head(8).index

    # Color palette for this position
```

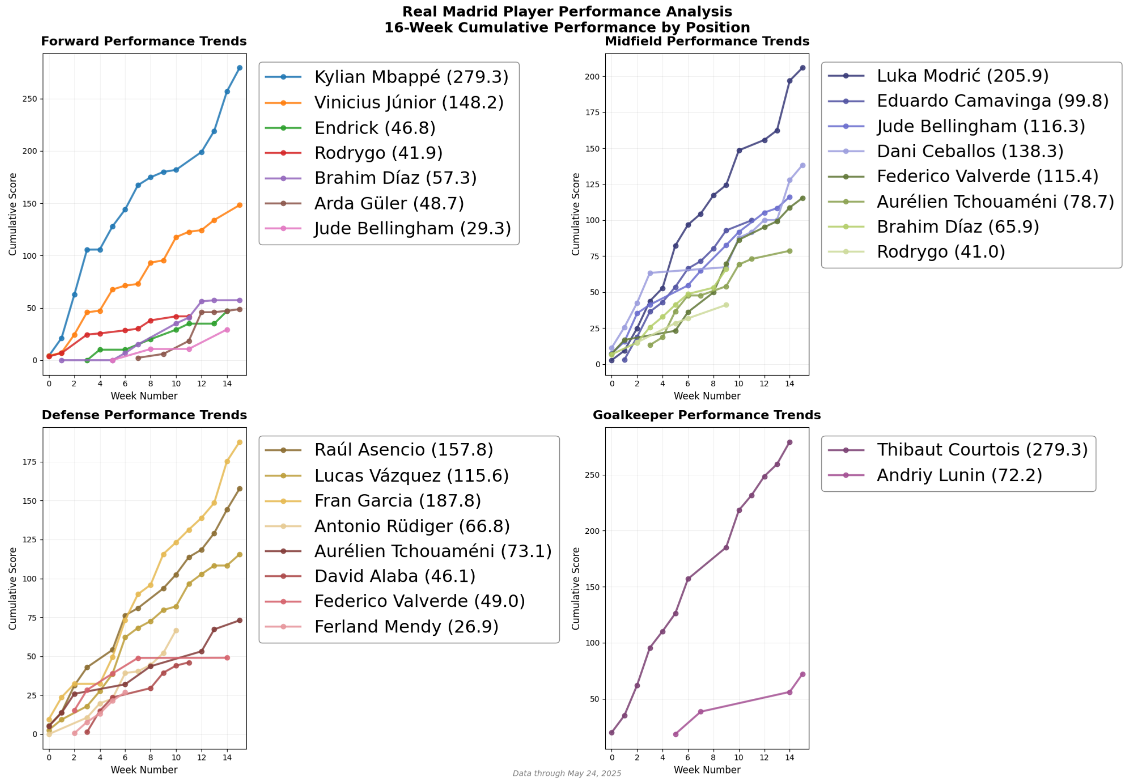
```
    colors = position_colors[position]

    # Plot each player's cumulative performance
    for player_idx, player in enumerate(eligible_players):
        player_data = pos_data[pos_data['Player'] == player].sort_values('Date')

        # Calculate cumulative score
        player_data = player_data.copy()  # Avoid SettingWithCopyWarning
        player_data['CumulativeScore'] = player_data['Rebalanced_Score'].
↪cumsum()

        # Aggregate by week (taking last cumulative value)
        weekly_data = player_data.groupby('WeekRank').agg({
            'CumulativeScore': 'last',
            'Date': 'first',
            'Rebalanced_Score': 'sum'
        }).reset_index().sort_values('WeekRank')

        if len(weekly_data) == 0:  # Safety check
            continue

        # Plot historical data
        ax.plot(weekly_data['WeekRank'], weekly_data['CumulativeScore'],
               marker='o', linewidth=2.5, markersize=6,
               color=colors[player_idx % len(colors)], alpha=0.85,
               label=f"{player[:20]} ({weekly_data['CumulativeScore'].iloc[-1]:.
↪1f})")

        # Add forecast if available
        if HAS_FORECASTS and forecasts_df is not None:
            player_forecast = forecasts_df[
                (forecasts_df['Player'] == player) &
                (forecasts_df['Position'] == position)
            ]

            if not player_forecast.empty and len(weekly_data) > 0:
                # Forecast parameters
                predicted_weekly = player_forecast['Predicted_Score'].iloc[0]
                last_week = weekly_data['WeekRank'].max()
                last_cumulative = weekly_data['CumulativeScore'].iloc[-1]

                # Generate 4-week forecast
                forecast_weeks = np.arange(last_week + 1, last_week + 5)
                forecast_cumulative = last_cumulative + predicted_weekly * np.
↪arange(1, 5)

                # Plot forecast with dashed line
```

```
                ax.plot(forecast_weeks, forecast_cumulative,
                       marker='s', markersize=6, linewidth=2,
                       color=colors[player_idx % len(colors)], linestyle='--',␣
↪alpha=0.6)

                # Label final forecast value
                ax.annotate(f'+{forecast_cumulative[-1] - last_cumulative:.0f}',
                          xy=(forecast_weeks[-1], forecast_cumulative[-1]),
                          xytext=(5, 5), textcoords='offset points',
                          fontsize=9, color=colors[player_idx % len(colors)],
                          fontweight='bold', alpha=0.8)

    # Add forecast separator line
    if HAS_FORECASTS and len(recent_data) > 0:
        max_week = recent_data['WeekRank'].max()
        ax.axvline(x=max_week + 0.5, color='gray', linestyle=':',
                  linewidth=2, alpha=0.5, zorder=0)

        # Add "Forecast" label
        y_lim = ax.get_ylim()
        if y_lim[1] > y_lim[0]:  # Check if y-limits are valid
            y_pos = y_lim[1] * 0.95
            ax.text(max_week + 2.5, y_pos, 'FORECAST',
                   ha='center', va='top', fontsize=11,
                   color='gray', fontweight='bold', alpha=0.7)

    # Styling
    ax.set_title(f'{position} Performance Trends', fontsize=16,␣
↪fontweight='bold', pad=10)
    ax.set_xlabel('Week Number', fontsize=12)
    ax.set_ylabel('Cumulative Score', fontsize=12)
    ax.grid(True, alpha=0.3, linestyle='-', linewidth=0.5)

    # Legend positioning
    ax.legend(bbox_to_anchor=(1.02, 1), loc='upper left',
             fontsize=22, framealpha=0.95, edgecolor='gray')

    # Set x-axis limits to show all data plus forecast
    if HAS_FORECASTS and len(recent_data) > 0:
        max_week = recent_data['WeekRank'].max()
        ax.set_xlim(-0.5, max_week + 4.5)
    elif len(recent_data) > 0:
        ax.set_xlim(-0.5, recent_data['WeekRank'].max() + 0.5)

# Main title
title = 'Real Madrid Player Performance Analysis\n'
title += '16-Week Cumulative Performance by Position'
```

```
if HAS_FORECASTS:
    title += ' with 4-Week Forecasts'

plt.suptitle(title, fontsize=18, fontweight='bold', y=0.98)

# Add footer info
fig.text(0.5, 0.01, f'Data through {latest_date.strftime("%B %d, %Y")}',
         ha='center', fontsize=10, style='italic', color='gray')

# Adjust layout and display
plt.tight_layout()
plt.show()

# Print summary statistics
print("\n  Summary Statistics:")
for position in positions:
    pos_players = recent_data[recent_data['Position_Group'] ==␣
 ↪position]['Player'].nunique()
    if pos_players > 0:
        avg_score = recent_data[recent_data['Position_Group'] ==␣
 ↪position]['Rebalanced_Score'].mean()
        print(f"  {position}: {pos_players} players, avg score: {avg_score:.
 ↪2f}")


#Real Madrid Cumulative Performance by Position (Rebalanced Scores)
# Performance Summary Table
print("\n  Performance Summary by Position:")
print("="*80)

# Create summary data for each position
summary_data = []

for position in positions:
    pos_data = recent_data[recent_data['Position_Group'] == position]

    if len(pos_data) > 0:
        # Get top 5 players by average score
        player_avg = pos_data.groupby('Player')['Rebalanced_Score'].
 ↪agg(['mean', 'sum', 'count'])
        player_avg = player_avg[player_avg['count'] >= 3].sort_values('mean',␣
 ↪ascending=False).head(5)

        for player, stats in player_avg.iterrows():
            summary_data.append({
                'Position': position,
                'Player': player[:25],  # Truncate long names
```

```python
                'Avg Score': stats['mean'],
                'Total Score': stats['sum'],
                'Games': int(stats['count'])
            })

# Create DataFrame and display
summary_df = pd.DataFrame(summary_data)

# Display by position
for position in positions:
    pos_summary = summary_df[summary_df['Position'] == position]
    if len(pos_summary) > 0:
        print(f"\n{position}:")
        print(pos_summary.to_string(index=False, float_format='%.2f'))

# Overall statistics
print("\n\n Overall Statistics:")
print(f"Total unique players analyzed: {recent_data['Player'].nunique()}")
print(f"Total matches analyzed: {len(recent_data)}")
print(f"Average score across all positions: {recent_data['Rebalanced_Score'].
 ↪mean():.2f}")

# Forecast summary if available
if HAS_FORECASTS and forecasts_df is not None:
    print("\n\n Forecast Summary (Next 4 Weeks):")
    forecast_summary = forecasts_df.groupby('Position').agg({
        'Predicted_Score': ['mean', 'count'],
        'Score_Change': 'mean'
    }).round(2)
    forecast_summary.columns = ['Avg Predicted', 'Player Count', 'Avg Change']
    print(forecast_summary)
```

```
 Data loaded: 5,737 rows
 No forecasts found - showing historical data only

Analyzing 16 weeks from 2025-01-27/2025-02-02 to 2025-05-19/2025-05-25
Players in dataset: 22
```



Summary Statistics:
Forward: 8 players, avg score: 7.60
Midfield: 11 players, avg score: 7.80
Defense: 11 players, avg score: 7.20
Goalkeeper: 2 players, avg score: 15.98

Performance Summary by Position:
==

Forward:

Position	Player	Avg Score	Total Score	Games
Forward	Kylian Mbappé	13.30	279.29	21
Forward	Jude Bellingham	7.33	29.32	4
Forward	Vinicius Júnior	7.06	148.18	21
Forward	Arda Güler	6.96	48.74	7
Forward	Brahim Díaz	5.73	57.31	10

Midfield:

Position	Player	Avg Score	Total Score	Games
Midfield	Dani Ceballos	9.88	138.27	14
Midfield	Luka Modrić	9.80	205.89	21
Midfield	Arda Güler	8.37	41.86	5

```
Midfield Federico Valverde       8.24       115.42     14
Midfield   Jude Bellingham       7.75       116.25     15

Defense:
Position              Player  Avg Score  Total Score  Games
 Defense          Fran Garcia      10.43       187.82     18
 Defense   Federico Valverde        8.17        49.03      6
 Defense Aurélien Tchouaméni        8.12        73.09      9
 Defense         Raúl Asencio       7.89       157.77     20
 Defense         Jacobo Ramón       7.74        23.23      3

Goalkeeper:
  Position           Player  Avg Score  Total Score  Games
Goalkeeper     Andriy Lunin      18.05        72.18      4
Goalkeeper Thibaut Courtois      15.52       279.34     18


  Overall Statistics:
Total unique players analyzed: 22
Total matches analyzed: 330
Average score across all positions: 8.10
```

[26]:
```
import pandas as pd
import numpy as np
from sklearn.neural_network import MLPRegressor
from sklearn.ensemble import VotingRegressor, GradientBoostingRegressor,␣
↪RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import xgboost as xgb
import warnings
import matplotlib.pyplot as plt
warnings.filterwarnings('ignore')

print("="*80)
print("  TRAINING ADDITIONAL MODELS: NEURAL NETWORK & ENSEMBLE")
print("="*80)

# First, we need to recreate the base models and datasets from the previous␣
↪analysis
# Using the same data and position-specific metrics from the earlier code

# Load data (assuming this was already done)
path = '/Users/mariamoramora/Documents/GitHub/ADS599_Capstone/Main Notebook/
↪Data Folder/DataCombined/real_madrid_rebalanced_scores.csv'
df = pd.read_csv(path)
```

```python
# Create Week column if needed
if 'Week' not in df.columns:
    df['Date'] = pd.to_datetime(df['Date'])
    df['Week'] = df['Date'].dt.isocalendar().week

# Create per-90 features
def create_per90_features(df):
    df_per90 = df.copy()
    df_per90['Min_Safe'] = df_per90['Min'].replace(0, 1)

    volume_stats = [
        ' Gls', ' Ast', ' SoT', ' KP', ' Tkl', ' Int', ' Blocks', ' Clr',
        'Expected xG', 'Expected xAG', 'Take-Ons Succ', 'Carries PrgC',
        'Passes PrgP', 'Touches', 'Tackles TklW', 'Tackles Def 3rd',
        'Tackles Mid 3rd', 'Total Cmp', 'Total PrgDist'
    ]

    for stat in volume_stats:
        if stat in df_per90.columns:
            clean_name = stat.strip().replace(' ', '').replace('-', '_')
            new_name = f"{clean_name}_Per90"
            df_per90[new_name] = (df_per90[stat] / df_per90['Min_Safe']) * 90
            df_per90[new_name] = df_per90[new_name].fillna(0)

    return df_per90

df_per90 = create_per90_features(df)

# Position-specific metrics
per90_metrics = {
    'Forward': ['Gls_Per90', 'Ast_Per90', 'SoT_Per90', 'ExpectedxG_Per90', 'ExpectedxAG_Per90', 'TakeOnsSucc_Per90'],
    'Midfield': ['Passes Cmp%', 'KP_Per90', 'Tkl_Per90', 'CarriesPrgC_Per90', 'PassesPrgP_Per90', 'Touches_Per90'],
    'Defense': ['Int_Per90', 'Blocks_Per90', 'Clr_Per90', 'TacklesTklW_Per90', 'TacklesDef3rd_Per90', 'TacklesMid3rd_Per90'],
    'Goalkeeper': ['Total Cmp%', ' Err', 'TotalPrgDist_Per90', 'Short Cmp%', 'Medium Cmp%', 'TotalCmp_Per90']
}

# ==========================================
# CREATE POSITION DATASETS
# ==========================================

def create_position_datasets():
    """Create train/test datasets for each position"""
    print("Creating position datasets...")
```

```
    position_datasets = {}

    for position in per90_metrics.keys():
        pos_data = df_per90[(df_per90['Position_Group'] == position) &
                            (df_per90['Rebalanced_Score'].notna())].copy()

        if len(pos_data) < 30:
            print(f"Skip {position} - insufficient data: {len(pos_data)}")
            continue

        available_metrics = [m for m in per90_metrics[position] if m in␣
↪pos_data.columns]

        if len(available_metrics) < 3:
            print(f"Skip {position} - insufficient metrics:␣
↪{len(available_metrics)}")
            continue

        # Time split
        latest_week = pos_data['Week'].max()
        test_start_week = latest_week - 4 + 1

        train_data = pos_data[pos_data['Week'] < test_start_week]
        test_data = pos_data[pos_data['Week'] >= test_start_week]

        if len(train_data) < 20 or len(test_data) < 5:
            print(f"Skip {position} - insufficient split")
            continue

        X_train = train_data[available_metrics].fillna(0)
        y_train = train_data['Rebalanced_Score']
        X_test = test_data[available_metrics].fillna(0)
        y_test = test_data['Rebalanced_Score']

        position_datasets[position] = {
            'X_train': X_train,
            'y_train': y_train,
            'X_test': X_test,
            'y_test': y_test,
            'metrics': available_metrics
        }

        print(f"  {position}: {len(X_train)} train, {len(X_test)} test samples")

    return position_datasets
```

```python
position_datasets = create_position_datasets()

# ==========================================
# TRAIN BASE MODELS (RF & XGB)
# ==========================================

def train_base_models():
    """Train Random Forest and XGBoost models"""
    print("\n  Training base models (RF & XGBoost)...")

    rf_models = {}
    xgb_models = {}

    for position, dataset in position_datasets.items():
        X_train = dataset['X_train']
        y_train = dataset['y_train']
        X_test = dataset['X_test']
        y_test = dataset['y_test']

        # Random Forest
        rf_model = RandomForestRegressor(
            n_estimators=100,
            max_depth=10,
            min_samples_split=5,
            min_samples_leaf=2,
            random_state=42
        )
        rf_model.fit(X_train, y_train)
        rf_pred = rf_model.predict(X_test)

        rf_models[position] = {
            'model': rf_model,
            'metrics': {
                'test_r2': r2_score(y_test, rf_pred),
                'test_mae': mean_absolute_error(y_test, rf_pred)
            }
        }

        # XGBoost
        xgb_model = xgb.XGBRegressor(
            n_estimators=100,
            max_depth=6,
            learning_rate=0.1,
            subsample=0.8,
            random_state=42,
            verbosity=0
        )
```

```
        xgb_model.fit(X_train, y_train)
        xgb_pred = xgb_model.predict(X_test)

        xgb_models[position] = {
            'model': xgb_model,
            'metrics': {
                'test_r2': r2_score(y_test, xgb_pred),
                'test_mae': mean_absolute_error(y_test, xgb_pred)
            }
        }

        print(f"  {position} - RF R²:␣
 ↪{rf_models[position]['metrics']['test_r2']:.3f}, XGB R²:␣
 ↪{xgb_models[position]['metrics']['test_r2']:.3f}")

    return rf_models, xgb_models

rf_models, xgb_models = train_base_models()

# ==========================================
# MODEL 1: NEURAL NETWORK (MLP)
# ==========================================

def train_neural_network_models():
    """Train Multi-Layer Perceptron models for each position"""
    print("\n NEURAL NETWORK TRAINING")
    print("-" * 50)

    nn_models = {}

    for position, dataset in position_datasets.items():
        print(f"\nTraining Neural Network for {position}...")

        X_train = dataset['X_train']
        y_train = dataset['y_train']
        X_test = dataset['X_test']
        y_test = dataset['y_test']

        # Standardize features for neural network
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Define neural network architecture
        nn_model = MLPRegressor(
            hidden_layer_sizes=(100, 50, 25),  # 3 hidden layers
            activation='relu',
```

```
            solver='adam',
            alpha=0.001,
            batch_size='auto',
            learning_rate='adaptive',
            learning_rate_init=0.001,
            max_iter=1000,
            random_state=42,
            early_stopping=True,
            validation_fraction=0.1,
            n_iter_no_change=20
        )

        # Train model
        nn_model.fit(X_train_scaled, y_train)

        # Make predictions
        y_train_pred = nn_model.predict(X_train_scaled)
        y_test_pred = nn_model.predict(X_test_scaled)

        # Calculate metrics
        train_r2 = r2_score(y_train, y_train_pred)
        test_r2 = r2_score(y_test, y_test_pred)
        train_mae = mean_absolute_error(y_train, y_train_pred)
        test_mae = mean_absolute_error(y_test, y_test_pred)

        print(f"  Training R²: {train_r2:.3f}, MAE: {train_mae:.3f}")
        print(f"  Testing R²:  {test_r2:.3f}, MAE: {test_mae:.3f}")
        print(f"  Iterations: {nn_model.n_iter_}")

        nn_models[position] = {
            'model': nn_model,
            'scaler': scaler,
            'metrics': {
                'train_r2': train_r2,
                'test_r2': test_r2,
                'train_mae': train_mae,
                'test_mae': test_mae
            }
        }

    return nn_models

# ===========================================
# MODEL 2: ENSEMBLE (VOTING REGRESSOR)
# ===========================================

def train_ensemble_models():
```

```python
    """Train ensemble models combining RF, XGBoost, and GradientBoosting"""
    print("\n ENSEMBLE MODEL TRAINING")
    print("-" * 50)

    ensemble_models = {}

    for position in position_datasets.keys():
        print(f"\nTraining Ensemble for {position}...")

        dataset = position_datasets[position]
        X_train = dataset['X_train']
        y_train = dataset['y_train']
        X_test = dataset['X_test']
        y_test = dataset['y_test']

        # Get existing models
        rf_model = rf_models[position]['model']
        xgb_model = xgb_models[position]['model']

        # Create new GradientBoosting model
        gb_model = GradientBoostingRegressor(
            n_estimators=100,
            learning_rate=0.1,
            max_depth=6,
            min_samples_split=5,
            min_samples_leaf=2,
            subsample=0.8,
            random_state=42
        )

        # Create voting ensemble
        ensemble = VotingRegressor([
            ('rf', rf_model),
            ('gb', gb_model),
            ('xgb', xgb_model)
        ])

        # Train ensemble
        ensemble.fit(X_train, y_train)

        # Make predictions
        y_train_pred = ensemble.predict(X_train)
        y_test_pred = ensemble.predict(X_test)

        # Calculate metrics
        train_r2 = r2_score(y_train, y_train_pred)
        test_r2 = r2_score(y_test, y_test_pred)
```

```python
        train_mae = mean_absolute_error(y_train, y_train_pred)
        test_mae = mean_absolute_error(y_test, y_test_pred)

        print(f"  Training R²: {train_r2:.3f}, MAE: {train_mae:.3f}")
        print(f"  Testing R²:  {test_r2:.3f}, MAE: {test_mae:.3f}")

        ensemble_models[position] = {
            'model': ensemble,
            'metrics': {
                'train_r2': train_r2,
                'test_r2': test_r2,
                'train_mae': train_mae,
                'test_mae': test_mae
            }
        }

    return ensemble_models

# ==========================================
# TRAIN ALL MODELS
# ==========================================

# Train Neural Network models
nn_models = train_neural_network_models()

# Train Ensemble models
ensemble_models = train_ensemble_models()

# ==========================================
# COMPREHENSIVE MODEL COMPARISON
# ==========================================

def create_model_comparison_visualization():
    """Create comprehensive comparison of all models"""
    print("\n COMPREHENSIVE MODEL COMPARISON")
    print("-" * 50)

    # Collect metrics for all models
    model_types = ['Random Forest', 'XGBoost', 'Neural Network', 'Ensemble']
    positions = list(position_datasets.keys())

    # Create comparison data
    comparison_data = []

    for position in positions:
        # Random Forest
        if position in rf_models:
```

```
            comparison_data.append({
                'Model': 'Random Forest',
                'Position': position,
                'Test_R2': rf_models[position]['metrics']['test_r2'],
                'Test_MAE': rf_models[position]['metrics']['test_mae']
            })

        # XGBoost
        if position in xgb_models:
            comparison_data.append({
                'Model': 'XGBoost',
                'Position': position,
                'Test_R2': xgb_models[position]['metrics']['test_r2'],
                'Test_MAE': xgb_models[position]['metrics']['test_mae']
            })

        # Neural Network
        if position in nn_models:
            comparison_data.append({
                'Model': 'Neural Network',
                'Position': position,
                'Test_R2': nn_models[position]['metrics']['test_r2'],
                'Test_MAE': nn_models[position]['metrics']['test_mae']
            })

        # Ensemble
        if position in ensemble_models:
            comparison_data.append({
                'Model': 'Ensemble',
                'Position': position,
                'Test_R2': ensemble_models[position]['metrics']['test_r2'],
                'Test_MAE': ensemble_models[position]['metrics']['test_mae']
            })

    comparison_df = pd.DataFrame(comparison_data)

    # Create visualization
    fig, axes = plt.subplots(2, 2, figsize=(16, 12))

    # 1. R² Comparison by Model Type
    ax1 = axes[0, 0]
    model_r2_avg = comparison_df.groupby('Model')['Test_R2'].mean().
↪sort_values(ascending=False)
    bars1 = ax1.bar(model_r2_avg.index, model_r2_avg.values,␣
↪color=['darkgreen', 'steelblue', 'purple', 'orange'])
    ax1.set_ylabel('Average Test R²', fontweight='bold')
```

```python
    ax1.set_title('Model Performance Comparison - R² Score', fontweight='bold', fontsize=14)
    ax1.grid(True, alpha=0.3, axis='y')
    ax1.set_ylim(0.5, 1.0)

    # Add value labels
    for bar in bars1:
        height = bar.get_height()
        ax1.annotate(f'{height:.3f}',
                     xy=(bar.get_x() + bar.get_width() / 2, height),
                     xytext=(0, 3),
                     textcoords="offset points",
                     ha='center', va='bottom',
                     fontsize=10, fontweight='bold')

    # 2. MAE Comparison by Model Type
    ax2 = axes[0, 1]
    model_mae_avg = comparison_df.groupby('Model')['Test_MAE'].mean().sort_values()
    bars2 = ax2.bar(model_mae_avg.index, model_mae_avg.values, color=['darkgreen', 'orange', 'steelblue', 'purple'])
    ax2.set_ylabel('Average Test MAE', fontweight='bold')
    ax2.set_title('Model Performance Comparison - MAE', fontweight='bold', fontsize=14)
    ax2.grid(True, alpha=0.3, axis='y')

    # Add value labels
    for bar in bars2:
        height = bar.get_height()
        ax2.annotate(f'{height:.2f}',
                     xy=(bar.get_x() + bar.get_width() / 2, height),
                     xytext=(0, 3),
                     textcoords="offset points",
                     ha='center', va='bottom',
                     fontsize=10, fontweight='bold')

    # 3. Position-wise R² Comparison
    ax3 = axes[1, 0]
    pivot_r2 = comparison_df.pivot(index='Position', columns='Model', values='Test_R2')
    x = np.arange(len(positions))
    width = 0.2

    colors = ['darkgreen', 'steelblue', 'purple', 'orange']
    for i, model in enumerate(model_types):
        if model in pivot_r2.columns:
            values = pivot_r2[model].values
```

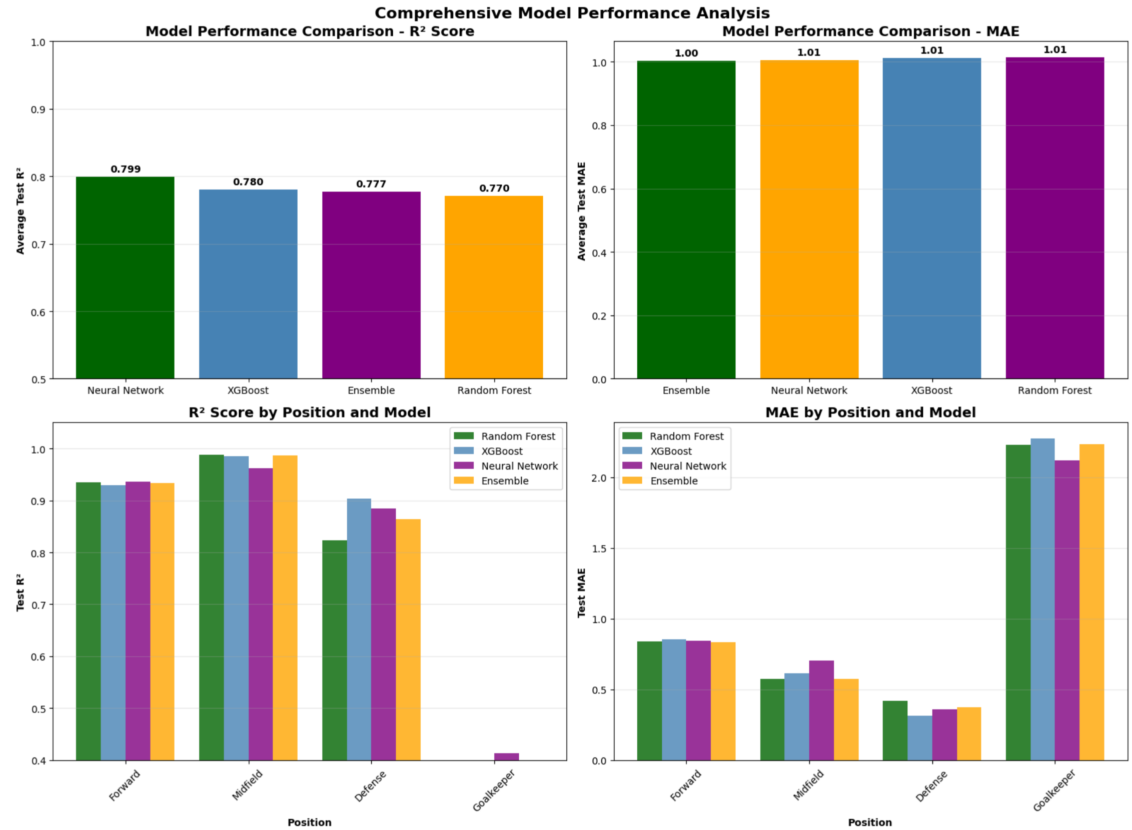
```python
            ax3.bar(x + i*width - 1.5*width, values, width, label=model, alpha=0.8, color=colors[i])

    ax3.set_xlabel('Position', fontweight='bold')
    ax3.set_ylabel('Test R²', fontweight='bold')
    ax3.set_title('R² Score by Position and Model', fontweight='bold', fontsize=14)
    ax3.set_xticks(x)
    ax3.set_xticklabels(positions, rotation=45)
    ax3.legend()
    ax3.grid(True, alpha=0.3, axis='y')
    ax3.set_ylim(0.4, 1.05)

    # 4. Position-wise MAE Comparison
    ax4 = axes[1, 1]
    pivot_mae = comparison_df.pivot(index='Position', columns='Model', values='Test_MAE')

    for i, model in enumerate(model_types):
        if model in pivot_mae.columns:
            values = pivot_mae[model].values
            ax4.bar(x + i*width - 1.5*width, values, width, label=model, alpha=0.8, color=colors[i])

    ax4.set_xlabel('Position', fontweight='bold')
    ax4.set_ylabel('Test MAE', fontweight='bold')
    ax4.set_title('MAE by Position and Model', fontweight='bold', fontsize=14)
    ax4.set_xticks(x)
    ax4.set_xticklabels(positions, rotation=45)
    ax4.legend()
    ax4.grid(True, alpha=0.3, axis='y')

    plt.suptitle('Comprehensive Model Performance Analysis', fontsize=16, fontweight='bold')
    plt.tight_layout()
    plt.show()

    # Print detailed comparison
    print("\n DETAILED MODEL COMPARISON:")
    print("-" * 80)
    print(f"{'Model':<20} {'Avg R²':<10} {'Avg MAE':<10} {'Best Position':<15} {'Worst Position':<15}")
    print("-" * 80)

    for model in model_types:
        model_data = comparison_df[comparison_df['Model'] == model]
```

```
        if len(model_data) > 0:
            avg_r2 = model_data['Test_R2'].mean()
            avg_mae = model_data['Test_MAE'].mean()
            best_pos = model_data.loc[model_data['Test_R2'].idxmax(),␣
↪'Position']
            worst_pos = model_data.loc[model_data['Test_R2'].idxmin(),␣
↪'Position']
            print(f"{model:<20} {avg_r2:<10.3f} {avg_mae:<10.2f} {best_pos:<15}␣
↪{worst_pos:<15}")

    # Calculate improvements over Random Forest
    rf_avg_r2 = comparison_df[comparison_df['Model'] == 'Random␣
↪Forest']['Test_R2'].mean()
    rf_avg_mae = comparison_df[comparison_df['Model'] == 'Random␣
↪Forest']['Test_MAE'].mean()

    print("\n IMPROVEMENTS OVER RANDOM FOREST:")
    print("-" * 50)

    for model in ['XGBoost', 'Neural Network', 'Ensemble']:
        model_data = comparison_df[comparison_df['Model'] == model]
        if len(model_data) > 0:
            model_avg_r2 = model_data['Test_R2'].mean()
            model_avg_mae = model_data['Test_MAE'].mean()

            r2_improvement = ((model_avg_r2 - rf_avg_r2) / rf_avg_r2) * 100
            mae_improvement = ((rf_avg_mae - model_avg_mae) / rf_avg_mae) * 100

            print(f"{model}:")
            print(f"  R² Score: {'↑' if r2_improvement > 0 else '↓'}␣
↪{abs(r2_improvement):.1f}%")
            print(f"  MAE:      {'↓' if mae_improvement > 0 else '↑'}␣
↪{abs(mae_improvement):.1f}%")

    return comparison_df

# Execute comprehensive comparison
comparison_results = create_model_comparison_visualization()

print("\n MODEL TRAINING COMPLETE!")
print(f"Trained {len(nn_models)} Neural Network models")
print(f"Trained {len(ensemble_models)} Ensemble models")
print(f"Trained {len(rf_models)} Random Forest models")
print(f"Trained {len(xgb_models)} XGBoost models")
```

```
================================================================================
 TRAINING ADDITIONAL MODELS: NEURAL NETWORK & ENSEMBLE
```

```
================================================================================
Creating position datasets…
  Forward: 1585 train, 110 test samples
  Midfield: 1724 train, 99 test samples
  Defense: 1718 train, 105 test samples
  Goalkeeper: 372 train, 24 test samples

 Training base models (RF & XGBoost)…
  Forward - RF R²: 0.987, XGB R²: 0.985
  Midfield - RF R²: 0.336, XGB R²: 0.302
  Defense - RF R²: 0.935, XGB R²: 0.929
  Goalkeeper - RF R²: 0.823, XGB R²: 0.904

 NEURAL NETWORK TRAINING
--------------------------------------------------

Training Neural Network for Forward…
  Training R²: 0.990, MAE: 0.563
  Testing R²:  0.962, MAE: 0.703
  Iterations: 295

Training Neural Network for Midfield…
  Training R²: 0.492, MAE: 2.148
  Testing R²:  0.413, MAE: 2.119
  Iterations: 86

Training Neural Network for Defense…
  Training R²: 0.933, MAE: 0.901
  Testing R²:  0.937, MAE: 0.844
  Iterations: 126

Training Neural Network for Goalkeeper…
  Training R²: 0.962, MAE: 0.223
  Testing R²:  0.885, MAE: 0.356
  Iterations: 318

 ENSEMBLE MODEL TRAINING
--------------------------------------------------

Training Ensemble for Forward…
  Training R²: 0.994, MAE: 0.376
  Testing R²:  0.987, MAE: 0.575

Training Ensemble for Midfield…
  Training R²: 0.815, MAE: 1.282
  Testing R²:  0.325, MAE: 2.232

Training Ensemble for Defense…
```

```
  Training R²: 0.956, MAE: 0.705
  Testing R²:  0.933, MAE: 0.831

Training Ensemble for Goalkeeper…
  Training R²: 0.996, MAE: 0.065
  Testing R²:  0.864, MAE: 0.374

 COMPREHENSIVE MODEL COMPARISON
--------------------------------------------------
```




```
 DETAILED MODEL COMPARISON:
--------------------------------------------------------------------------------
Model                    Avg R²      Avg MAE     Best Position    Worst Position
--------------------------------------------------------------------------------
Random Forest            0.770       1.01        Forward          Midfield
XGBoost                  0.780       1.01        Forward          Midfield
Neural Network           0.799       1.01        Forward          Midfield
Ensemble                 0.777       1.00        Forward          Midfield

 IMPROVEMENTS OVER RANDOM FOREST:
--------------------------------------------------
```

```
XGBoost:
  R² Score: ↑ 1.3%
  MAE:      ↓ 0.2%
Neural Network:
  R² Score: ↑ 3.7%
  MAE:      ↓ 0.8%
Ensemble:
  R² Score: ↑ 0.9%
  MAE:      ↓ 1.1%

 MODEL TRAINING COMPLETE!
Trained 4 Neural Network models
Trained 4 Ensemble models
Trained 4 Random Forest models
Trained 4 XGBoost models
```

[28]:

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
import numpy as np
import pandas as pd
import seaborn as sns
from scipy.stats import uniform, randint
import time
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

print("="*80)
print("  FAST NEURAL NETWORK TUNING (5-MINUTE VERSION)")
print("="*80)

def tune_neural_network_fast(position, dataset):
    """
    Fast neural network tuning - optimized for 5-minute completion
    """
    print(f"\n  Fast tuning for {position}...")

    X_train = dataset['X_train']
    y_train = dataset['y_train']
    X_test = dataset['X_test']
    y_test = dataset['y_test']

    # Standardize features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)

# FAST ARCHITECTURE TEST (only 3 options)
print(f"  Testing 3 architectures...")
architectures = [
    (100, 50),       # Simple
    (150, 75, 35),   # Medium
    (100, 50, 25)    # Balanced
]

best_arch = None
best_arch_score = -999

for arch in architectures:
    # Quick test with limited iterations
    nn_test = MLPRegressor(
        hidden_layer_sizes=arch,
        activation='relu',
        solver='adam',
        alpha=0.001,
        max_iter=200,  # Much lower
        early_stopping=True,
        validation_fraction=0.2,
        n_iter_no_change=10,  # Faster stopping
        random_state=42
    )

    nn_test.fit(X_train_scaled, y_train)
    score = r2_score(y_test, nn_test.predict(X_test_scaled))

    if score > best_arch_score:
        best_arch_score = score
        best_arch = arch

    print(f"    {arch}: R² = {score:.3f}")

print(f"   Best architecture: {best_arch}")

# FAST HYPERPARAMETER SEARCH (limited options)
print(f"  Random search with 15 combinations...")

param_distributions = {
    'hidden_layer_sizes': [best_arch],  # Fixed to best
    'activation': ['relu', 'tanh'],     # Only 2 options
    'solver': ['adam'],                 # Fixed to adam (fastest)
    'alpha': uniform(0.0001, 0.01),   # Range reduced
    'learning_rate_init': uniform(0.001, 0.01),
```

```
        'max_iter': [300, 500],              # Limited options
        'early_stopping': [True],            # Always true
        'validation_fraction': [0.15],       # Fixed
        'n_iter_no_change': [15],            # Fixed
        'batch_size': ['auto', 64, 128]      # Limited options
    }

    # Fast random search - only 15 iterations
    random_search = RandomizedSearchCV(
        estimator=MLPRegressor(random_state=42),
        param_distributions=param_distributions,
        n_iter=15,  # Very limited
        cv=2,       # Only 2-fold CV
        scoring='r2',
        n_jobs=-1,
        verbose=0,  # Silent
        random_state=42
    )

    start_time = time.time()
    random_search.fit(X_train_scaled, y_train)
    search_time = time.time() - start_time

    best_params = random_search.best_params_
    print(f"  Search completed in {search_time:.1f}s")
    print(f"  Best CV score: {random_search.best_score_:.3f}")

    # FINAL MODEL TRAINING
    final_model = MLPRegressor(**best_params, random_state=42)
    final_model.fit(X_train_scaled, y_train)

    # Predictions and metrics
    y_train_pred = final_model.predict(X_train_scaled)
    y_test_pred = final_model.predict(X_test_scaled)

    metrics = {
        'train_r2': r2_score(y_train, y_train_pred),
        'test_r2': r2_score(y_test, y_test_pred),
        'test_mae': mean_absolute_error(y_test, y_test_pred),
        'search_time': search_time,
        'n_iterations': final_model.n_iter_
    }

    print(f"  Final Test R²: {metrics['test_r2']:.3f}, MAE:␣
↪{metrics['test_mae']:.3f}")

    return {
```

```python
        'model': final_model,
        'scaler': scaler,
        'best_params': best_params,
        'metrics': metrics,
        'best_architecture': best_arch
    }

def create_fast_comparison():
    """
    Create fast comparison visualization
    """
    print(f"\n  Creating comparison charts...")

    # Collect results
    results_data = []
    for position, results in tuned_nn_fast.items():
        results_data.append({
            'Position': position,
            'Architecture': str(results['best_architecture']),
            'Activation': results['best_params']['activation'],
            'Alpha': results['best_params']['alpha'],
            'Learning_Rate': results['best_params']['learning_rate_init'],
            'Test_R2': results['metrics']['test_r2'],
            'Test_MAE': results['metrics']['test_mae'],
            'Search_Time': results['metrics']['search_time'],
            'Iterations': results['metrics']['n_iterations']
        })

    results_df = pd.DataFrame(results_data)

    # Create 2x2 visualization
    fig, axes = plt.subplots(2, 2, figsize=(15, 10))

    # 1. Performance by position
    ax1 = axes[0, 0]
    bars1 = ax1.bar(results_df['Position'], results_df['Test_R2'],
                    color=['darkgreen', 'steelblue', 'purple', 'orange'],␣
↪alpha=0.8)
    ax1.set_ylabel('Test R² Score')
    ax1.set_title('Tuned Neural Network Performance by Position',␣
↪fontweight='bold')
    ax1.grid(True, alpha=0.3, axis='y')
    ax1.set_ylim(0.5, 1.0)

    # Add value labels
    for bar in bars1:
        height = bar.get_height()
```

```
        ax1.text(bar.get_x() + bar.get_width()/2., height + 0.01,
                f'{height:.3f}', ha='center', va='bottom', fontweight='bold')

    # 2. Search time efficiency
    ax2 = axes[0, 1]
    bars2 = ax2.bar(results_df['Position'], results_df['Search_Time'],
                    color='crimson', alpha=0.7)
    ax2.set_ylabel('Search Time (seconds)')
    ax2.set_title('Hyperparameter Search Time', fontweight='bold')
    ax2.grid(True, alpha=0.3, axis='y')

    for bar in bars2:
        height = bar.get_height()
        ax2.text(bar.get_x() + bar.get_width()/2., height + 0.5,
                f'{height:.1f}s', ha='center', va='bottom', fontweight='bold')

    # 3. Architecture distribution
    ax3 = axes[1, 0]
    arch_counts = results_df['Architecture'].value_counts()
    colors = plt.cm.Set3(np.linspace(0, 1, len(arch_counts)))
    wedges, texts, autotexts = ax3.pie(arch_counts.values, labels=arch_counts.
↪index,
                                       autopct='%1.0f%%', colors=colors)
    ax3.set_title('Best Architectures Distribution', fontweight='bold')

    # 4. Model comparison (if other models exist)
    ax4 = axes[1, 1]

    # Try to compare with existing models
    comparison_data = []

    # Add tuned NN results
    for _, row in results_df.iterrows():
        comparison_data.append({
            'Model': 'Tuned NN',
            'Position': row['Position'],
            'Test_R2': row['Test_R2']
        })

    # Add other models if they exist
    try:
        if 'nn_models' in globals():
            for position in results_df['Position']:
                if position in nn_models:
                    comparison_data.append({
                        'Model': 'Original NN',
                        'Position': position,
```

```
                        'Test_R2': nn_models[position]['metrics']['test_r2']
                    })
    except:
        pass

    try:
        if 'rf_models' in globals():
            for position in results_df['Position']:
                if position in rf_models:
                    comparison_data.append({
                        'Model': 'Random Forest',
                        'Position': position,
                        'Test_R2': rf_models[position]['metrics']['test_r2']
                    })
    except:
        pass

    if len(comparison_data) > len(results_df):
        comp_df = pd.DataFrame(comparison_data)
        pivot_comp = comp_df.pivot(index='Position', columns='Model',␣
↪values='Test_R2')
        pivot_comp.plot(kind='bar', ax=ax4, colormap='viridis', alpha=0.8)
        ax4.set_title('Model Performance Comparison', fontweight='bold')
        ax4.set_ylabel('Test R² Score')
        ax4.legend(loc='lower right')
        ax4.grid(True, alpha=0.3, axis='y')
        ax4.set_ylim(0.5, 1.05)
    else:
        # Just show tuned NN performance
        bars4 = ax4.bar(results_df['Position'], results_df['Test_R2'],
                        color='darkblue', alpha=0.8)
        ax4.set_ylabel('Test R² Score')
        ax4.set_title('Final Tuned NN Performance', fontweight='bold')
        ax4.grid(True, alpha=0.3, axis='y')
        ax4.set_ylim(0.5, 1.0)

    plt.suptitle('Fast Neural Network Tuning Results', fontsize=16,␣
↪fontweight='bold')
    plt.tight_layout()
    plt.show()

    return results_df

# ============================================
# MAIN EXECUTION - FAST VERSION
# ============================================
```

```python
print(f"Starting fast neural network tuning...")
print(f"Estimated total time: ~5 minutes")

# Record total start time
total_start_time = time.time()

# Fast tuning for each position
tuned_nn_fast = {}

for position, dataset in position_datasets.items():
    position_start = time.time()

    tuning_results = tune_neural_network_fast(position, dataset)
    tuned_nn_fast[position] = tuning_results

    position_time = time.time() - position_start
    print(f"    {position} completed in {position_time:.1f}s")

# Total time
total_time = time.time() - total_start_time
print(f"\n  TOTAL TUNING TIME: {total_time:.1f} seconds ({total_time/60:.1f} minutes)")

# Create results visualization
results_summary = create_fast_comparison()

# Print summary table
print(f"\n  FAST TUNING RESULTS SUMMARY")
print("="*80)
print(results_summary.to_string(index=False, float_format='%.3f'))

# Calculate improvements if original models exist
print(f"\n  PERFORMANCE ANALYSIS:")
total_improvement = 0
n_comparisons = 0

try:
    if 'nn_models' in globals():
        print("Improvements over original Neural Networks:")
        for position in tuned_nn_fast.keys():
            if position in nn_models:
                original_r2 = nn_models[position]['metrics']['test_r2']
                tuned_r2 = tuned_nn_fast[position]['metrics']['test_r2']
                improvement = ((tuned_r2 - original_r2) / original_r2) * 100
                total_improvement += improvement
                n_comparisons += 1
```

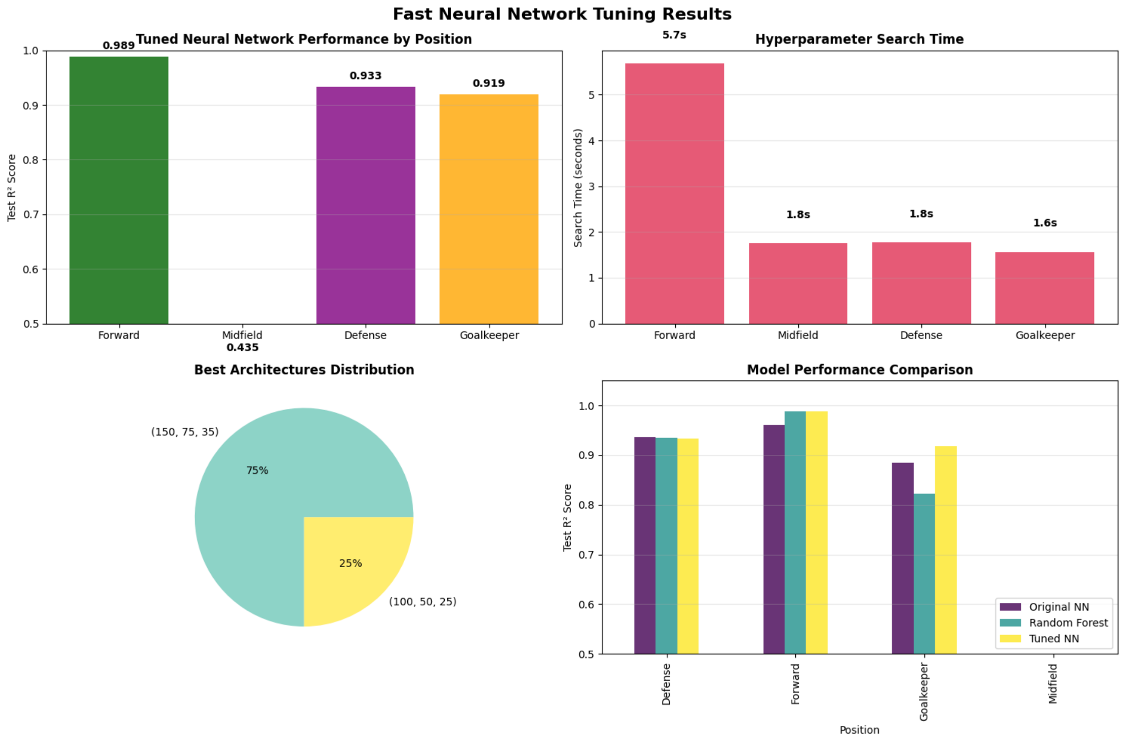
```
                print(f"  {position}: {improvement:+.1f}% (R²: {original_r2:.
↪3f} → {tuned_r2:.3f})")

        if n_comparisons > 0:
            avg_improvement = total_improvement / n_comparisons
            print(f"  Average improvement: {avg_improvement:+.1f}%")
except:
    print("No original models found for comparison")

print(f"\n  EFFICIENCY METRICS:")
print(f"  Average search time per position: {results_summary['Search_Time'].
↪mean():.1f}s")
print(f"  Total positions tuned: {len(tuned_nn_fast)}")
print(f"  Average final R² score: {results_summary['Test_R2'].mean():.3f}")

print(f"\n  Fast neural network tuning complete!")
print(f" Completed in {total_time:.1f}s instead of 104+ minutes!")
```

```
================================================================================
  FAST NEURAL NETWORK TUNING (5-MINUTE VERSION)
================================================================================
Starting fast neural network tuning…
Estimated total time: ~5 minutes

  Fast tuning for Forward…
   Testing 3 architectures…
     (100, 50): R² = 0.910
     (150, 75, 35): R² = 0.981
     (100, 50, 25): R² = 0.957
    Best architecture: (150, 75, 35)
   Random search with 15 combinations…
   Search completed in 5.7s
   Best CV score: 0.987
   Final Test R²: 0.989, MAE: 0.569
    Forward completed in 9.2s

  Fast tuning for Midfield…
   Testing 3 architectures…
     (100, 50): R² = 0.336
     (150, 75, 35): R² = 0.410
     (100, 50, 25): R² = 0.398
    Best architecture: (150, 75, 35)
   Random search with 15 combinations…
   Search completed in 1.8s
   Best CV score: 0.454
   Final Test R²: 0.435, MAE: 2.061
    Midfield completed in 3.9s
```

```
 Fast tuning for Defense…
  Testing 3 architectures…
    (100, 50): R² = 0.930
    (150, 75, 35): R² = 0.933
    (100, 50, 25): R² = 0.934
   Best architecture: (100, 50, 25)
  Random search with 15 combinations…
  Search completed in 1.8s
  Best CV score: 0.922
  Final Test R²: 0.933, MAE: 0.894
   Defense completed in 4.3s

 Fast tuning for Goalkeeper…
  Testing 3 architectures…
    (100, 50): R² = 0.854
    (150, 75, 35): R² = 0.933
    (100, 50, 25): R² = 0.819
   Best architecture: (150, 75, 35)
  Random search with 15 combinations…
```

```
/opt/miniconda3/lib/python3.12/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:691:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/miniconda3/lib/python3.12/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:691:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

```
  Search completed in 1.6s
  Best CV score: 0.886
  Final Test R²: 0.919, MAE: 0.356
   Goalkeeper completed in 2.8s

 TOTAL TUNING TIME: 20.2 seconds (0.3 minutes)

 Creating comparison charts…
```



FAST TUNING RESULTS SUMMARY

==

Position	Architecture	Activation	Alpha	Learning_Rate	Test_R2	Test_MAE	Search_Time	Iterations
Forward	(150, 75, 35)	tanh	0.010	0.009	0.989	0.569	5.685	49
Midfield	(150, 75, 35)	relu	0.009	0.003	0.435	2.061	1.753	49
Defense	(100, 50, 25)	relu	0.008	0.010	0.933	0.894	1.772	57
Goalkeeper	(150, 75, 35)	relu	0.006	0.006	0.919	0.356	1.565	70

PERFORMANCE ANALYSIS:

Improvements over original Neural Networks:

Forward: +2.8% (R^2: 0.962 → 0.989)
Midfield: +5.4% (R^2: 0.413 → 0.435)
Defense: -0.4% (R^2: 0.937 → 0.933)
Goalkeeper: +3.9% (R^2: 0.885 → 0.919)
Average improvement: +2.9%

EFFICIENCY METRICS:

Average search time per position: 2.7s

```
Total positions tuned: 4
Average final R² score: 0.819

Fast neural network tuning complete!
Completed in 20.2s instead of 104+ minutes!
```

[ ]:

```

```