

Project-1-1 - Salem Buhluli-262000880

FIFA World Cup 2026 - Player Performance Analysis

Assignment tasks:

1. Handle missing values
2. Handle outliers
3. Perform data visualization

Dataset: `fifa_world_cup_2026_player_performance.csv` (from Kaggle)

```
In [1]: import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

DATA = 'Desktop/midocean/'
```

```
In [2]: df = pd.read_csv(DATA + 'fifa_world_cup_2026_player_performance.csv')
df.head()
```

```
Out[2]:
```

	player_id	player_name	age	nationality	team	jersey_number	position	height_cm
0	P00055	Rodri Fati	26	Spanish	Spain	3	Goalkeeper	195
1	P00070	Ansu Le Normand	19	Spanish	Spain	18	Midfielder	178
2	P00066	Gavi Ramos	18	Spanish	Spain	14	NaN	177
3	P00073	Pedro Cubarsi	20	Spanish	Spain	21	Forward	182
4	P00059	Alvaro Oyarzabal	23	Spanish	Spain	7	Defender	191

5 rows × 75 columns

```
In [3]: print('Shape:', df.shape)
df.dtypes.value_counts()
```

Shape: (54600, 75)

```
Out[3]: int64      42
float64     19
object      14
Name: count, dtype: int64
```

1) Handle Missing Values

```
In [4]: # Count of missing values per column (only columns that have any)
missing = df.isnull().sum()
missing[missing > 0].sort_values(ascending=False)
```

```
Out[4]: market_value_eur    5460
pass_accuracy             4368
preferred_foot            3822
top_speed_kmh             3276
position                  2730
dtype: int64
```

```
In [5]: # Drop columns with more than 30% missing values
missing_percentage = df.isnull().mean() * 100
threshold = 30
columns_to_drop = missing_percentage[missing_percentage > threshold].index
df = df.drop(columns=columns_to_drop)
print(f"Columns dropped (>30% missing): {list(columns_to_drop)}")
print(f"Remaining columns: {df.shape[1]}")
```

```
Columns dropped (>30% missing): []
Remaining columns: 75
```

```
In [6]: # Fill numeric columns with the median, categorical columns with the mode
for column in df.select_dtypes(include=['float64', 'int64']).columns:
    df[column] = df[column].fillna(df[column].median())

for column in df.select_dtypes(include=['object']).columns:
    df[column] = df[column].fillna(df[column].mode()[0])

print('Total missing values after handling:', df.isnull().sum().sum())
```

```
Total missing values after handling: 0
```

Create a classification target

The dataset has no ready label, so we derive a binary **performance class** (1 = High performer , 0 = Regular) by splitting on the median player rating.

```
In [7]: median_rating = df['player_rating'].median()
df['classification'] = (df['player_rating'] >= median_rating).astype(int)
print('Median player rating:', round(median_rating, 2))
print(df['classification'].value_counts())
```

```
Median player rating: 5.5
classification
1    27621
0    26979
Name: count, dtype: int64
```

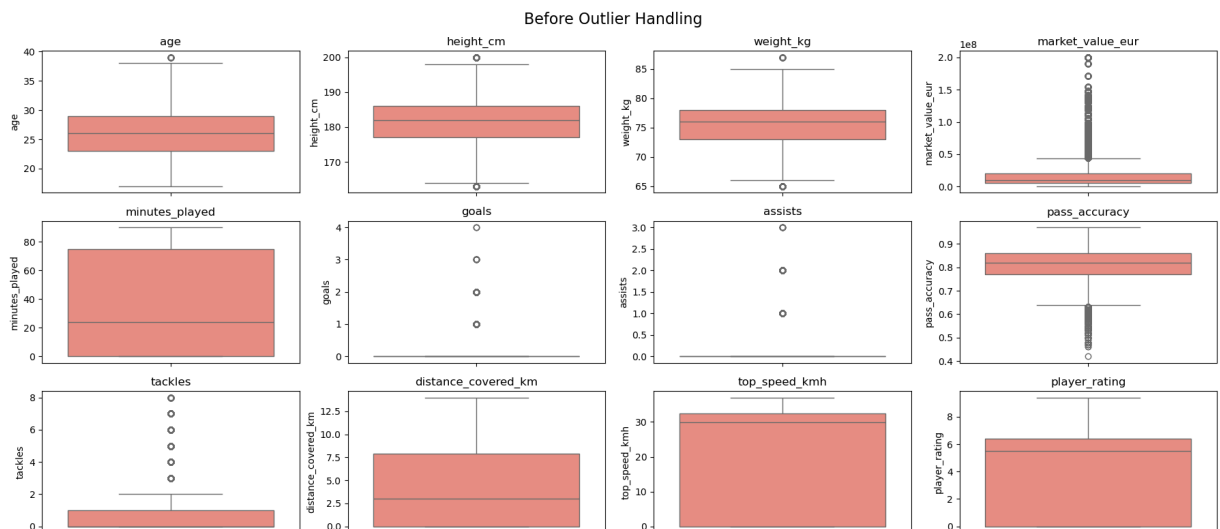
2) Handle Outliers (IQR capping)

We focus on the main numeric performance features.

```
In [8]: num_cols = ['age', 'height_cm', 'weight_kg', 'market_value_eur', 'minutes_played',  
                  'goals', 'assists', 'pass_accuracy', 'tackles',  
                  'distance_covered_km', 'top_speed_kmh', 'player_rating']  
num_cols = [col for col in num_cols if col in df.columns]  
num_cols
```

```
Out[8]: ['age',  
         'height_cm',  
         'weight_kg',  
         'market_value_eur',  
         'minutes_played',  
         'goals',  
         'assists',  
         'pass_accuracy',  
         'tackles',  
         'distance_covered_km',  
         'top_speed_kmh',  
         'player_rating']
```

```
In [9]: # Boxplots BEFORE handling outliers  
plt.figure(figsize=(18, 8))  
for i, col in enumerate(num_cols):  
    plt.subplot(3, 4, i + 1)  
    sns.boxplot(y=df[col], color='salmon')  
    plt.title(col)  
plt.suptitle('Before Outlier Handling', fontsize=16)  
plt.tight_layout()  
plt.show()
```



```
In [10]: # Cap outliers using the IQR method (winsorization)  
def cap_outliers_iqr(data, cols):  
    for col in cols:  
        Q1 = data[col].quantile(0.25)  
        Q3 = data[col].quantile(0.75)  
        IQR = Q3 - Q1  
        lower, upper = Q1 - 1.5 * IQR, Q3 + 1.5 * IQR
```

```

        data[col] = data[col].clip(lower, upper)
    return data

df = cap_outliers_iqr(df, num_cols)
print('Outliers capped for:', num_cols)

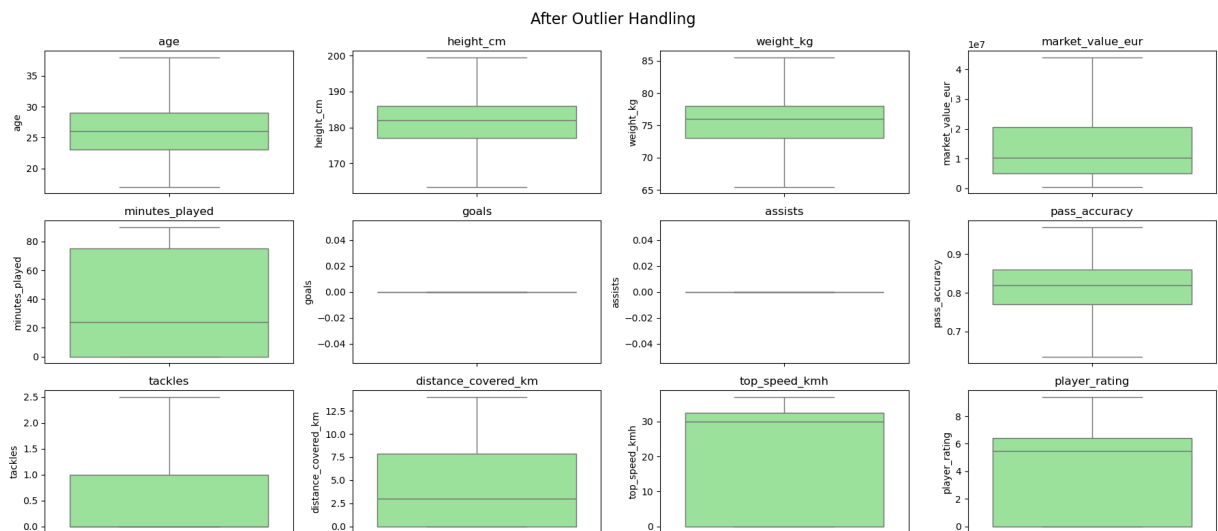
```

Outliers capped for: ['age', 'height_cm', 'weight_kg', 'market_value_eur', 'minutes_played', 'goals', 'assists', 'pass_accuracy', 'tackles', 'distance_covered_km', 'top_speed_kmh', 'player_rating']

```

In [11]: # Boxplots AFTER handling outliers
plt.figure(figsize=(18, 8))
for i, col in enumerate(num_cols):
    plt.subplot(3, 4, i + 1)
    sns.boxplot(y=df[col], color='lightgreen')
    plt.title(col)
plt.suptitle('After Outlier Handling', fontsize=16)
plt.tight_layout()
plt.show()

```

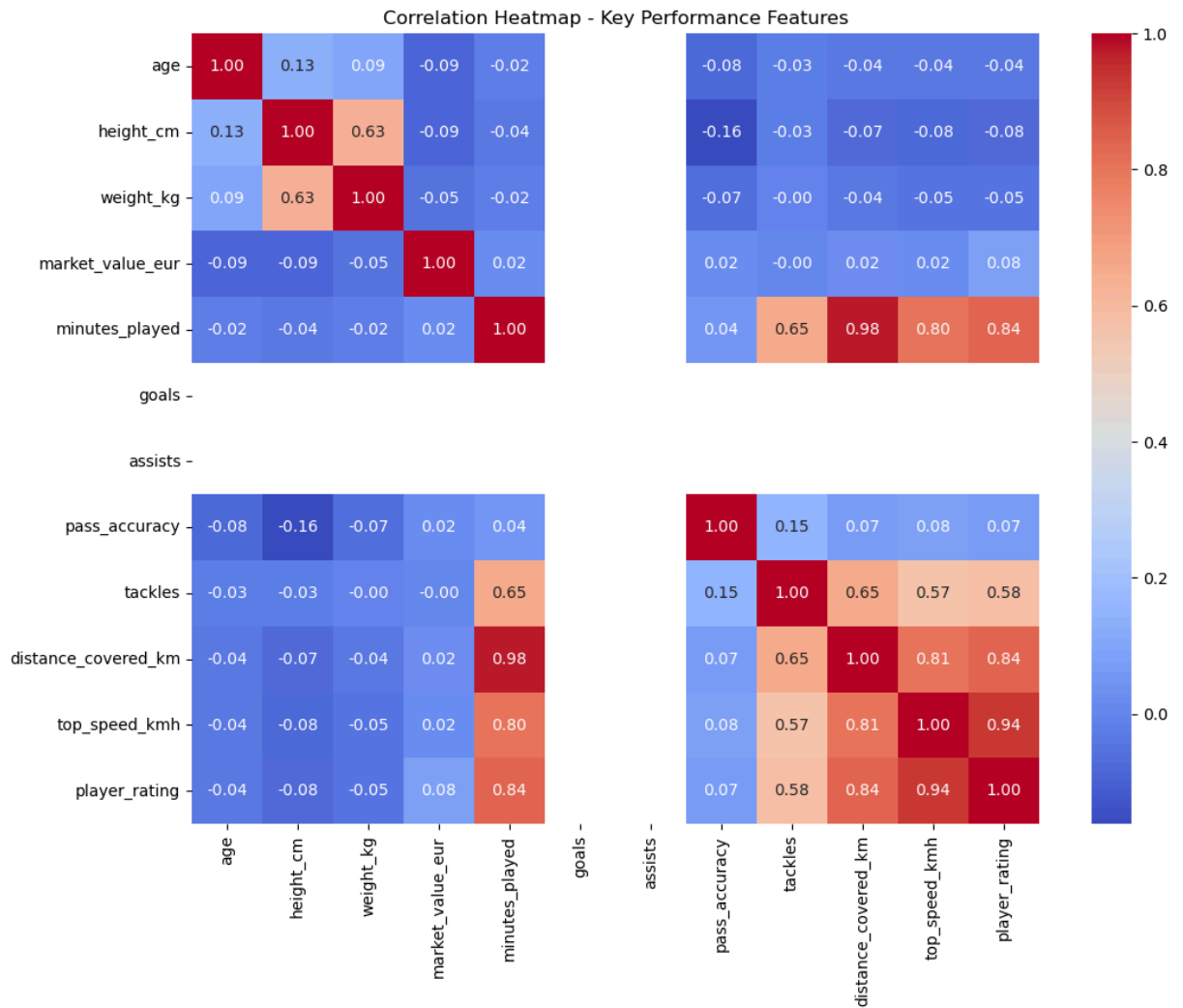


3) Data Visualization

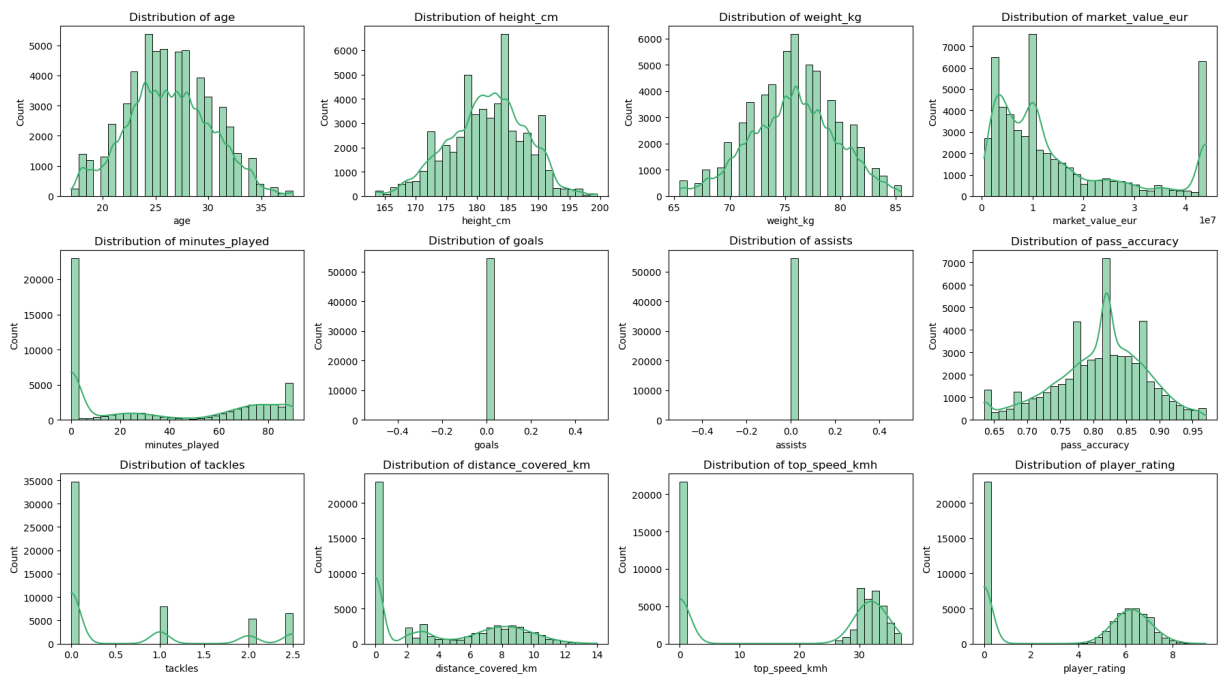
```

In [12]: # Correlation heatmap of the main numeric features
plt.figure(figsize=(12, 9))
sns.heatmap(df[num_cols].corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap - Key Performance Features')
plt.show()

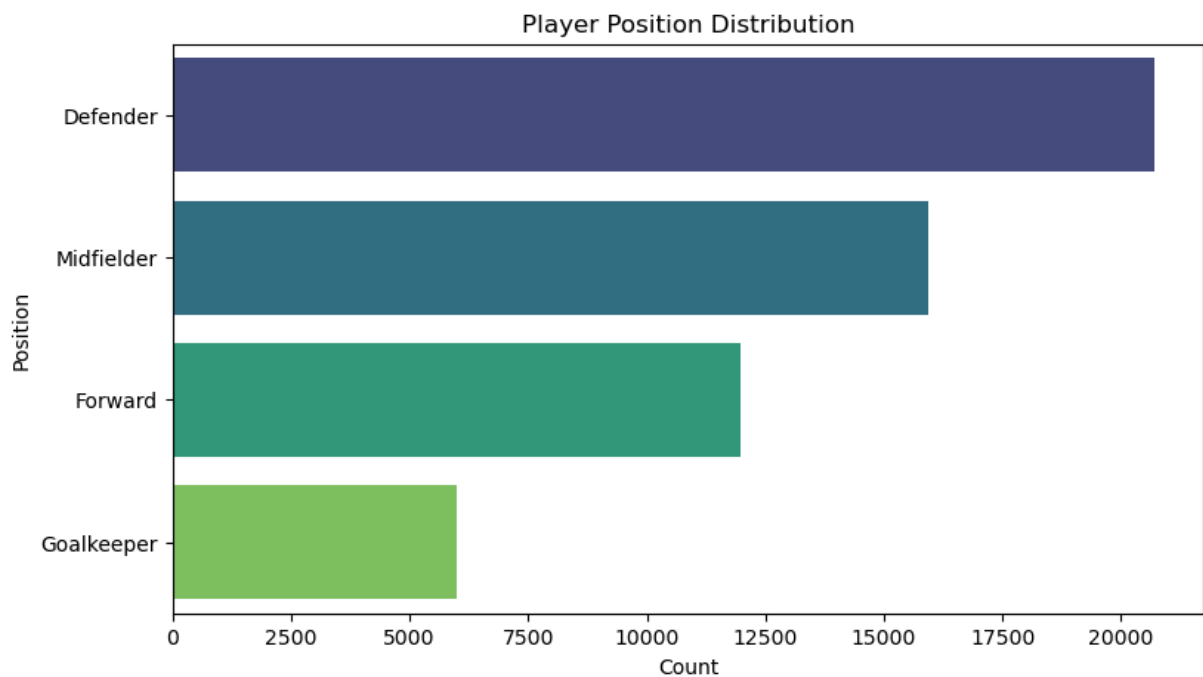
```



```
In [13]: # Distributions of key numeric features
plt.figure(figsize=(18, 10))
for i, col in enumerate(num_cols):
    plt.subplot(3, 4, i + 1)
    sns.histplot(df[col], kde=True, bins=30, color='mediumseagreen')
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```

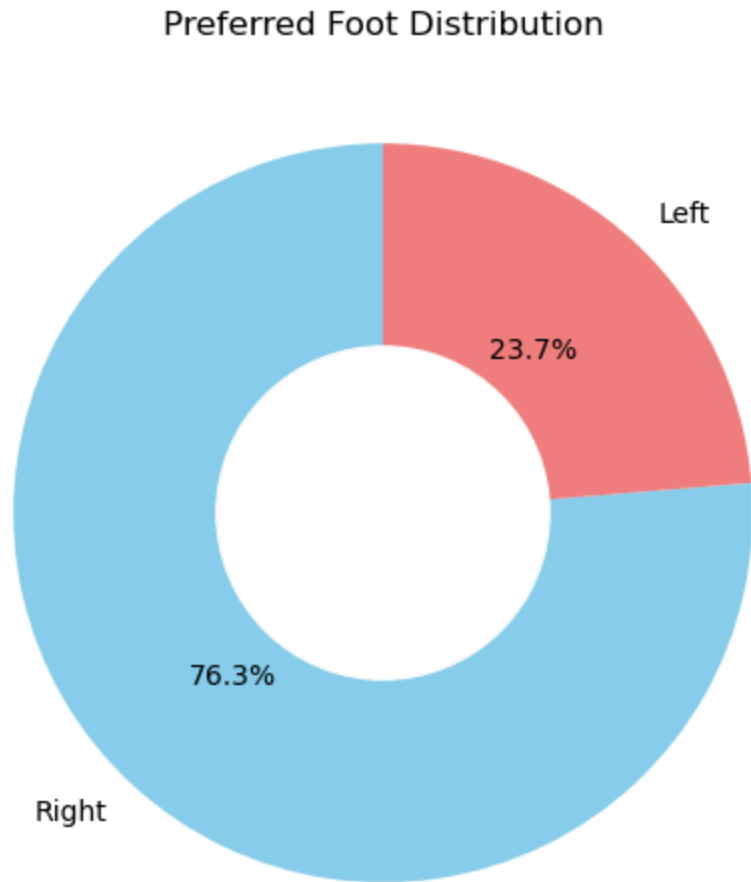


```
In [14]: # Player position distribution (count plot)
plt.figure(figsize=(9, 5))
order = df['position'].value_counts().index
sns.countplot(y='position', data=df, order=order, palette='viridis')
plt.title('Player Position Distribution')
plt.xlabel('Count')
plt.ylabel('Position')
plt.show()
```

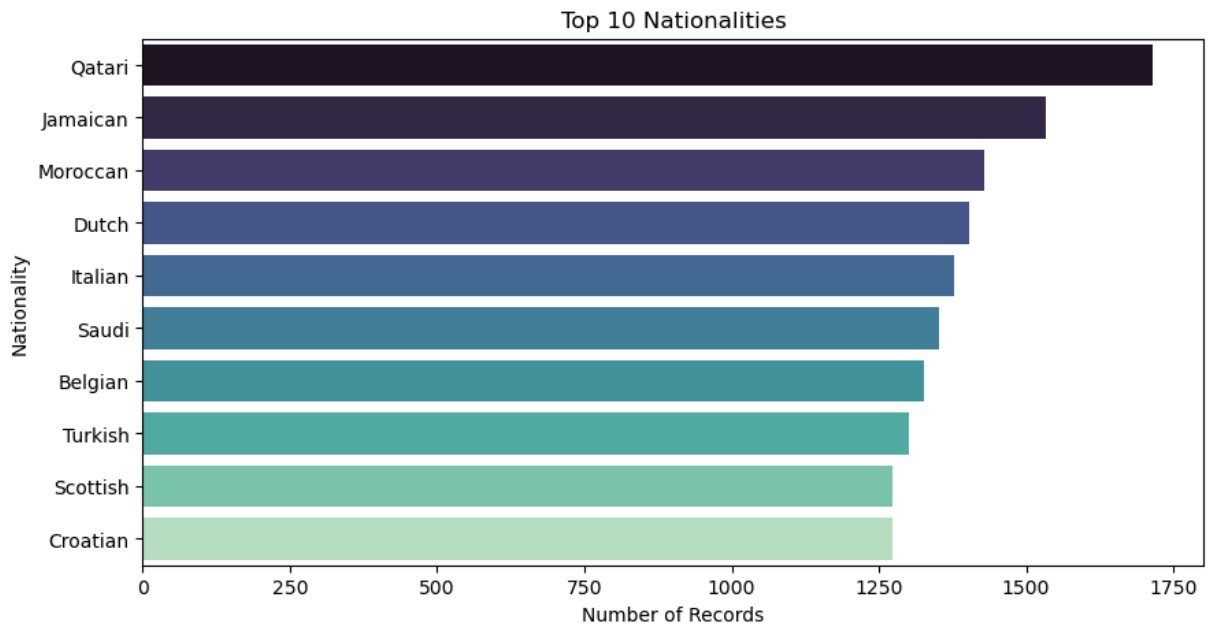


```
In [15]: # Preferred foot distribution (pie / donut chart)
foot_counts = df['preferred_foot'].value_counts()
plt.figure(figsize=(6, 6))
plt.pie(foot_counts, labels=foot_counts.index, autopct='%1.1f%%',
        startangle=90, colors=['skyblue', 'lightcoral', 'lightgreen'])
```

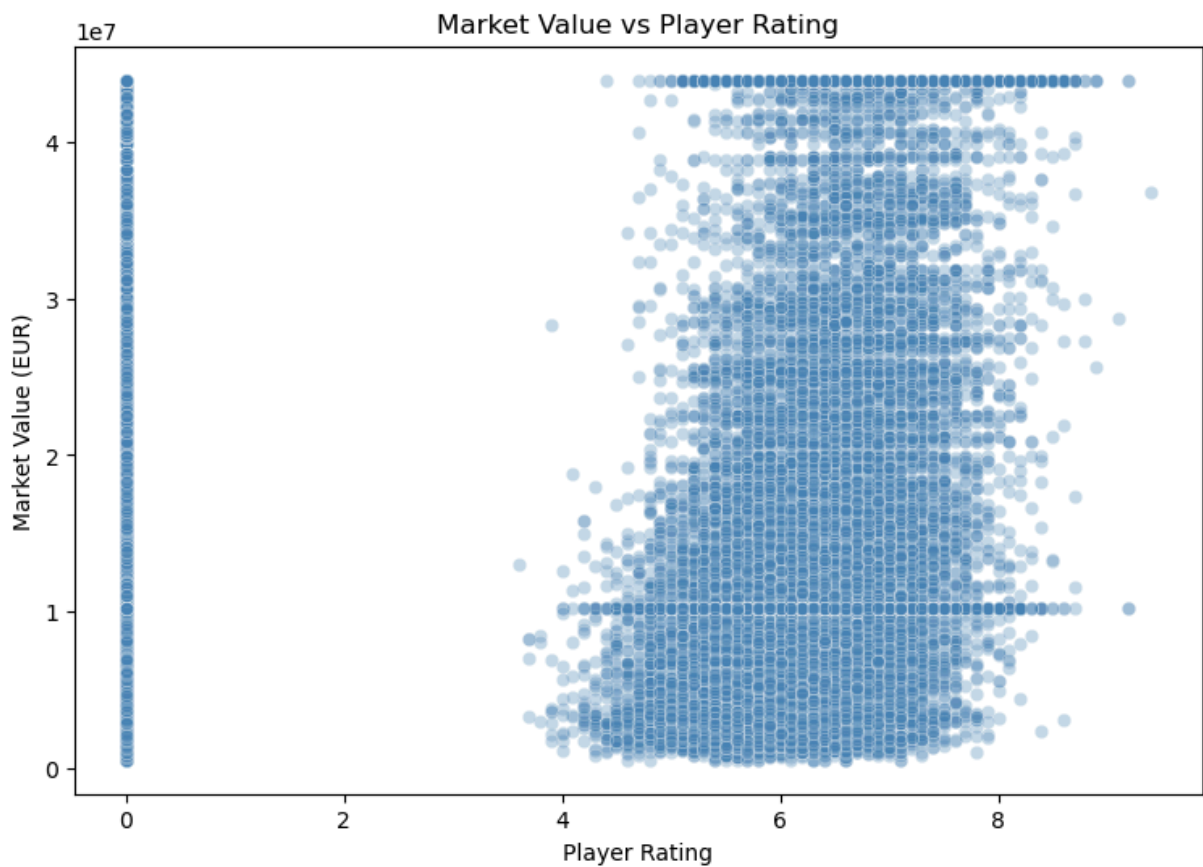
```
centre = plt.Circle((0, 0), 0.45, color='white')
plt.gca().add_artist(centre)
plt.title('Preferred Foot Distribution')
plt.show()
```



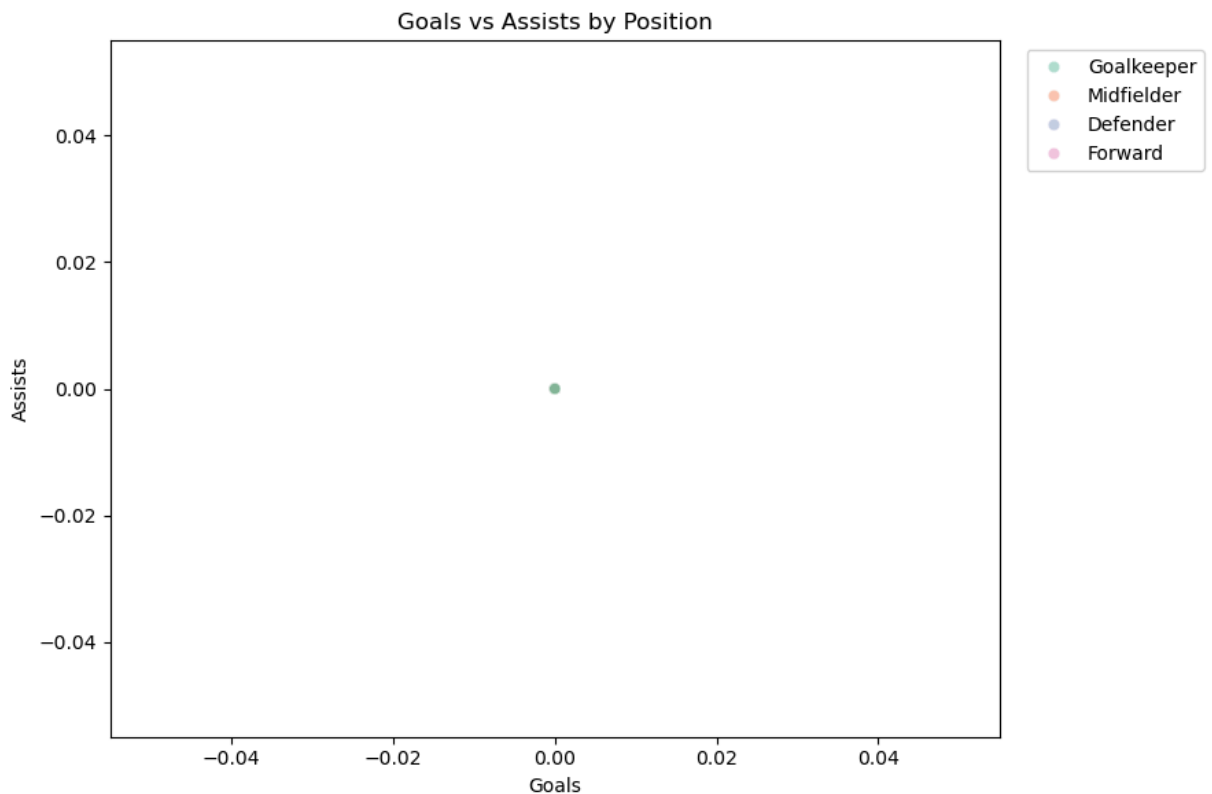
```
In [16]: # Top 10 nationalities by number of player records
plt.figure(figsize=(10, 5))
top_nat = df['nationality'].value_counts().head(10)
sns.barplot(x=top_nat.values, y=top_nat.index, palette='mako')
plt.title('Top 10 Nationalities')
plt.xlabel('Number of Records')
plt.ylabel('Nationality')
plt.show()
```



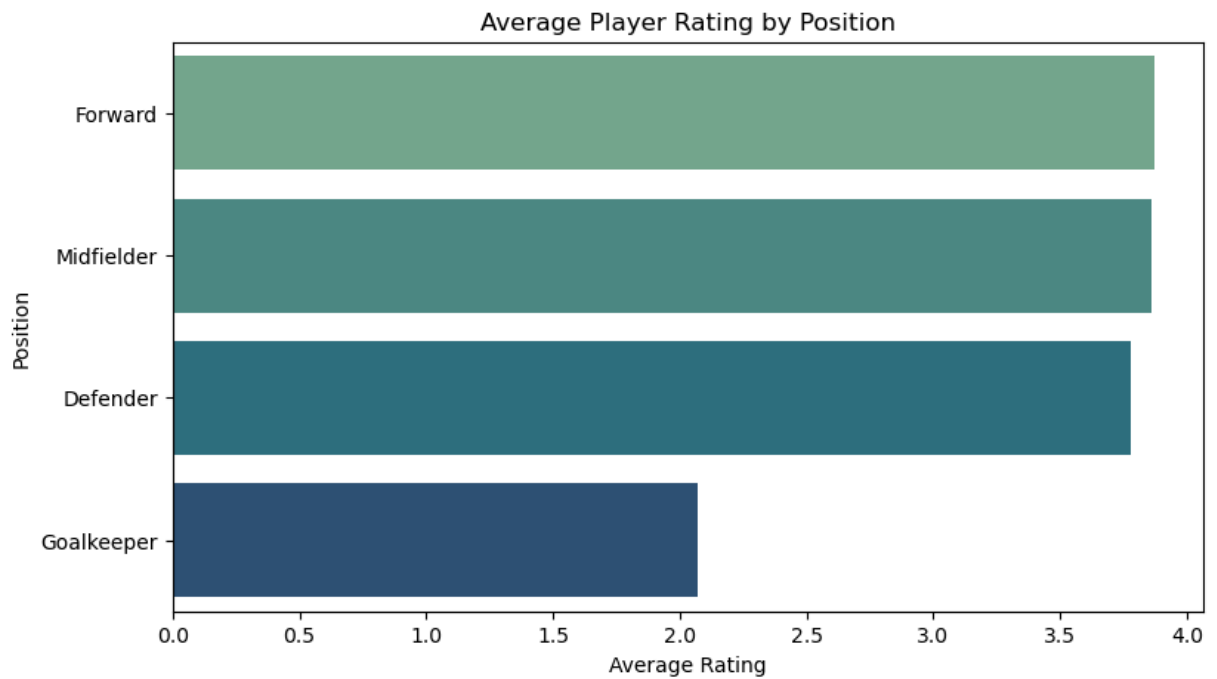
```
In [17]: # Market value vs Player rating (scatter)
plt.figure(figsize=(9, 6))
sns.scatterplot(x='player_rating', y='market_value_eur', data=df,
                alpha=0.3, color='steelblue')
plt.title('Market Value vs Player Rating')
plt.xlabel('Player Rating')
plt.ylabel('Market Value (EUR)')
plt.show()
```



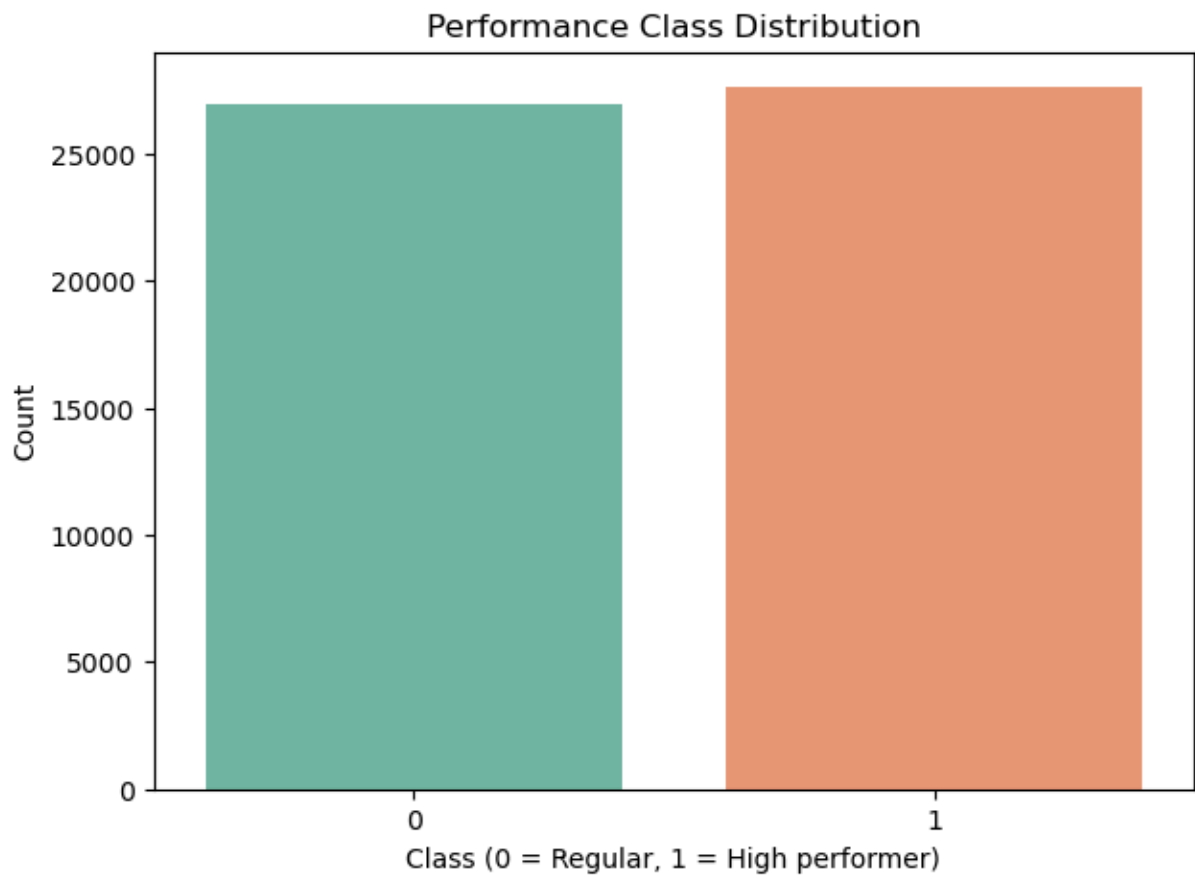
```
In [18]: # Goals vs Assists colored by position
plt.figure(figsize=(9, 6))
sns.scatterplot(x='goals', y='assists', data=df, hue='position',
                alpha=0.5, palette='Set2')
plt.title('Goals vs Assists by Position')
plt.xlabel('Goals')
plt.ylabel('Assists')
plt.legend(bbox_to_anchor=(1.02, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



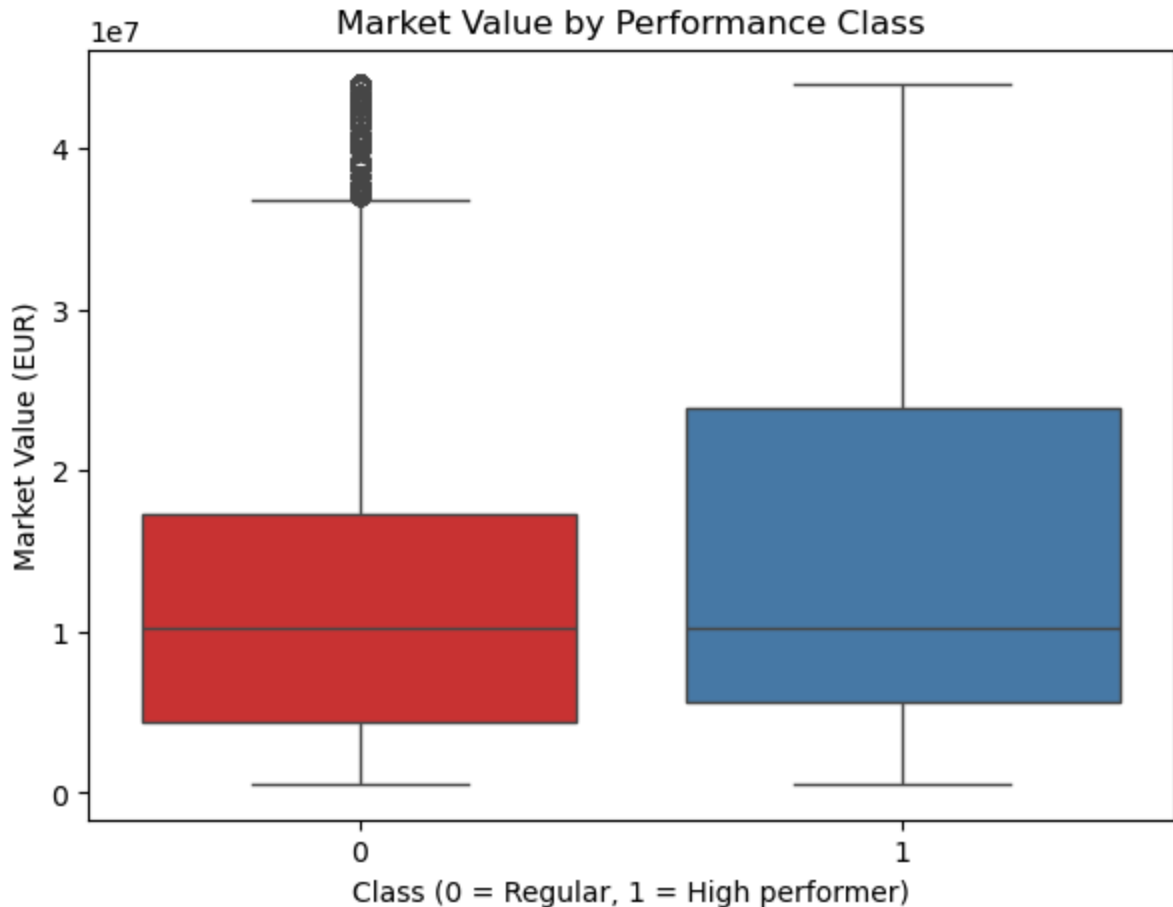
```
In [19]: # Average player rating by position (bar chart)
plt.figure(figsize=(9, 5))
pos_rating = df.groupby('position')['player_rating'].mean().sort_values(ascending=False)
sns.barplot(x=pos_rating.values, y=pos_rating.index, palette='crest')
plt.title('Average Player Rating by Position')
plt.xlabel('Average Rating')
plt.ylabel('Position')
plt.show()
```



```
In [20]: # Performance class distribution (0 = Regular, 1 = High performer)
plt.figure(figsize=(7, 5))
sns.countplot(x='classification', data=df, palette='Set2')
plt.title('Performance Class Distribution')
plt.xlabel('Class (0 = Regular, 1 = High performer)')
plt.ylabel('Count')
plt.show()
```



```
In [21]: # Market value by performance class
plt.figure(figsize=(7, 5))
sns.boxplot(x='classification', y='market_value_eur', data=df, palette='Set1')
plt.title('Market Value by Performance Class')
plt.xlabel('Class (0 = Regular, 1 = High performer)')
plt.ylabel('Market Value (EUR)')
plt.show()
```



```
In [22]: # Goals vs Assists colored by performance class
plt.figure(figsize=(9, 6))
sns.scatterplot(x='goals', y='assists', data=df, hue='classification',
                alpha=0.4, palette='Set1')
plt.title('Goals vs Assists by Performance Class')
plt.xlabel('Goals')
plt.ylabel('Assists')
plt.legend(title='Class')
plt.show()
```

