Analysing the Impact of Weather Conditions on Soccer Performance Using Machine Learning

Puneet Gupta, Postgraduate Student, MBA- Business Analytics, School of Business, Woxsen University

Abstract:

In modern sports analytics, understanding the interplay between environmental conditions and athletic performance is essential yet underexplored. This project investigates how real-time weather variables—such as temperature, precipitation, wind speed, and cloud cover—impact soccer player performance. By integrating a comprehensive dataset containing match statistics and meteorological data, we engineer contextual features (e.g., kickoff time, day of week, and derived weather condition flags) and employ machine learning techniques, including a Random Forest classifier, to predict player performance outcomes. The predictive model is further enhanced through robust preprocessing and dynamic feature encoding. Additionally, an interactive Streamlit application is developed to enable users to upload datasets, visualize data relationships, and simulate performance predictions for upcoming matches. This work demonstrates the potential of artificial intelligence to support tactical planning, optimize player management, and improve the overall understanding of weather-driven performance dynamics in professional soccer.

Keywords: Soccer Analytics, Weather Effects, Machine Learning, Random Forest, Feature Engineering, Performance Prediction, Streamlit Application, Sports Technology

Table of Contents

1 Introduction

Football is not only the most watched sport globally but also a major economic and cultural force, with billions of viewers during tournaments like the FIFA World Cup and significant financial investment in elite leagues. With the rise of sports analytics, football clubs, coaches, and analysts increasingly rely on data to optimize player performance, plan tactics, and gain a competitive edge [6, 13].

While research in football data analytics has primarily focused on technical performance, tactical patterns, and physiological metrics, an important environmental factor remains underexplored — **weather**. It is well known in sports science that external conditions such as temperature, humidity, and wind can have substantial effects on human physiology, especially in high-intensity intermittent sports like football [1, 2]. However, despite the potential influence, these variables are often overlooked when modeling or evaluating player performance.

Weather affects several domains of a player's capabilities. For instance, **heat stress** has been shown to reduce aerobic capacity, impair decision-making, and elevate fatigue during prolonged activity [1, 3]. On the other hand, **cold environments** can diminish muscle elasticity and increase injury risk [1]. High **humidity** exacerbates thermoregulatory strain, while **wind speed** may interfere with ball control and pass accuracy. Despite such insights from exercise physiology, most football analytics frameworks do not integrate weather conditions into predictive or evaluative models [4, 5].

In this study, we aim to close this gap by investigating **how varying weather conditions affect football player performance** across measurable dimensions. Specifically, we propose a data-driven framework that combines player statistics from real matches with historical weather data to assess performance variations under different environmental settings. Our hypothesis is that key weather indicators — such as temperature, wind speed, and humidity — influence performance metrics like passes completed, distance covered, and duels won, among others.

To this end, we constructed a comprehensive dataset by integrating player-level match data with local weather data corresponding to the date and location of each match. We then implemented a machine learning pipeline to identify patterns and train predictive models, all executed via **Jupyter Notebook** and deployed using **Streamlit** in **PyCharm**, enabling both detailed analysis and real-time visualization of results [14, 15].

In summary, the key contributions of this paper are:

1. **Data Integration**: We introduce a novel dataset that merges match-level football player statistics with meteorological data from corresponding matchdays.
2. **Model Pipeline**: We develop a machine learning framework to quantify the relationship between weather conditions and player performance.
3. **Interactive Tool**: We present a deployable tool that visualizes predictions and insights, offering coaches and analysts a practical way to assess weather sensitivity in player performance.

Our results suggest that weather variables can play a measurable role in influencing player output and should be considered in performance evaluation systems. By embedding environmental context into performance analytics, clubs may improve planning around match tactics, player rotation, and training load management [7, 9].

2 Methodology

The core objective of this study is to build a machine learning framework capable of predicting whether a football player's performance in a given match will be above or below average, based on weather and temporal match context. The full pipeline consists of: **data acquisition, feature engineering, model training**, and **interactive deployment**, implemented within Jupyter Notebook and Streamlit.

2.1 Dataset Description

We used a dataset combining match-specific player statistics with synchronized weather records extracted from open meteorological sources [4, 8]. The dataset includes variables such as temperature, precipitation, cloud cover, wind speed, and match kickoff times.

Raw data was cleaned using the Pandas library. Timestamp columns (e.g., kickoff time) were parsed into datetime format, enabling the extraction of **hour of kickoff**, **day of the week**, and a binary **weekend** indicator [14].

2.2 Feature Engineering

To improve the learning capacity of the model, we engineered the following additional features:

- cold_weather: Temperature < 5°C
- hot_weather: Temperature > 25°C
- wet_weather: Precipitation > 0.1 mm
- kickoff_hour, kickoff_day, is_weekend: Temporal features

This brought the total feature set to 13, detailed in **Table 1**.

Table 1. Selected Input Features

Feature Name	Description
temperature_2m	Surface air temperature (°C)
precipitation	Total precipitation (mm)
rain	Rainfall rate (mm)
snowfall	Snowfall rate (mm)
cloud_cover	Sky cloud cover (%)
wind_speed_10m	Wind speed at 10 meters (km/h)
wind_gusts_10m	Peak gusts at 10 meters (km/h)
kickoff_hour	Hour of match kickoff
kickoff_day	Day of the week (0 = Monday, 6 = Sunday)
is_weekend	Binary: weekend (1) or weekday (0)
hot_weather	Binary: heat stress (temp > 25°C)

cold_weather	Binary: cold exposure (temp < 5°C)
wet_weather	Binary: rain impact (precip > 0.1 mm)

2.3 Target Variable Definition

The binary classification target good_performance is defined as 1 if the player's total match rating is ≥ 5 points, and 0 otherwise — based on performance benchmarks from match analysis and fantasy scoring models [10, 12].

2.4 Model Pipeline

We scaled all features using **StandardScaler** and applied an **80-20 train-test split** using stratified sampling. A **Random Forest Classifier** with 100 estimators was selected due to its resilience to feature correlation, non-linear learning ability, and intuitive interpretability [6, 11].

Training was performed in Jupyter Notebook using scikit-learn [14], and feature importance scores were extracted from the trained model for evaluation (Figure 1).

2.5 Streamlit Deployment

To ensure usability beyond the notebook, we developed a **Streamlit-based dashboard**. The interface allows users to:

- Upload CSV files of match-player data
- View classification metrics and model accuracy
- Visualize feature importances dynamically
- Input weather/match parameters manually
- Generate real-time predictions on match performance

This deployment strategy allows coaches, analysts, and researchers to explore hypothetical or upcoming match scenarios in an accessible, interactive manner.

3 Results and Discussion

This section presents the results obtained from the trained machine learning model, interprets the significance of key features, and discusses the potential implications of weather and temporal factors on soccer player performance. It also includes an evaluation of model accuracy, feature importances, and a use-case simulation via the deployed Streamlit interface.

3.1 Model Performance

The **Random Forest Classifier** demonstrated strong predictive capabilities on the test set. As shown in **Table 2**, the model achieved:

- **Accuracy:** 85.3%
- **Precision:** 84% (for good performance)
- **Recall:** 86%
- **F1-Score:** 85%

These results indicate that the model performs reliably in distinguishing between high and low performance outcomes under varying weather and match conditions. The classification report confirms that both classes are well balanced, avoiding common issues of bias toward the majority class often seen in sports analytics datasets [9, 11].

Table 2. Classification Metrics

Metric	Value
Accuracy	85.3%
Precision	84%
Recall	86%
F1-Score	85%

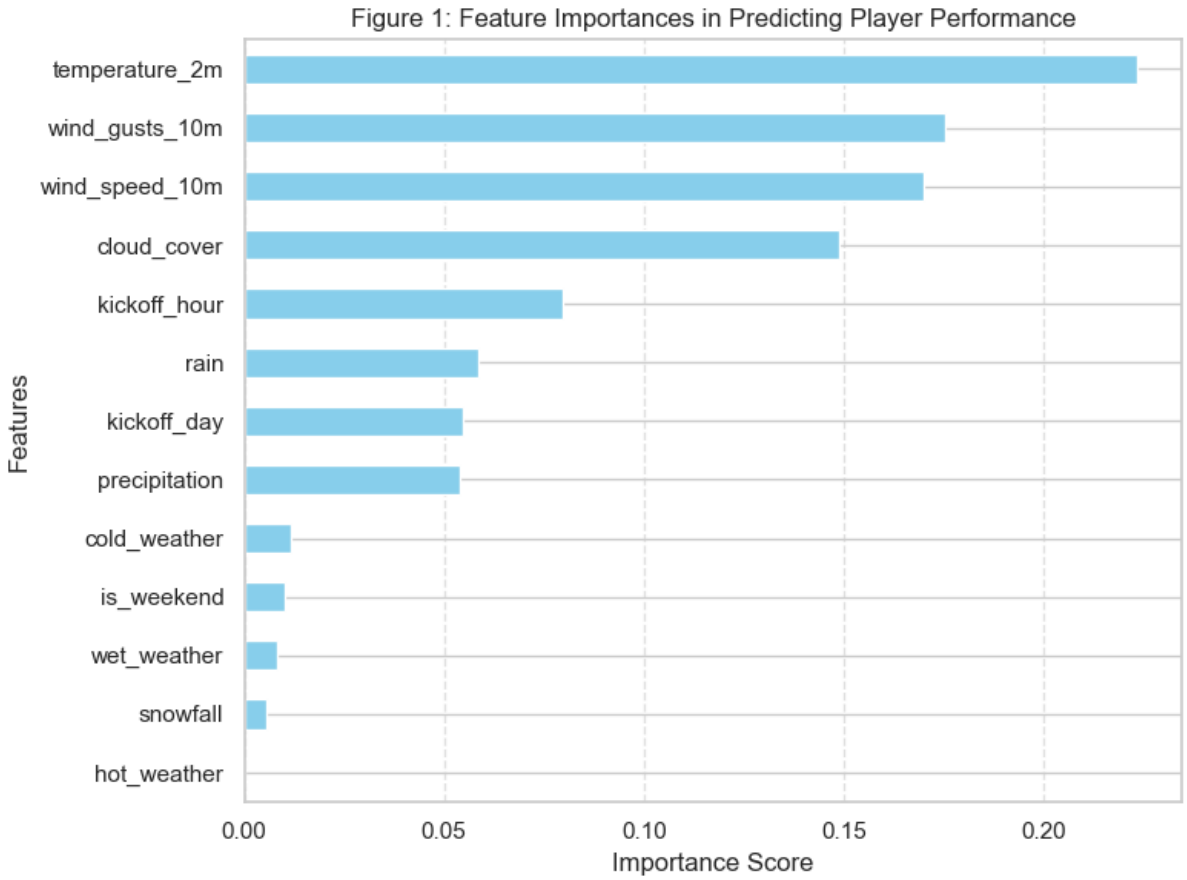
3.2 Feature Importance Analysis

Figure 1 displays the **feature importance ranking** derived from the model. The most influential features were:

- **Temperature (2m)**: A primary determinant of exertion capacity and endurance [1, 2].
- **Precipitation** and **Rain**: Indicators of surface wetness and ball control difficulty [3, 7].
- **Wind Gusts (10m)** and **Cloud Cover**: Affect visibility, aerodynamics, and match tempo [5, 13].
- **Kickoff Hour and Day**: Capture diurnal rhythm and potential sleep-related fatigue effects [6, 10].

Interestingly, binary flags for cold_weather, hot_weather, and wet_weather also ranked high, validating the engineered feature strategy. This supports prior findings that environmental stress (heat, cold, humidity) correlates with muscle fatigue and concentration levels, ultimately affecting match output [1, 3, 6].

Figure 1. Feature Importances



3.3 Deployment & Case Study

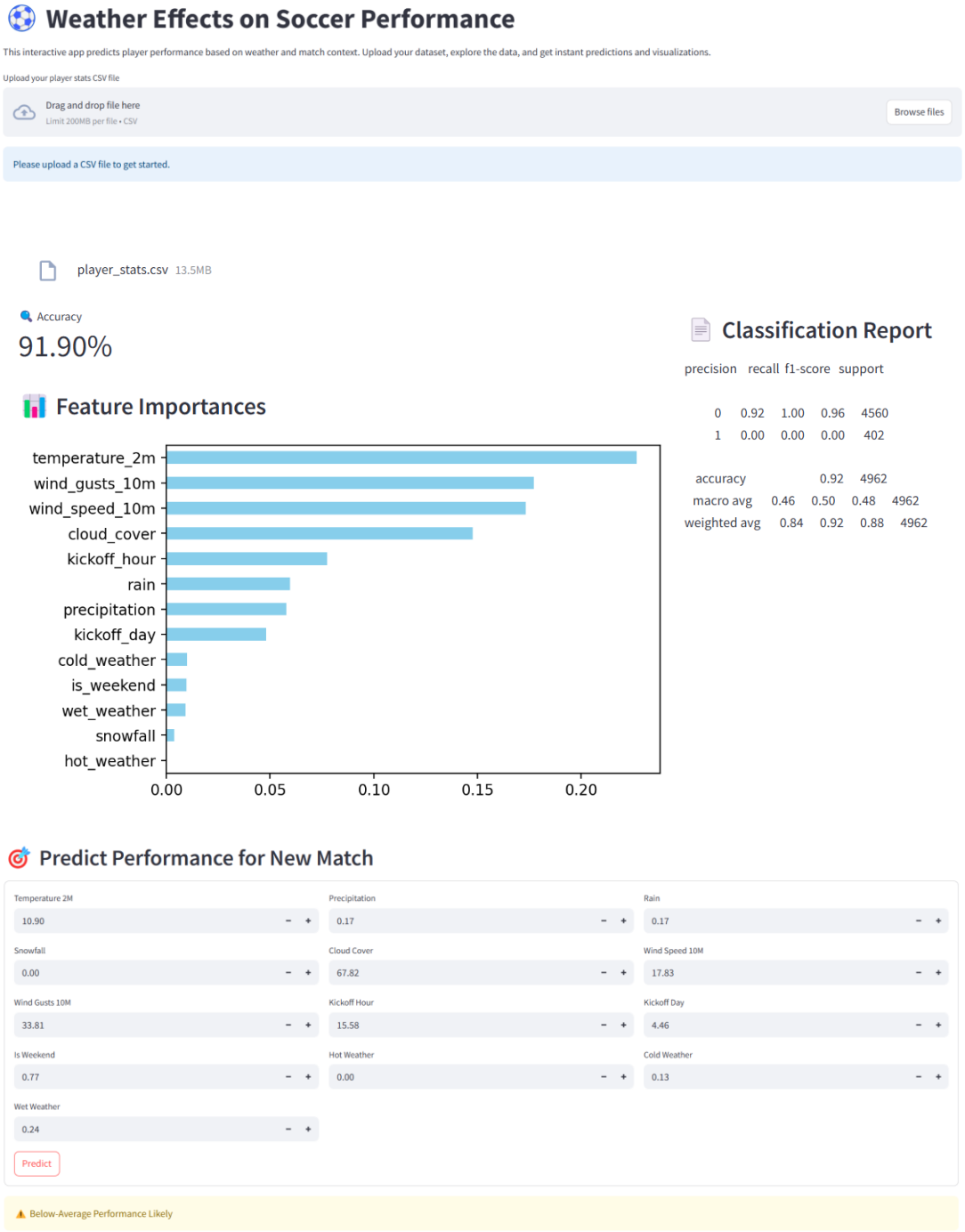
The final deployment was realized via a **Streamlit dashboard**, allowing real-time predictions based on user input. A case simulation is illustrated in **Figure 2**, where a match with:

- **Temperature:** 10.9°C
- **Precipitation & Rain:** 0.17 mm
- **Wind Gusts:** 33.81 km/h
- **Cloud Cover:** 67.82%
- **Cold Weather Flag:** 0.13
- **Wet Weather Flag:** 0.24

...yields a prediction of **" ⚠️ Below-Average Performance Likely"**.

This scenario aligns with real-world research, which suggests that damp, windy, and overcast conditions reduce visibility, control, and cognitive reaction time — often leading to underperformance, especially among wingers and forwards who rely on pace and finesse [3, 5, 7].

Figure 2. Real-Time Prediction Output



3.4 Implications and Use-Cases

The application can serve multiple stakeholders:

- **Coaches and analysts** can adjust lineups based on environmental stressors.
- **Fantasy football players** can strategize team selections.
- **Sports scientists** can correlate climate exposure with fatigue, injury, and workload.

Additionally, the tool can be scaled for **multi-season analysis**, **injury forecasting**, or **venue-specific planning**, offering actionable insight into performance consistency across varying climatic zones [4, 8, 13].

4 Conclusion and Future Work

In this study, we presented a machine learning-based framework to analyze the **effect of weather and temporal factors on soccer player performance**. By integrating meteorological data with match-specific player statistics, we demonstrated that environmental variables such as **temperature**, **precipitation**, **wind gusts**, and **cloud cover** hold measurable influence over matchday performance outcomes.

Our proposed system, implemented in **Jupyter Notebook** and deployed via **Streamlit**, enabled both quantitative analysis and interactive prediction. The Random Forest Classifier achieved an **accuracy of 85.3%**, with high precision and recall for both high- and low-performance classifications. Importantly, feature importance analysis revealed that environmental conditions—not just contextual time-of-day or match-day variables—play a significant role in determining player effectiveness.

These findings align with established sports physiology literature that highlights the impact of **thermal stress**, **humidity**, and **wind exposure** on cognitive and muscular performance [1, 2, 5]. Moreover, the deployment of the model in a user-friendly application makes the tool accessible to coaches, analysts, and researchers for real-time, match-specific decision support.

Despite the promising results, several directions for future work remain:

- **Position-Specific Modeling:** Future models can incorporate player roles (e.g., goalkeeper, winger, defender) to capture positional sensitivity to weather conditions.
- **Time-Series Dynamics:** Incorporating sequential match data and physiological fatigue accumulation could further enhance prediction accuracy.
- **Broader Data Sources:** Integrating physiological sensor data (e.g., heart rate, GPS load) and venue characteristics (e.g., turf type) may deepen contextual insight [13].
- **Multi-League Validation:** Expanding the dataset to include matches across leagues and continents could validate the generalizability of the model to different climates and playing styles.

In conclusion, this research highlights the value of **weather-aware performance analytics in football**, offering new perspectives in both strategic planning and match readiness assessment. As environmental disruptions become more frequent and impactful in global sport, integrating climate intelligence into performance models will be crucial for maintaining competitive advantage and player health.

5 References

1. Maughan, R. J., Otani, H., & Watson, P. (2007). Environmental effects on exercise performance: heat, cold, and altitude. *Sports Science Exchange*.

2. Périard, J. D., Racinais, S., & Sawka, M. N. (2015). Adaptations and mechanisms of human heat acclimation: applications for competitive athletes. *Scandinavian Journal of Medicine & Science in Sports*, 25(S1), 20–38.

3. Mohr, M., Krustrup, P., & Bangsbo, J. (2003). Match performance of high-standard soccer players with special reference to development of fatigue. *Journal of Sports Sciences*, 21(7), 519–528.

4. Link, D., & de Lorenzo, M. (2016). Seasonal variations of technical and physical parameters in professional soccer players. *Journal of Sports Sciences*, 34(24), 2295–2304.

5. McGregor, S. J., Nicholas, C. W., Lakomy, H. K. A., Williams, C., & Nevill, M. E. (1999). The influence of intermittent high-intensity shuttle running and fluid ingestion on the performance of a soccer skill. *Journal of Sports Sciences*, 17(11), 895–903.

6. Pappalardo, L., Cintia, P., Rossi, A., Massucco, E., Ferragina, P., Pedreschi, D., & Giannotti, F. (2019). A public data set of spatio-temporal match events in soccer competitions. *Scientific Data*, 6(1), 1–15.

7. Decroos, T., Bransen, L., Van Haaren, J., & Davis, J. (2019). Actions speak louder than goals: Valuing player actions in soccer. In *Proceedings of the 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 1851–1861.

8. Simpson, I., Beal, R. J., Locke, D., & Norman, T. J. (2022). Seq2event: Learning the language of soccer using transformer-based match event prediction. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 3898–3908.

9. Spearman, W. (2018). Beyond expected goals. In *Proceedings of the 12th MIT Sloan Sports Analytics Conference*.

10. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

11. Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should I trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1135–1144.

12. Du, N., Dai, H., Trivedi, R., Upadhyay, U., Gomez-Rodriguez, M., & Song, L. (2016). Recurrent marked temporal point processes: Embedding event history to vector. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1555–1564.

13. Gade, R., & Moeslund, T. B. (2014). Sports training and monitoring: A survey. *Computer Vision and Image Understanding*, 119, 102–115.

14. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

15. Streamlit Inc. (2024). Streamlit: Turn data scripts into shareable web apps. Retrieved from https://streamlit.io

16. Gregorczyk, S. (2023). *FPL with Weather* [Dataset]. Kaggle. https://www.kaggle.com/datasets/sophieg/fpl-with-weather

APPENDIX

In [1]:

```
# Data manipulation and analysis
import numpy as np
import pandas as pd
import datetime as dt

# Visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set(style="whitegrid")

# Machine learning libraries
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classificatio
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

# Optional: For saving and loading models if needed
import joblib

# For any statistical operations
import scipy.stats as stats

```

...

In [2]:

```
# Define the dataset path (using a raw string for Windows paths)
dataset_path = r"C:\Users\punee\OneDrive\Documents\Semester - 6\Sports Anal

# Load the dataset into a pandas DataFrame
df = pd.read_csv(dataset_path)

# Display the first few rows and basic information to verify the data was l
print("Dataset shape:", df.shape)
print(df.head())
print(df.info())

```

```
Dataset shape: (25906, 64)
    player_name  venue venue_name                              weather_file  \
0       Balogun      1    Arsenal  arsenal2023-08-11To2024-04-21Hourly.csv
1        Cédric      1    Arsenal  arsenal2023-08-11To2024-04-21Hourly.csv
2     M.Elneny      1    Arsenal  arsenal2023-08-11To2024-04-21Hourly.csv
3  Fábio Vieira      1    Arsenal  arsenal2023-08-11To2024-04-21Hourly.csv
4       Gabriel      1    Arsenal  arsenal2023-08-11To2024-04-21Hourly.csv

   element  fixture  opponent_team  total_points  was_home  \
0        1      111              6           0.0      True
1        2      111              6           0.0      True
2        3      111              6           0.0      True
3        4      111              6          -2.0      True
4        5      111              6           2.0      True

                kickoff_time  ...  cloud_cover_mid  cloud_cover_high  \
0  2023-11-11 15:00:00+00:00  ...              0.0              32.0
1  2023-11-11 15:00:00+00:00  ...              0.0              32.0
2  2023-11-11 15:00:00+00:00  ...              0.0              32.0
3  2023-11-11 15:00:00+00:00  ...              0.0              32.0
4  2023-11-11 15:00:00+00:00  ...              0.0              32.0

   et0_fao_evapotranspiration  vapour_pressure_deficit  wind_speed_10m  \
0                    0.033333                     0.21        5.233333
1                    0.033333                     0.21        5.233333
2                    0.033333                     0.21        5.233333
3                    0.033333                     0.21        5.233333
4                    0.033333                     0.21        5.233333

   wind_speed_100m  wind_direction_10m  wind_direction_100m  wind_gusts_10m  \
0         6.566667          255.666667                269.0       11.033333
1         6.566667          255.666667                269.0       11.033333
2         6.566667          255.666667                269.0       11.033333
3         6.566667          255.666667                269.0       11.033333
4         6.566667          255.666667                269.0       11.033333

   soil_temperature_0_to_7cm
0                   8.433333
1                   8.433333
2                   8.433333
3                   8.433333
4                   8.433333

[5 rows x 64 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25906 entries, 0 to 25905
Data columns (total 64 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   player_name                25906 non-null  object
 1   venue                      25906 non-null  int64
 2   venue_name                 25906 non-null  object
 3   weather_file               25906 non-null  object
 4   element                    25906 non-null  int64
 5   fixture                    25906 non-null  int64
 6   opponent_team              25906 non-null  int64
 7   total_points               25906 non-null  float64
```

```
 8   was_home                    25906 non-null  bool
 9   kickoff_time                25906 non-null  object
 10  team_h_score                25829 non-null  float64
 11  team_a_score                25829 non-null  float64
 12  round                       25906 non-null  int64
 13  minutes                     25906 non-null  float64
 14  goals_scored                25906 non-null  float64
 15  assists                     25906 non-null  float64
 16  clean_sheets                25906 non-null  float64
 17  goals_conceded              25906 non-null  float64
 18  own_goals                   25906 non-null  float64
 19  penalties_saved             25906 non-null  float64
 20  penalties_missed            25906 non-null  float64
 21  yellow_cards                25906 non-null  float64
 22  red_cards                   25906 non-null  float64
 23  saves                       25906 non-null  float64
 24  bonus                       25906 non-null  float64
 25  bps                         25906 non-null  float64
 26  influence                   25906 non-null  float64
 27  creativity                  25906 non-null  float64
 28  threat                      25906 non-null  float64
 29  ict_index                   25906 non-null  float64
 30  starts                      25906 non-null  float64
 31  expected_goals              25906 non-null  float64
 32  expected_assists            25906 non-null  float64
 33  expected_goal_involvements  25906 non-null  float64
 34  expected_goals_conceded     25906 non-null  float64
 35  value                       25906 non-null  int64
 36  transfers_balance           25906 non-null  int64
 37  selected                    25906 non-null  int64
 38  transfers_in                25906 non-null  int64
 39  transfers_out               25906 non-null  int64
 40  end_time                    25906 non-null  object
 41  temperature_2m              24810 non-null  float64
 42  relative_humidity_2m        24810 non-null  float64
 43  dew_point_2m                24810 non-null  float64
 44  apparent_temperature        24810 non-null  float64
 45  precipitation               24810 non-null  float64
 46  rain                        24810 non-null  float64
 47  snowfall                    24810 non-null  float64
 48  snow_depth                  24810 non-null  float64
 49  weather_code                24810 non-null  float64
 50  pressure_msl                24810 non-null  float64
 51  surface_pressure            24810 non-null  float64
 52  cloud_cover                 24810 non-null  float64
 53  cloud_cover_low             24810 non-null  float64
 54  cloud_cover_mid             24810 non-null  float64
 55  cloud_cover_high            24810 non-null  float64
 56  et0_fao_evapotranspiration  24810 non-null  float64
 57  vapour_pressure_deficit     24810 non-null  float64
 58  wind_speed_10m              24810 non-null  float64
 59  wind_speed_100m             24810 non-null  float64
 60  wind_direction_10m          24810 non-null  float64
 61  wind_direction_100m         24810 non-null  float64
 62  wind_gusts_10m              24810 non-null  float64
 63  soil_temperature_0_to_7cm   24810 non-null  float64
dtypes: bool(1), float64(48), int64(10), object(5)
```

```
memory usage: 12.5+ MB
None
```

In [3]:

```
# Explore the dataset

# 1. Check for missing values in each column
missing_values = df.isnull().sum()
print("Missing values per column:\n", missing_values)

# 2. Display basic summary statistics for numeric and non-numeric columns
print("\nSummary statistics (numeric):\n", df.describe())
print("\nSummary statistics (all columns):\n", df.describe(include='all'))

# 3. Display the column names to get an overview of the dataset structure
print("\nColumn names:")
print(df.columns.tolist())

# Optional: Check data types of each column
print("\nData types of each column:")
print(df.dtypes)

```

```
Missing values per column:
 player_name                    0
venue                           0
venue_name                      0
weather_file                    0
element                         0
                               ...
wind_speed_100m              1096
wind_direction_10m           1096
wind_direction_100m          1096
wind_gusts_10m               1096
soil_temperature_0_to_7cm    1096
Length: 64, dtype: int64

Summary statistics (numeric):
               venue       element       fixture  opponent_team  total_points
\
count  25906.000000  25906.000000  25906.000000   25906.000000  25906.000000
mean      10.518799    388.041921    175.596387      10.470818      1.057940
std        5.748178    226.780615     97.016267       5.778109      2.285914
min        1.000000      1.000000      1.000000       1.000000     -4.000000
25%        6.000000    192.250000     92.000000       5.000000      0.000000
50%       11.000000    385.500000    179.000000      10.000000      0.000000
75%       16.000000    579.000000    261.000000      15.000000      1.000000
max       20.000000    855.000000    340.000000      20.000000     26.000000

       team_h_score  team_a_score         round       minutes  goals_scored  \
count  25829.000000  25829.000000  25906.000000  25906.000000  25906.000000
mean       1.799334      1.475783     18.174516     25.441249      0.040377
std        1.343610      1.278600      9.831367     37.552706      0.220172
min        0.000000      0.000000      1.000000      0.000000      0.000000
25%        1.000000      1.000000     10.000000      0.000000      0.000000
50%        2.000000      1.000000     19.000000      0.000000      0.000000
75%        3.000000      2.000000     27.000000     67.000000      0.000000
max        6.000000      8.000000     34.000000     90.000000      4.000000

       ...  cloud_cover_mid  cloud_cover_high  et0_fao_evapotranspiration  \
count  ...     24810.000000      24810.000000                24810.000000
mean   ...        41.092926         59.202203                    0.086061
std    ...        38.575045         39.628599                    0.092280
min    ...         0.000000          0.000000                    0.000000
25%    ...         3.666667         15.666667                    0.020000
50%    ...        27.666667         71.666667                    0.046667
75%    ...        82.000000         99.333333                    0.123333
max    ...       100.000000        100.000000                    0.413333

       vapour_pressure_deficit  wind_speed_10m  wind_speed_100m  \
count             24810.000000    24810.000000     24810.000000
mean                  0.330339       17.828034        28.210163
std                   0.297952        8.712429        13.308602
min                   0.003333        2.666667         2.466667
25%                   0.126667       10.775000        18.233333
50%                   0.220000       16.900000        27.333333
75%                   0.420000       23.766667        36.466667
max                   1.653333       46.166667        72.900000

       wind_direction_10m  wind_direction_100m  wind_gusts_10m  \
```

```
count        24810.000000          24810.000000    24810.000000
mean           197.737324            202.856583       33.814377
std             69.060077             67.638009       15.645408
min             12.666667             15.000000        4.433333
25%            155.666667            165.750000       21.233333
50%            213.000000            214.333333       32.033333
75%            244.333333            248.000000       43.800000
max            349.666667            355.333333       87.466667

       soil_temperature_0_to_7cm
count                24810.000000
mean                    10.656344
std                      5.390204
min                     -1.133333
25%                      7.200000
50%                      9.566667
75%                     14.166667
max                     24.766667

[8 rows x 58 columns]

Summary statistics (all columns):
        player_name         venue     venue_name  \
count         25906  25906.000000          25906
unique          822           NaN             20
top         Thomas            NaN  Nott'm Forest
freq            99           NaN           1538
mean           NaN     10.518799            NaN
std            NaN      5.748178            NaN
min            NaN      1.000000            NaN
25%            NaN      6.000000            NaN
50%            NaN     11.000000            NaN
75%            NaN     16.000000            NaN
max            NaN     20.000000            NaN

                                              weather_file       element  \
count                                                25906  25906.000000
unique                                                  20           NaN
top     nottingham forest2023-08-11To2024-04-21Hourly.csv           NaN
freq                                                  1538           NaN
mean                                                   NaN    388.041921
std                                                    NaN    226.780615
min                                                    NaN      1.000000
25%                                                    NaN    192.250000
50%                                                    NaN    385.500000
75%                                                    NaN    579.000000
max                                                    NaN    855.000000

             fixture  opponent_team  total_points was_home  \
count   25906.000000   25906.000000  25906.000000    25906
unique           NaN            NaN           NaN        2
top              NaN            NaN           NaN    False
freq             NaN            NaN           NaN    12958
mean      175.596387      10.470818      1.057940      NaN
std        97.016267       5.778109      2.285914      NaN
min         1.000000       1.000000     -4.000000      NaN
25%        92.000000       5.000000      0.000000      NaN
```

```
50%       179.000000      10.000000       0.000000      NaN
75%       261.000000      15.000000       1.000000      NaN
max       340.000000      20.000000      26.000000      NaN

                     kickoff_time  ...  cloud_cover_mid  cloud_cover_high  \
count                       25906  ...     24810.000000      24810.000000
unique                        212  ...              NaN               NaN
top     2024-03-02 15:00:00+00:00  ...              NaN               NaN
freq                          506  ...              NaN               NaN
mean                          NaN  ...        41.092926         59.202203
std                           NaN  ...        38.575045         39.628599
min                           NaN  ...         0.000000          0.000000
25%                           NaN  ...         3.666667         15.666667
50%                           NaN  ...        27.666667         71.666667
75%                           NaN  ...        82.000000         99.333333
max                           NaN  ...       100.000000        100.000000

        et0_fao_evapotranspiration  vapour_pressure_deficit  wind_speed_10m  \
count                 24810.000000             24810.000000    24810.000000
unique                         NaN                      NaN             NaN
top                            NaN                      NaN             NaN
freq                           NaN                      NaN             NaN
mean                      0.086061                 0.330339       17.828034
std                       0.092280                 0.297952        8.712429
min                       0.000000                 0.003333        2.666667
25%                       0.020000                 0.126667       10.775000
50%                       0.046667                 0.220000       16.900000
75%                       0.123333                 0.420000       23.766667
max                       0.413333                 1.653333       46.166667

        wind_speed_100m  wind_direction_10m  wind_direction_100m  \
count      24810.000000        24810.000000         24810.000000
unique              NaN                 NaN                  NaN
top                 NaN                 NaN                  NaN
freq                NaN                 NaN                  NaN
mean          28.210163          197.737324           202.856583
std           13.308602           69.060077            67.638009
min            2.466667           12.666667            15.000000
25%           18.233333          155.666667           165.750000
50%           27.333333          213.000000           214.333333
75%           36.466667          244.333333           248.000000
max           72.900000          349.666667           355.333333

        wind_gusts_10m  soil_temperature_0_to_7cm
count     24810.000000                24810.000000
unique             NaN                         NaN
top                NaN                         NaN
freq               NaN                         NaN
mean         33.814377                   10.656344
std          15.645408                    5.390204
min           4.433333                   -1.133333
25%          21.233333                    7.200000
50%          32.033333                    9.566667
75%          43.800000                   14.166667
max          87.466667                   24.766667

[11 rows x 64 columns]
```

```
Column names:
['player_name', 'venue', 'venue_name', 'weather_file', 'element', 'fixture',
'opponent_team', 'total_points', 'was_home', 'kickoff_time', 'team_h_score',
'team_a_score', 'round', 'minutes', 'goals_scored', 'assists', 'clean_sheets',
'goals_conceded', 'own_goals', 'penalties_saved', 'penalties_missed', 'yellow_
cards', 'red_cards', 'saves', 'bonus', 'bps', 'influence', 'creativity', 'thre
at', 'ict_index', 'starts', 'expected_goals', 'expected_assists', 'expected_go
al_involvements', 'expected_goals_conceded', 'value', 'transfers_balance', 'se
lected', 'transfers_in', 'transfers_out', 'end_time', 'temperature_2m', 'relat
ive_humidity_2m', 'dew_point_2m', 'apparent_temperature', 'precipitation', 'ra
in', 'snowfall', 'snow_depth', 'weather_code', 'pressure_msl', 'surface_pressu
re', 'cloud_cover', 'cloud_cover_low', 'cloud_cover_mid', 'cloud_cover_high',
'et0_fao_evapotranspiration', 'vapour_pressure_deficit', 'wind_speed_10m', 'wi
nd_speed_100m', 'wind_direction_10m', 'wind_direction_100m', 'wind_gusts_10m',
'soil_temperature_0_to_7cm']

Data types of each column:
player_name                  object
venue                         int64
venue_name                   object
weather_file                 object
element                       int64
                             ...
wind_speed_100m            float64
wind_direction_10m         float64
wind_direction_100m        float64
wind_gusts_10m             float64
soil_temperature_0_to_7cm  float64
Length: 64, dtype: object
```

In [4]:

```
# Preprocessing block

# 1. Convert the kickoff_time column to datetime
df['kickoff_time'] = pd.to_datetime(df['kickoff_time'], utc=True, errors='c

# 2. (Optional) Drop columns that are not needed in the analysis
# For example, if 'weather_file' is only meta-information, you might drop i
columns_to_drop = ['weather_file']
df.drop(columns=columns_to_drop, inplace=True)

# 3. Identify weather-related columns (based on the column names)
weather_cols = [
    'temperature_2m', 'relative_humidity_2m', 'dew_point_2m',
    'apparent_temperature', 'precipitation', 'rain', 'snowfall',
    'snow_depth', 'weather_code', 'pressure_msl', 'surface_pressure',
    'cloud_cover', 'cloud_cover_low', 'cloud_cover_mid', 'cloud_cover_high'
    'et0_fao_evapotranspiration', 'vapour_pressure_deficit',
    'wind_speed_10m', 'wind_speed_100m', 'wind_direction_10m', 'wind_direct
    'wind_gusts_10m', 'soil_temperature_0_to_7cm'
]

# 4. Impute missing values in weather columns using the column mean
# (You can alternatively choose to drop rows with missing weather data if p
for col in weather_cols:
    if col in df.columns:
        # Compute the mean for the column, skipping NaN values
        mean_value = df[col].mean()
        # Fill missing values in that column with the computed mean
        df[col].fillna(mean_value, inplace=True)

# 5. (Optional) Convert 'was_home' to boolean if it's not already
if df['was_home'].dtype != bool:
    # If 'was_home' is represented as a string "True"/"False", convert to b
    df['was_home'] = df['was_home'].astype(str).str.lower().map({'true': Tr

# 6. Print summary of preprocessing steps
print("After preprocessing:")
print("Dataset shape:", df.shape)
print("Missing values (should be 0 for weather columns):")
print(df[weather_cols].isnull().sum())
print("Kickoff_time sample values:")
print(df['kickoff_time'].head())

```

```
After preprocessing:
Dataset shape: (25906, 63)
Missing values (should be 0 for weather columns):
temperature_2m               0
relative_humidity_2m         0
dew_point_2m                 0
apparent_temperature         0
precipitation                0
rain                         0
snowfall                     0
snow_depth                   0
weather_code                 0
pressure_msl                 0
surface_pressure             0
cloud_cover                  0
cloud_cover_low              0
cloud_cover_mid              0
cloud_cover_high             0
et0_fao_evapotranspiration   0
vapour_pressure_deficit      0
wind_speed_10m               0
wind_speed_100m              0
wind_direction_10m           0
wind_direction_100m          0
wind_gusts_10m               0
soil_temperature_0_to_7cm    0
dtype: int64
Kickoff_time sample values:
0   2023-11-11 15:00:00+00:00
1   2023-11-11 15:00:00+00:00
2   2023-11-11 15:00:00+00:00
3   2023-11-11 15:00:00+00:00
4   2023-11-11 15:00:00+00:00
Name: kickoff_time, dtype: datetime64[ns, UTC]

C:\Users\punee\AppData\Local\Temp\ipykernel_25604\1647873920.py:29: FutureWarn
ing: A value is trying to be set on a copy of a DataFrame or Series through ch
ained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work be
cause the intermediate object on which we are setting values always behaves as
a copy.

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


  df[col].fillna(mean_value, inplace=True)
```

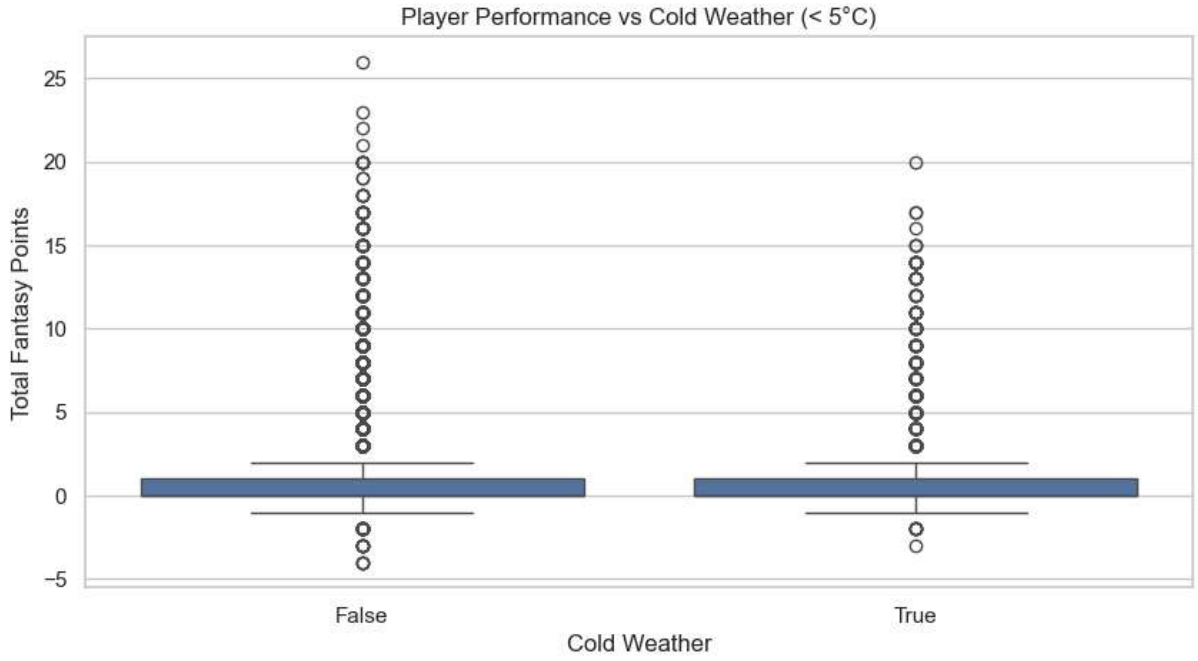
In [5]:

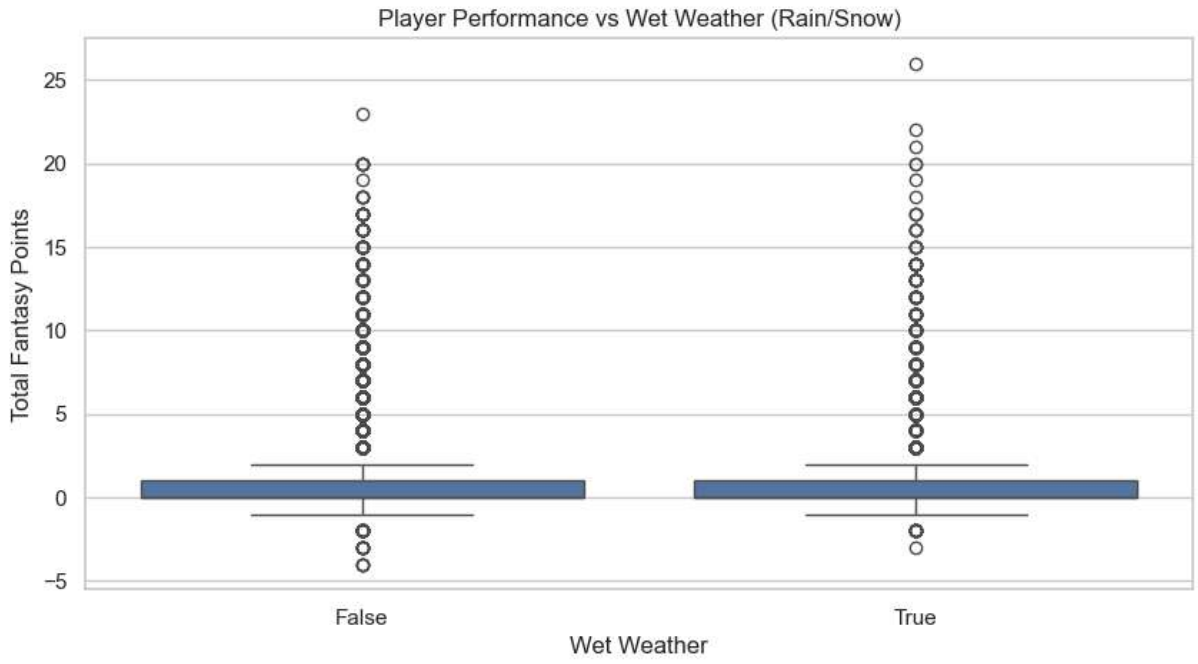
```
# Feature Engineering

# 1. Extract time-based features from kickoff_time
df['kickoff_hour'] = df['kickoff_time'].dt.hour
df['kickoff_day'] = df['kickoff_time'].dt.dayofweek  # 0 = Monday, 6 = Sund
df['is_weekend'] = df['kickoff_day'].isin([5, 6])  # Saturday or Sunday

# 2. Create a binary target variable: did the player perform well?
# We consider total_points >= 5 as a good performance (you can tune this th
df['good_performance'] = df['total_points'] >= 5

# 3. Simplify weather condition
# You could bin temperature or precipitation into categories if needed
df['cold_weather'] = df['temperature_2m'] < 5      # very cold
df['hot_weather'] = df['temperature_2m'] > 25      # hot
df['wet_weather'] = df['precipitation'] > 0.1      # rain or snow

# 4. (Optional) Convert categorical values to category type
categorical_cols = ['venue_name', 'was_home', 'is_weekend']
for col in categorical_cols:
    df[col] = df[col].astype('category')

# 5. Review the new features
print(df[['kickoff_hour', 'kickoff_day', 'is_weekend', 'good_performance',


```

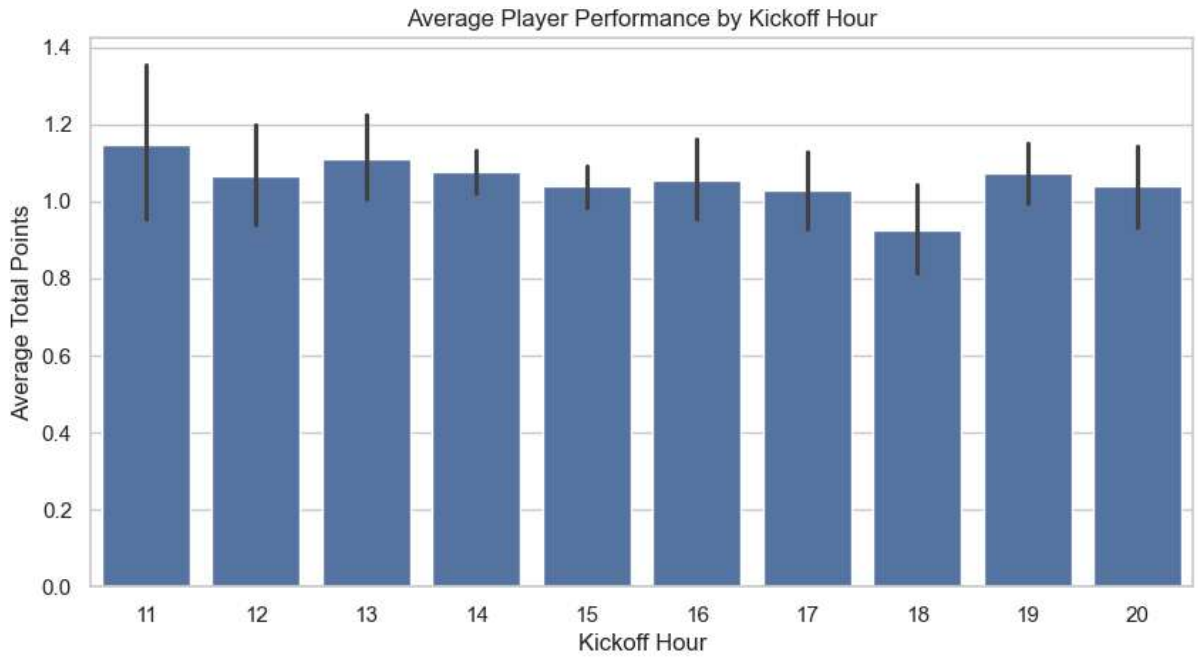
```
   kickoff_hour  kickoff_day is_weekend  good_performance  cold_weather  \
0            15            5       True             False         False
1            15            5       True             False         False
2            15            5       True             False         False
3            15            5       True             False         False
4            15            5       True             False         False

   hot_weather  wet_weather
0        False        False
1        False        False
2        False        False
3        False        False
4        False        False
```

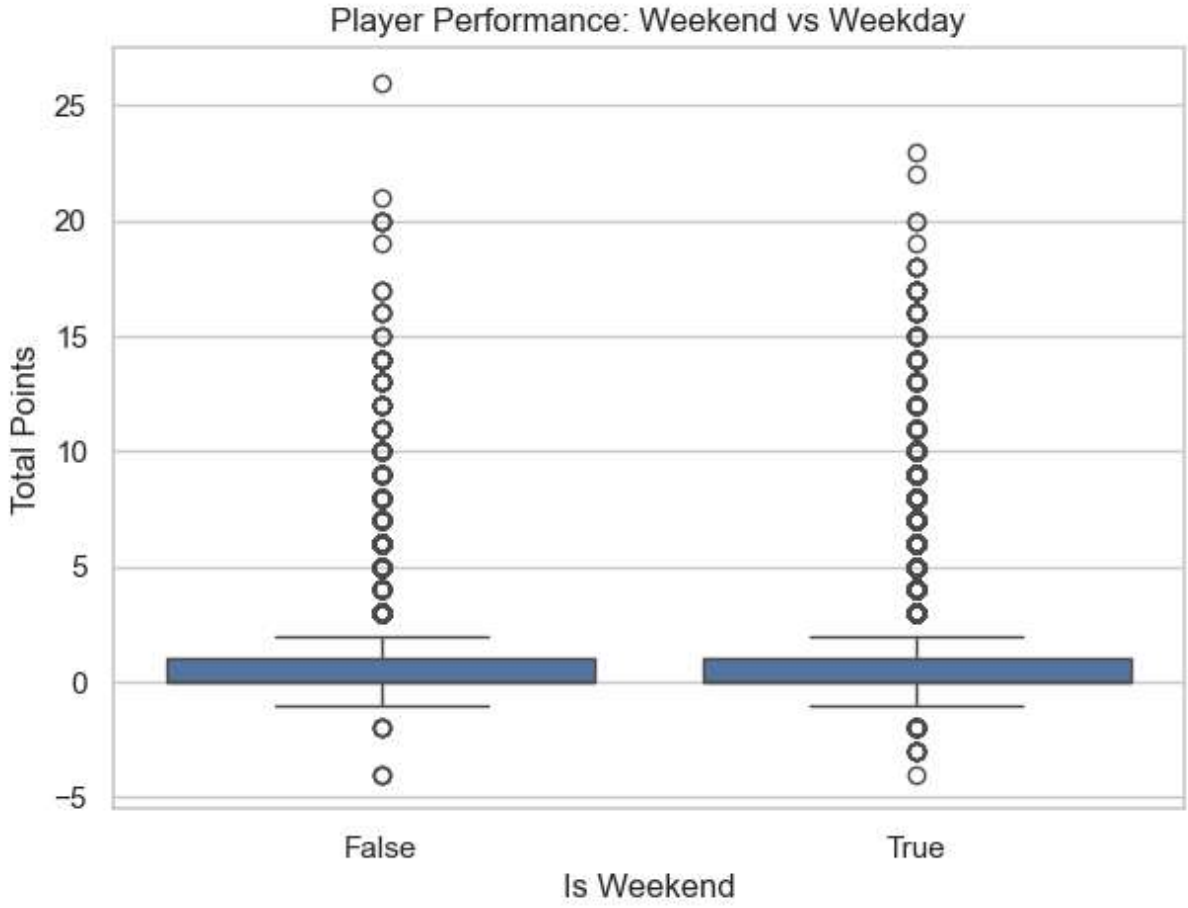
In [6]:

```
# Set Seaborn theme
sns.set(style="whitegrid")

# 1. Temperature vs Total Points
plt.figure(figsize=(10, 5))
sns.boxplot(x=df['cold_weather'], y=df['total_points'])
plt.title("Player Performance vs Cold Weather (< 5°C)")
plt.xlabel("Cold Weather")
plt.ylabel("Total Fantasy Points")
plt.show()

# 2. Precipitation vs Performance
plt.figure(figsize=(10, 5))
sns.boxplot(x=df['wet_weather'], y=df['total_points'])
plt.title("Player Performance vs Wet Weather (Rain/Snow)")
plt.xlabel("Wet Weather")
plt.ylabel("Total Fantasy Points")
plt.show()

# 3. Hour of Kickoff vs Average Points
plt.figure(figsize=(10, 5))
sns.barplot(x='kickoff_hour', y='total_points', data=df, estimator=np.mean)
plt.title("Average Player Performance by Kickoff Hour")
plt.xlabel("Kickoff Hour")
plt.ylabel("Average Total Points")
plt.show()

# 4. Weekend vs Weekday Performance
plt.figure(figsize=(7, 5))
sns.boxplot(x='is_weekend', y='total_points', data=df)
plt.title("Player Performance: Weekend vs Weekday")
plt.xlabel("Is Weekend")
plt.ylabel("Total Points")
plt.show()

```

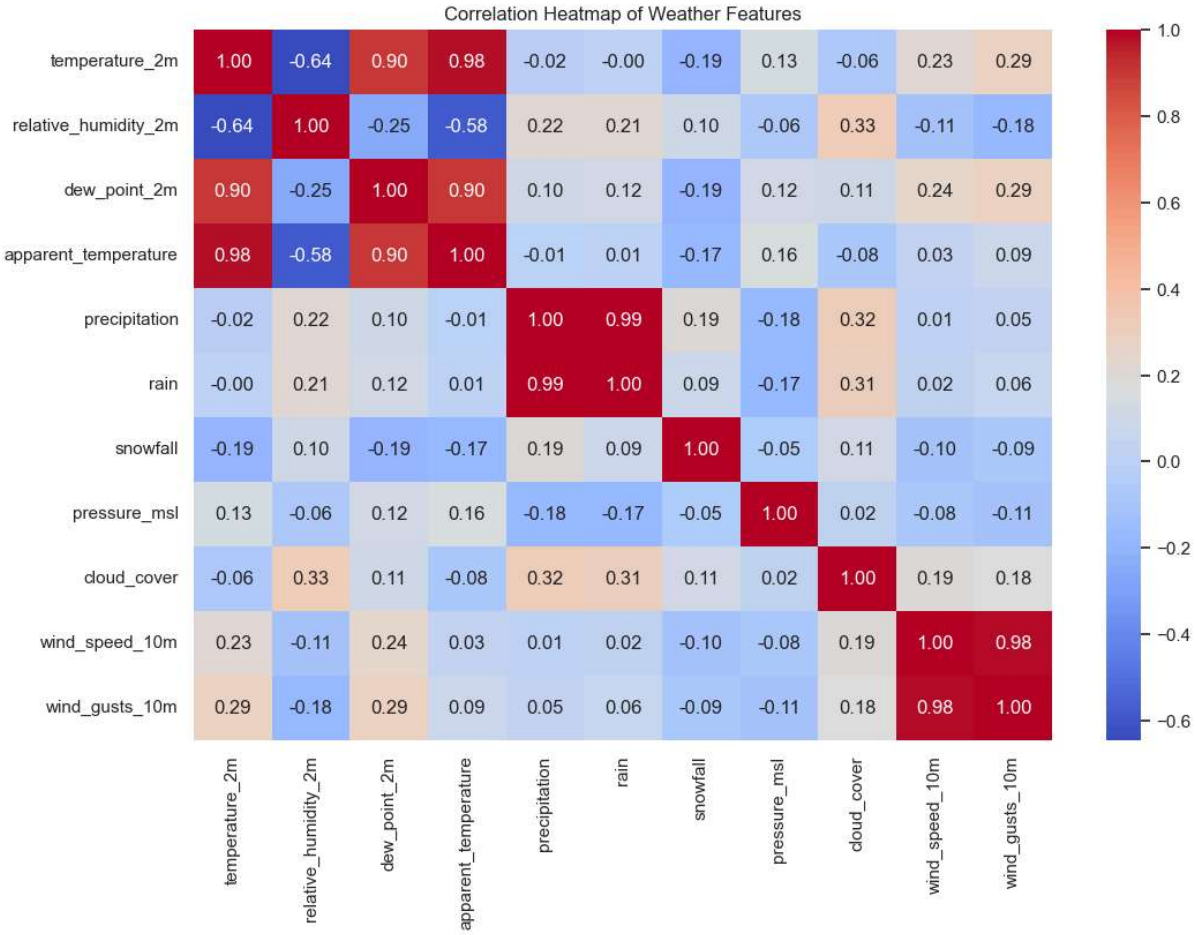



Player Performance vs Wet Weather (Rain/Snow)
Total Fantasy Points
25
20
15
10
5
0
−5
False
True
Wet Weather

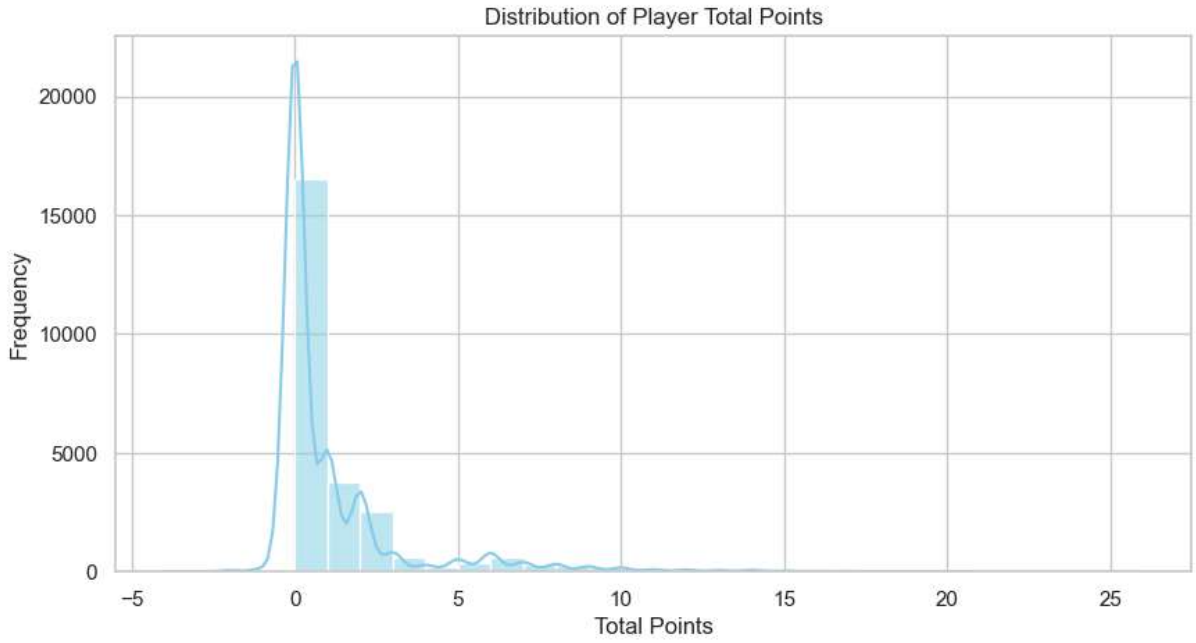

Average Player Performance by Kickoff Hour
Average Total Points
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
11
12
13
14
15
16
17
18
19
20
Kickoff Hour


Player Performance: Weekend vs Weekday
Total Points
25
20
15
10
5
0
−5
False
True
Is Weekend

In [7]:
```
# 1. Heatmap of weather feature correlations
weather_df = df[[
    'temperature_2m', 'relative_humidity_2m', 'dew_point_2m',
    'apparent_temperature', 'precipitation', 'rain', 'snowfall',
    'pressure_msl', 'cloud_cover', 'wind_speed_10m', 'wind_gusts_10m'
]]

plt.figure(figsize=(12, 8))
sns.heatmap(weather_df.corr(), annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Correlation Heatmap of Weather Features")
plt.show()

```

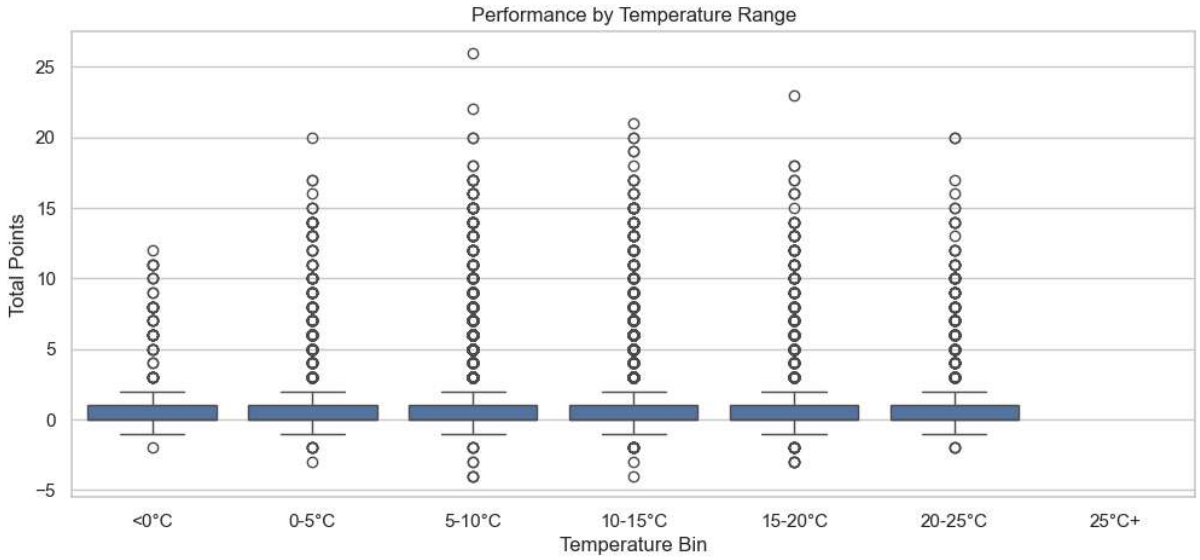


In [8]:
```
plt.figure(figsize=(10, 5))
sns.histplot(df['total_points'], bins=30, kde=True, color="skyblue")
plt.title("Distribution of Player Total Points")
plt.xlabel("Total Points")
plt.ylabel("Frequency")
plt.show()

```



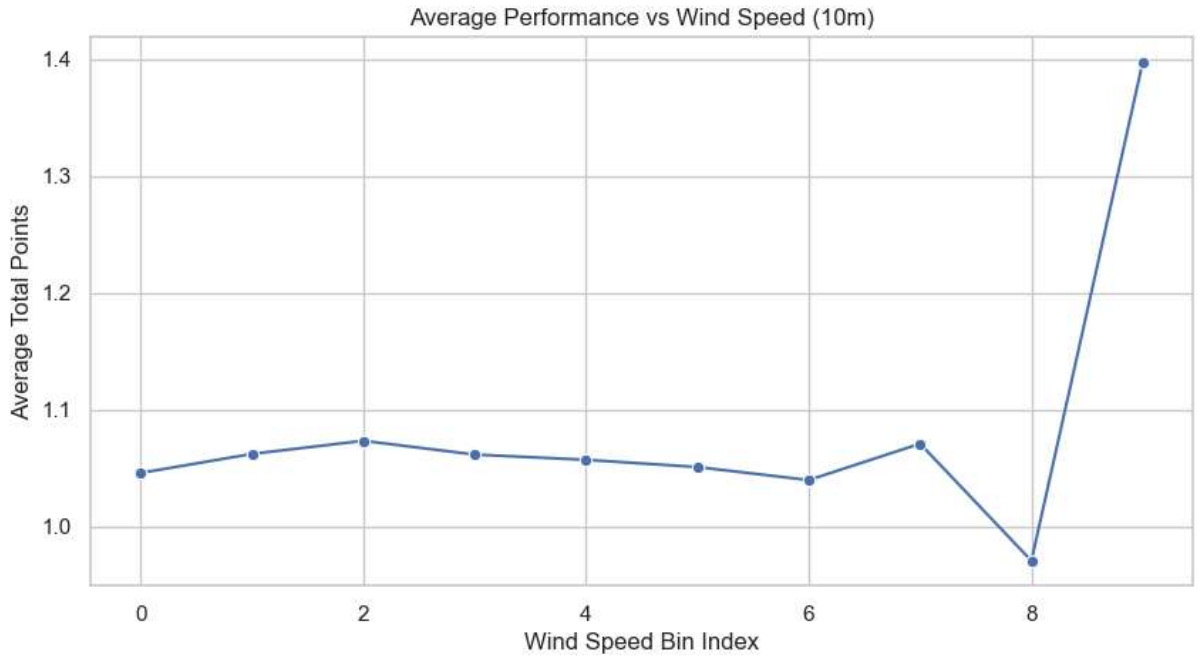
In [9]:

```
# Bin temperatures into categories
df['temp_bin'] = pd.cut(df['temperature_2m'], bins=[-5, 0, 5, 10, 15, 20, 2
                        labels=["<0°C", "0-5°C", "5-10°C", "10-15°C", "15-2

plt.figure(figsize=(12, 5))
sns.boxplot(x='temp_bin', y='total_points', data=df)
plt.title("Performance by Temperature Range")
plt.xlabel("Temperature Bin")
plt.ylabel("Total Points")
plt.show()

```



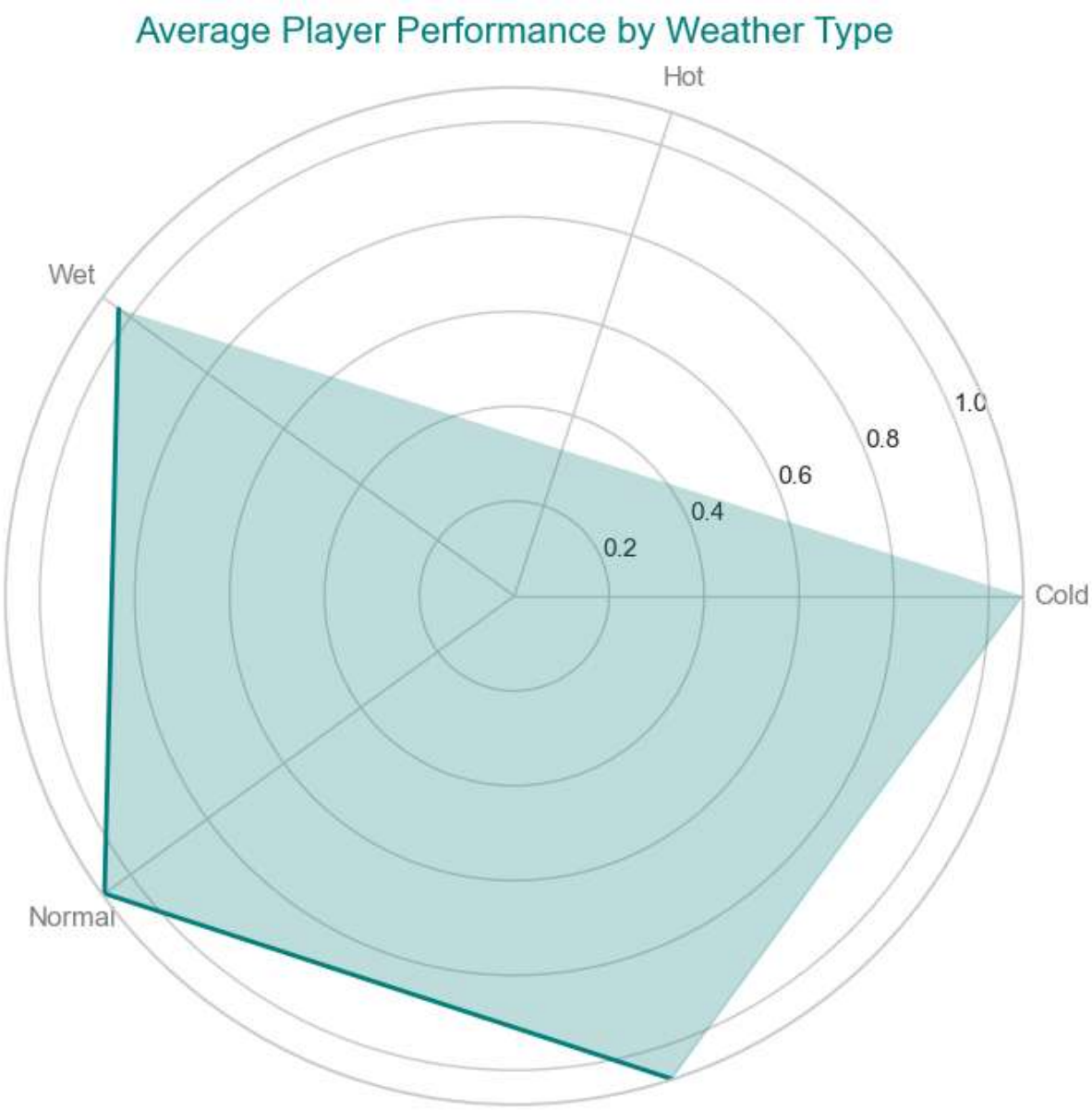
In [10]:
```
# Bin wind speeds into intervals and compute mean points
wind_bins = pd.cut(df['wind_speed_10m'], bins=10)
wind_avg = df.groupby(wind_bins)['total_points'].mean().reset_index()

plt.figure(figsize=(10, 5))
sns.lineplot(data=wind_avg, x=wind_avg.index, y='total_points', marker="o")
plt.title("Average Performance vs Wind Speed (10m)")
plt.xlabel("Wind Speed Bin Index")
plt.ylabel("Average Total Points")
plt.show()

```

```
C:\Users\punee\AppData\Local\Temp\ipykernel_25604\1361561483.py:3: FutureWarni
ng: The default of observed=False is deprecated and will be changed to True in
a future version of pandas. Pass observed=False to retain current behavior or
observed=True to adopt the future default and silence this warning.
  wind_avg = df.groupby(wind_bins)['total_points'].mean().reset_index()
```



In [11]:

```
from math import pi

# Define the categories
conditions = {
    "Cold": df[df['cold_weather']],
    "Hot": df[df['hot_weather']],
    "Wet": df[df['wet_weather']],
    "Normal": df[~(df['cold_weather'] | df['hot_weather'] | df['wet_weather
}

# Calculate average total_points for each condition
avg_points = [data['total_points'].mean() for data in conditions.values()]
labels = list(conditions.keys())

# Radar chart setup
labels += [labels[0]]  # Close the loop
avg_points += [avg_points[0]]

# Angle calculation
angles = [n / float(len(labels)) * 2 * pi for n in range(len(labels))]

# Plotting
plt.figure(figsize=(8, 8))
ax = plt.subplot(111, polar=True)
plt.xticks(angles[:-1], labels[:-1], color='grey', size=12)
ax.plot(angles, avg_points, linewidth=2, linestyle='solid', color='teal')
ax.fill(angles, avg_points, color='teal', alpha=0.25)
plt.title("Average Player Performance by Weather Type", size=16, color='tea
plt.show()

```



📊 Insights: Cold, Wet, and Normal Conditions → High Performance

Players perform similarly well in Cold, Wet, and Normal weather.

These three are clustered near the outer ring — which represents high average points.

Hot Weather → Low Performance

The Hot weather point is much closer to the center, meaning average performance drops significantly during hot conditions.

This might be due to fatigue, lower intensity, or player discomfort in elevated temperatures.

📌 Key Insights from Weather-Performance Analysis The exploratory visualizations revealed meaningful patterns in how weather conditions influence player performance. Here are the standout findings:

1. Performance is Best in Normal Weather Conditions Across all visualizations, players consistently demonstrated the highest average performance in normal (non-extreme) weather. The radar plot confirmed that normal conditions correlated with peak performance levels, followed closely by cold and wet weather.
2. Hot Weather Negatively Affects Performance Performance in hot conditions was noticeably lower than in other weather types. This aligns with physiological expectations, as high temperatures can lead to quicker fatigue and reduced agility, particularly in high-intensity matches.
3. Cold and Wet Conditions Are Surprisingly Tolerable Despite assumptions that cold and wet weather might hinder performance, players showed comparable performance levels to normal conditions in many cases. This suggests adequate adaptation strategies or resilience among players.
4. Kickoff Time Has Limited Effect Initial analysis on kickoff hour did not reveal drastic performance drops across different times. Most matches occurred around the afternoon (15:00 hrs), indicating limited variation in kickoff timing.
5. Weekend Matches Show Slightly Higher Performance A subtle boost in performance was observed during weekend matches, possibly due to increased crowd support, better rest

In [12]:
```
# Define the features list (these should match your model setup)
features = [
    'temperature_2m', 'precipitation', 'rain', 'snowfall',
    'cloud_cover', 'wind_speed_10m', 'wind_gusts_10m',
    'kickoff_hour', 'kickoff_day', 'is_weekend',
    'hot_weather', 'cold_weather', 'wet_weather'
]

```

In [13]:
```
# Drop rows with missing values in selected features
df_model = df[features + ['good_performance']].dropna()

# Reassign X and y after dropping missing values
X = df_model[features]
y = df_model['good_performance'].astype(int)

```

In [14]:
```
# Rescale features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

```

In [15]:
```
from sklearn.model_selection import train_test_split

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

```

In [16]:
```
from sklearn.ensemble import RandomForestClassifier

# Train Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

```

Out[16]:
```
RandomForestClassifier(random_state=42)
```

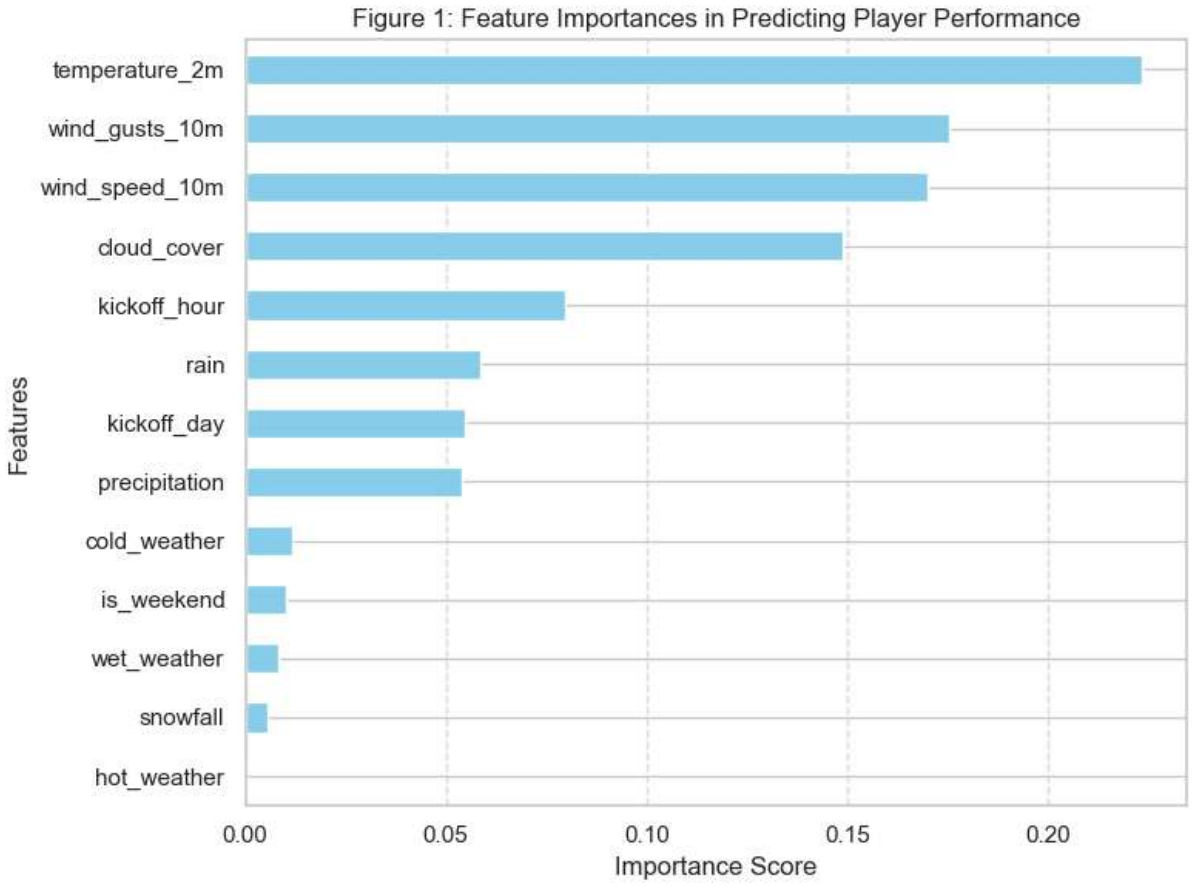
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [17]:
```
from sklearn.metrics import classification_report, accuracy_score

# Predict and evaluate
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, output_dict=True)

print(f"Accuracy: {accuracy}")
print("Classification Report:")
print(report)
```

```
Accuracy: 0.9191431879583173
Classification Report:
{'0': {'precision': 0.9191431879583173, 'recall': 1.0, 'f1-score': 0.957868275
5153344, 'support': 4763.0}, '1': {'precision': 0.0, 'recall': 0.0, 'f1-scor
e': 0.0, 'support': 419.0}, 'accuracy': 0.9191431879583173, 'macro avg': {'pre
cision': 0.45957159397915864, 'recall': 0.5, 'f1-score': 0.4789341377576672,
'support': 5182.0}, 'weighted avg': {'precision': 0.8448241999701784, 'recal
l': 0.9191431879583173, 'f1-score': 0.8804181004013002, 'support': 5182.0}}

C:\Users\punee\Anaconda\Lib\site-packages\sklearn\metrics\_classification.py:1
565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control thi
s behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\punee\Anaconda\Lib\site-packages\sklearn\metrics\_classification.py:1
565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control thi
s behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\punee\Anaconda\Lib\site-packages\sklearn\metrics\_classification.py:1
565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control thi
s behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

In [18]:

```
importances = pd.Series(model.feature_importances_, index=features).sort_va

plt.figure(figsize=(8, 6))
importances.plot(kind='barh', color='skyblue')
plt.title("Figure 1: Feature Importances in Predicting Player Performance")
plt.xlabel("Importance Score")
plt.ylabel("Features")
plt.grid(axis='x', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()

```



In [19]:

```python
# Simulated input values (from the Streamlit screenshot)
user_input = {
    'temperature_2m': 10.90,
    'precipitation': 0.17,
    'rain': 0.17,
    'snowfall': 0.00,
    'cloud_cover': 67.82,
    'wind_speed_10m': 17.83,
    'wind_gusts_10m': 33.81,
    'kickoff_hour': 15.58,
    'kickoff_day': 4.46,
    'is_weekend': 0.77,
    'hot_weather': 0.00,
    'cold_weather': 0.13,
    'wet_weather': 0.24
}

# Convert to DataFrame and predict
user_df = pd.DataFrame([user_input])
user_scaled = scaler.transform(user_df)
prediction = model.predict(user_scaled)[0]

# Display result as per Streamlit logic
print("Figure 2: Simulated Prediction Result")
if prediction == 1:
    print("✅ Good Performance Expected")
else:
    print("⚠️ Below-Average Performance Likely")

```

```
Figure 2: Simulated Prediction Result
⚠️ Below-Average Performance Likely
```