In [34]:
```
# Imports and Directory Setup

import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns

# Define base directory for the project
project_dir = r"C:\Users\matth\OneDrive\Documents\data_science_project\pre

# Define the directory where the processed (cleaned) data is stored
processed_data_dir = os.path.join(project_dir, "data", "processed_data")

# Define the path to the combined match data file
match_data_file = os.path.join(processed_data_dir, "all_seasons_match_data

# Define the directory paths for saving outputs
output_dir = os.path.join(project_dir, "output")
figures_dir = os.path.join(output_dir, "figures")
tables_dir = os.path.join(output_dir, "tables")

print("Required libraries imported.")
print(f"Processed data directory: {processed_data_dir}")
print(f"Match data file path: {match_data_file}")
print(f"Figures output directory: {figures_dir}")
print(f"Tables output directory: {tables_dir}")

# Set a default style for plots
sns.set_theme(style="whitegrid")
```

```
Required libraries imported.
Processed data directory: C:\Users\matth\OneDrive\Documents\data_science_
project\premier-league-home-advantage\data\processed_data
Match data file path: C:\Users\matth\OneDrive\Documents\data_science_proj
ect\premier-league-home-advantage\data\processed_data\all_seasons_match_d
ata.csv
Figures output directory: C:\Users\matth\OneDrive\Documents\data_science_
project\premier-league-home-advantage\output\figures
Tables output directory: C:\Users\matth\OneDrive\Documents\data_science_p
roject\premier-league-home-advantage\output\tables
```

In [35]:

```
# Load Match Data

print(f"\nLoading match data from: {match_data_file}")
try:
    match_df = pd.read_csv(match_data_file, encoding='utf-8')
    print(f" Successfully loaded match data. Shape: {match_df.shape}")
except FileNotFoundError:
    print(f"Error: File not found at {match_data_file}.")
    print(" Please ensure the file exists in the processed_data directory.
    match_df = None
except Exception as e:
    print(f"An error occurred loading the match data: {e}")
    match_df = None
```

```
Loading match data from: C:\Users\matth\OneDrive\Documents\data_science_p
roject\premier-league-home-advantage\data\processed_data\all_seasons_matc
h_data.csv
 Successfully loaded match data. Shape: (2280, 12)
```

In [36]:
```
# Initial Data Inspection

# Check if data loaded successfully before inspecting
if match_df is not None:
    print("\n--- First 5 Rows (Head) ---")
    display(match_df.head())

    print("\n--- DataFrame Info ---")
    # Simply call .info() - Jupyter will display its output
    match_df.info()

    print("\n--- Summary Statistics ---")
    # Include 'all' to get stats for object/category columns too
    display(match_df.describe(include='all'))

else:
    print("\nCannot perform inspection because match data failed to load."
```

```
--- First 5 Rows (Head) ---
```

	dayofweek	date	start_time	home_team	home_xg	score	away_xg	away_team	atter
0	Fri	2018-08-10	20:00	Manchester Utd	1.5	Score: 2 - 1	1.8	Leicester City	7
1	Sat	2018-08-11	12:30	Newcastle Utd	1.0	Score: 1 - 2	2.0	Tottenham	5
2	Sat	2018-08-11	15:00	Fulham	0.7	Score: 0 - 2	1.0	Crystal Palace	2
3	Sat	2018-08-11	15:00	Bournemouth	2.2	Score: 2 - 0	1.4	Cardiff City	1
4	Sat	2018-08-11	15:00	Huddersfield	0.3	Score: 0 - 3	1.9	Chelsea	2

```
--- DataFrame Info ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2280 entries, 0 to 2279
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   dayofweek   2280 non-null   object 
 1   date        2280 non-null   object 
 2   start_time  2280 non-null   object 
 3   home_team   2280 non-null   object 
 4   home_xg     2280 non-null   float64
 5   score       2280 non-null   object 
 6   away_xg     2280 non-null   float64
 7   away_team   2280 non-null   object 
 8   attendance  2280 non-null   float64
 9   venue       2280 non-null   object 
 10  referee     2280 non-null   object 
 11  season      2280 non-null   object 
dtypes: float64(3), object(9)
memory usage: 213.9+ KB

--- Summary Statistics ---
```

	dayofweek	date	start_time	home_team	home_xg	score	away_xg	away_
count	2280	2280	2280	2280	2280.000000	2280	2280.000000	
unique	7	718	24	28	NaN	46	NaN	
top	Sat	2024-05-19	15:00	Manchester Utd	NaN	Score: 1 - 1	NaN	W
freq	1065	10	712	114	NaN	239	NaN	
mean	NaN	NaN	NaN	NaN	1.518333	NaN	1.254167	
std	NaN	NaN	NaN	NaN	0.855207	NaN	0.761324	
min	NaN	NaN	NaN	NaN	0.000000	NaN	0.000000	
25%	NaN	NaN	NaN	NaN	0.900000	NaN	0.700000	
50%	NaN	NaN	NaN	NaN	1.400000	NaN	1.100000	
75%	NaN	NaN	NaN	NaN	2.100000	NaN	1.700000	
max	NaN	NaN	NaN	NaN	7.000000	NaN	4.600000	

In [37]:
```
# Feature Engineering from Score

# Check if match_df exists
if 'match_df' in locals() and match_df is not None:
    print("\nPerforming feature engineering (Goals, Difference, Points)...

    # Make a copy to avoid SettingWithCopyWarning
    match_df = match_df.copy()

    # Check if 'score' column exists
    if 'score' in match_df.columns:
        try:
            # Extract Home and Away Goals
            # Expected format: "Score: H - A"
            split_scores = match_df['score'].str.split(': ', expand=True)[
            # Convert to numeric, coercing errors to NaN
            match_df['home_goals'] = pd.to_numeric(split_scores[0].str.str
            match_df['away_goals'] = pd.to_numeric(split_scores[1].str.str

            # Check for parsing errors (NaNs) - only if needed for debuggi
            if match_df['home_goals'].isnull().any() or match_df['away_goa
                 print("Warning: Some scores may not have been parsed corr

            # Calculate Goal Difference
            match_df['goal_difference'] = match_df['home_goals'] - match_d

            # Calculate Home Points
            conditions = [
                match_df['home_goals'] > match_df['away_goals'], # Home Wi
                match_df['home_goals'] == match_df['away_goals'], # Draw
                match_df['home_goals'] < match_df['away_goals'] # Home Los
            ]
            point_values = [3, 1, 0]
            match_df['home_points'] = np.select(conditions, point_values,

            print("Feature engineering complete.")

            # Verification
            print("\n--- Verification of new columns (first 5 rows) ---")
            display(match_df[['score', 'home_goals', 'away_goals', 'goal_d

        except Exception as e:
            print(f"Error during feature engineering: {e}")
            # Ensure columns exist as NaN if error occurs
            if 'home_goals' not in match_df.columns: match_df['home_goals'
            if 'away_goals' not in match_df.columns: match_df['away_goals'
            if 'goal_difference' not in match_df.columns: match_df['goal_d
            if 'home_points' not in match_df.columns: match_df['home_point

    else:
        print("Error: 'score' column not found in DataFrame.")

else:
    print("\nSkipping feature engineering because match_df DataFrame not a
Performing feature engineering (Goals, Difference, Points)...
Feature engineering complete.

--- Verification of new columns (first 5 rows) ---
```

	score	home_goals	away_goals	goal_difference	home_points
0	Score: 2 - 1	2	1	1	3.0
1	Score: 1 - 2	1	2	-1	0.0
2	Score: 0 - 2	0	2	-2	0.0
3	Score: 2 - 0	2	0	2	3.0
4	Score: 0 - 3	0	3	-3	0.0

In [38]:

```
# Define Analysis Periods ('COVID-Affected' vs 'Normal')

# Check if match_df exists
if 'match_df' in locals() and match_df is not None:
    print("\nDefining analysis periods based on attendance...")
    print(" Confirmed no missing values in 'attendance' column.")

    # Define the threshold based on user information
    attendance_threshold = 10000
    print(f" Using attendance threshold: < {attendance_threshold} for 'COV

    # Create the 'period' column using np.where
    match_df['period'] = np.where(match_df['attendance'] < attendance_thre

    print(" Created 'period' column.")

    # Verification
    print("\n--- Value Counts for 'period' column ---")
    print(match_df['period'].value_counts())

else:
    print("\nSkipping period definition because match_df DataFrame not ava
```

```
Defining analysis periods based on attendance...
 Confirmed no missing values in 'attendance' column.
 Using attendance threshold: < 10000 for 'COVID-Affected'.
 Created 'period' column.

--- Value Counts for 'period' column ---
Normal            1812
COVID-Affected     468
Name: period, dtype: int64
```

In [39]:
```
# Compare Home Advantage Metrics Across Periods

# Check if match_df exists and has the required columns
if 'match_df' in locals() and match_df is not None and 'period' in match_d
    print("\nCalculating average home advantage metrics per period...")

    # Group by the 'period' column and calculate the mean for relevant met
    period_comparison = match_df.groupby('period')[[
        'goal_difference',
        'home_points'
    ]].mean().reset_index()

    print("\n--- Comparison of Average Metrics ('COVID-Affected' vs 'Norma
    # Display the resulting comparison table
    display(period_comparison)
    print("-" * 60)

else:
    print("\nSkipping comparison because match_df or required columns ('pe
    print(" Ensure the previous cells creating these have been run success
```

```
Calculating average home advantage metrics per period...

--- Comparison of Average Metrics ('COVID-Affected' vs 'Normal') ---
```

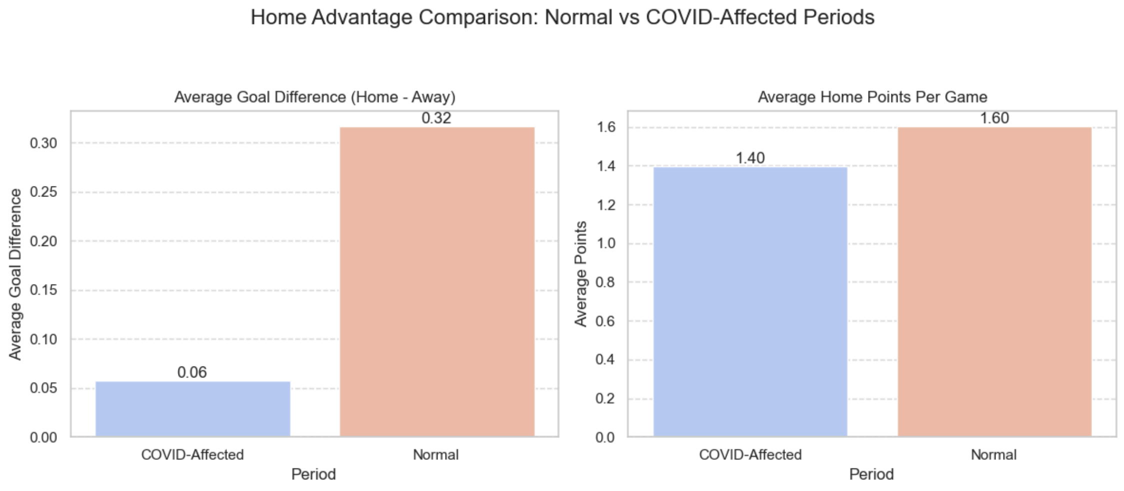
	period	goal_difference	home_points
0	COVID-Affected	0.057692	1.397436
1	Normal	0.316777	1.603753

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

In [40]:

```
# Visualise Home Advantage Metrics Across Periods

# Check if the period_comparison DataFrame exists from the previous cell
if 'period_comparison' in locals() and period_comparison is not None:
    print("\nGenerating bar charts comparing metrics across periods...")

    # Create Figure with Two Subplots
    # Create a figure and a set of subplots (1 row, 2 columns)
    fig, axes = plt.subplots(1, 2, figsize=(12, 5)) # Adjust figsize as ne
    fig.suptitle('Home Advantage Comparison: Normal vs COVID-Affected Peri

    # Plot 1: Average Goal Difference
    sns.barplot(ax=axes[0], x='period', y='goal_difference', data=period_c
    axes[0].set_title('Average Goal Difference (Home - Away)')
    axes[0].set_xlabel('Period')
    axes[0].set_ylabel('Average Goal Difference')
    axes[0].grid(axis='y', linestyle='--', alpha=0.7)
    # Add exact values on bars
    for container in axes[0].containers:
        axes[0].bar_label(container, fmt='%.2f')


    # Plot 2: Average Home Points
    sns.barplot(ax=axes[1], x='period', y='home_points', data=period_compa
    axes[1].set_title('Average Home Points Per Game')
    axes[1].set_xlabel('Period')
    axes[1].set_ylabel('Average Points')
    axes[1].grid(axis='y', linestyle='--', alpha=0.7)
    # Add exact values on bars
    for container in axes[1].containers:
        axes[1].bar_label(container, fmt='%.2f')


    # Final Touches & Saving
    plt.tight_layout(rect=[0, 0, 1, 0.97]) # Adjust layout to prevent over

    # Define filename and save
    plot_filename_period_comp = os.path.join(figures_dir, 'period_comparis
    try:
        plt.savefig(plot_filename_period_comp, bbox_inches='tight', dpi=30
        print(f" Comparison plots saved successfully to: {plot_filename_pe
    except Exception as e:
        print(f" An error occurred while saving the comparison plots: {e}"

    # Display the plot
    plt.show()

else:
    print("\nSkipping visualisation because period_comparison DataFrame no
    print(" Ensure the previous cell calculating this has been run success
```

```
Generating bar charts comparing metrics across periods...
 Comparison plots saved successfully to: C:\Users\matth\OneDrive\Document
s\data_science_project\premier-league-home-advantage\output\figures\perio
d_comparison_bar_charts.png
```


Home Advantage Comparison: Normal vs COVID-Affected Periods
Average Goal Difference (Home - Away)
Average Goal Difference
Period
COVID-Affected
Normal
0.06
0.32
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Average Home Points Per Game
Average Points
1.40
1.60
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6

In [41]:
```
# Regression Analysis: Attendance Impact on Goal Difference

# Import the statsmodels formula API
import statsmodels.formula.api as smf

# Check if match_df exists and has the required columns
if 'match_df' in locals() and match_df is not None and 'period' in match_d
    print("\nSetting up regression analysis (goal_difference ~ attendance)

    # Filter out the 'COVID-Affected' period
    regression_df = match_df[match_df['period'] == 'Normal'].copy()
    print(f" Using {len(regression_df)} matches from the 'Normal' period f

    if len(regression_df) > 0:
        # Define and Fit the OLS Model
        formula = 'goal_difference ~ attendance'
        print(f" Fitting OLS model with formula: {formula}")

        try:
            model = smf.ols(formula=formula, data=regression_df)
            results = model.fit()

            # Display Regression Results
            print("\n--- OLS Regression Results ---")
            print(results.summary())
            print("-" * 80)

        except Exception as e:
            print(f"An error occurred during model fitting or summary gene
    else:
        print("Skipping regression: No data available after filtering for

else:
    print("\nSkipping regression because match_df or required columns not
    print(" Ensure the previous cells creating these have been run success
Setting up regression analysis (goal_difference ~ attendance)...
 Using 1812 matches from the 'Normal' period for regression.
 Fitting OLS model with formula: goal_difference ~ attendance

--- OLS Regression Results ---
                            OLS Regression Results
==========================================================================
=====
Dep. Variable:        goal_difference   R-squared:
0.057
Model:                            OLS   Adj. R-squared:
0.057
Method:                 Least Squares   F-statistic:
110.2
Date:                Mon, 21 Apr 2025   Prob (F-statistic):           4.5
8e-25
Time:                        12:41:20   Log-Likelihood:                -3
734.2
No. Observations:                1812   AIC:
7472.
Df Residuals:                    1810   BIC:
7483.
Df Model:                           1
Covariance Type:            nonrobust
==========================================================================
=====
                coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------
-----
Intercept     -0.7863      0.114     -6.886      0.000      -1.010      -
0.562
attendance  2.822e-05   2.69e-06     10.497      0.000    2.29e-05    3.3
5e-05
==========================================================================
=====
Omnibus:                       35.626   Durbin-Watson:
2.006
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               7
1.229
Skew:                          -0.043   Prob(JB):                     3.4
1e-16
Kurtosis:                       3.967   Cond. No.                     1.0
9e+05
==========================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The condition number is large, 1.09e+05. This might indicate that the
re are
strong multicollinearity or other numerical problems.
--------------------------------------------------------------------------
-------
```

In [42]:
```
# Advanced Regression Prep: Calculate Team Strength Proxies

print("\nLoading seasonal data to calculate team strength proxies...")

# Define the path to the seasonal data file
seasonal_data_file = os.path.join(processed_data_dir, "all_seasons_team_da

# Load the seasonal team stats data
try:
    seasonal_df = pd.read_csv(seasonal_data_file, encoding='utf-8')
    print(f" Successfully loaded seasonal stats data from: {seasonal_data_
except FileNotFoundError:
    print(f"Error: File not found at {seasonal_data_file}.")
    print(" This file is required to create team strength proxies.")
    seasonal_df = None
except Exception as e:
    print(f"An error occurred loading the seasonal stats data: {e}")
    seasonal_df = None

# Proceed only if seasonal data loaded successfully
if seasonal_df is not None:
    print("\nSelecting and renaming columns for strength proxies...")
    # Select the necessary columns: team, season, and the xG difference me
    # Use home_xg_diff_per90 as a proxy for team strength when playing at
    # Use away_xg_diff_per90 as a proxy for team strength when playing awa
    try:
        team_strength_proxies = seasonal_df[[
            'team',
            'season',
            'home_xg_diff_per90',
            'away_xg_diff_per90'
        ]].copy() # Use .copy() to avoid SettingWithCopyWarning when renam

        # Rename columns for clarity when merging later
        team_strength_proxies.rename(columns={
            'home_xg_diff_per90': 'home_strength_proxy',
            'away_xg_diff_per90': 'away_strength_proxy'
        }, inplace=True)

        print(" Created team strength proxy DataFrame.")

        # Verification
        print("\n--- Team Strength Proxies DataFrame Info ---")
        team_strength_proxies.info()
        print("\n--- Team Strength Proxies DataFrame Head ---")
        display(team_strength_proxies.head())

    except KeyError as e:
        print(f"Error: A required column ({e}) was not found in {seasonal_
        print(" Cannot create strength proxies.")
        team_strength_proxies = None
    except Exception as e:
        print(f"An error occurred selecting/renaming columns: {e}")
        team_strength_proxies = None

else:
    print("\nSkipping strength proxy creation because seasonal data failed
Loading seasonal data to calculate team strength proxies...
 Successfully loaded seasonal stats data from: C:\Users\matth\OneDrive\Do
cuments\data_science_project\premier-league-home-advantage\data\processed
_data\all_seasons_team_data.csv

Selecting and renaming columns for strength proxies...
 Created team strength proxy DataFrame.

--- Team Strength Proxies DataFrame Info ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120 entries, 0 to 119
Data columns (total 4 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   team                 120 non-null    object 
 1   season               120 non-null    object 
 2   home_strength_proxy  120 non-null    float64
 3   away_strength_proxy  120 non-null    float64
dtypes: float64(2), object(2)
memory usage: 3.9+ KB

--- Team Strength Proxies DataFrame Head ---
```

	team	season	home_strength_proxy	away_strength_proxy
0	Manchester City	2018-2019	1.93	1.21
1	Liverpool	2018-2019	1.52	0.84
2	Chelsea	2018-2019	0.87	0.29
3	Tottenham	2018-2019	0.36	0.04
4	Arsenal	2018-2019	0.58	-0.27

In [43]:

```
# Advanced Regression Prep: Merge Home Team Strength

# Check if prerequisite DataFrames exist and have necessary columns
if ('match_df' in locals() and match_df is not None and
    'team_strength_proxies' in locals() and team_strength_proxies is not N
    'season' in match_df.columns and 'home_team' in match_df.columns and
    'team' in team_strength_proxies.columns and 'home_strength_proxy' in t

    print("\nMerging home team strength proxy...")
    match_df_merged_home = match_df.copy() # Start with a copy

    # Merge based on home_team and season
    match_df_merged_home = pd.merge(
        match_df_merged_home,
        team_strength_proxies[['team', 'season', 'home_strength_proxy']],
        how='left',
        left_on=['home_team', 'season'],
        right_on=['team', 'season']
    )
    # Rename merged column and drop redundant 'team' column from merge
    match_df_merged_home.rename(columns={'home_strength_proxy': 'home_team
    match_df_merged_home.drop(columns=['team'], inplace=True)

    # Check for NaNs introduced by the merge
    home_nans = match_df_merged_home['home_team_strength_proxy'].isnull().
    if home_nans > 0:
        print(f"Warning: Found {home_nans} rows with missing home strength
    else:
        print(" Home team merge complete (no NaNs found).")

else:
    print("\nSkipping home merge due to missing prerequisites.")
    match_df_merged_home = None # Ensure variable exists but is None
```

```
Merging home team strength proxy...
 Home team merge complete (no NaNs found).
```

In [44]:

```python
# Advanced Regression Prep: Merge Away Team Strength

# Check if the result from the previous merge step exists and prerequisite
if ('match_df_merged_home' in locals() and match_df_merged_home is not Non
    'team_strength_proxies' in locals() and team_strength_proxies is not N
    'season' in match_df_merged_home.columns and 'away_team' in match_df_m
    'team' in team_strength_proxies.columns and 'away_strength_proxy' in t

    print("\nMerging away team strength proxy...")
    # Continue working with the result of the previous merge
    match_df_merged = match_df_merged_home.copy() # Rename for final merge

    # Merge based on away_team and season
    match_df_merged = pd.merge(
        match_df_merged,
        team_strength_proxies[['team', 'season', 'away_strength_proxy']],
        how='left',
        left_on=['away_team', 'season'],
        right_on=['team', 'season']
    )
    # Rename merged column and drop redundant 'team' column from merge
    match_df_merged.rename(columns={'away_strength_proxy': 'away_team_stre
    match_df_merged.drop(columns=['team'], inplace=True)

    # Check for NaNs introduced by the merge
    away_nans = match_df_merged['away_team_strength_proxy'].isnull().sum()
    if away_nans > 0:
        print(f"Warning: Found {away_nans} rows with missing away strength
    else:
        print(" Away team merge complete (no NaNs found).")

else:
    print("\nSkipping away merge because prerequisite DataFrame from home
    match_df_merged = None # Ensure variable exists but is None
```

```
Merging away team strength proxy...
 Away team merge complete (no NaNs found).
```

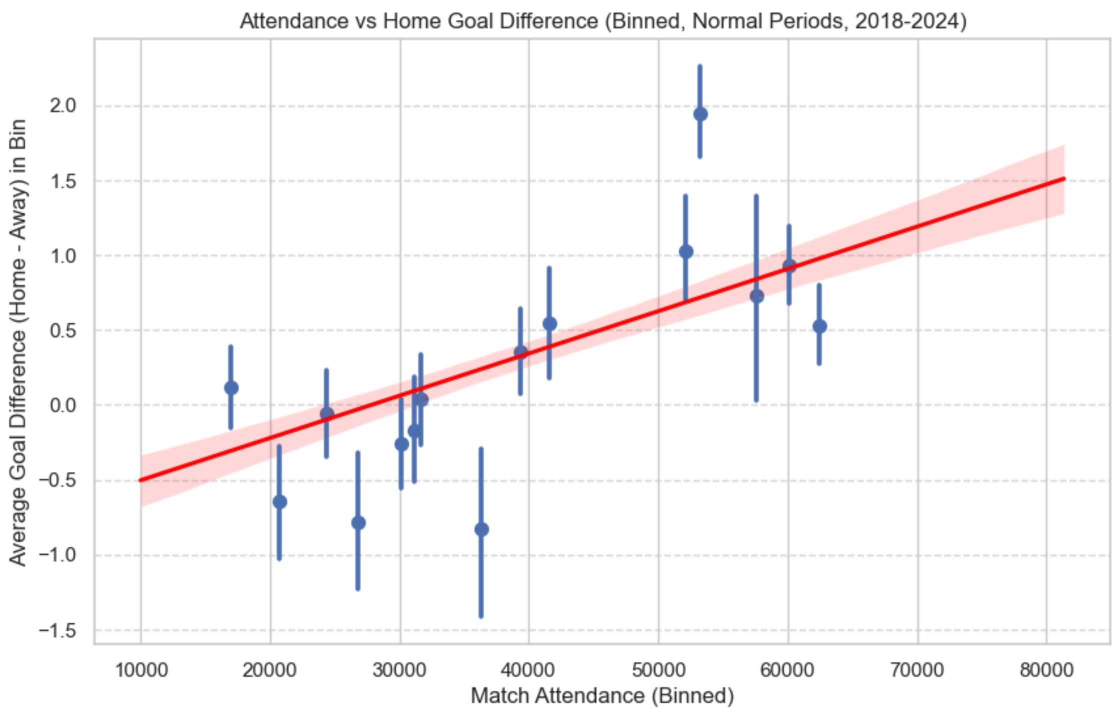
In [45]:
```
# Advanced Regression Analysis: Attendance + Team Strength Impact

# Check if necessary prerequisites exist
if ('match_df_merged' in locals() and match_df_merged is not None and
    'period' in match_df_merged.columns and
    'goal_difference' in match_df_merged.columns and
    'attendance' in match_df_merged.columns and
    'home_team_strength_proxy' in match_df_merged.columns and
    'away_team_strength_proxy' in match_df_merged.columns):

    print("\nSetting up advanced regression analysis (controlling for team

    # Filter Data
    regression_df_adv = match_df_merged[match_df_merged['period'] == 'Norm
    print(f" Using {len(regression_df_adv)} matches initially from the 'No

    # Handle NaNs in Regression Columns
    regression_cols_adv = ['goal_difference', 'attendance', 'home_team_str
    rows_before_drop = len(regression_df_adv)
    regression_df_adv.dropna(subset=regression_cols_adv, inplace=True) # K
    rows_after_drop = len(regression_df_adv)
    if rows_before_drop != rows_after_drop:
        print(f" Dropped {rows_before_drop - rows_after_drop} rows due to
    else:
        print(" No rows dropped due to NaNs.")

    # Define and Fit the OLS Model
    formula_adv = 'goal_difference ~ attendance + home_team_strength_proxy
    print(f"\nFitting OLS model with formula: {formula_adv}")

    try:
        model_adv = smf.ols(formula=formula_adv, data=regression_df_adv)
        results_adv = model_adv.fit()

        # Display Regression Results
        print("\n--- OLS Regression Results (Advanced Model) ---")
        print(results_adv.summary())
        print("-" * 80)

    except NameError as e:
         print(f"Error during model fitting: {e}")
    except Exception as e:
        print(f"An error occurred during model fitting or summary generati


else:
    print("\nSkipping advanced regression because merged DataFrame or requ
    print(" Ensure the previous cells preparing 'match_df_merged' have run
Setting up advanced regression analysis (controlling for team strengt
h)...
 Using 1812 matches initially from the 'Normal' period.
 No rows dropped due to NaNs.

Fitting OLS model with formula: goal_difference ~ attendance + home_team_
strength_proxy + away_team_strength_proxy

--- OLS Regression Results (Advanced Model) ---
                            OLS Regression Results
==========================================================================
=====
Dep. Variable:        goal_difference   R-squared:
0.282
Model:                            OLS   Adj. R-squared:
0.280
Method:                 Least Squares   F-statistic:
236.2
Date:                Mon, 21 Apr 2025   Prob (F-statistic):          2.71
e-129
Time:                        12:41:39   Log-Likelihood:                -3
488.2
No. Observations:                1812   AIC:
6984.
Df Residuals:                    1808   BIC:
7006.
Df Model:                           3
Covariance Type:            nonrobust
==========================================================================
===================
                               coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------
-------------------
Intercept                   -0.5903      0.107     -5.531      0.000
-0.800      -0.381
attendance                6.621e-06    2.7e-06      2.449      0.014
1.32e-06    1.19e-05
home_team_strength_proxy     1.0590      0.067     15.813      0.000
0.928       1.190
away_team_strength_proxy    -1.0875      0.065    -16.852      0.000
-1.214      -0.961
==========================================================================
=====
Omnibus:                       21.498   Durbin-Watson:
1.998
Prob(Omnibus):                  0.000   Jarque-Bera (JB):                  3
5.864
Skew:                          -0.015   Prob(JB):                        1.6
3e-08
Kurtosis:                       3.689   Cond. No.                        1.2
1e+05
==========================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The condition number is large, 1.21e+05. This might indicate that the
re are
```

```
strong multicollinearity or other numerical problems.
-----------------------------------------------------------------------------
-------
```

In [46]:

```
# Visualise Simple Regression Relationship (Binned Scatter Plot)

# Check if match_df exists and has the required columns
if ('match_df' in locals() and match_df is not None and
    'period' in match_df.columns and
    'goal_difference' in match_df.columns and
    'attendance' in match_df.columns):

    # Prepare Data (using the same subset as the simple regression model)
    simple_reg_df = match_df[match_df['period'] == 'Normal'].copy()

    # Ensure required columns are numeric and drop NaNs
    simple_reg_df['attendance'] = pd.to_numeric(simple_reg_df['attendance'
    simple_reg_df['goal_difference'] = pd.to_numeric(simple_reg_df['goal_d
    simple_reg_df.dropna(subset=['attendance', 'goal_difference'], inplace

    if len(simple_reg_df) > 0:
        # Create Binned Scatter Plot with Regression Line
        print("\nGenerating binned scatter plot with regression line...")
        plt.figure(figsize=(10, 6))

        # Use seaborn.regplot with x_bins
        num_bins = 15
        sns.regplot(x='attendance', y='goal_difference', data=simple_reg_d
                    x_bins=num_bins,
                    line_kws={'color':'red'}) # Make regression line red

        # Add Titles and Labels
        plt.title(f'Attendance vs Home Goal Difference (Binned, Normal Per
        plt.xlabel('Match Attendance (Binned)')
        plt.ylabel('Average Goal Difference (Home - Away) in Bin')
        plt.grid(axis='y', linestyle='--', alpha=0.7)

        # Define Filename and Save
        plot_filename_binned_reg = os.path.join(figures_dir, 'attendance_v
        try:
            plt.savefig(plot_filename_binned_reg, bbox_inches='tight', dpi
            print(f"Binned regression plot saved successfully to: {plot_fi
        except Exception as e:
            print(f"An error occurred while saving the binned regression p

        # Display the plot
        plt.show()

    else:
        print("Skipping visualisation: No data available after filtering/c

else:
    print("\nSkipping binned regression visualisation because match_df or
```

```
Generating binned scatter plot with regression line...
Binned regression plot saved successfully to: C:\Users\matth\OneDrive\Doc
uments\data_science_project\premier-league-home-advantage\output\figures
\attendance_vs_goaldiff_binned_regplot.png
```


Attendance vs Home Goal Difference (Binned, Normal Periods, 2018-2024)
Average Goal Difference (Home - Away) in Bin
Match Attendance (Binned)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
10000
20000
30000
40000
50000
60000
70000
80000

In [47]:

```python
# Time/Day Effects Prep: Process Start Time

# Check if prerequisite DataFrame exists
if 'match_df_merged' in locals() and match_df_merged is not None:
    print("\nProcessing 'start_time' to create time categories...")

    if 'start_time' in match_df_merged.columns:
        try:
            # Convert 'start_time' to datetime objects
            times = pd.to_datetime(match_df_merged['start_time'], format='
            # Extract the hour
            match_df_merged['kick_off_hour'] = times.dt.hour

            # Handle potential parsing errors
            if match_df_merged['kick_off_hour'].isnull().any():
                print(f"Warning: Could not parse start time for {match_df_

            # Define time categories based on kick-off hour
            conditions = [
                match_df_merged['kick_off_hour'] < 13,
                (match_df_merged['kick_off_hour'] >= 13) & (match_df_merge
                match_df_merged['kick_off_hour'] > 17
            ]
            categories = ['Early', 'Midday', 'Late']
            # Assign categories, default to 'Unknown' if hour is NaN
            match_df_merged['time_category'] = np.select(conditions, categ
            print(" Created 'time_category' column.")

            # Verification for this step
            print("\n--- Value Counts for 'time_category':")
            print(match_df_merged['time_category'].value_counts(dropna=Fal
            print("\n--- Head showing 'start_time', 'kick_off_hour', 'time
            display(match_df_merged[['start_time', 'kick_off_hour', 'time_

        except Exception as e:
            print(f" Error processing 'start_time': {e}")
            if 'time_category' not in match_df_merged.columns: match_df_me
    else:
        print(" 'start_time' column not found. Cannot create time categori
        if 'time_category' not in match_df_merged.columns: match_df_merged
else:
    print("\nSkipping start time processing because match_df_merged DataFr
```

```
Processing 'start_time' to create time categories...
 Created 'time_category' column.

--- Value Counts for 'time_category':
Midday    1428
Late       640
Early      212
Name: time_category, dtype: int64

--- Head showing 'start_time', 'kick_off_hour', 'time_category':
```

	start_time	kick_off_hour	time_category
0	20:00	20	Late
1	12:30	12	Early
2	15:00	15	Midday
3	15:00	15	Midday
4	15:00	15	Midday

In [48]:

```
# Time/Day Effects Prep: Process Date for Day of Week

# Check if prerequisite DataFrame exists
if 'match_df_merged' in locals() and match_df_merged is not None:
    print("\nProcessing 'date' to create 'day_of_week'...")

    if 'date' in match_df_merged.columns:
        try:
            # Ensure date column is datetime type
            if not pd.api.types.is_datetime64_any_dtype(match_df_merged['d
                match_df_merged['date'] = pd.to_datetime(match_df_merged['
                print(" Converted 'date' column to datetime.")

            # Check if conversion resulted in NaT dates before extracting
            if match_df_merged['date'].isnull().any():
                 print(f"Warning: Could not parse date for {match_df_merge

            # Extract day name (NaT dates will result in NaN day_of_week)
            match_df_merged['day_of_week'] = match_df_merged['date'].dt.da
            print(" Created 'day_of_week' column.")

            # Verification for this step
            print("\n--- Value Counts for 'day_of_week':")
            # dropna=False shows count of NaNs if any dates failed convers
            print(match_df_merged['day_of_week'].value_counts(dropna=False
            print("\n--- Head showing 'date' and 'day_of_week':")
            display(match_df_merged[['date', 'day_of_week']].head())

        except Exception as e:
             print(f" Error processing 'date': {e}")
             if 'day_of_week' not in match_df_merged.columns: match_df_mer
    else:
        print(" 'date' column not found. Cannot create day of week.")
        if 'day_of_week' not in match_df_merged.columns: match_df_merged['
else:
    print("\nSkipping date processing because match_df_merged DataFrame no
```

```
Processing 'date' to create 'day_of_week'...
 Converted 'date' column to datetime.
 Created 'day_of_week' column.

--- Value Counts for 'day_of_week':
Saturday     1065
Sunday        607
Wednesday     200
Monday        130
Tuesday       128
Thursday       81
Friday         69
Name: day_of_week, dtype: int64

--- Head showing 'date' and 'day_of_week':
```

	date	day_of_week
0	2018-08-10	Friday
1	2018-08-11	Saturday
2	2018-08-11	Saturday
3	2018-08-11	Saturday
4	2018-08-11	Saturday

In [49]:
```
# Time/Day Effects Analysis: Calculate Averages by Time Category

# Check if match_df_merged exists and has the required columns
if ('match_df_merged' in locals() and match_df_merged is not None and
    'time_category' in match_df_merged.columns and
    'goal_difference' in match_df_merged.columns and
    'home_points' in match_df_merged.columns):

    print("\nCalculating average home advantage metrics per kick-off time

    # Create a DataFrame without NaN values in the columns needed for grou
    valid_categories = ['Early', 'Midday', 'Late'] # Define valid categori
    calc_df = match_df_merged[
        match_df_merged['time_category'].isin(valid_categories) &
        match_df_merged['goal_difference'].notna() &
        match_df_merged['home_points'].notna()
    ].copy()

    if not calc_df.empty:
        # Group by 'time_category' and Calculate Means
        time_category_comparison = calc_df.groupby('time_category')[[
            'goal_difference',
            'home_points'
        ]].mean().reset_index()

        # Order Categories Logically
        time_order = ['Early', 'Midday', 'Late']
        try:
            # Convert 'time_category' to a categorical type with the speci
            time_category_comparison['time_category'] = pd.Categorical(
                time_category_comparison['time_category'],
                categories=time_order,
                ordered=True
            )
            # Sort the DataFrame based on the categorical order
            time_category_comparison = time_category_comparison.sort_value
            print(" Ordered results by time category: Early, Midday, Late.
        except Exception as e:
            print(f" Warning: Could not reorder categories: {e}. Results m

        print("\n--- Comparison of Average Metrics by Time Category ---")
        # Display the resulting comparison table
        display(time_category_comparison)
        print("-" * 60)

    else:
        print(" Warning: No valid data found for time category calculation
        time_category_comparison = None # Ensure variable exists but is No

else:
    print("\nSkipping time category calculation: 'match_df_merged' DataFra
```

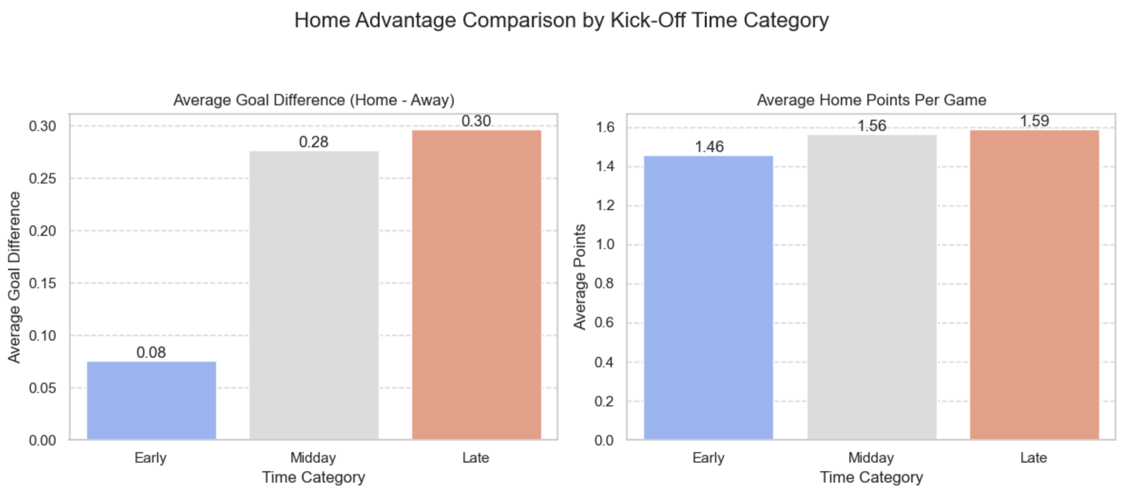
```
Calculating average home advantage metrics per kick-off time category...
 Ordered results by time category: Early, Midday, Late.

--- Comparison of Average Metrics by Time Category ---
```

	time_category	goal_difference	home_points
0	Early	0.075472	1.457547
2	Midday	0.276611	1.564426
1	Late	0.296875	1.589063

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

In [50]:
```
# Visualise Home Advantage Metrics Across Time Categories

# Check if the time_category_comparison DataFrame exists from the previous
if 'time_category_comparison' in locals() and time_category_comparison is
    print("\nGenerating bar charts comparing metrics across time categorie

    # Define the logical order for plotting categories
    # Filter time_order list to ensure only categories present in the data
    time_order = [cat for cat in ['Early', 'Midday', 'Late'] if cat in tim

    if not time_order:
         print("Warning: No valid categories ('Early', 'Midday', 'Late') f
    else:
        # Create the Plots
        # Create a figure and a set of subplots (1 row, 2 columns)
        fig, axes = plt.subplots(1, 2, figsize=(12, 5))
        fig.suptitle('Home Advantage Comparison by Kick-Off Time Category'

        # Plot 1: Average Goal Difference
        sns.barplot(ax=axes[0], x='time_category', y='goal_difference', da
                    palette='coolwarm', order=time_order)
        axes[0].set_title('Average Goal Difference (Home - Away)')
        axes[0].set_xlabel('Time Category')
        axes[0].set_ylabel('Average Goal Difference')
        axes[0].grid(axis='y', linestyle='--', alpha=0.7)
        # Add exact values on bars
        for container in axes[0].containers:
            axes[0].bar_label(container, fmt='%.2f')

        # Plot 2: Average Home Points
        sns.barplot(ax=axes[1], x='time_category', y='home_points', data=t
                    palette='coolwarm', order=time_order)
        axes[1].set_title('Average Home Points Per Game')
        axes[1].set_xlabel('Time Category')
        axes[1].set_ylabel('Average Points')
        axes[1].grid(axis='y', linestyle='--', alpha=0.7)
        # Add exact values on bars
        for container in axes[1].containers:
            axes[1].bar_label(container, fmt='%.2f')

        # Final Touches & Saving
        plt.tight_layout(rect=[0, 0, 1, 0.97]) # Adjust layout to prevent

        # Define filename and save
        plot_filename_time_cat_comp = os.path.join(figures_dir, 'time_cate
        try:
            # Ensure the figures directory exists
            os.makedirs(figures_dir, exist_ok=True)
            plt.savefig(plot_filename_time_cat_comp, bbox_inches='tight',
            print(f" Time category comparison plots saved successfully to:
        except Exception as e:
            print(f" An error occurred while saving the time category comp

        # Display the Plot
        plt.show()

else:
    print("\nSkipping visualisation: 'time_category_comparison' DataFrame
    print(" Ensure the cell calculating time_category_comparison ran succe
```

```
Generating bar charts comparing metrics across time categories...
 Time category comparison plots saved successfully to: C:\Users\matth\One
Drive\Documents\data_science_project\premier-league-home-advantage\output
\figures\time_category_comparison_bar_charts.png
```



In [51]:

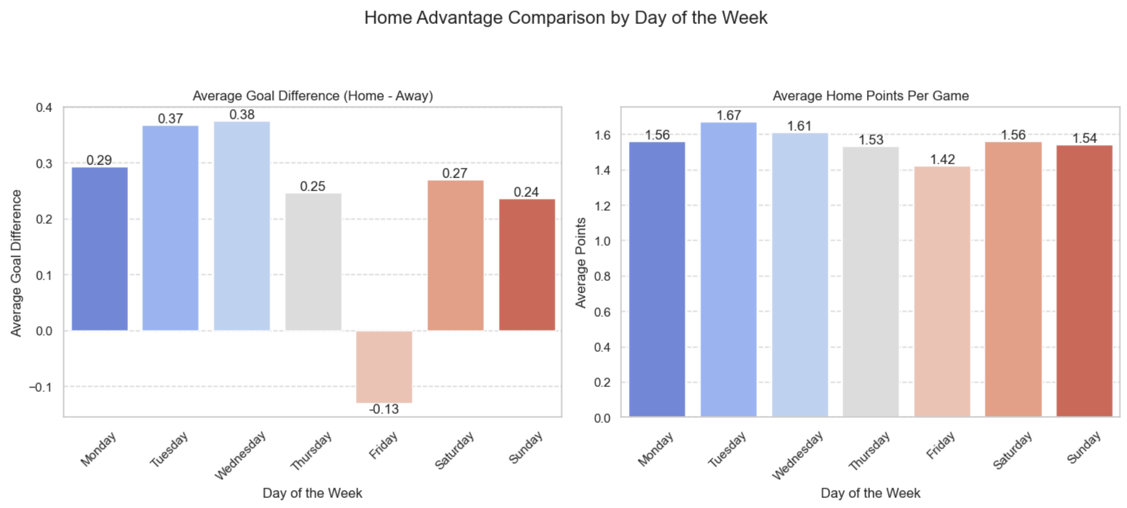
```
# Time/Day Effects Analysis: Calculate Averages by Day of Week

# Check if match_df_merged exists and has the required columns
if ('match_df_merged' in locals() and match_df_merged is not None and
    'day_of_week' in match_df_merged.columns and
    'goal_difference' in match_df_merged.columns and
    'home_points' in match_df_merged.columns):

    print("\nCalculating average home advantage metrics per day of the wee

    # Create a DataFrame without NaN values in the columns needed for grou
    valid_days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
    calc_df_dow = match_df_merged[
        match_df_merged['day_of_week'].isin(valid_days) &
        match_df_merged['goal_difference'].notna() &
        match_df_merged['home_points'].notna()
    ].copy()

    if not calc_df_dow.empty:
        # Group by 'day_of_week' and Calculate Means
        day_of_week_comparison = calc_df_dow.groupby('day_of_week')[[
            'goal_difference',
            'home_points'
        ]].mean().reset_index()

        # Order Days Logically
        # Define the desired order for the days
        day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday
        try:
            # Convert 'day_of_week' to a categorical type with the specifi
            day_of_week_comparison['day_of_week'] = pd.Categorical(
                day_of_week_comparison['day_of_week'],
                categories=day_order,
                ordered=True
            )
            # Sort the DataFrame based on the categorical order
            day_of_week_comparison = day_of_week_comparison.sort_values('d
            print(" Ordered results by day of the week (Mon-Sun).")
        except Exception as e:
            print(f" Warning: Could not reorder days: {e}. Results might b

        print("\n--- Comparison of Average Metrics by Day of Week ---")
        # Display the resulting comparison table
        display(day_of_week_comparison)
        print("-" * 60)

    else:
         print(" Warning: No valid data found for day of week calculation
         day_of_week_comparison = None # Ensure variable exists but is Non

else:
    print("\nSkipping day of week calculation: 'match_df_merged' DataFrame
```

```

Calculating average home advantage metrics per day of the week...
 Ordered results by day of the week (Mon-Sun).

--- Comparison of Average Metrics by Day of Week ---
```

	day_of_week	goal_difference	home_points
1	Monday	0.292308	1.561538
5	Tuesday	0.367188	1.671875
6	Wednesday	0.375000	1.610000
4	Thursday	0.246914	1.530864
0	Friday	-0.130435	1.420290
2	Saturday	0.269484	1.561502
3	Sunday	0.235585	1.542010

In [52]:
```
# Visualise Home Advantage Metrics Across Days of the Week

# Check if the day_of_week_comparison DataFrame exists from the previous c
if 'day_of_week_comparison' in locals() and day_of_week_comparison is not
    print("\nGenerating bar charts comparing metrics across days of the we

    # Define the logical order for plotting days
    day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', '

    # Check if the DataFrame is empty before proceeding
    if day_of_week_comparison.empty:
         print("Warning: The 'day_of_week_comparison' DataFrame is empty.
    else:
        # Create the Plots
        # Create a figure and a set of subplots (1 row, 2 columns)
        fig, axes = plt.subplots(1, 2, figsize=(14, 6)) # Slightly wider f
        fig.suptitle('Home Advantage Comparison by Day of the Week', fonts

        # Plot 1: Average Goal Difference
        # Plot directly using day_of_week_comparison, relying on previous
        sns.barplot(ax=axes[0], x='day_of_week', y='goal_difference', data
                    palette='coolwarm', order=day_order)
        axes[0].set_title('Average Goal Difference (Home - Away)')
        axes[0].set_xlabel('Day of the Week')
        axes[0].set_ylabel('Average Goal Difference')
        axes[0].tick_params(axis='x', rotation=45) # Rotate x-axis labels
        axes[0].grid(axis='y', linestyle='--', alpha=0.7)
        # Add exact values on bars
        for container in axes[0].containers:
            axes[0].bar_label(container, fmt='%.2f')

        # Plot 2: Average Home Points
        # Plot directly using day_of_week_comparison, relying on previous
        sns.barplot(ax=axes[1], x='day_of_week', y='home_points', data=day
                    palette='coolwarm', order=day_order)
        axes[1].set_title('Average Home Points Per Game')
        axes[1].set_xlabel('Day of the Week')
        axes[1].set_ylabel('Average Points')
        axes[1].tick_params(axis='x', rotation=45) # Rotate x-axis labels
        axes[1].grid(axis='y', linestyle='--', alpha=0.7)
        # Add exact values on bars
        for container in axes[1].containers:
            axes[1].bar_label(container, fmt='%.2f')

        # Final Touches & Saving
        plt.tight_layout(rect=[0, 0, 1, 0.96]) # Adjust layout

        # Define filename and save
        plot_filename_dow_comp = os.path.join(figures_dir, 'day_of_week_co
        try:
            # Ensure the figures directory exists
            os.makedirs(figures_dir, exist_ok=True)
            plt.savefig(plot_filename_dow_comp, bbox_inches='tight', dpi=3
            print(f" Day of week comparison plots saved successfully to: {
        except Exception as e:
            print(f" An error occurred while saving the day of week compar

        # Display the Plot
        plt.show()

else:
    print("\nSkipping visualisation: 'day_of_week_comparison' DataFrame no
```

```
Generating bar charts comparing metrics across days of the week...
 Day of week comparison plots saved successfully to: C:\Users\matth\OneDr
ive\Documents\data_science_project\premier-league-home-advantage\output\f
igures\day_of_week_comparison_bar_charts.png
```



In [53]:
```
# Enhanced Regression Analysis: Including Time/Day Effects

# Check if necessary prerequisites exist in the merged DataFrame
if ('match_df_merged' in locals() and match_df_merged is not None and
    'period' in match_df_merged.columns and
    'goal_difference' in match_df_merged.columns and
    'attendance' in match_df_merged.columns and
    'home_team_strength_proxy' in match_df_merged.columns and
    'away_team_strength_proxy' in match_df_merged.columns and
    'time_category' in match_df_merged.columns and
    'day_of_week' in match_df_merged.columns):

    print("\nSetting up enhanced regression analysis (controlling for stre

    # Filter Data
    regression_df_adv = match_df_merged[match_df_merged['period'] == 'Norm
    print(f" Using {len(regression_df_adv)} matches initially from the 'No

    # Handle NaNs in required regression columns
    regression_cols_adv = [
        'goal_difference', 'attendance', 'home_team_strength_proxy',
        'away_team_strength_proxy', 'time_category', 'day_of_week'
        ]
    rows_before_drop = len(regression_df_adv)
    regression_df_adv.dropna(subset=regression_cols_adv, inplace=True)
    rows_after_drop = len(regression_df_adv)
    if rows_before_drop != rows_after_drop:
        print(f" Dropped {rows_before_drop - rows_after_drop} rows due to
    else:
        print(" No rows dropped due to NaNs in regression columns.")

    # Check if data remains after NaN drop before proceeding
    if not regression_df_adv.empty:
        # Define and Fit the OLS Model
        # Use C() to handle categorical variables (time_category, day_of_w
        formula_adv = 'goal_difference ~ attendance + home_team_strength_p
        print(f"\nFitting OLS model with formula: {formula_adv}")

        try:
            model_adv = smf.ols(formula=formula_adv, data=regression_df_ad
            results_adv = model_adv.fit()

            # Display Regression Results
            print("\n--- OLS Regression Results (Enhanced Model with Time/
            print(results_adv.summary())
            print("-" * 80)

        except NameError as e:
             # Added check for smf import in case of NameError
             print(f"Error during model fitting: {e}. Check if statsmodels
        except Exception as e:
            print(f"An error occurred during model fitting or summary gene
    else:
         print("Skipping regression: No data available after filtering for

else:
    # Handle case where prerequisite DataFrame or columns are missing
    print("\nSkipping enhanced regression: 'match_df_merged' DataFrame or
    print(" Ensure the previous cells preparing the merged data and time/d
Setting up enhanced regression analysis (controlling for strength, attend
ance, time, and day)...
 Using 1812 matches initially from the 'Normal' period.
 No rows dropped due to NaNs in regression columns.

Fitting OLS model with formula: goal_difference ~ attendance + home_team_
strength_proxy + away_team_strength_proxy + C(time_category) + C(day_of_w
eek)

--- OLS Regression Results (Enhanced Model with Time/Day) ---
                            OLS Regression Results
==============================================================================
=====
Dep. Variable:        goal_difference   R-squared:
0.284
Model:                            OLS   Adj. R-squared:
0.279
Method:                 Least Squares   F-statistic:
64.76
Date:                Mon, 21 Apr 2025   Prob (F-statistic):          6.12
e-122
Time:                        12:42:18   Log-Likelihood:                 -3
485.7
No. Observations:                1812   AIC:
6995.
Df Residuals:                    1800   BIC:
7061.
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
======================
                               coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
----------------------
Intercept                   -0.8556      0.355     -2.407      0.016
-1.553      -0.158
C(time_category)[T.Late]     0.0081      0.247      0.033      0.974
-0.476       0.492
C(time_category)[T.Midday]  -0.0206      0.148     -0.139      0.889
-0.311       0.269
C(day_of_week)[T.Monday]     0.1748      0.301      0.580      0.562
-0.416       0.765
C(day_of_week)[T.Saturday]   0.2705      0.321      0.842      0.400
-0.360       0.901
C(day_of_week)[T.Sunday]     0.2795      0.330      0.848      0.396
-0.367       0.926
C(day_of_week)[T.Thursday]   0.1764      0.326      0.541      0.589
-0.463       0.816
C(day_of_week)[T.Tuesday]    0.5725      0.296      1.932      0.054
-0.009       1.154
C(day_of_week)[T.Wednesday]  0.3243      0.277      1.173      0.241
-0.218       0.867
attendance                6.588e-06   2.73e-06      2.416      0.016
1.24e-06    1.19e-05
home_team_strength_proxy     1.0611      0.067     15.758      0.000
0.929       1.193
away_team_strength_proxy    -1.0909      0.066    -16.573      0.000
-1.220      -0.962
==============================================================================
=====
Omnibus:                       19.616   Durbin-Watson:                   
2.003
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               3
1.819
Skew:                          -0.007   Prob(JB):                     1.2
3e-07
Kurtosis:                       3.649   Cond. No.                     7.8
1e+05
==========================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The condition number is large, 7.81e+05. This might indicate that the
re are
strong multicollinearity or other numerical problems.
--------------------------------------------------------------------------
-------
```

```
In [ ]:
```