Author: David Schenck
GitHub: https://github.com/daviderics/mls_attendance (https://github.com/daviderics/mls_attendance)
November, 2023

Modeling

In this notebook, I create different types of models that attempt to predict attendance of MLS matches.

Types of models:

1. Linear Regression
2. XGBoost Regression
3. Random Forest Regression
4. K-Nearest Neighbors Regression

Business Problem

Major League Soccer (MLS) is the top professional soccer league in the United States and Canada. It began play in 1996 and has been a driving force in growing the sport of soccer in the U.S. ever since.

A major source of income for a league and its teams is ticket sales. MLS exists in a crowded sports landscape in which it competes with both other professional sports and college sports. Avoiding competing head-to-head with the National Football League (NFL) is a major reason that MLS starts its season in late February and ends in early December, out of phase with most other soccer leagues worldwide which start in August and end in May.

The goal of this project is to determine when MLS should hold their matches in order to maximize attendance. I will use seasons ranging from 2018 to 2023 to perform my analysis, but I will exclude 2020 because almost the entire season was played without fans in attendance due to the COVID-19 pandemic.

I will look at the following factors related to time:

1. Time of day. Matches start as early as noon and as late as 10 pm.
2. Day of the week. Most matches are played on weekends, a fair number are played on Wednesdays, and a smaller number are played on other weekdays.
3. Month. The season starts in February and ends in December, though the regular season ends in October.
4. Year.

I will also look at these factors related to weather:

1. Temperature. I am especially looking to see if especially hot or cold temperatures negatively affect attendance.

2. Rain and snow. I hypothesize that precipitation could reduce attendance.
3. Windspeed.

The models will also incorporate these factors:

1. Regular season vs. playoffs: I expect playoff matches to have higher attendance.
2. Home openers: In data_exploration.ipynb, there was some evidence that home openers tend to have better attendance than average.
3. Rivalry matches: MLS has quite a few rivalries that could help drive fans to attend matches.
4. Matches without a real home team: Each team has a stadium that it uses for its home matches. However, there have been cases where matches had to be held in neutral locations. There is reason to believe these matches have MUCH lower attendance than average.

Note about the models

While I am creating regression models that could make predictions of attendance for single matches, the goal of this work is not to use the model to make match-by-match predictions. I expect there to be a lot of variance in attendance that is not related to the factors listed above, making a single prediction somewhat unreliable. Instead, the goal is to look at the way the factors are incorporated into the models to see what is important and how it affects attendance.

Data Collection

The match and weather data were collected using this notebook:
https://github.com/daviderics/mls_attendance/blob/main/notebooks/collect_match_weather_da
(https://github.com/daviderics/mls_attendance/blob/main/notebooks/collect_match_weather_d

The actual data is saved as a CSV file (mls_with_weather.csv).

Someone who wants to reproduce this work can either choose to run collect_match_weather_data.ipynb themselves to generate the data or find the data in the notebooks/ directory. There will be a slight difference in the data just because I collected the data before the 2023 MLS season ended, so collecting the data now will collect a few more matches.

Data Exploration

I explored trends in the data using this notebook:
https://github.com/daviderics/mls_attendance/blob/main/notebooks/data_exploration.ipynb
(https://github.com/daviderics/mls_attendance/blob/main/notebooks/data_exploration.ipynb)

Here are the conclusions I made based on the data exploration:

Months: There is some evidence that attendance does change from month to month. Other factors might be responsible, but it does look like attendance might improve as the season progresses.

Day of the week: It looks like Sundays might be the best for attendance while Wednesdays are the worst. However, it is possible that Sundays had the highest average attendance because more marquee matches were scheduled for Sundays.

Time of day: There is some evidence that early/mid afternoon matches might be slightly better than late afternoon or night matches.

Rivals: It does look like there could be a significant increase in attendance when two rivals are playing.

Home team, away team: The models will definitely need to take into account which teams are playing as there is significant variation between them.

Weather: Cold temperatures do appear to be correlated with lower attendance. Other than that, there were not any clear trends with other weather features.

The models that I create should do a better job of finding real trends because the models will do a better job of accounting for correlations between features. For example, it is possible that once the relative averages of each home team are taken into account, some

Reproducability

I tested the reproducability of this notebook in Google colab. The entire notebook ran without error.

In order to get the notebook to run correctly on colab, I needed to get model_funcs.py, which contains supplementary functions for evaluating the models, and mls_with_weather.csv, the data. If you clone the entire GitHub repository for this project, then the code should work without the cell below, in which case you can set on_colab to False.

In [1]:
```
# Included for reproducability

#on_colab = True
on_colab = False

if on_colab:
    # Get supplementary functions from model_funcs.py
    !wget https://github.com/daviderics/mls_attendance/raw/main/notebool

    # Get data that was produced by collect_match_weather_data.ipynb
    !wget https://github.com/daviderics/mls_attendance/raw/main/notebool
```

In [2]:

```
import numpy as np
import scipy.stats as sp
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import statsmodels.api as sm
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score
from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
import warnings
warnings.filterwarnings('ignore', category=FutureWarning)
from model_funcs import *
```

In [3]:
```
# Read in data
mlsall_df = pd.read_csv('mls_with_weather.csv', index_col=0)
mlsall_df
```

Out[3]:

	round	day	date	local_time	home_team	home_score	away_score	away_tean
0	Regular Season	Sat	2018-03-03	13.0	27	0	2	
1	Regular Season	Sat	2018-03-03	14.5	9	4	0	
2	Regular Season	Sun	2018-03-04	14.0	24	0	1	1
3	Regular Season	Sun	2018-03-04	15.0	28	2	1	1
4	Regular Season	Sat	2018-03-03	19.0	20	2	0	1
...	...	...	...	...	...	...	...	..
2280	Regular Season	Sat	2023-10-21	19.0	5	0	1	2
2281	Regular Season	Sat	2023-10-21	20.0	26	0	2	2
2282	Regular Season	Sat	2023-10-21	20.0	25	3	1	1
2283	Wild Card Round	Wed	2023-10-25	19.5	18	5	2	
2284	Wild Card Round	Wed	2023-10-25	20.5	25	0	0	2

2285 rows × 29 columns

Below, I drop matches for which weather data is missing and the attendance is 0. If the attendance is 0, it either means the information was missing or fans were not allowed due to the pandemic.

In [4]:
```
# Drop anything with missing data
mlsall_df.dropna(inplace=True)
# Drop matches with attendance=0
mlsall_df.drop(mlsall_df[mlsall_df['attendance']==0].index, inplace=Tru
mlsall_df
```

Out[4]:

	round	day	date	local_time	home_team	home_score	away_score	away_team
0	Regular Season	Sat	2018-03-03	13.0	27	0	2	6
1	Regular Season	Sat	2018-03-03	14.5	9	4	0	0
2	Regular Season	Sun	2018-03-04	14.0	24	0	1	11
3	Regular Season	Sun	2018-03-04	15.0	28	2	1	14
4	Regular Season	Sat	2018-03-03	19.0	20	2	0	16
...	...	...	...	...	...	...	...	...
2278	Regular Season	Sat	2023-10-21	18.0	21	1	3	9
2279	Regular Season	Sat	2023-10-21	18.0	28	1	1	11
2280	Regular Season	Sat	2023-10-21	19.0	5	0	1	22
2281	Regular Season	Sat	2023-10-21	20.0	26	0	2	24
2282	Regular Season	Sat	2023-10-21	20.0	25	3	1	12

2229 rows × 29 columns

After dropping no weather, no attendance matches, there are 2229 matches remaining.

In [5]:
```
# Get data types of each column
mlsall_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2229 entries, 0 to 2282
Data columns (total 29 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   round             2229 non-null   object
 1   day               2229 non-null   object
 2   date              2229 non-null   object
 3   local_time        2229 non-null   float64
 4   home_team         2229 non-null   int64
 5   home_score        2229 non-null   int64
 6   away_score        2229 non-null   int64
 7   away_team         2229 non-null   int64
 8   attendance        2229 non-null   int64
 9   stadium           2229 non-null   object
 10  latitude          2229 non-null   float64
 11  longitude         2229 non-null   float64
 12  playoff           2229 non-null   int64
 13  att_div_capacity  2229 non-null   float64
 14  real_home_team    2229 non-null   int64
 15  same_conf         2229 non-null   int64
 16  rivals            2229 non-null   int64
 17  temperature       2229 non-null   float64
 18  rain              2229 non-null   float64
 19  snow              2229 non-null   float64
 20  cloudcover        2229 non-null   float64
 21  windspeed         2229 non-null   float64
 22  windgust          2229 non-null   float64
 23  rain_sum          2229 non-null   float64
 24  snow_sum          2229 non-null   float64
 25  date_year         2229 non-null   int64
 26  date_month        2229 non-null   int64
 27  date_day          2229 non-null   int64
 28  home_opener       2229 non-null   int64
dtypes: float64(12), int64(13), object(4)
memory usage: 522.4+ KB
```

Target variable

There are two options for the target variable: attendance and attendance/capacity. I plan on using home team and away team as categorical parameters in the model, so even if I choose to use raw attendance, there will be parameters that effectively normalize the attendance per team. Therefore, I will use raw attendance as the target variable.

In [6]:
```
# Target variable
#y = mlsall_df['att_div_capacity']
y = mlsall_df['attendance']
```

Evaluating Models

Throughout this notebook, I will use two numerical metrics for evaluating the fits: root-mean-square error (RMSE) and R^2.

The RMSE will give me a sense of the typical amount by which the predicted attendance deviates from the true attendance. Smaller values are better because it means more accurate predictions.

R^2 will indicate how much of the variance in the attendance is explained by a given model. Larger values (near 1) are better.

Linear Regression

The way I am going to create the linear regression model is by iteratively trying different combinations of inputs and evaluating the success of the model based on the RMSE of the residuals and the R-squared of the fit.

Model 1

To start, I will just fit a single constant to the data.

In [7]:
```
X = create_X(mlsall_df, columns=[], add_constant=True)
linreg = sm.OLS(y,X).fit()
linreg.summary()
```

Out[7]:

OLS Regression Results

Dep. Variable:	attendance	**R-squared:**	0.000
Model:	OLS	**Adj. R-squared:**	0.000
Method:	Least Squares	**F-statistic:**	nan
Date:	Mon, 04 Dec 2023	**Prob (F-statistic):**	nan
Time:	13:32:36	**Log-Likelihood:**	-23554.
No. Observations:	2229	**AIC:**	4.711e+04
Df Residuals:	2228	**BIC:**	4.712e+04
Df Model:	0		
Covariance Type:	nonrobust		

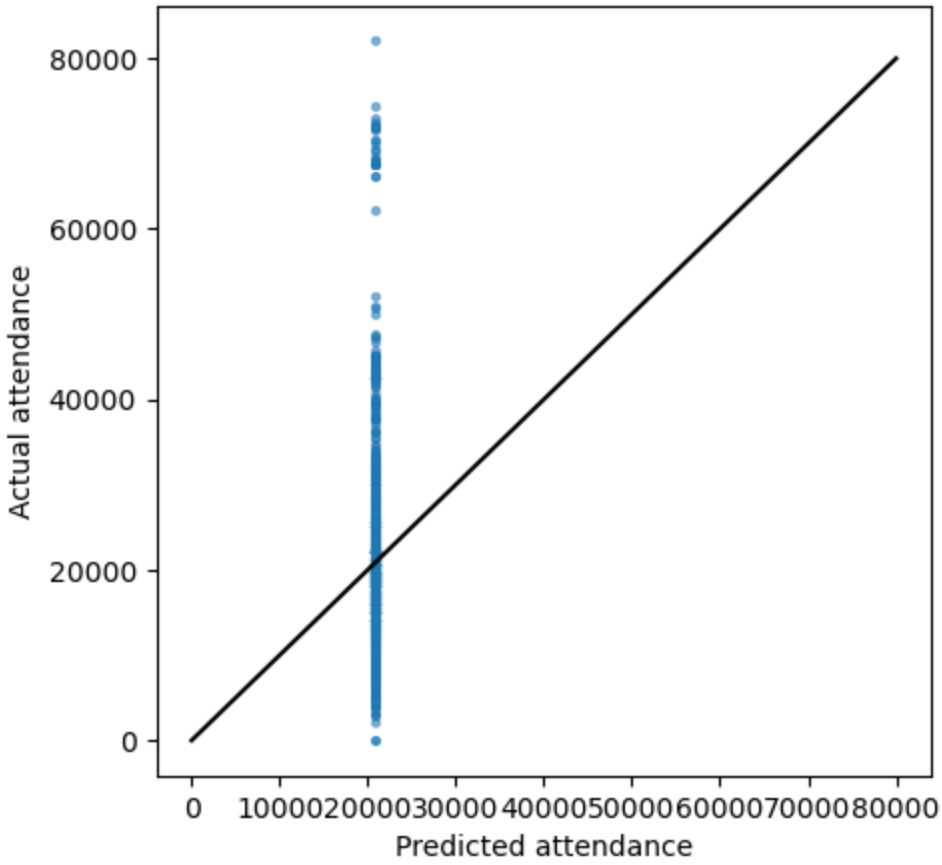
	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	2.082e+04	199.049	104.579	0.000	2.04e+04	2.12e+04

Omnibus:	1176.355	**Durbin-Watson:**	1.850
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	9640.591
Skew:	2.370	**Prob(JB):**	0.00
Kurtosis:	12.019	**Cond. No.**	1.00

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [8]:
```
evaluate_linear_model(linreg, X, y)
```

```
RMSE of residuals: 9397.534829728174
R-squared: 1.1102230246251565e-16
```



This model predicts 20,816 for each match. Not surprisingly, this is a pretty bad fit as it does not account for any of the trends in the data.

The RMSE for this fit was 9,398. The goal of subsequent models will be to reduce this number.

Model 2

This model will add home and away team parameters. I expect this to improve the fit quite a lot because it will account for differences in stadium capacities.

In [9]:
```
X = create_X(mlsall_df, columns=['home_team','away_team'], add_constant=
linreg = sm.OLS(y,X).fit()
linreg.summary()
```

Out[9]:

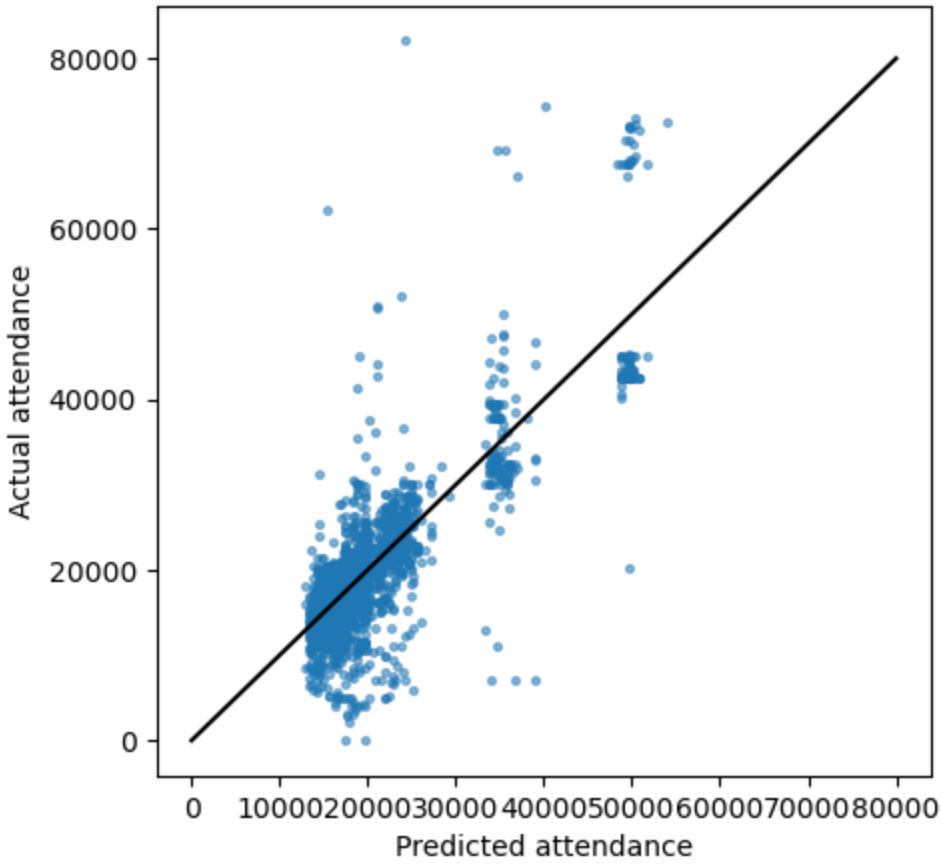
OLS Regression Results

Dep. Variable:	attendance	**R-squared:**	0.662
Model:	OLS	**Adj. R-squared:**	0.653
Method:	Least Squares	**F-statistic:**	75.95
Date:	Mon, 04 Dec 2023	**Prob (F-statistic):**	0.00
Time:	13:32:37	**Log-Likelihood:**	-22345.
No. Observations:	2229	**AIC:**	4.480e+04
Df Residuals:	2172	**BIC:**	4.513e+04
Df Model:	56		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.969e+04	845.914	58.746	0.000	4.8e+04	5.14e+04

In [10]:
```
evaluate_linear_model(linreg, X, y)
```

```
RMSE of residuals: 5533.834547035362
R-squared: 0.6619592263521832
```



Accounting for the teams involved in the match improved the fit. The RMSE of the residuals dropped from 9,398 to 5,534.

The R-squared is now at 0.662. The parameters for home and away team will be kept for all subsequent models.

For note, the constant in the fit (49,694) is a lot higher than it was for model 1. This is because the new variables are all in reference to Atlanta United which has the highest average attendance for its home matches. This does strangely mean that the constant assumes a match between Atlanta and Atlanta, but that is not actually a problem.

Model 3

In this model, I incorporate a few time-based parameters:

1. Day of the week
2. Month (reference case is a match in July)
3. Local time of kick off (categorical)
4. Year (2018 is reference case)

In [11]:
```
X = create_X(mlsall_df,
             columns=['home_team','away_team',
                      'day','date_month','local_time','date_year'],
             add_constant=True)
linreg = sm.OLS(y,X).fit()
linreg.summary()
```

Out[11]:

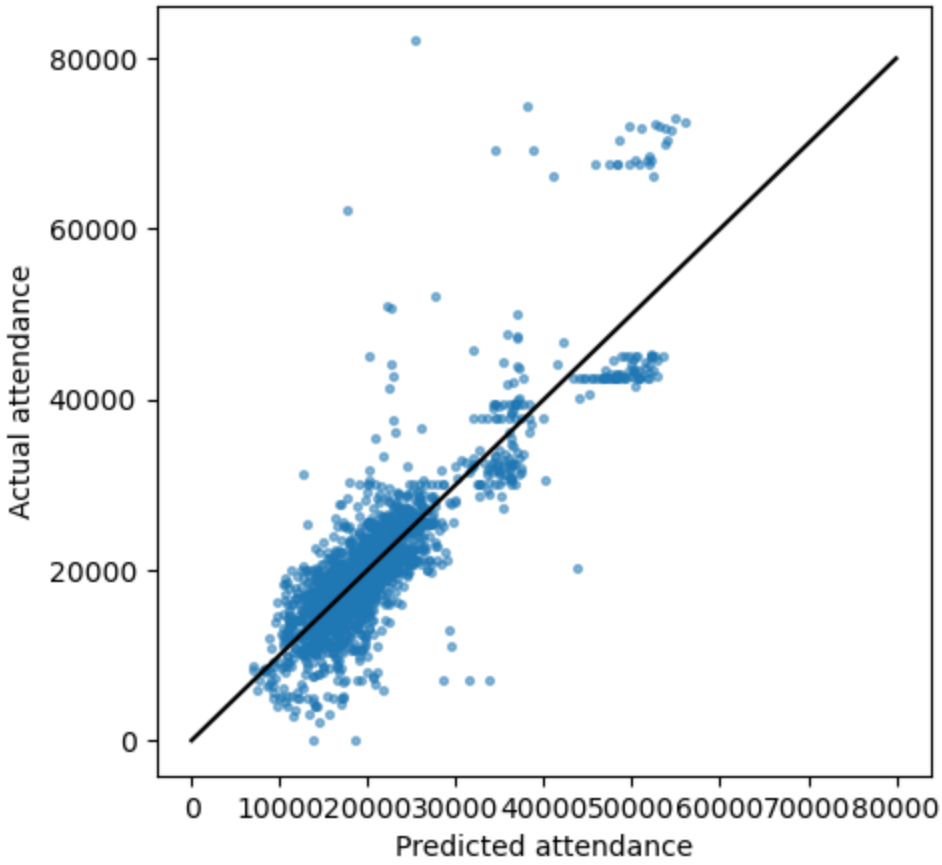
OLS Regression Results

Dep. Variable:	attendance	**R-squared:**	0.728
Model:	OLS	**Adj. R-squared:**	0.719
Method:	Least Squares	**F-statistic:**	82.59
Date:	Mon, 04 Dec 2023	**Prob (F-statistic):**	0.00
Time:	13:32:38	**Log-Likelihood:**	-22102.
No. Observations:	2229	**AIC:**	4.435e+04
Df Residuals:	2158	**BIC:**	4.475e+04
Df Model:	70		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
date_month	543.4748	49.962	10.878	0.000	445.495	641.454

In [12]:
```
evaluate_linear_model(linreg, X, y)
```

```
RMSE of residuals: 4978.200192318331
R-squared: 0.7281977294597555
```



The fit improved a bit. The R-squared is now 0.728 and the RMSE is 4,978.

Day of the week: All of the parameters for day of the week were negative, indicating that Saturday is the best day to hold matches. However, only Tuesday and Wednesday had p-values below 0.05. The result for Wednesday is quite telling. MLS holds a lot of matches on Wednesdays, but the model here indicates that those matches expect over 3000 fewer attendees than on Saturdays.

Month: The one month parameter is statistically significant according to the p-value. The value of the parameter is 543, which means that attendance goes up by 543 people on average for every month that goes by. That is a significant increase for a season that spans 11 months. However, this number might be biased high because the model does not yet account for the effect of playoff matches which happen at the end of the season. Once that is incorporated into the model, I expect the value of the month parameter to decrease a bit.

Kick off time: None of the parameters were statistically significant at the 0.05 level. The numbers indicate that matches played between 2 and 5 did the best. It is important to keep in mind that almost all of the matches like this were played on the weekend.

Year: Both 2021 and 2022 had significantly lower attendance than 2018. The effect of the pandemic was quite clear in 2021 when the average attendance was about 6000 lower than

Model 4

This model incorporates a few binary factors:

1. Playoff or not
2. Home opener or not
3. Rivals or not
4. Real home team or not

For now, I am going to keep all of the parameters from Model 3, but I am going to keep my eye on the local_time variables to see if they are significant or not.

In [13]:
```
X = create_X(mlsall_df,
             columns=['home_team','away_team',
                      'day','date_month','local_time','date_year',
                      'playoff','home_opener','rivals','real_home_team'],
             add_constant=True)
linreg = sm.OLS(y,X).fit()
linreg.summary()
```

Out[13]:

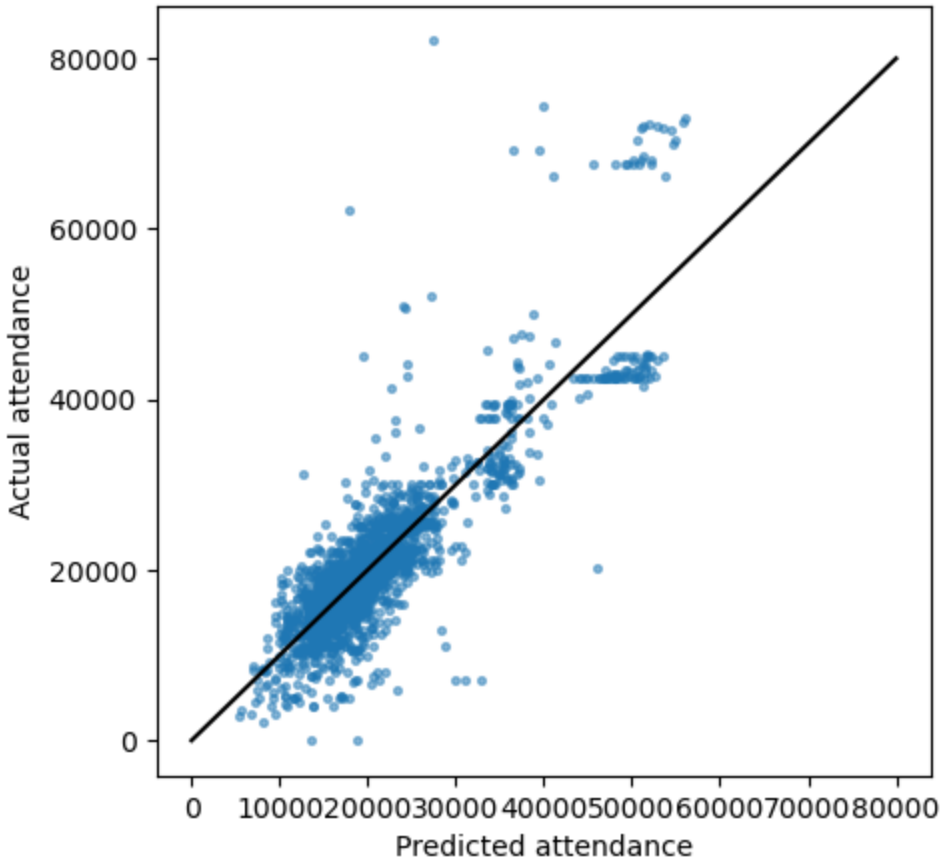
OLS Regression Results

Dep. Variable:	attendance	**R-squared:**	0.739
Model:	OLS	**Adj. R-squared:**	0.730
Method:	Least Squares	**F-statistic:**	82.30
Date:	Mon, 04 Dec 2023	**Prob (F-statistic):**	0.00
Time:	13:32:38	**Log-Likelihood:**	-22058.
No. Observations:	2229	**AIC:**	4.427e+04
Df Residuals:	2154	**BIC:**	4.469e+04
Df Model:	74		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
date_month	581.6798	55.051	10.566	0.000	473.722	689.638

In [14]:
```
evaluate_linear_model(linreg, X, y)
```

```
RMSE of residuals: 4885.263804900885
R-squared: 0.7387365449922851
```



The fit improved slightly. The R-squared is up to 0.739 and the RMSE is down to 4,885.

Playoff: I was surprised that the parameter for indicating whether a match was a playoff match was not significant at the 0.05 level.

Home openers: The parameter for home openers was significant and it indicates that home openers attract about 2,200 more people than would otherwise be expected.

Rivals: This parameter was also significant. Rivalry matches attract about 2,300 more people on average. Certain teams in MLS don't have rivalries, yet. Can we get Minnesota and Nashville to hate each other?

Real home team: Matches in which the home team was actually the home team averaged nearly 7,100 more fans.

The parameter for month is actually higher now (582).

Wednesdays and Thursdays are both statistically worse than Saturdays for attendance and the effect is quite large. Both get over 2000 fewer people per match.

None of the parameters for kick off time are significant still.

Model 5

This model will incorporate weather parameters

1. Whether it rained prior to match or at start of match.
2. Whether it snowed prior to match or at start of match.
3. Whether it was below 40 degrees Fahrenheit.
4. Whether it was above 90 degrees Fahrenheit.
5. Windspeed.

In [15]:
```
X = create_X(mlsall_df,
             columns=['home_team','away_team',
                      'day','date_month','local_time','date_year',
                      'playoff','home_opener','rivals','real_home_team',
                      'rain','snow','temperature','windspeed'],
             add_constant=True)
linreg = sm.OLS(y,X).fit()
linreg.summary()
```

Out[15]:

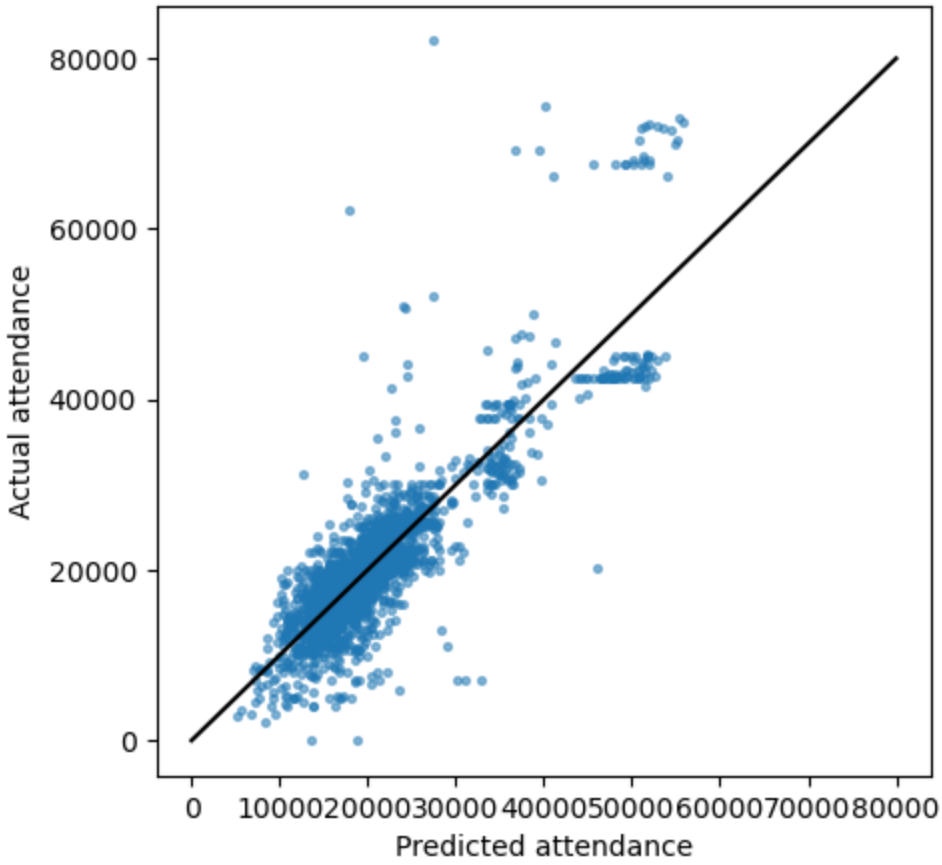
OLS Regression Results

Dep. Variable:	attendance	**R-squared:**	0.739
Model:	OLS	**Adj. R-squared:**	0.729
Method:	Least Squares	**F-statistic:**	77.02
Date:	Mon, 04 Dec 2023	**Prob (F-statistic):**	0.00
Time:	13:32:39	**Log-Likelihood:**	-22057.
No. Observations:	2229	**AIC:**	4.427e+04
Df Residuals:	2149	**BIC:**	4.473e+04
Df Model:	79		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
date_month	572.1905	56.687	10.094	0.000	461.023	683.358

In [16]:
```
evaluate_linear_model(linreg, X, y)
```

```
RMSE of residuals: 4888.42769570239
R-squared: 0.7390052731571286
```



There was no improvement to the fit and no indication that any of the weather parameters had an effect on the attendance. Weather parameters will not be included in the fit for subsequent models.

Model 6

For this model, I am going to remove parameters that were not significant. This means I will remove the following:

1. All weather parameters.
2. Kick off time parameters.

In [17]:
```
X = create_X(mlsall_df,
             columns=['home_team','away_team',
                      'day','date_month','date_year',
                      'playoff','home_opener','rivals','real_home_team'],
             add_constant=True)
linreg = sm.OLS(y,X).fit()
linreg.summary()
```

Out[17]:

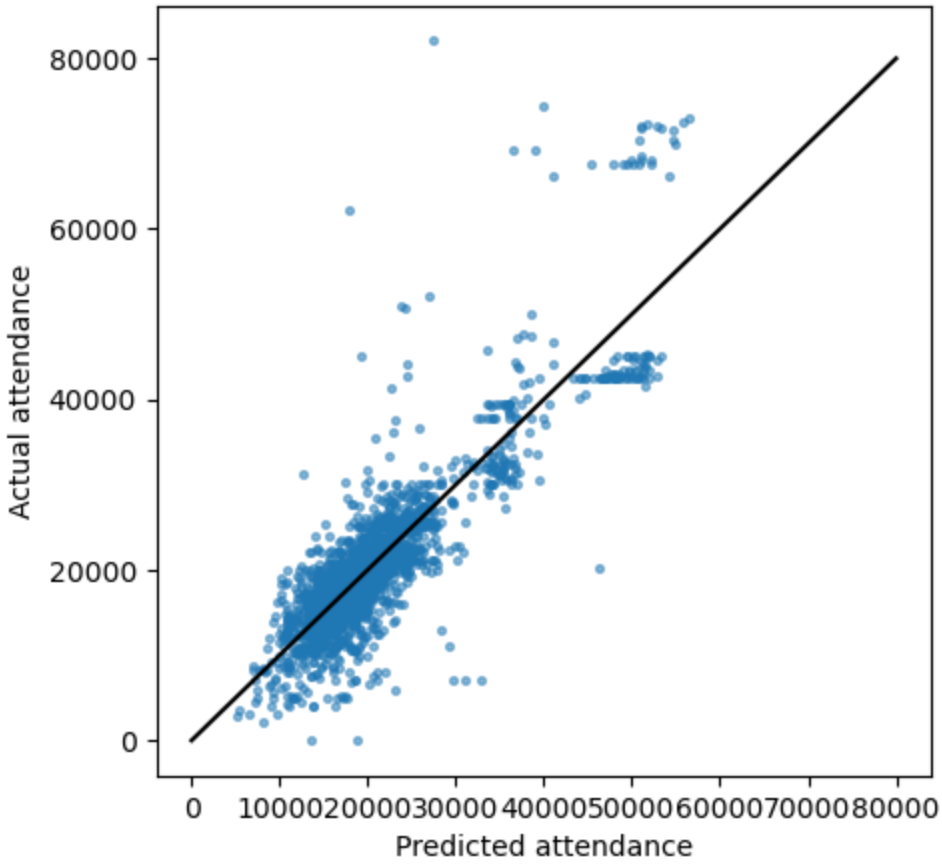
OLS Regression Results

Dep. Variable:	attendance	**R-squared:**	0.738
Model:	OLS	**Adj. R-squared:**	0.730
Method:	Least Squares	**F-statistic:**	85.76
Date:	Mon, 04 Dec 2023	**Prob (F-statistic):**	0.00
Time:	13:32:40	**Log-Likelihood:**	-22059.
No. Observations:	2229	**AIC:**	4.426e+04
Df Residuals:	2157	**BIC:**	4.467e+04
Df Model:	71		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
date_month	585.3170	54.818	10.678	0.000	477.816	692.818

In [18]:
```
evaluate_linear_model(linreg, X, y)
```

```
RMSE of residuals: 4884.843861590792
R-squared: 0.7384176459978459
```



This model has an R-squared of 0.738 and the RMSE is 4,885.

This is the best performing linear model. Below, I dig into the results a little more to see how well the model performed.

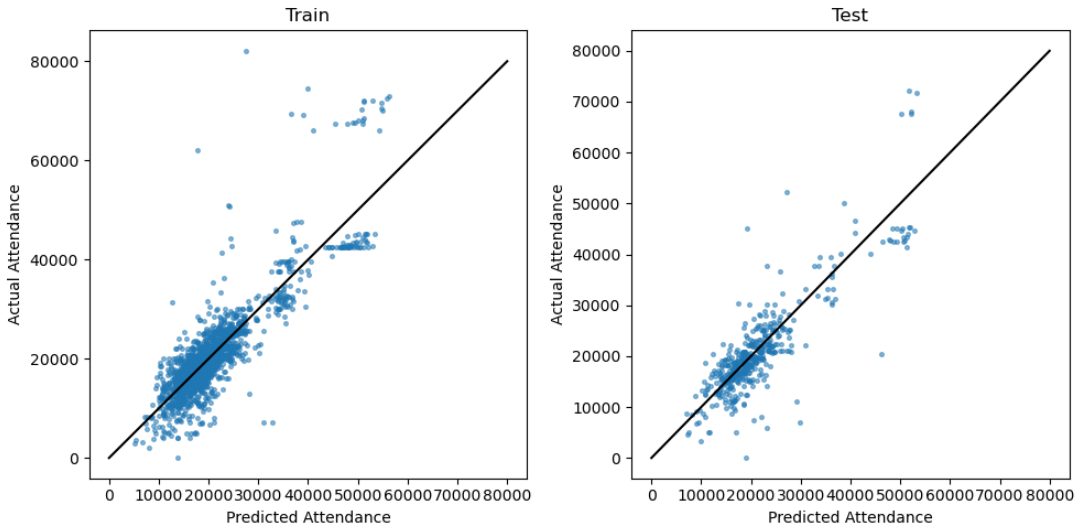
First, I am going to be using a train-test split for the other types of models I will try. In order to make a more direct comparison to those models, I would like to evaluate the linear regression model on those splits.

In [19]:
```
# Set random seed that will be used for ALL train-test splits
rando = 23
```

In [20]:
```
# Split the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=r
```

In [21]:
```
evaluate_model_split(linreg, X_train, y_train, X_test, y_test)
```

```
RMSE:
Train: 4752.0
Test: 5012.6

R-squared:
Train: 0.74
Test: 0.733
```



Despite fitting all of the data without the split, there does not appear to be an issue with overfitting. This is not surprising as a linear regression model is likely too simple to overfit the data.

For the other types of models, the benchmark performance for the test data is the following:

RMSE = 5,012.6

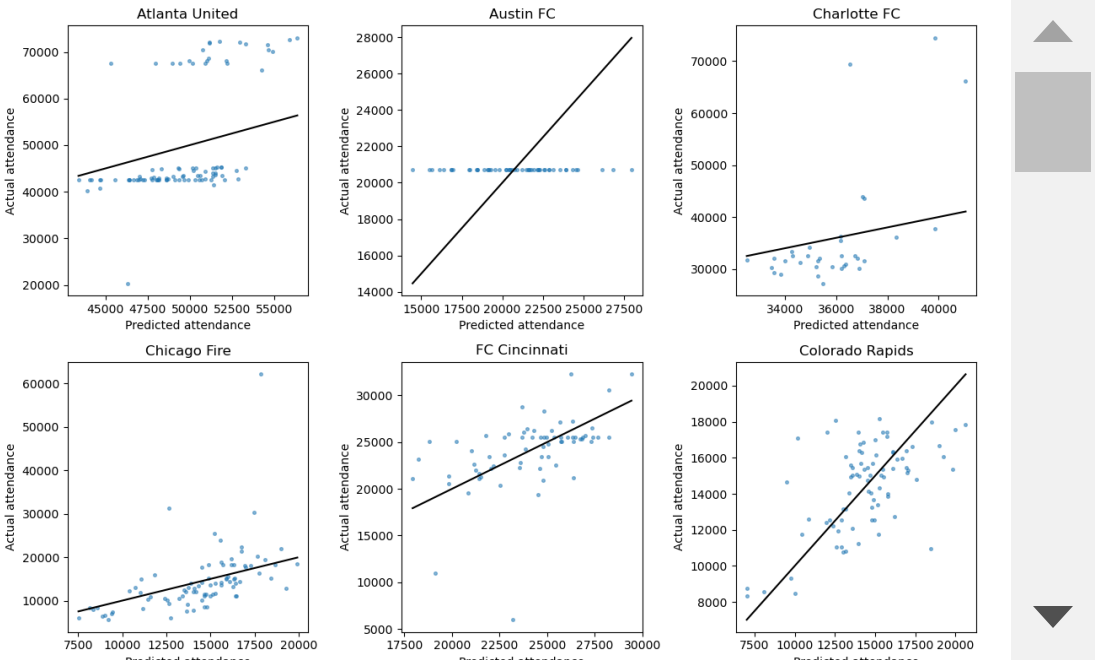
R-squared = 0.733

I want to look at how the fit performed on a team-by-team basis. The dictionary below turns the IDs in home_team and away_team into the actual team names.

In [22]:
```
team_names = {0: 'Atlanta United',
              1: 'Austin FC',
              2: 'Charlotte FC',
              3: 'Chicago Fire',
              4: 'FC Cincinnati',
              5: 'Colorado Rapids',
              6: 'Columbus Crew',
              7: 'FC Dallas',
              8: 'D.C. United',
              9: 'Houston Dynamo',
              10: 'Los Angeles Galaxy',
              11: 'Los Angeles FC',
              12: 'Minnesota United',
              13: 'Inter Miami',
              14: 'CF Montreal',
              15: 'Nashville SC',
              16: 'New England Revolution',
              17: 'New York City FC',
              18: 'New York Red Bulls',
              19: 'Orlando City',
              20: 'Philadelphia Union',
              21: 'Portland Timbers',
              22: 'Real Salt Lake',
              23: 'San Jose Earthquakes',
              24: 'Seattle Sounders',
              25: 'Sporting Kansas City',
              26: 'St. Louis FC',
              27: 'Toronto FC',
              28: 'Vancouver Whitecaps'}
```

In [23]:
```
# Get predictions from the model
y_pred_lr = linreg.predict(X)
```

In [24]:
```
# Plot actual attendance vs. predicted attendance for each team's home r
fig, ax = plt.subplots(ncols=3, nrows=10, figsize=(12,40))

for ht in range(29):
    r = ht//3
    c = ht%3

    y_pred_ht = y_pred_lr[mlsall_df['home_team']==ht]

    ax[r,c].scatter(y_pred_ht,
                    y[mlsall_df['home_team']==ht],
                    s=7,
                    alpha=0.5)

    ax[r,c].set_title(team_names[ht])
    ax[r,c].set_xlabel('Predicted attendance')
    ax[r,c].set_ylabel('Actual attendance')

    ax[r,c].plot([np.min(y_pred_ht),np.max(y_pred_ht)],
                 [np.min(y_pred_ht),np.max(y_pred_ht)],
                 color='black')

fig.tight_layout()
```



The predictions are quite solid for some teams, but not for others.

The model struggles the most with teams that change the capacity of their stadiums. Atlanta United is a good example. Usually, they use a capacity around 45,000, but then expand it to over 70,000 for other matches. The changing capacity means that the model struggles to fit the data which is grouped into two parts of the graph. In the case of Atlanta, I can tell pretty clearly what capacity was being used in a given match, but that is not the case for all teams.

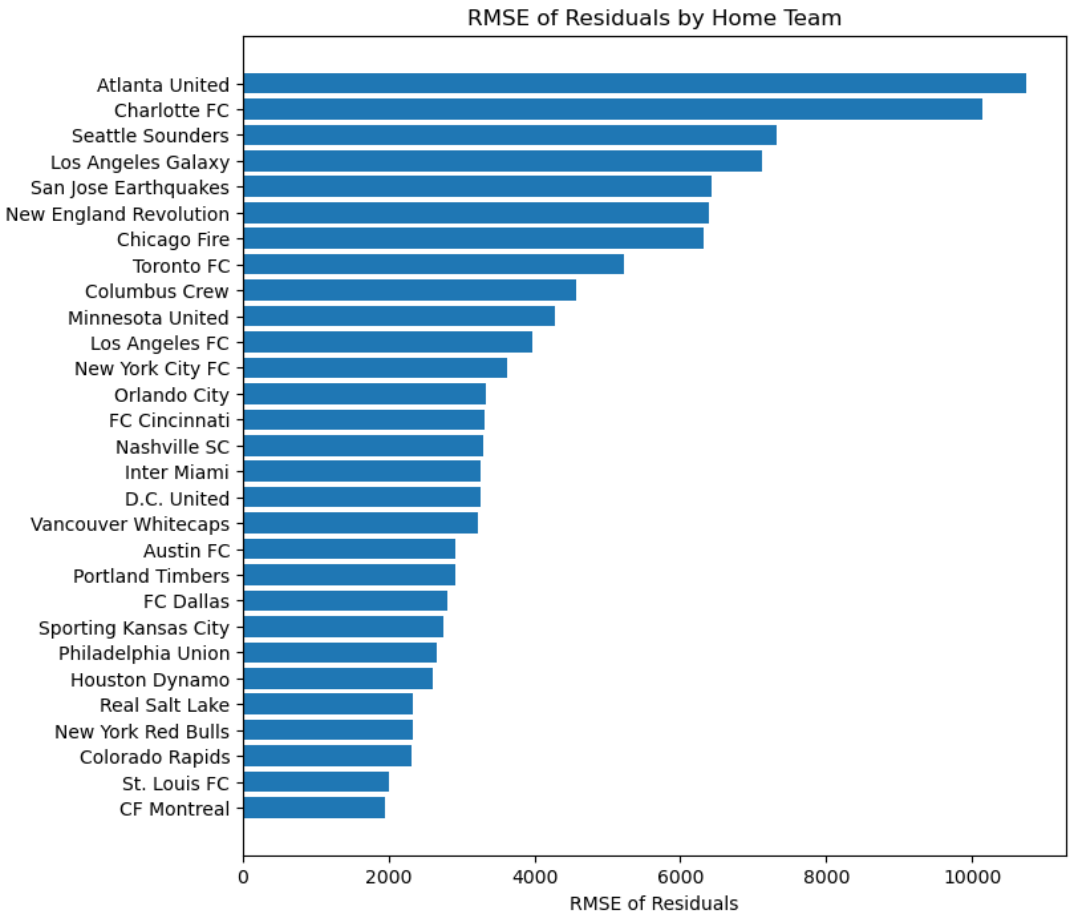
The artifically reduced capacity of Gillette Stadium, used by the New England Revolution, is listed as 20,000, but many of their matches have a higher attendance than that. In contrast to teams like Atlanta United, there seems to be a continuum of values above 20,000 rather than one unique, higher capacity. This implies that New England uses a less rigid capacity for its home matches, making it impossible to tell what capacity they used, if they used one at all.

The model also struggled with teams that had very consistent home attendance. The most obvious cases are Austin FC and St. Louis FC who have had the same (or nearly the same attendance) for every home match they have played. The model expects the attendance to vary from match to match depending on changes in the parameters, but there just isn't any variance for these teams. Similar issues are present for these teams that just have less variance in attendance: Los Angeles FC, Minnesota United, and Sporting Kansas City.

These teams outline a second problem with the model which is that there is a true maximum capacity for each stadium that cannot be exceeded, but the linear model does not

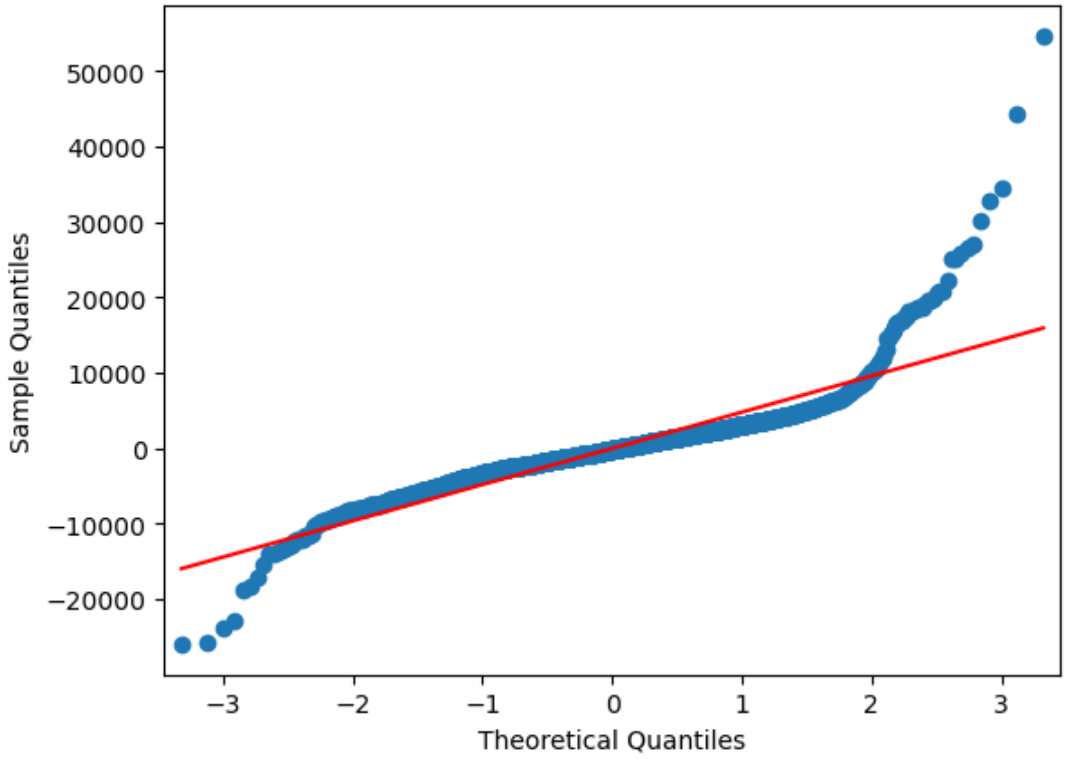
One metric for judging how well the model fits each team is the RMSE. Below, I plot the RMSE for each teams' home matches.

In [25]:
```
# RMSE of residuals for each home team
fig, ax = plt.subplots(figsize=(8,8))

rmse_team = []
for ht in range(29):
    rmse_team.append(np.sqrt(np.var(linreg.resid[mlsall_df['home_team']=

ax.barh(y=np.arange(29), width=np.sort(rmse_team))

ax.set_yticks(np.arange(29))
ax.set_yticklabels([team_names[x] for x in np.argsort(rmse_team)])
ax.set_xlabel('RMSE of Residuals')
ax.set_title('RMSE of Residuals by Home Team');
```



Unsurprisingly, the highest RMSE values are for the 3 teams that often change their stadium capacity (Atlanta, Charlotte, and Seattle). For 19 of the 29 teams, the RMSE is below 4,000.

In [26]:
```
# Make Quantile-Quantile plot to compare to normal distribution
sm.qqplot(linreg.resid,line='s');
```



The Q-Q plot above shows that the distribution of residuals is close to normal except at the edges.

XGBoost Regression

Next, I use an extreme gradient boosting regression model, XGBRegressor.

Model 1

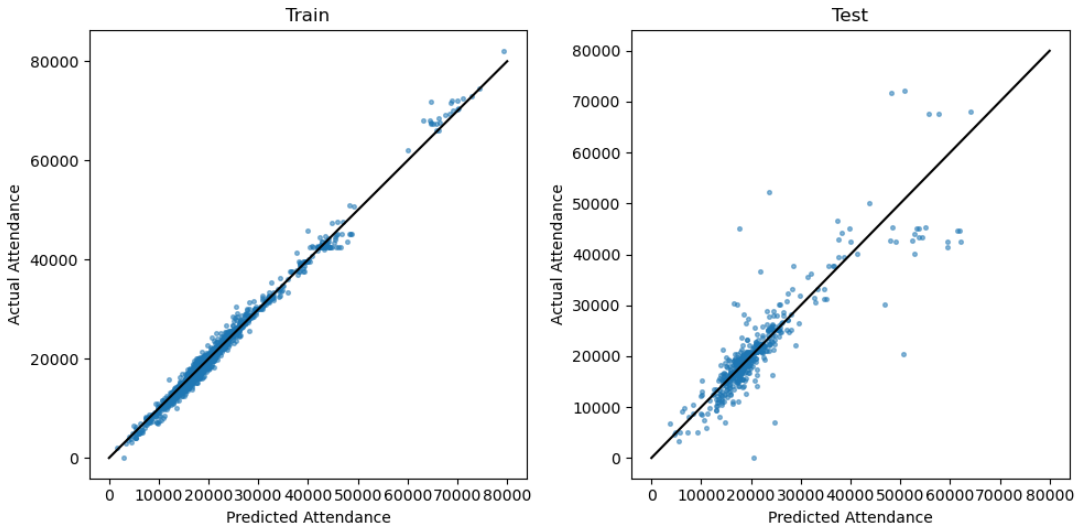
First, I want to see how well XGBRegressor performs with the default hyperparameters.

I am going to include all of the possible columns in my input data.

In [27]:
```
# Create input DataFrame, X
X = create_X(mlsall_df, columns=['home_team','away_team',
                                 'day','date_month','date_year','local_t
                                 'playoff','home_opener','rivals','real_
                                 'temperature','rain','snow','windspeed
             drop_first=False)
# Apply the same train-test split that was used earlier
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=r
```

In [28]:
```
# Instantiate XGBClassifier
xgb = XGBRegressor()

# Fit model
xgb.fit(X_train, y_train)

# Evaluate XGBRegressor model
evaluate_model_split(xgb, X_train, y_train, X_test, y_test)
```

```
RMSE:
Train: 971.3
Test: 4814.8

R-squared:
Train: 0.989
Test: 0.754
```



The model is overfitting quite a bit as the R-squared value is 0.989 for the train dataset, but just 0.754 for the test dataset. The RMSE is also higher for the test data. Some hyperparameters will need to be adjusted to improve the fit.

Both the RMSE and R-squared values are better for the test data than they were for the linear regression model. Below, I start adjusting hyperparameters to improve the model further.

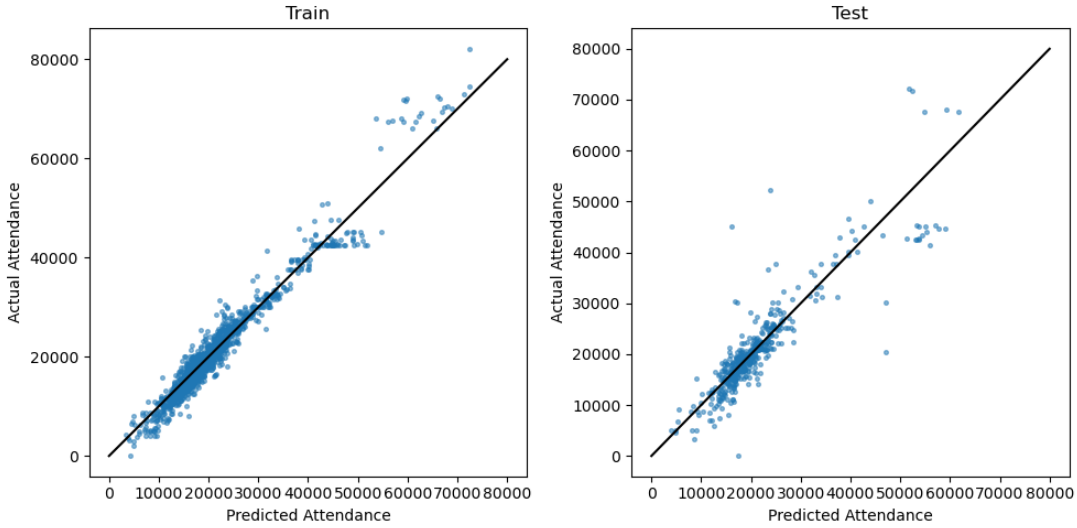
Model 2

Below, I use a custom function, xgb_gridsearch, which can be found in model_funcs.py. First, I just want to try out different tree depths and number of estimators.

In [29]:
```
# Create parameter grid
param_grid_xgb = {
    'max_depth': [3,4,5,6,7,8,9,None],
    'n_estimators': [50,100,200,250,300,350,400],
    'booster': ['gbtree']
}

# Search for optimal parameter combination
xgb_best, best_params, params, scores = model_gridsearch('xgb',param_gr
```

In [30]:
```
# Evaluate XGBRegressor model
print(best_params)
evaluate_model_split(xgb_best, X_train, y_train, X_test, y_test)
```

```
{'max_depth': 4, 'n_estimators': 100, 'booster': 'gbtree'}
RMSE:
Train: 2026.9
Test: 4467.4

R-squared:
Train: 0.953
Test: 0.788
```



The best model had a max_depth of 4 and 100 estimators.

The performance improved a bit from the last model. The RMSE and R-squared are now 4,467.4 and 0.788, respectively.

The model did overfit as the metrics for the training data was significantly better than the test data. Some regularization might come in handy.

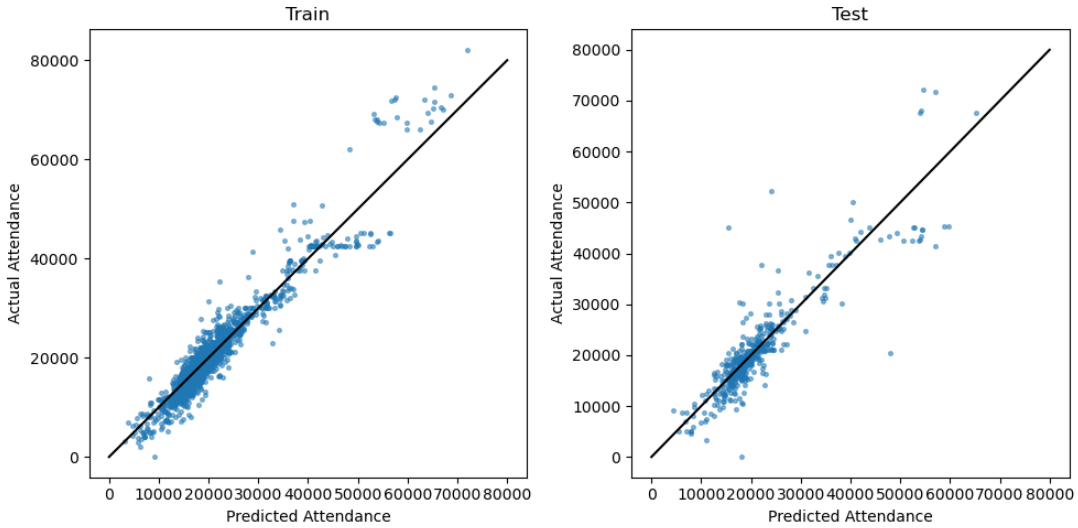
Model 3

Next, I introduce learning_rate, reg_lambda, and subsample to the parameter grid. The hope is that reg_lambda and subsample will help with the overfitting.

In [31]:
```
# Create parameter grid
param_grid_xgb = {
    'max_depth': [3,4,5,6],
    'n_estimators': [25,50,75,100,125,150,200,250,300],
    'booster': ['gbtree'],
    'learning_rate':[0.3,0.4,0.5,0.6],
    'reg_alpha':[0.01,0.1,1,10],
    'subsample':[0.7,0.8,0.9,1]
}

# Search for optimal parameter combination
xgb_best, best_params, params, scores = model_gridsearch('xgb',param_gr
```

In [32]:
```
# Evaluate XGBRegressor model
print(best_params)
evaluate_model_split(xgb_best, X_train, y_train, X_test, y_test)
```

```
{'max_depth': 4, 'n_estimators': 25, 'booster': 'gbtree', 'learning_ra
te': 0.6, 'reg_alpha': 10, 'subsample': 0.9}
RMSE:
Train: 2685.7
Test: 4408.7

R-squared:
Train: 0.917
Test: 0.793
```



The fit once again improved a bit. The new RMSE and R-squared values are 4,408.7 and 0.793.

The best parameter values were:

1. max_depth = 4
2. n_estimators = 25
3. learning_rate = 0.6
4. reg_alpha = 10
5. subsample = 0.9

The overfitting is still present.

Number of estimators

Below, I want to check to see how the performance changes as the number of estimators increases. All other hyperparameters will take on the values they had in the best model so far.

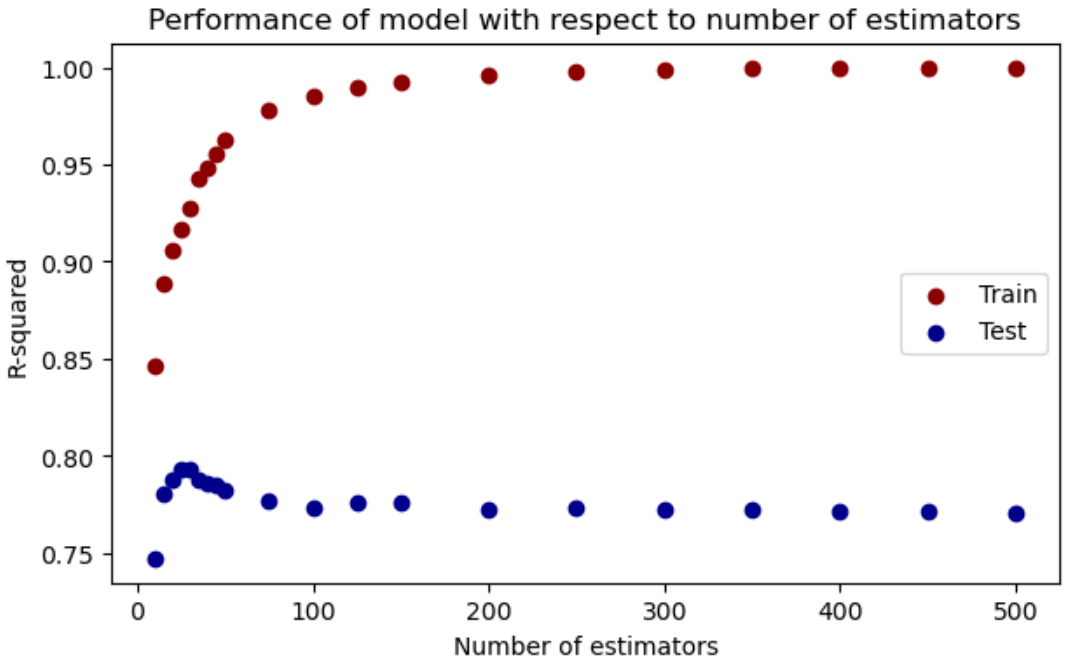
In [33]:
```
# Create parameter grid
param_grid_xgb = {
    'max_depth': [4],
    'n_estimators': [10,15,20,25,30,35,40,45,50,75,100,125,150,200,250,
    'booster': ['gbtree'],
    'learning_rate':[0.6],
    'reg_alpha':[10],
    'subsample':[0.9]
}

# Search for optimal parameter combination
xgb_best, best_params, params, scores = model_gridsearch('xgb',param_gr
```

In [34]:
```
best_params
```

Out[34]:
```
{'max_depth': 4,
 'n_estimators': 25,
 'booster': 'gbtree',
 'learning_rate': 0.6,
 'reg_alpha': 10,
 'subsample': 0.9}
```

In [35]:
```
# plot R-squared values vs. number of estimators
fig, ax = plt.subplots(figsize=(7,4))

ax.scatter(param_grid_xgb['n_estimators'],scores[:,0],label='Train', co
ax.scatter(param_grid_xgb['n_estimators'],scores[:,1],label='Test',colo
ax.set_xlabel('Number of estimators')
ax.set_ylabel('R-squared')
ax.set_title('Performance of model with respect to number of estimators

ax.legend()
```

Out[35]: <matplotlib.legend.Legend at 0x1df83fb2740>



The R-squared value for the test data maxes out at 0.793 when the number of estimators is 25. After that, the R-squared value begins to drop a bit.

Summary for XGBRegressor: The XGB model was able to outperform the linear regression model. The best R-squared value was just below 0.8 and the best RMSE was 4,408.7. Next, I will try random forest regression to see if it can outperform these metrics.

Random Forest Regression

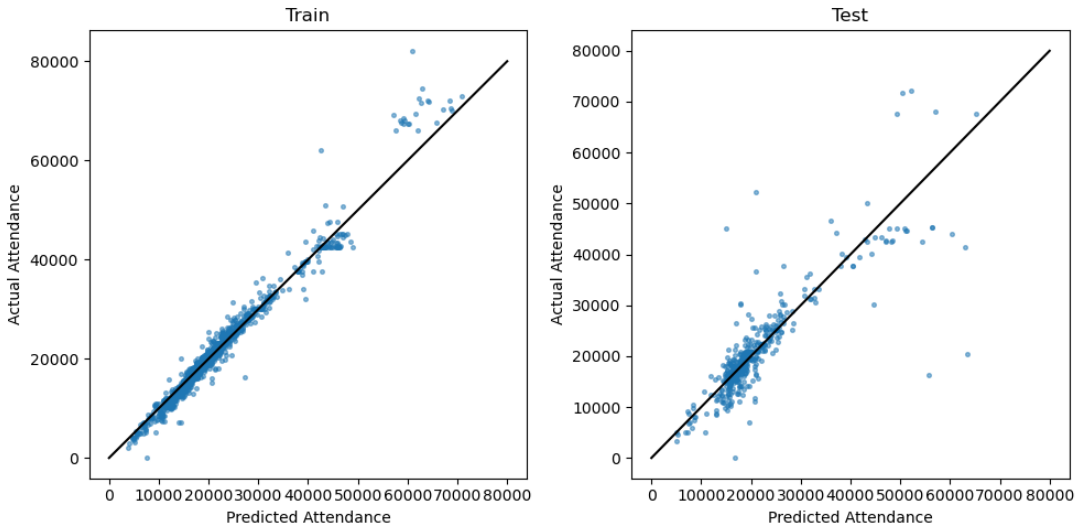
The random forest regressor will use the same training and test data as the XGB regressor.

Model 1

First, I will try the default hyperparameters for RandomForestRegressor.

In [36]:
```
# Create instance of RandomForestRegressor
rfr = RandomForestRegressor()

# Fit the model
rfr.fit(X_train, y_train)

# Evaluate the fit
evaluate_model_split(rfr, X_train, y_train, X_test, y_test)
```

```
RMSE:
Train: 1619.1
Test: 5167.5

R-squared:
Train: 0.97
Test: 0.716
```



The R-squared for this model is 0.725 and the RMSE is 5,084.0. Like the default XGBRegressor model, this is significantly overfitting.

Model 2

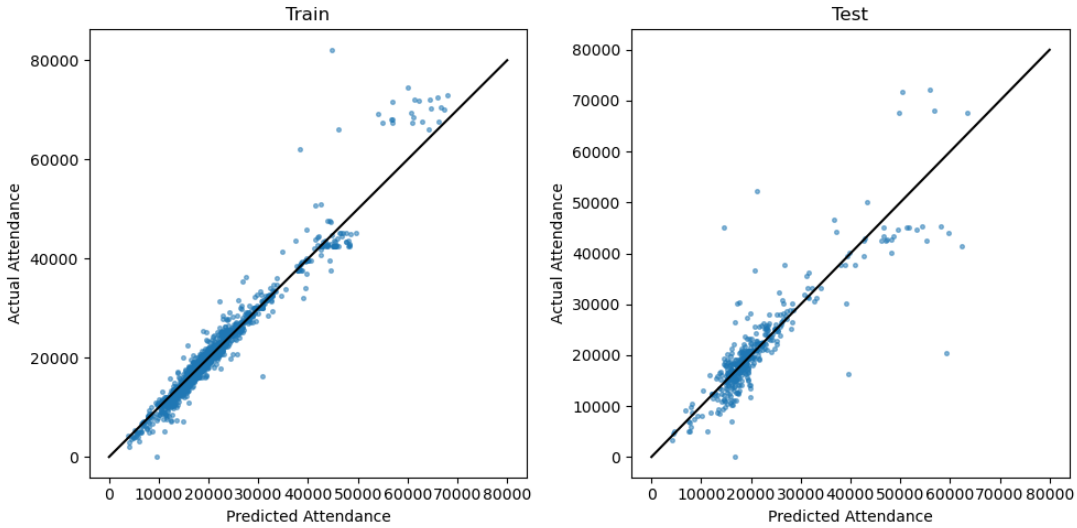
Next, I do a grid search on the following parameters:

1. Number of estimators
2. Max depth
3. Minimum samples to split a node

In [37]:
```
# Create parameter grid
param_grid_rfr = {
    'n_estimators': [5,10,15,20,25,30,40,50],
    'max_depth': [20,30,40,50,60,70,80],
    'min_samples_split': [2,3,4,5,6,7]
}

# Search for optimal parameter combination
rfr_best, best_params, params, scores = model_gridsearch('rfr', param_gr
```

In [38]:
```
print(best_params)
evaluate_model_split(rfr_best, X_train, y_train, X_test, y_test)
```

```
{'n_estimators': 30, 'max_depth': 60, 'min_samples_split': 5}
RMSE:
Train: 2117.1
Test: 4780.2

R-squared:
Train: 0.948
Test: 0.757
```



The fit on the test data did improve a bit. The RMSE is 4,650.8 and the R-squared is 0.77. The model is still overfitting quite a bit.

The performance of this model is worse than the best XGB model. This is not surprising since XGB also uses decision trees, but uses a more sophisticated algorithm to improve the fit each iteration.

K Nearest Neighbors

The last type of model I will try is a K-Nearest Neighbors model. Specifically, I will use KNeighborsRegressor.

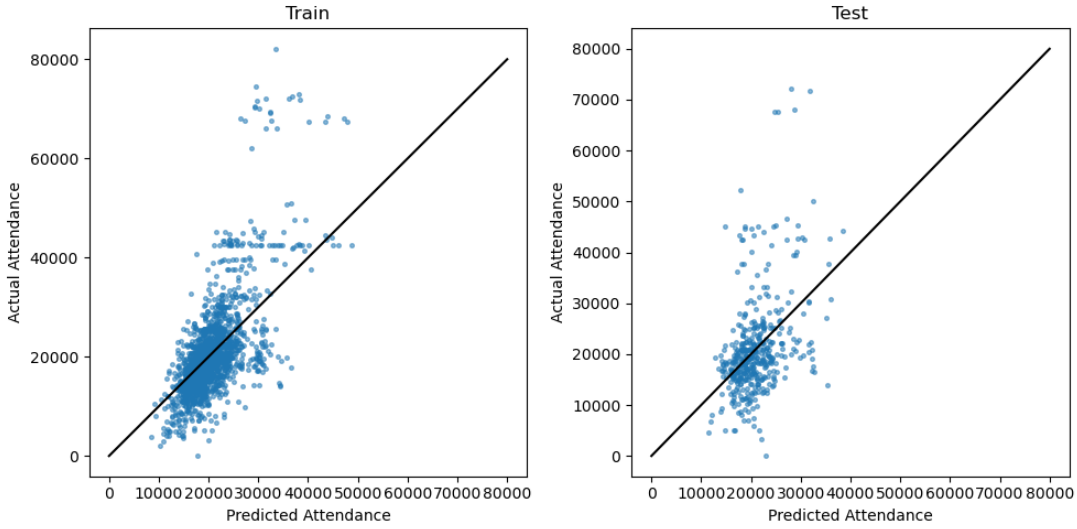
In theory, I think this type of model could be quite effective if there was enough data. However, I expect that this model will struggle to match the performance of the other models because certain features have less data coverage.

Model 1

I start with the default hyperparameters for KNeighborsRegressor.

In [39]:
```
# Create instance of KNeighborsRegressor
knn = KNeighborsRegressor()

# Fit the model
knn.fit(X_train, y_train)

# Evaluate the fit
evaluate_model_split(knn, X_train, y_train, X_test, y_test)
```

```
RMSE:
Train: 7040.4
Test: 8853.3

R-squared:
Train: 0.429
Test: 0.167
```



The model performed very poorly on both the training and test data. The hyperparameters need some tuning to improve the model.

Both the XGBRegressor and RandomForestRegressor performed quite well without tuning anything. This might confirm what I said above about this model struggling.

Model 2

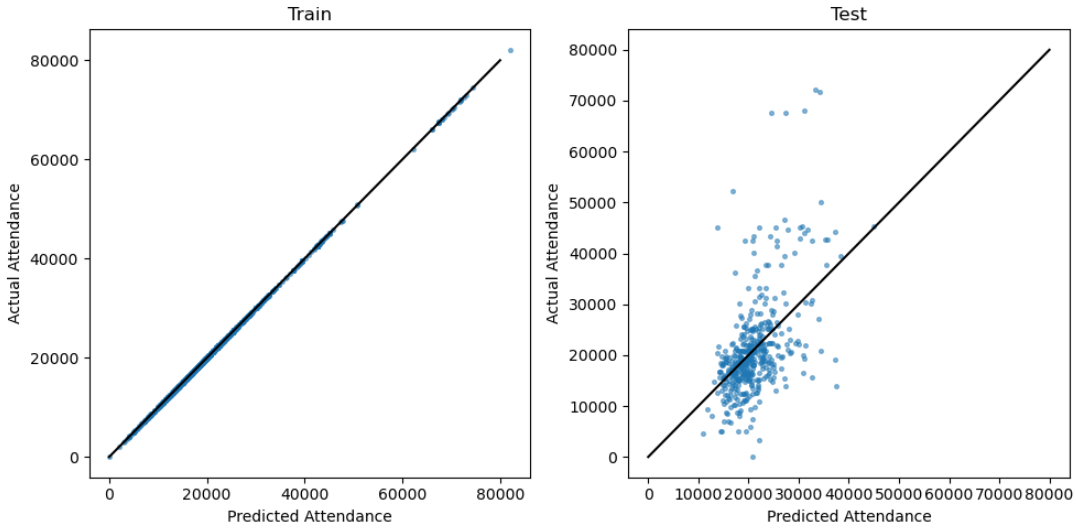
Next, I adjust values for the following parameters:

1. Number of neighbors
2. Minkowski metric exponent, p.
3. Whether to weight with distance or not.

In [40]:
```
# Create parameter grid
param_grid_knn = {
    'n_neighbors': [5,10,15,20,25,30,40,50],
    'p': [1,1.5,2,2.5,3],
    'weights': ['uniform','distance']
}

# Search for optimal parameter combination
knn_best, best_params, params, scores = model_gridsearch('knn', param_gr
```

In [41]:
```
# Evaluate the fit
print(best_params)
evaluate_model_split(knn_best, X_train, y_train, X_test, y_test)
```

```
{'n_neighbors': 5, 'p': 1, 'weights': 'distance'}
RMSE:
Train: 0.0
Test: 8131.9

R-squared:
Train: 1.0
Test: 0.297
```



The model suffers from severe overfitting and the performance on the test data is still very bad (RMSE=8,131.9 and R-squared=0.297). Since I already have other types of models that perform well, I am going to stop here for K-Nearest Neighbors.

Summary of Models

The best performing instance of each model is summarized in the table below.

Model	RMSE (Test)	R^2 (Test)	RMSE (Training)	R^2 (Training)
XGBoost	4,408.7	0.793	2,685.7	0.917
Random Forest	4,650.8	0.770	2,289.8	0.940
Linear Regression	5,012.6	0.733	4,752.0	0.740
K-Nearest Neighbors	8,131.9	0.207	0.0	0.000

The RMSE and R^2 values are for the test dataset which comprises 20% of the data. For each model except linear regression, the model was trained on the other 80% of the data. The train-test split was not used for linear regression because there was no concern that the model would overfit.

Overall performance

The first three types of models (linear regression, XGBoost, and Random Forest) each achieved RMSE values around or below 5,000 and R-squared values between 0.73 and 0.8. However, the K-Nearest Neighbors model struggled to match this level of performance. There might have been some room for improvement by tweaking the scaling of the input variables, but I doubt the predictions would have been as good as the other three.

As stated at the top of this notebook, the match-by-match predictions are not stellar. The best RMSE was around 4,400. This means that we expect a typical prediction to be off by about this much on average. This is not a problem, though. The goal of this project was to look for trends that MLS could use to improve attendance over the course of a season, not to target specific matches to try to improve attendance. There are trends that can be useful which will be discussed in the **Exploring the Model** section of this notebook.

The Best Performing Model

The model that performed the best based on having the lowest RMSE and highest R-squared was the XGBoost Regression model. The model did suffer from some overfitting,

Exploring the Model

In this section, I primarily explore the best-fitting XGBoost model, but I am also going to mention the linear regression model sometimes for comparison.

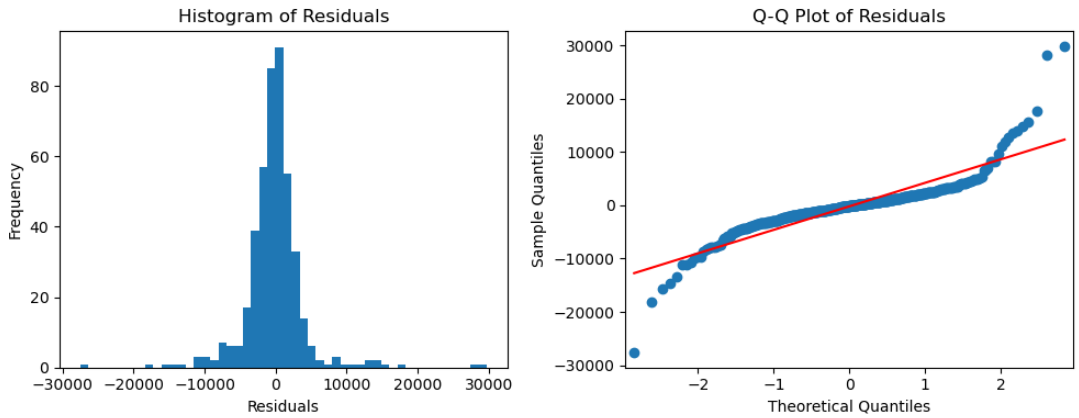
Residuals

First, I want to look into the residuals. I already know that the RMSE is around 4,400 for the best model, but I want to look at how it is distributed.

In [42]:
```
# I will use the blue and red from the MLS logo in some figures
mls_blue = '#001F5B'
mls_red = '#DF231A'
```

In [43]:
```
# Calculate residuals of test data
xgb_resid = y_test-xgb_best.predict(X_test)

# Kurtosis
print(f"Kurtosis of Residuals: {sp.kurtosis(xgb_resid)}")
print(f"Proportion of predictions within 1000 of actual value: {np.sum(
print(f"Proportion of predictions within 2000 of actual value: {np.sum(
print(f"Proportion of predictions within 3000 of actual value: {np.sum(
print(f"Proportion of predictions within 4000 of actual value: {np.sum(

# Plot residuals and a Q-Q plot
fig, ax = plt.subplots(ncols=2, figsize=(10,4))

# Residual distribution
ax[0].hist(xgb_resid, bins=50)
ax[0].set_xlabel('Residuals')
ax[0].set_ylabel('Frequency')
ax[0].set_title('Histogram of Residuals')

# Q-Q plot
sm.qqplot(xgb_resid,line='s',ax=ax[1])
ax[1].set_title('Q-Q Plot of Residuals')

fig.tight_layout();
```

```
Kurtosis of Residuals: 13.481246785624222
Proportion of predictions within 1000 of actual value: 0.3452914798206
278
Proportion of predictions within 2000 of actual value: 0.6008968609865
47
Proportion of predictions within 3000 of actual value: 0.7399103139013
453
Proportion of predictions within 4000 of actual value: 0.8363228699551
569
```



The distribution of the residuals appears pretty symmetric, but more centrally concentrated than a normal distribution. This is verified by the kurtosis value of 13.5. A normal distribution would have a kurtosis of 0, while positive values imply more central concentration.

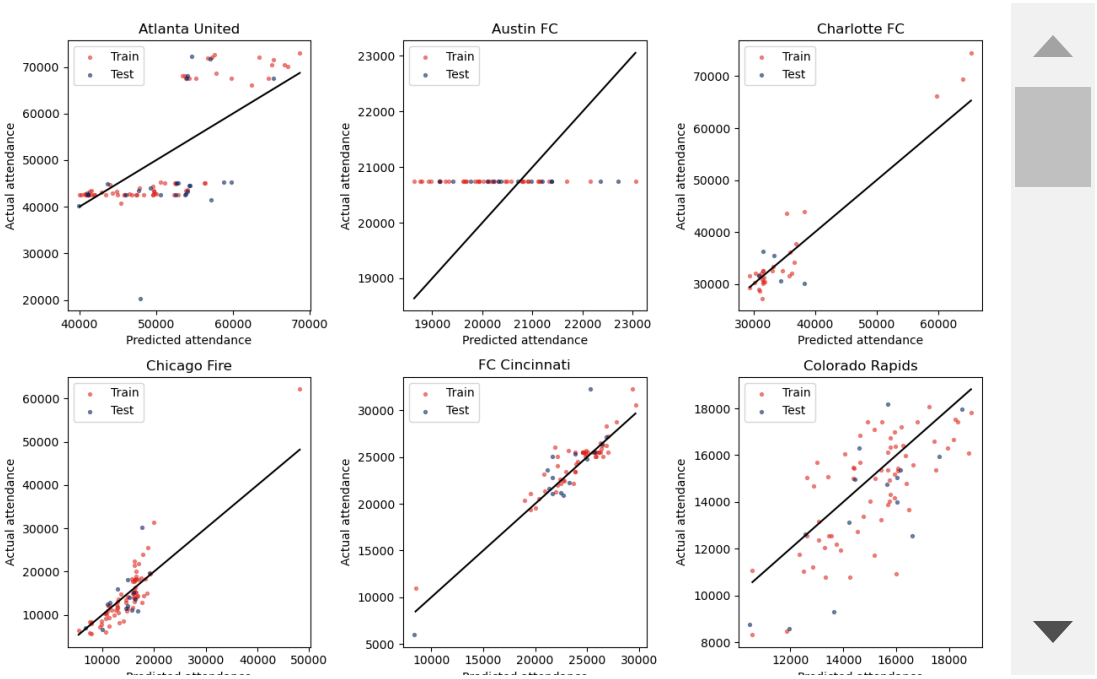
The Q-Q plot on the right shows how there are some large outliers on the edges. This is seen in the histogram as there are some residuals that are above 10,000, even as high as almost 30,000. However, the vast majority of the residuals are smaller than that. Even

Performance by Team

Below, I plot the actual attendance vs. the predicted attendance for each team's home matches. The goal is to see whether the model performed consistently well for each team, or if it was better for some teams compared to others.

In [44]:

```
# Get predictions from the best XGB model
y_pred_train_xgb = xgb_best.predict(X_train)
y_pred_test_xgb = xgb_best.predict(X_test)

# Plot actual attendance vs. predicted attendance for each team's home m
fig, ax = plt.subplots(ncols=3, nrows=10, figsize=(12,40))

for ht in range(29):
    r = ht//3
    c = ht%3

    train_select = X_train[f"home_team_{ht}"]==1
    test_select = X_test[f"home_team_{ht}"]==1

    # Training data
    ax[r,c].scatter(y_pred_train_xgb[train_select],
                    y_train[train_select],
                    s=8,
                    alpha=0.5,
                    label='Train',
                    color=mls_red)

    # Test data
    ax[r,c].scatter(y_pred_test_xgb[test_select],
                    y_test[test_select],
                    s=8,
                    alpha=0.5,
                    label='Test',
                    color=mls_blue)

    ax[r,c].set_title(team_names[ht])
    ax[r,c].set_xlabel('Predicted attendance')
    ax[r,c].set_ylabel('Actual attendance')

    # Plot x=y line
    ax[r,c].plot([np.min(y_pred_train_xgb[train_select]),np.max(y_pred_t
                 [np.min(y_pred_train_xgb[train_select]),np.max(y_pred_t
                 color='black')

    ax[r,c].legend()

fig.tight_layout();
```

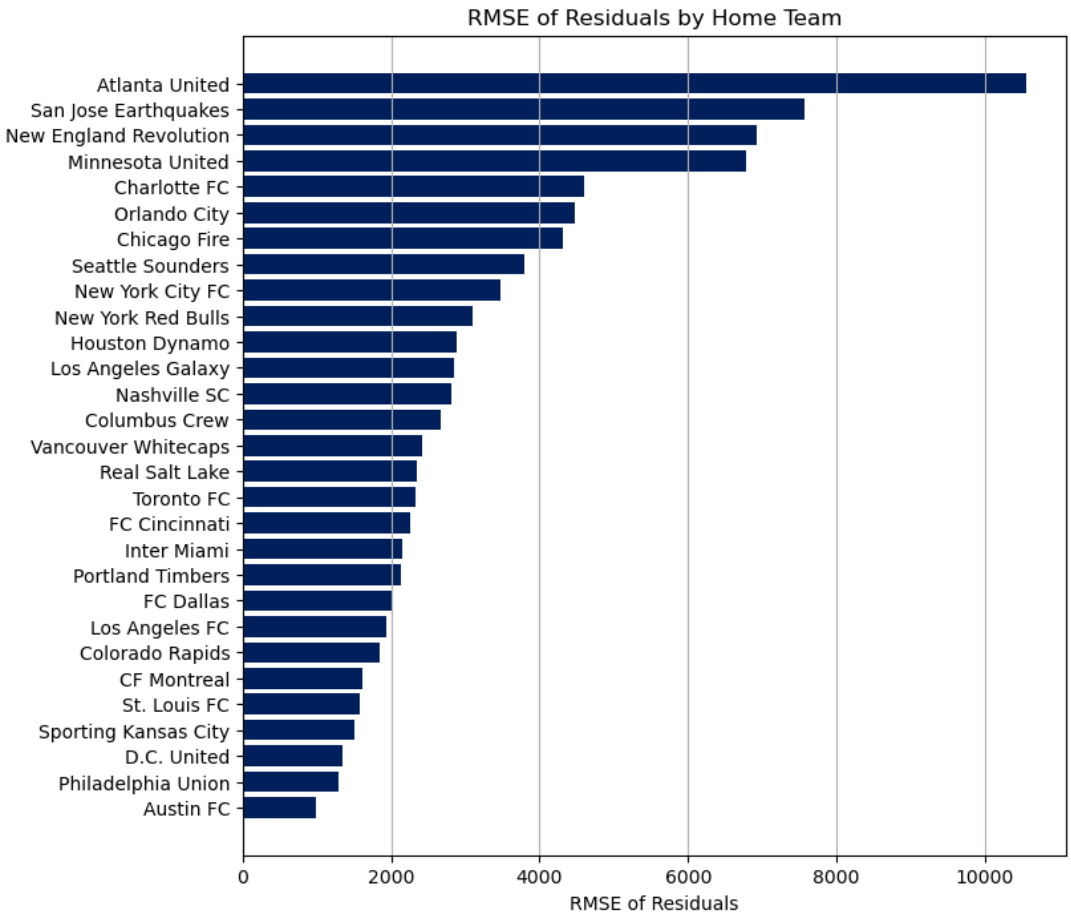


The first thing I notice is that the model struggled with the teams who change the capacity of their stadiums, like Atlanta United (first plot in top left). Even on the training data, it tends to overestimate the attendance when the lower capacity is used and underestimate it when the larger capacity is used.

It can be a little hard to tell how good the different fits are because not all of the plots are on the same scale.

Below, I look at the RMSE of the residuals for each team's home matches. Only the matches in the test dataset are used.

In [45]:
```
# RMSE of residuals for each home team
fig, ax = plt.subplots(figsize=(8,8))

rmse_team = []
for ht in range(29):
    rmse_team.append(np.sqrt(np.var(xgb_resid[mlsall_df['home_team']==ht

ax.barh(y=np.arange(29), width=np.sort(rmse_team), color=mls_blue)
ax.grid(axis='x')

ax.set_yticks(np.arange(29))
ax.set_yticklabels([team_names[x] for x in np.argsort(rmse_team)])
ax.set_xlabel('RMSE of Residuals')
ax.set_title('RMSE of Residuals by Home Team');
```



Not surprisingly, the highest RMSE was for Atlanta United whose trouble fitting I discussed earlier.

The second highest RMSE belongs to San Jose, but I think that is primarily caused by a single match that had a much higher attendance than predicted because the match was staged in a special location with much higher capacity.

The third highest RMSE belongs to New England. This also changes its capacity from match to match. The only other team with an RMSE above 5,000 was Minnesota United that also had a match with a higher than normal capacity.

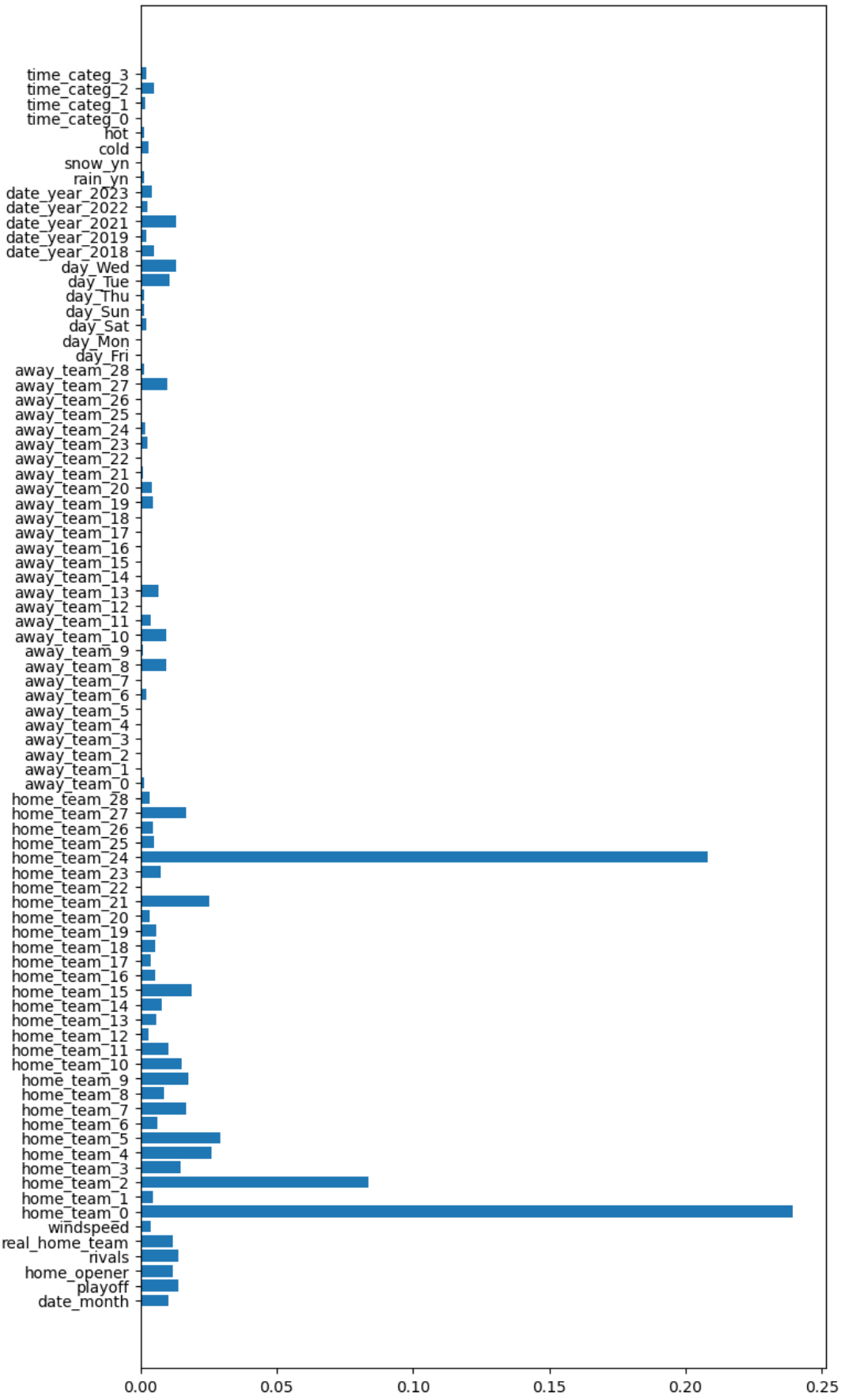
Other than the 4 teams with the highest RMSE values, all the others have RMSE values below 5,000 and most are at 3,000 or below.

Feature Importance

The main goal of this project was to figure out what factors had the greatest effect on attendance. I start by looking at the feature importance calculated by XGBoost below.

In [46]:

```
# Feature importance bar chart
fig, ax = plt.subplots(figsize=(8,16))

ax.barh(y=range(len(xgb_best.feature_importances_)), width=xgb_best.fea

ax.set_yticks(range(len(xgb_best.feature_importances_)))
ax.set_yticklabels(xgb_best.feature_names_in_);
```


time_categ_3
time_categ_2
time_categ_1
time_categ_0
hot
cold
snow_yn
rain_yn
date_year_2023
date_year_2022
date_year_2021
date_year_2019
date_year_2018
day_Wed
day_Tue
day_Thu
day_Sun
day_Sat
day_Mon
day_Fri
away_team_28
away_team_27
away_team_26
away_team_25
away_team_24
away_team_23
away_team_22
away_team_21
away_team_20
away_team_19
away_team_18
away_team_17
away_team_16
away_team_15
away_team_14
away_team_13
away_team_12
away_team_11
away_team_10
away_team_9
away_team_8
away_team_7
away_team_6
away_team_5
away_team_4
away_team_3
away_team_2
away_team_1
away_team_0
home_team_28
home_team_27
home_team_26
home_team_25
home_team_24
home_team_23
home_team_22
home_team_21
home_team_20
home_team_19
home_team_18
home_team_17
home_team_16
home_team_15
home_team_14
home_team_13
home_team_12
home_team_11
home_team_10
home_team_9
home_team_8
home_team_7
home_team_6
home_team_5
home_team_4
home_team_3
home_team_2
home_team_1
home_team_0
windspeed
real_home_team
rivals
home_opener
playoff
date_month
0.00
0.05
0.10
0.15
0.20
0.25

The most important features are those related to which team is at home. Part of the reason for this is simply that some teams have larger stadiums. The largest value in the plot above belongs to home_team_0, which is Atlanta United who play their matches in a big NFL stadium.

The trouble with the built-in feature_importances that are provided by the model is that they are normalized to add to 1 and it is hard to tell what actual effect each feature has on the attendance, especially because it does not indicate whether it has a positive or negative effect. I wrote my own function, get_feature_importance, that will attempt to measure the effect of each feature in more practical terms. The way I do this is by using the model on fake data, changing the input to see how the average attendance changes.

The fake data is generated below using the gen_fake_data function. The data contains each possible matchup between the 29 teams in MLS so that everyone plays everyone else once at home and once away (812 matches). The matches all assume the same default values for the other inputs at first. Those defaults are:

1. Match held on a Saturday in July, 2018 at 7:30 pm.
2. Not a playoff match, home opener, or rivalry match.
3. No rain or snow and the temperature is not considered hot or cold.
4. The real home team is present.

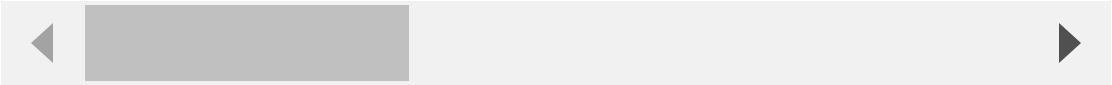
For each feature, I change the feature to compare it to the predicted attendance given the default values. For example, if I want to know the effect of playing matches on Wednesday instead of Saturday, I change all the matches to Wednesday and get the average of the predictions so it can be compared to the default average.

In [47]:
```
# Generate fake data
X_fake = gen_fake_data(X_train)
X_fake
```

Out[47]:

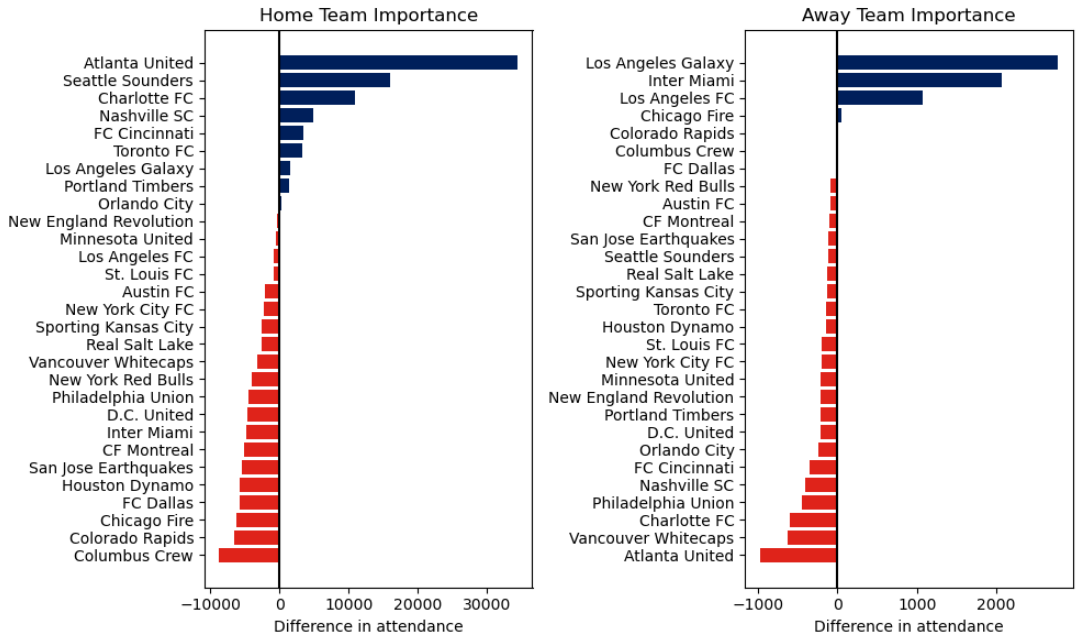
	date_month	playoff	home_opener	rivals	real_home_team	windspeed	home_team
1	0	0	0	0	1	11	
2	0	0	0	0	1	11	
3	0	0	0	0	1	11	
4	0	0	0	0	1	11	
5	0	0	0	0	1	11	
...	...	...	...	...	...	...	
835	0	0	0	0	1	11	
836	0	0	0	0	1	11	
837	0	0	0	0	1	11	
838	0	0	0	0	1	11	
839	0	0	0	0	1	11	

812 rows × 84 columns



In [48]:
```
# Get feature importance for the best XGB model
xgb_importance, xgb_import_ht = get_feature_importance(xgb_best, X_fake
```

In [49]:
```
# Home team and away team feature importance
fig, ax = plt.subplots(ncols=2, figsize=(10,6))

ht_importance = [xgb_importance[key] for key in xgb_importance.keys() i
ax[0].barh(y=range(29),
           width=np.sort(ht_importance),
           color=[mls_blue if x>=0 else mls_red for x in np.sort(ht_impo
ax[0].set_title('Home Team Importance')
ax[0].set_yticks(range(29))
ax[0].set_yticklabels([team_names[x] for x in np.argsort(ht_importance)]
ax[0].set_xlabel('Difference in attendance')
ax[0].axvline(0,color='black')

at_importance = [xgb_importance[key] for key in xgb_importance.keys() i
ax[1].barh(y=range(29),
           width=np.sort(at_importance),
           color=[mls_blue if x>=0 else mls_red for x in np.sort(at_impo
ax[1].set_title('Away Team Importance')
ax[1].set_yticks(range(29))
ax[1].set_yticklabels([team_names[x] for x in np.argsort(at_importance)]
ax[1].set_xlabel('Difference in attendance')
ax[1].axvline(0,color='black')

fig.tight_layout();
```



The bars in the graphs above show how much more or less attendance each team gets in its home matches (left) and away matches (right) compared to the average match.

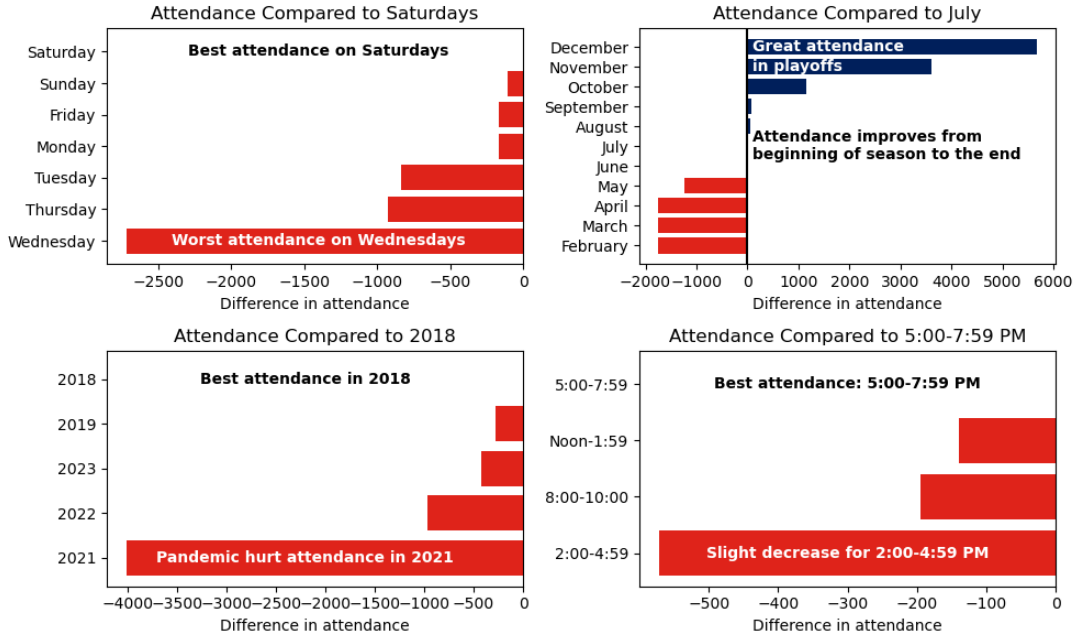
The graph on the left mostly accounts for differences in size between each teams' stadiums, it also incorporates information about how well those teams fill their stadiums. For example, the Columbus Crew get about 9,000 fewer people per match than the average. This is mostly because their stadium only holds 20,011 people, but also because the Columbus Crew don't always sell out their matches.

The three teams that tend to get the highest attendance when they go on the road are the two Los Angeles teams and Inter Miami.

In [50]:

```
#Day, month, year, and kick off time feature importance
fig, ax = plt.subplots(ncols=2, nrows=2, figsize=(10,6))

# Day of the week
day_names = ['Sunday','Monday','Tuesday','Wednesday','Thursday','Friday
day_importance = [xgb_importance[key] for key in xgb_importance.keys() :

ax[0,0].barh(y=range(7),
             width=np.sort(day_importance),
             color=[mls_blue if x>=0 else mls_red for x in np.sort(day_i
ax[0,0].set_yticks(range(7))
ax[0,0].set_yticklabels(day_names[x] for x in np.argsort(day_importance
ax[0,0].set_title('Attendance Compared to Saturdays')
ax[0,0].set_xlabel('Difference in attendance')
ax[0,0].text(-1400,6,'Best attendance on Saturdays',fontsize=10,fontweig
             horizontalalignment='center',verticalalignment='center')
ax[0,0].text(-1400,0,'Worst attendance on Wednesdays',fontsize=10,fontwe
             horizontalalignment='center',verticalalignment='center',co

# Month
month_names = ['February','March','April','May','June','July','August',
month_importance = [xgb_importance[key] for key in xgb_importance.keys(

ax[0,1].barh(y=range(11),
             width=np.sort(month_importance),
             color=[mls_blue if x>=0 else mls_red for x in np.sort(montl
ax[0,1].set_yticks(range(11))
ax[0,1].set_yticklabels(month_names[x] for x in np.argsort(month_importa
ax[0,1].axvline(0,color='black')
ax[0,1].set_title('Attendance Compared to July')
ax[0,1].set_xlabel('Difference in attendance')
ax[0,1].text(100,5,'Attendance improves from \nbeginning of season to th
             horizontalalignment='left',verticalalignment='center')
ax[0,1].text(100,10,'Great attendance',fontsize=10,fontweight='bold',
             horizontalalignment='left',verticalalignment='center',colo
ax[0,1].text(100,9,'in playoffs',fontsize=10,fontweight='bold',
             horizontalalignment='left',verticalalignment='center',colo

# Year
year_names = ['2018','2019','2021','2022','2023']
year_importance = [xgb_importance[key] for key in xgb_importance.keys()

ax[1,0].barh(y=range(5),
             width=np.sort(year_importance),
             color=[mls_blue if x>=0 else mls_red for x in np.sort(year_
ax[1,0].set_yticks(range(5))
ax[1,0].set_yticklabels(year_names[x] for x in np.argsort(year_importanc
ax[1,0].set_title('Attendance Compared to 2018')
ax[1,0].set_xlabel('Difference in attendance')
ax[1,0].text(-2200,4,'Best attendance in 2018',fontsize=10,fontweight='b
             horizontalalignment='center',verticalalignment='center')
ax[1,0].text(-2200,0,'Pandemic hurt attendance in 2021',fontsize=10,font
             horizontalalignment='center',verticalalignment='center',co

# Kick off time
kotime_names = ['Noon-1:59','2:00-4:59','5:00-7:59','8:00-10:00']
kotime_importance = [xgb_importance[key] for key in xgb_importance.keys
```

```
ax[1,1].barh(y=range(4),
             width=np.sort(kotime_importance),
             color=[mls_blue if x>=0 else mls_red for x in np.sort(kotir
ax[1,1].set_yticks(range(4))
ax[1,1].set_yticklabels(kotime_names[x] for x in np.argsort(kotime_impor
ax[1,1].set_title('Attendance Compared to 5:00-7:59 PM')
ax[1,1].set_xlabel('Difference in attendance')
ax[1,1].text(-300,3,'Best attendance: 5:00-7:59 PM',fontsize=10,fontweig
             horizontalalignment='center',verticalalignment='center')
ax[1,1].text(-300,0,'Slight decrease for 2:00-4:59 PM',fontsize=10,fontw
             horizontalalignment='center',verticalalignment='center',col

fig.tight_layout();
```



Day of the Week: All the bars showing the change in attendance compared to matches on Saturdays (hence why there is no bar for Saturday). The most important trend here is that Wednesday matches get over 2,700 fewer people than an average Saturday match. To some extent, MLS has to play some midweek matches, but there are changes they can make to reduce the number of those matches.

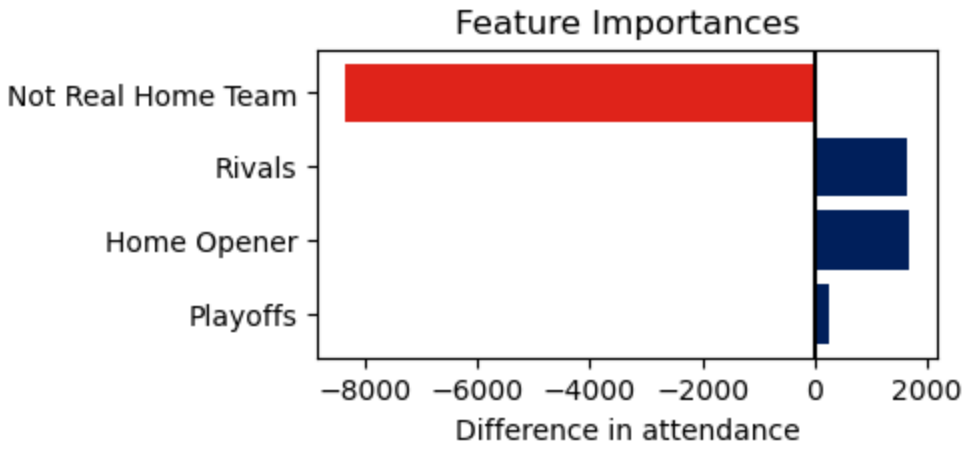
Month: There is a clear trend that attendance tends to improve as the season progresses from February to December. The most likely reason for this is that matches become more important as the playoff picture becomes clearer and then the playoffs actually start in late October. The big takeaway for me is that people want to see matches where the stakes are high.

Year: The fact that 2021 got about 4,000 fewer people per match was mostly due to the pandemic.

Kick Off Time: The best time to kick off a match according to the model is between 5:00 pm and 7:59 pm. However, the change in attendance does not appear to be huge as the kick off time changes. In fact, the linear regression model found similar numbers, but they were not

significant in that model

In [51]:
```
# Playoffs, home openers, rivals, and real home teams importance
fig, ax = plt.subplots(figsize=(4,2))

feat_importance = [xgb_importance[key] for key in xgb_importance.keys()




feat_names = ['Playoffs','Home Opener','Rivals','Not Real Home Team']
ax.barh(y=range(4),
        width=feat_importance,
        color=[mls_blue if x>=0 else mls_red for x in feat_importance])
ax.axvline(0, color='black')
ax.set_title('Feature Importances')
ax.set_yticks(range(4))
ax.set_yticklabels(feat_names)
ax.set_xlabel('Difference in attendance');
```



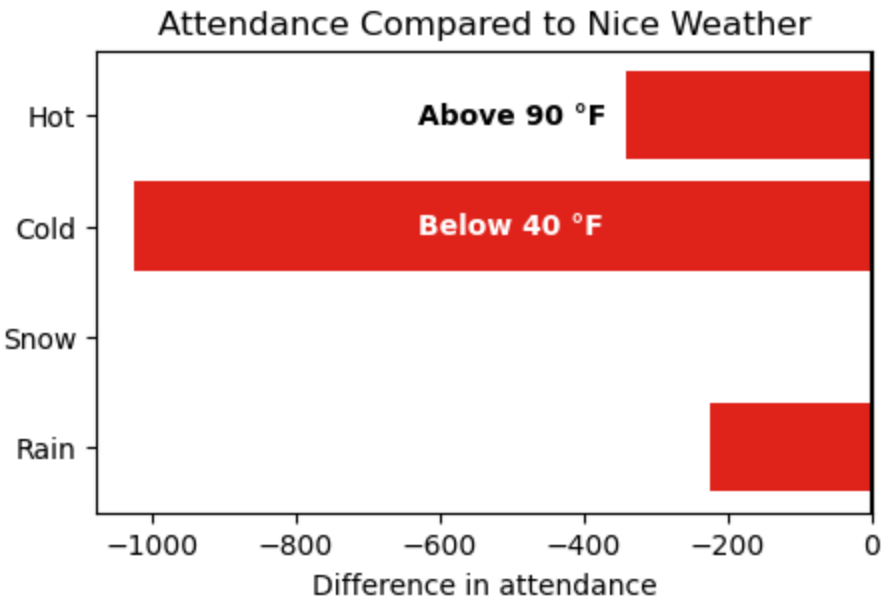
Not Real Home Team: The biggest effect is not having a real home team in a match. When that happened, the average attendance is expected to be 8,000 lower than average, a huge decrease.

Rivals: When rivals play, the attendance is expected to go up about 2,000.

Home Openers: While February and March are predicted to have the worst attendance, home openers are predicted to have a significant boost in attendance, around 2,000.

Playoffs: This one is interesting because the model says that the attendance is not expected to go up because a match is a playoff match. However, I think this is wrong. Earlier, we saw that November and December had the highest predicted attendance. The thing that makes those months special is that the playoffs happen during those months. I think the reason the playoff parameter in the model has such a small effect is because that effect is already built in to the model through the month parameters.

In [52]:
```
# Weather parameters
fig, ax = plt.subplots(figsize=(5,3))

feat_importance = [xgb_importance[key] for key in xgb_importance.keys()




feat_names = ['Rain','Snow','Cold','Hot']
ax.barh(y=range(4),
        width=feat_importance,
        color=[mls_blue if x>=0 else mls_red for x in feat_importance])

ax.axvline(0, color='black')
ax.set_title('Attendance Compared to Nice Weather')
ax.set_yticks(range(4))
ax.set_yticklabels(feat_names)
ax.set_xlabel('Difference in attendance')
ax.text(-500,3,'Above 90 \N{DEGREE SIGN}F',fontsize=10,fontweight='bold
            horizontalalignment='center',verticalalignment='center')
ax.text(-500,2,'Below 40 \N{DEGREE SIGN}F',fontsize=10,fontweight='bold
            horizontalalignment='center',verticalalignment='center',co
```



Hot and cold weather: According to the model, cold and hot temperatures don't seem to have a major effect. The model does say that cold weather (below 40 degrees Fahrenheit) is expected to get about 1,000 fewer people on average, but that is a smaller effect than many of the other factors.

Rain and snow: The model does not indicate there is much of an effect on attendance when it rains or snows. However, I think this might be mostly due to low sample size.

Recommendations

Weekends better than midweek: The model indicated that matches played on Wednesdays get about 2,700 fewer fans than Saturday matches and Tuesdays/Thursdays get almost 1,000 fewer fans. Attendance would improve if MLS could play more of their matches on weekends rather than midweek.

It is impossible to avoid playing some midweek matches because there simply aren't enough weeks during the season for the teams to play all their matches on the weekend, but there are steps MLS can take to improve the situation. Currently, MLS teams also compete in a few other competitions: the US Open Cup, the CONCACAF Champions Cup, and the Leagues Cup. The latter two competitions are somewhat redundant since they are both competitions that include teams from other North American countries (the Champions Cup is all of North America while the Leagues Cup is an MLS vs. Liga MX competition). At least in 2023, a big reason teams had to play so many midweek matches is because the Leagues Cup forced MLS to pause its season for a full month. This wouldn't be such a problem if the Leagues Cup was well attended, but the average was just 17,257, significantly lower than MLS matches. Even worse, some teams were eliminated from the Leagues Cup after the first week, meaning they didn't play a match for about three weeks. I think MLS should reconsider the format of the Leagues Cup, particularly if it is worth it to pause MLS for such a long time. Without the Leagues Cup, MLS would have gotten back about 5 weekends.

Make regular season matter: The attendance improved from the beginning of the season to the end of the season. I think a big part of this trend is that matches become more intriguing as the playoffs approach because the stakes of each match get bigger. People want to watch matches that matter. The problem is that many fans feel like most of the regular season does not matter that much because so many teams make the playoffs, so a team does not actually have to perform that well to qualify. In most European leagues, a team has to finish with the best overall record to be crowned champion, but in MLS, over half of the teams qualify for the playoffs (in 2023, 18 teams out of 29 make the playoffs). Sporting Kansas City provides a great example of how low the bar is. In their first 10 matches, they got 0 wins, 3 ties, and 7 losses. This was the worst 10 match beginning to a season in history, and yet they recovered to make the playoffs, then proceeded to beat the 1-seed in the West, St. Louis FC. This shows that a team can perform quite poorly, but still make the playoffs. Then the team just needs to go on a hot streak to make a run in the playoffs.

The relative importance of regular season matches would increase if fewer teams made the playoffs, but this also comes with a downside. The fewer playoff spots there are, then the earlier teams will be eliminated from playoff contention. Once a team is unable to make the playoffs, their matches cease to matter.

I do have an idea for a compromise between number of teams making the playoffs and rewarding teams for finishing higher in the standings. In the Australian Football League (AFL), which plays Australian rules football, they have a format in which the top 8 teams make the playoffs, but the top 4 start with a pretty significant advantage (click this link for the exact details: https://en.wikipedia.org/wiki/AFL_final_eight_system

(https://en.wikipedia.org/wiki/AFL_final_eight_system)). I think this format would do a good job of allowing enough teams into the playoffs while still rewarding the best-performing teams.

Favor warmer cities in winter: The model indicated that matches with temperatures below 40 degrees Fahrenheit had about 1,000 fewer fans on average. This seems to support the decision made by MLS to not play matches during most of the winter. The conditions would be even worse if MLS tried to hold matches in January and early February. Presumably, the attendance would also drop further.

MLS is forced to play some matches in February. Otherwise, they would not have enough time to actually play all of the matches they have scheduled. MLS could improve overall attendance by favoring warmer cities, meaning cities at lower latitude, during the first couple weeks of the year. This means playing February matches in places like Los Angeles, Houston, and Miami rather than places like Minnesota, New York, and Toronto.

Other considerations:

Sharing stadium with NFL team: The three teams with the highest overall attendance (Atlanta United, Seattle Sounders, and Charlotte FC) all share a stadium with an NFL team. Their advantage is pretty obvious: they have a larger stadium that they can utilize when demand is high. One argument against having a huge stadium is that the fan experience is not as fun when the stadium is less than half full, but this is not a big issue in my opinion. When there is not enough demand to fill the whole stadium, teams just sell tickets for the sections closest to the field, making for a similar environment as a smaller stadium.

While the top three teams in attendance indicate that having a larger stadium is better, it is not a guarantee of success. The Chicago Fire moved out of their smaller stadium in 2020 to move back into Soldier Field, which they share with the Chicago Bears of the NFL. The move has not led to a significant increase in attendance. This could possibly be due to the stadium being less easily accessible than the stadiums in Atlanta or Seattle or possibly just because the Fire have not been competitive for a long time.

I would not recommend that a team that already has their own stadium abandon it to move into a larger stadium. However, I do think this is an option that future expansion teams should explore. For example, if Phoenix or Las Vegas are awarded expansion teams, they should strongly consider using the stadiums owned by the Cardinals and Raiders of the NFL. They would have the potential for higher attendance and they would not have to pay to build a new stadium.

Promote rivalries: The model indicates that rivalry matches get about 2,000 more people on average. I think MLS would benefit from promoting these rivalries, but I would caution that they should not try to manufacture rivalries. Instead, they should make sure announcers and articles on their website highlight the history between teams. A good recent example of this is FC Cincinnati and the New York Red Bulls. They are not close enough geographically to be automatic rivals, but a rivalry was kindled when a player switched from Cincinnati to New York, leaving with some not so kind words for his former club. These are the kinds of

In [53]:
```
# Remake plot of actual attendance vs. predicted attendance for presenta
fig, ax = plt.subplots(figsize=(5,5))

ax.scatter(xgb_best.predict(X_test)/1000,
           y_test/1000,
           s=7,
           alpha=0.3,
           color=mls_blue)
ax.plot([0,80],[0,80], color='black')
ax.set_title('Actual Attendance vs. Predicted Attendance')
ax.set_xlabel('Predicted Attendance (Thousands)')
ax.set_ylabel('Actual Attendance (Thousands)')
ax.text(0,80,f"R$^2$ = {np.round(r2_score(y_test, xgb_best.predict(X_tes
ax.text(0,75,f"RMSE = {np.round(mse(y_test, xgb_best.predict(X_test), sq
ax.text(50,5,'This data not \nused to train model')
```

Out[53]:
```
Text(50, 5, 'This data not \nused to train model')
```