Data Exploration and Feature Engineering

In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sb
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor


pd.set_option('display.max_columns', None)  # To display all columns
pd.set_option('display.width', None)        # To avoid line wrapping
pd.set_option('display.max_rows', 100)
```

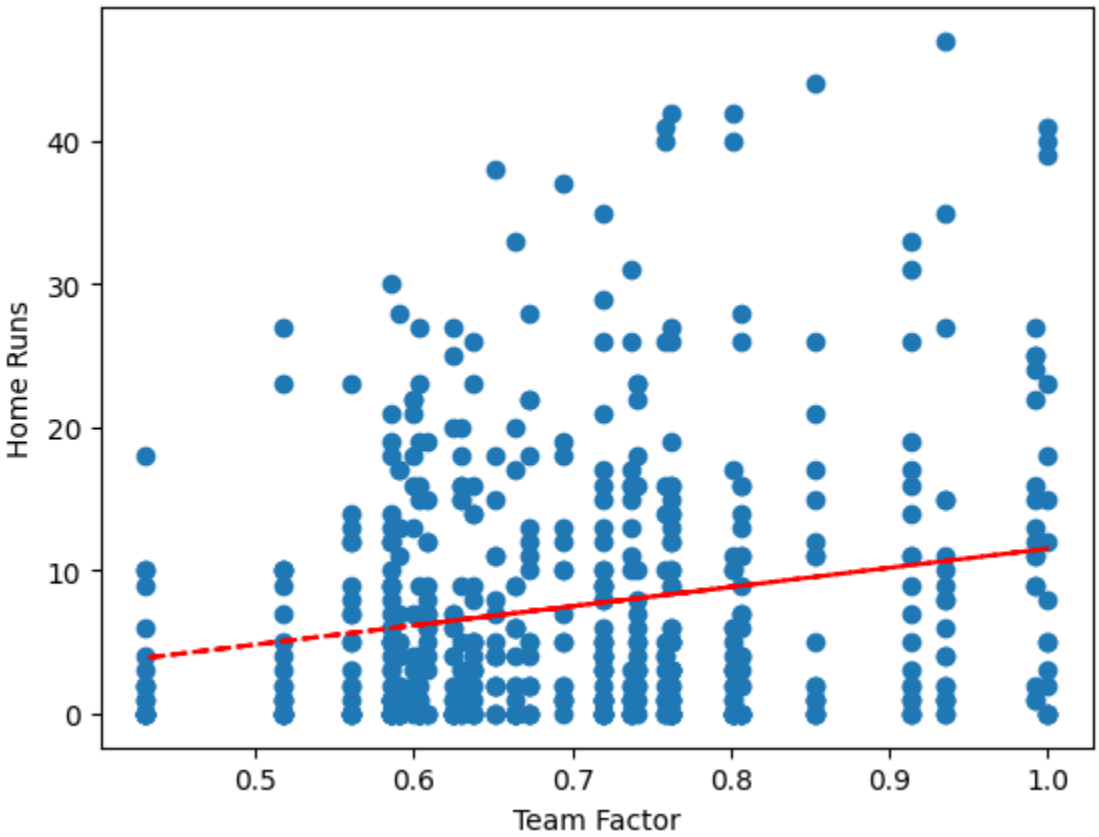
Team Factor Feature

This proposed feature will theoretically help to account for how a players home field affects the number of homeruns a player may hit. Formula to calculate it will be:

$$teamfactor = \frac{teamscore}{max(teamscore)}$$

In [2]:
```
# Loading a 2015 team stats data set into a dataframe
team_stats = pd.read_csv("../../data/raw/team_stats_2015.csv")

# adding a column for team factor feature
team_stats.insert(0, "tmFactor", 0)

# populating the team factor column for each team
team_stats["tmFactor"] = team_stats["HR"]/max(team_stats["HR"])

# creating a dictionary of Team: team Factor
team_factor = team_stats.set_index("Team")["tmFactor"].to_dict()

# loading a season dataset and cleaning out some columns that are duplicates or
# appear to be blank on many rows based on a quick scan in excel
players =  pd.read_csv("../../data/raw/MLB_player_stats_2015.csv")
players = players[players['PA'] >= 20]
players = players.drop(columns=['Name.1', 'Team.1','UBR', 'wGDP', 'XBR'], axis=1)
#####players = players.drop(columns=['Name.1', 'Team.1'], axis=1)
# adding and populating team factor column on player stats data set
players.insert(0,  "tmFactor", 0)
players["tmFactor"] = players["Team"].map(team_factor)

# drop nan rows for team factor.
# These are caused by players that played for multiple teams in a season as their T

players = players.dropna(subset = ['tmFactor'])
```

```
# assigning columns to variables to for scatter plot and trendline
x = players["tmFactor"]
y = players["HR"]
z = np.polyfit(x, y, 1) # fits a linear polynomial
p = np.poly1d(z)    # creates a polynomial function from coefficients from z


plt.scatter(x,y)
plt.plot(x, p(x), "r--")
plt.xlabel("Team Factor")
plt.ylabel("Home Runs")
plt.show()
```



Centering the team factor

A mean centered feature is "centered around the mean" by replacing the value with the value - mean value. In ths case:

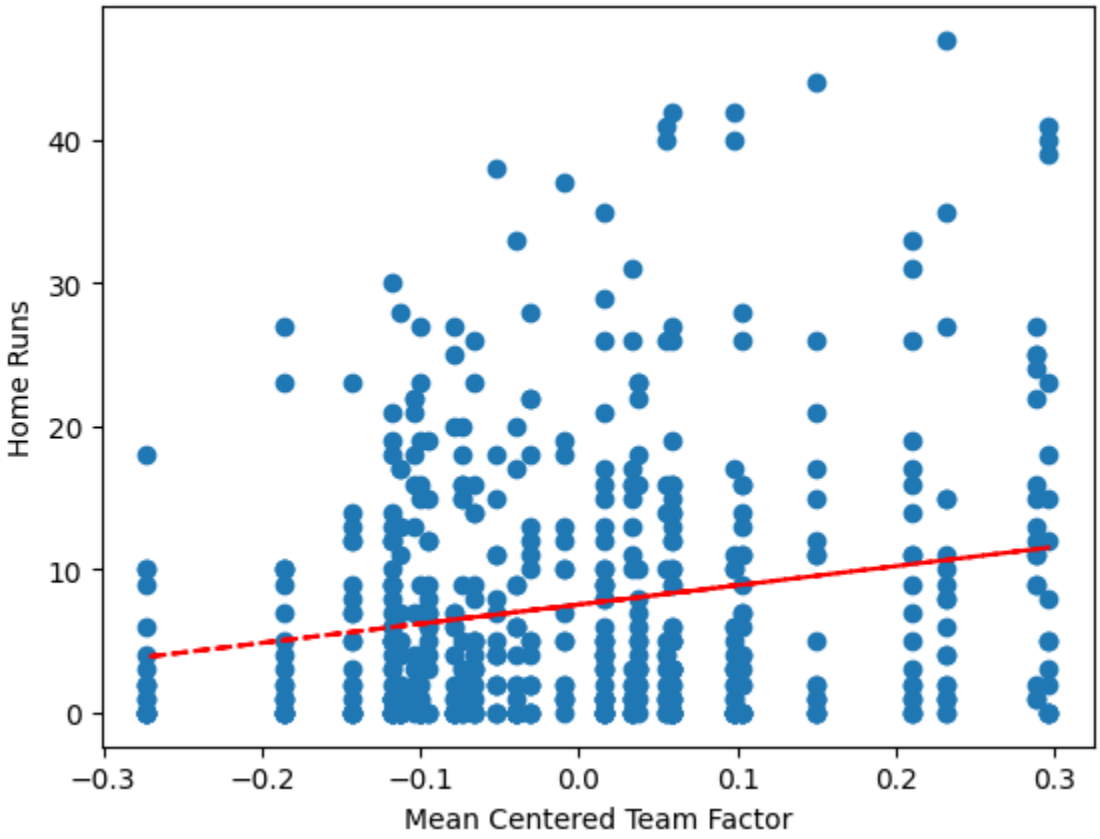
centered team factor = team factor - mean team factor

or

$x_i = x_i - \bar{x}$

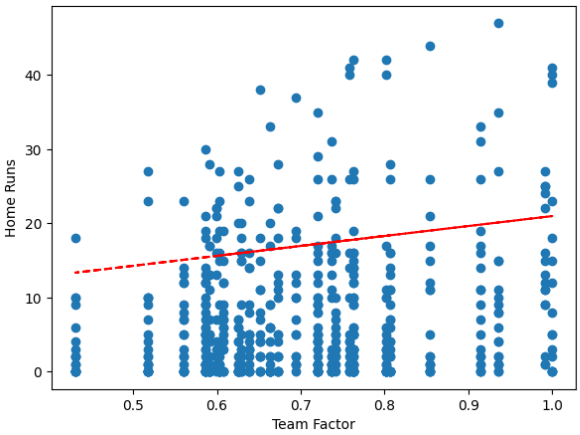
In [3]:
```
x_tm_factor_centered = players['tmFactor'] - players['tmFactor'].mean()

y = players["HR"]
```

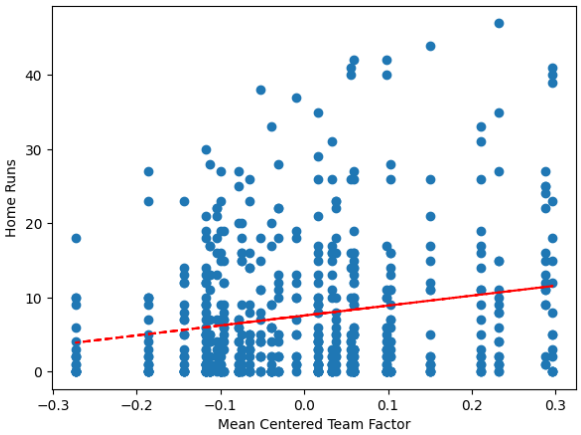
```
z = np.polyfit(x_tm_factor_centered, y, 1) # fits a linear polynomial
p = np.poly1d(z)    # creates a polynomial function from coefficients from z


plt.scatter(x_tm_factor_centered,y)
plt.plot(x_tm_factor_centered, p(x_tm_factor_centered), "r--")
plt.xlabel("Mean Centered Team Factor")
plt.ylabel("Home Runs")
plt.show()
```



In [4]:
```
#from matplotlib.pyplot import subplots

fig, axes = plt.subplots(nrows=1,
                         ncols=2,
                         figsize=(15,5))

axes[1].scatter(x_tm_factor_centered,y)
axes[1].plot(x_tm_factor_centered, p(x_tm_factor_centered), "r--")
axes[1].set_xlabel("Mean Centered Team Factor")
axes[1].set_ylabel("Home Runs")

axes[0].scatter(x,y)
axes[0].plot(x, p(x), "r--")
axes[0].set_xlabel("Team Factor")
axes[0].set_ylabel("Home Runs")
```

Out[4]:
```
Text(0, 0.5, 'Home Runs')
```





mean centering does not change the shape of the data, but it does affect the location of the intercept of the plot line. This can help with colinearity and will be used for a second iteration of our linear regression model.

Handling players who played for more than one team

$$Weightedteamfactor = \frac{\sum_i^n team_i Factor \times team_i PAs}{\sum_i^n team_i PA}$$

This value will be calculated using the player stats with team splits and then inserted to the main data set.

Player ages

Players ages will be scaled using the following formula:

$$\text{scaled player age} = \frac{\text{player age}}{50}$$

Season Factor

A scaled season factor will be used to account for changes to the game from season to season, such as the shortened 2020 season, balls started to be kept in humidors, or the introduction of the pitch clock.

This stat will be calculated much like the team factor stat

$$\text{Season factor} = \frac{seasonHRs}{max(seasonHRs)}$$

The code snippet below if from the data processing script that shows how the season factor is generated and applied to the data set.

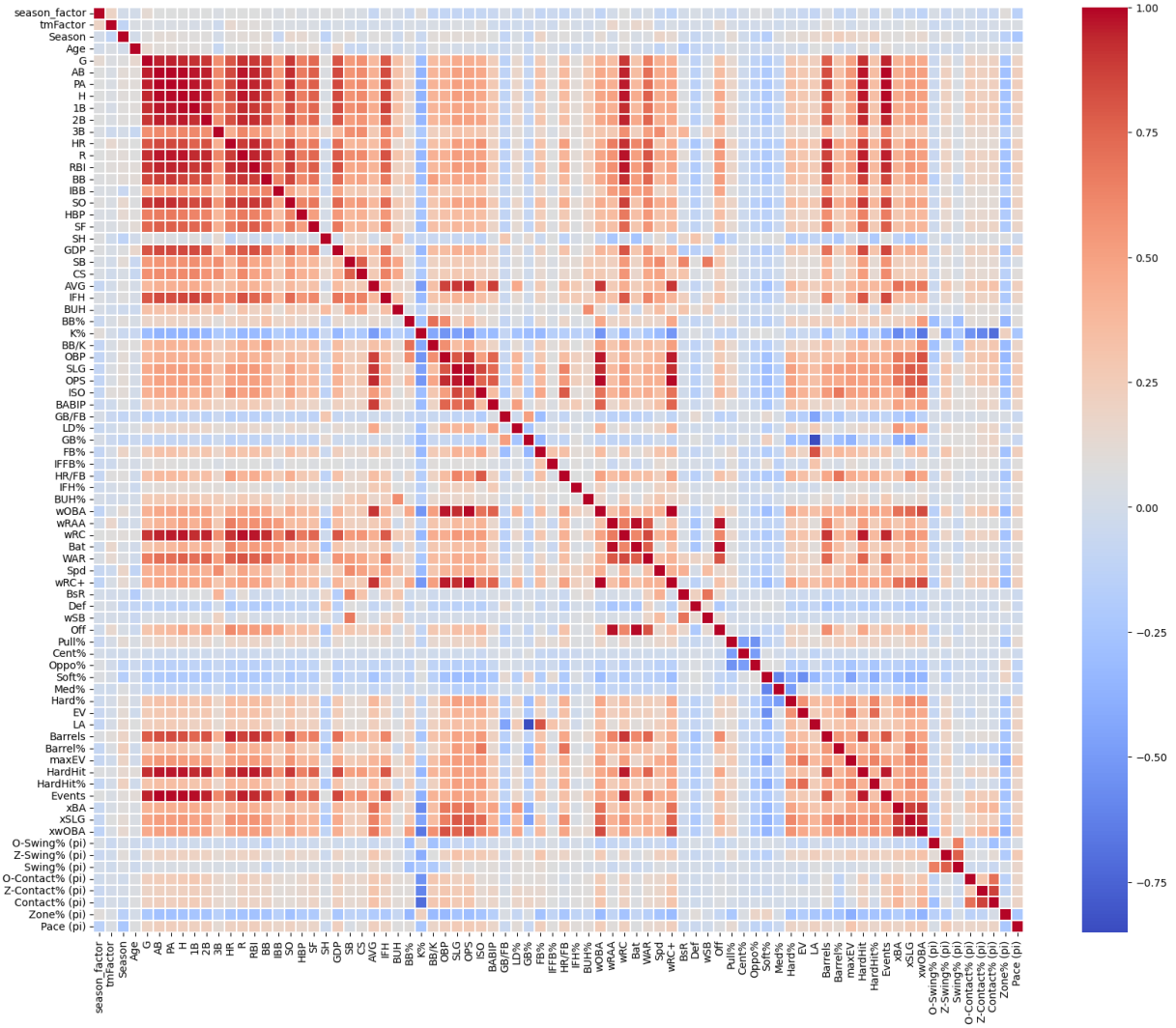
```
    # while looping through the datasets for each season
        # Create a new row in a DataFrame format to store the total
```

```
number of homeruns for the sesaon
    new_row = pd.DataFrame({'season': [season], 'HR_total':
[team_stats_df['HR'].sum()]})

    # Concatenate the new row with the existing DataFrame to store
season total homeruns
    season_factor_df = pd.concat([season_factor_df, new_row],
ignore_index=True)

    # Combine data into the main dataset
    combined_data_df = pd.concat([combined_data_df,
player_stats_df], ignore_index=True)
# end of loop

# generate dictionary of season factors
season_factor_df.insert(0, "season_factor", 0)
season_factor_df['season_factor'] = season_factor_df['HR_total']/
max(season_factor_df['HR_total'])
season_factor_dict = season_factor_df.set_index('season')
['season_factor'].to_dict()

# Add season factor to the dataframe
combined_data_df.insert(0, "season_factor", 0)
combined_data_df["season_factor"] =
combined_data_df["Season"].map(season_factor_dict)
```

Selecting features for initial linear regression model

I will start by examining correlation by generating a correlation matrix

In [5]:
```
import seaborn as sns
#players_df = pd.read_csv('../../data/processed/nicks_dataset.csv')
players_df = pd.read_csv('../../data/processed/combined_player_stats.csv')
players_df = players_df.drop(columns=['Name', 'Team', 'Name.1', 'Team.1','NameASCII

correlation_matrix = players_df.corr()
plt.figure(figsize=(20,16))
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm', fmt='.2f', linewidths
plt.show()
```



In [6]:
```
target = "HR"
target_corr = correlation_matrix[target].sort_values(ascending=False)

print("Feature Correlation with Target:")
print(target_corr.to_string())
```

```
Feature Correlation with Target:
HR                  1.000000
Barrels             0.950002
RBI                 0.940067
wRC                 0.915917
R                   0.890957
HardHit             0.888977
SO                  0.858090
PA                  0.855113
AB                  0.847008
H                   0.844005
BB                  0.843139
2B                  0.819400
Events              0.816571
G                   0.797992
WAR                 0.766209
1B                  0.754913
GDP                 0.717416
SF                  0.669875
wRAA                0.659078
IFH                 0.628069
Bat                 0.621447
Off                 0.603960
IBB                 0.599864
HBP                 0.593339
ISO                 0.575268
xSLG                0.568554
SLG                 0.517860
HR/FB               0.508497
Barrel%             0.493972
maxEV               0.490829
xwOBA               0.483705
OPS                 0.476265
wRC+                0.447066
wOBA                0.444578
3B                  0.432689
CS                  0.407991
SB                  0.385568
HardHit%            0.385252
xBA                 0.367920
OBP                 0.361691
Hard%               0.361371
AVG                 0.349121
BB/K                0.331721
FB%                 0.322664
EV                  0.320795
Spd                 0.300939
LA                  0.260207
Pace (pi)           0.208012
BABIP               0.206944
BB%                 0.201704
Pull%               0.191890
Z-Swing% (pi)       0.186259
BUH%                0.169587
tmFactor            0.142175
BUH                 0.136768
```

```
O-Contact% (pi)    0.127109
Contact% (pi)      0.121226
Z-Contact% (pi)    0.119886
LD%                0.119317
Season             0.093558
IFH%               0.076445
Age                0.073826
season_factor      0.073323
IFFB%              0.064368
Swing% (pi)        0.026969
wSB               -0.009978
O-Swing% (pi)     -0.043837
Cent%             -0.044266
BsR               -0.056351
SH                -0.102046
GB%               -0.123391
Med%              -0.137617
Oppo%             -0.148177
GB/FB             -0.158458
Soft%             -0.186363
Def               -0.224620
Zone% (pi)        -0.248399
K%                -0.250783
```

In [7]:
```
players_df.head
```

Out[7]:
```
<bound method NDFrame.head of       season_factor  tmFactor  Season   Age    G   A
B   PA    H  1B  2B  3B  \
0          0.724469  0.603448    2015  0.76   26   42   52    5   4   0   0
1          0.724469  0.762931    2015  0.46    3    2    2    0   0   0   0
2          0.724469  0.806034    2015  0.68   63  181  217   43  27   9   0
3          0.724469  0.431034    2015  0.76  113  407  436  122  88  24   1
4          0.724469  0.560345    2015  0.48   39  137  161   33  13  11   4
...             ...       ...     ...   ...  ...  ...  ...  ...  ..  ..  ..
8495       0.804752  0.683544    2024  0.58  118  297  325   64  41  14   5
8496       0.804752  0.696203    2024  0.46  155  542  602  135  77  34   1
8497       0.804752  0.561181    2024  0.60   15   33   39    8   6   1   1
8498       0.804752  0.827004    2024  0.48  138  497  547  105  66  20   2
8499       0.804752  0.888905    2024  0.58   42   70   88    9   6   3   0

      HR   R  RBI  BB  IBB   SO  HBP  SF  SH  GDP  SB  CS       AVG  IFH  BUH  \
0      1   1    5   0    0   20    0   1   9    1   0   0  0.119048    0    0
1      0   0    0   0    0    0    0   0   0    1   0   0  0.000000    0    0
2      7  24   21  32    1   38    1   0   3    4   0   0  0.237569    1    0
3      9  38   49  19    2   37    7   3   0   19   0   2  0.299754    7    0
4      5  25   22  16    0   41    5   2   1    3   6   2  0.240876    3    0
...   ..  ..  ...  ..  ...  ...  ...  ..  ..  ...  ..  ..       ...  ...  ...
8495   4  32   23  24    0   69    2   2   0    3  16   0  0.215488    3    0
8496  23  70   77  39    1  140   16   1   4   14  30  10  0.249077    9    1
8497   0   3    1   5    0   10    0   0   1    1   1   0  0.242424    1    0
8498  17  60   49  38    0  188    4   5   3    7  25   3  0.211268    8    1
8499   0  11    5  14    0   24    1   0   3    0   2   1  0.128571    1    1

           BB%        K%      BB/K       OBP       SLG       OPS       ISO  \
0     0.000000  0.384615  0.000000  0.116279  0.190476  0.306755  0.071429
1     0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
2     0.147465  0.175115  0.842105  0.355140  0.403315  0.758455  0.165746
3     0.043578  0.084862  0.513514  0.339450  0.429975  0.769425  0.130221
4     0.099379  0.254658  0.390244  0.337500  0.489051  0.826551  0.248175
...        ...       ...       ...       ...       ...       ...       ...
8495  0.073846  0.212308  0.347826  0.276923  0.336700  0.613623  0.121212
8496  0.064784  0.232558  0.278571  0.317726  0.442804  0.760530  0.193727
8497  0.128205  0.256410  0.500000  0.342105  0.333333  0.675439  0.090909
8498  0.069470  0.343693  0.202128  0.270221  0.362173  0.632394  0.150905
8499  0.159091  0.272727  0.583333  0.282353  0.171429  0.453782  0.042857

          BABIP     GB/FB       LD%       GB%       FB%     IFFB%     HR/FB  \
0     0.181818  5.000000  0.217391  0.652174  0.130435  0.000000  0.333333
1     0.000000  1.000000  0.000000  0.500000  0.500000  0.000000  0.000000
2     0.264706  1.240000  0.211268  0.436620  0.352113  0.020000  0.140000
3     0.310440  1.626168  0.246649  0.466488  0.286863  0.121495  0.084112
4     0.301075  1.054054  0.224490  0.397959  0.377551  0.108108  0.135135
...        ...       ...       ...       ...       ...       ...       ...
8495  0.265487  0.773196  0.248908  0.327511  0.423581  0.041237  0.041237
8496  0.294737  1.142857  0.172932  0.441103  0.385965  0.090909  0.149351
8497  0.347826  1.571429  0.181818  0.500000  0.318182  0.000000  0.000000
8498  0.296296  1.118644  0.196141  0.424437  0.379421  0.110169  0.144068
8499  0.195652  0.583333  0.155556  0.311111  0.533333  0.083333  0.000000

          IFH%      BUH%      wOBA       wRAA        wRC        Bat       WAR  \
0     0.000000  0.000000  0.129967  -7.613813  -1.766938  -7.717518 -0.076850
1     0.000000  0.000000  0.000000  -0.500539  -0.275659  -0.510371 -0.025590
```

```
2     0.016129  0.000000  0.335944   3.941941  28.341397   4.152935  0.511462
3     0.040230  0.000000  0.333060   6.915596  55.939390   5.395772  0.930997
4     0.076923  0.000000  0.353456   5.177591  23.280414   4.568318  1.489093
...        ...       ...       ...        ...        ...        ...       ...
8495  0.040000  0.000000  0.270277 -10.437673  27.580200  -9.259699  1.078802
8496  0.051136  0.125000  0.329956   9.582577  80.003376   9.723791  3.489527
8497  0.090909  0.000000  0.304744  -0.170604   4.391541  -0.105197  0.257332
8498  0.060606  0.166667  0.276421 -14.862482  49.124524 -11.140251  1.432776
8499  0.071429  0.250000  0.228490  -5.759897   4.534174  -5.989754 -0.429462

          Spd        wRC+       BsR       Def       wSB        Off     Pull%  \
0     0.142857  -27.370685 -0.007843  5.395016 -0.007843  -7.725361  0.312500
1     0.142857 -100.000000  0.000000  0.207501  0.000000  -0.510371  0.000000
2     1.080357  116.424439 -2.715686 -3.321893 -0.115685   1.437249  0.363014
3     1.736382  110.620923 -4.803810 -5.289096 -1.003811   0.591961  0.423592
4     8.036122  124.351495  1.049130  3.438364  0.349129   5.617447  0.353535
...        ...         ...       ...       ...       ...        ...       ...
8495  8.182890   75.099510  5.055838  3.800079  2.671208  -4.203861  0.391304
8496  5.427539  114.116711  1.796802  2.170734  0.916029  11.520593  0.503686
8497  6.500663   97.642603  0.155080  1.139834  0.113183   0.049883  0.375000
8498  6.394459   82.200736  4.285689  2.467187  2.932599  -6.854562  0.466877
8499  4.777140   44.068385  0.156304 -1.256555 -0.170747  -5.833450  0.326531

         Cent%     Oppo%     Soft%      Med%     Hard%         EV         LA  \
0     0.468750  0.218750  0.437500  0.406250  0.156250  84.500738  -9.018391
1     0.500000  0.500000  0.000000  1.000000  0.000000  76.032501  16.334999
2     0.363014  0.273973  0.198630  0.506849  0.294521  87.681620   9.801383
3     0.335121  0.241287  0.201072  0.538874  0.260054  85.400034  10.583399
4     0.343434  0.303030  0.202020  0.505051  0.292929  90.904075  11.568464
...        ...       ...       ...       ...       ...        ...        ...
8495  0.373913  0.234783  0.121739  0.591304  0.286957  87.978473  18.683960
8496  0.299754  0.196560  0.137592  0.535627  0.326781  88.491275  12.305902
8497  0.333333  0.291667  0.083333  0.791667  0.125000  86.866048  14.331496
8498  0.328076  0.205047  0.104101  0.611987  0.283912  89.197045  14.237215
8499  0.326531  0.346939  0.122449  0.632653  0.244898  88.563941  21.244562

      Barrels   Barrel%    maxEV  HardHit  HardHit%  Events    xBA   xSLG  \
0         1.0  0.031250  104.322      7.0  0.218750      32  0.120  0.193
1         0.0  0.000000   82.962      0.0  0.000000       2  0.070  0.089
2         8.0  0.054795  108.544     48.0  0.328767     146  0.243  0.409
3        11.0  0.029491  109.962     95.0  0.254692     373  0.277  0.400
4         2.0  0.020202  110.026     43.0  0.434343      99  0.241  0.347
...       ...       ...      ...      ...       ...     ...    ...    ...
8495     10.0  0.043478  108.147     75.0  0.326087     230  0.238  0.364
8496     34.0  0.083538  111.306    157.0  0.385749     407  0.236  0.414
8497      1.0  0.041667  105.293      7.0  0.291667      24  0.228  0.324
8498     28.0  0.088328  109.136    118.0  0.372240     317  0.199  0.347
8499      2.0  0.040816  108.933     13.0  0.265306      49  0.169  0.250

      xwOBA  O-Swing% (pi)  Z-Swing% (pi)  Swing% (pi)  O-Contact% (pi)  \
0     0.137       0.301887       0.476190     0.424581         0.312500
1     0.070       0.500000       1.000000     0.600000         0.500000
2     0.342       0.129676       0.527619     0.355292         0.615385
3     0.318       0.363023       0.740902     0.544818         0.769517
4     0.312       0.216561       0.601852     0.412226         0.500000
...     ...            ...            ...          ...              ...
```

```
8495  0.294       0.292642       0.639017     0.473179         0.605714
8496  0.317       0.312771       0.660808     0.485802         0.530556
8497  0.302       0.161290       0.505618     0.364238         0.500000
8498  0.271       0.289641       0.703336     0.512895         0.357664
8499  0.271       0.134078       0.617925     0.396419         0.583333

      Z-Contact% (pi)  Contact% (pi)  Zone% (pi)  Pace (pi)
0            0.816667       0.710526    0.703911  22.119048
1            1.000000       0.666667    0.200000  20.250000
2            0.888087       0.844985    0.566955  22.139738
3            0.923379       0.870180    0.481092  22.454822
4            0.861538       0.768061    0.507837  20.627706
...               ...            ...         ...        ...
8495         0.884615       0.802030    0.521217  17.656489
8496         0.839096       0.739209    0.497160  18.721625
8497         0.866667       0.800000    0.589404  17.292035
8498         0.756410       0.652751    0.539659  18.735020
8499         0.763359       0.735484    0.542199  18.495050

[8500 rows x 78 columns]>
```

Variance inflation factor

In [8]:
```
# Compute VIF for each feature
features = players_df.drop(columns=[target]).dropna(ignore_index=True)  # All colum
features.head
```

Out[8]:
```
<bound method NDFrame.head of       season_factor  tmFactor  Season   Age    G   A
B   PA    H  1B  2B  3B  \
0          0.724469  0.603448    2015  0.76   26   42   52    5   4   0   0
1          0.724469  0.762931    2015  0.46    3    2    2    0   0   0   0
2          0.724469  0.806034    2015  0.68   63  181  217   43  27   9   0
3          0.724469  0.431034    2015  0.76  113  407  436  122  88  24   1
4          0.724469  0.560345    2015  0.48   39  137  161   33  13  11   4
...             ...       ...     ...   ...  ...  ...  ...  ...  ..  ..  ..
7462       0.804752  0.683544    2024  0.58  118  297  325   64  41  14   5
7463       0.804752  0.696203    2024  0.46  155  542  602  135  77  34   1
7464       0.804752  0.561181    2024  0.60   15   33   39    8   6   1   1
7465       0.804752  0.827004    2024  0.48  138  497  547  105  66  20   2
7466       0.804752  0.888905    2024  0.58   42   70   88    9   6   3   0

       R  RBI  BB  IBB   SO  HBP  SF  SH  GDP  SB  CS       AVG  IFH  BUH  \
0      1    5   0    0   20    0   1   9    1   0   0  0.119048    0    0
1      0    0   0    0    0    0   0   0    1   0   0  0.000000    0    0
2     24   21  32    1   38    1   0   3    4   0   0  0.237569    1    0
3     38   49  19    2   37    7   3   0   19   0   2  0.299754    7    0
4     25   22  16    0   41    5   2   1    3   6   2  0.240876    3    0
...   ..  ...  ..  ...  ...  ...  ..  ..  ...  ..  ..       ...  ...  ...
7462  32   23  24    0   69    2   2   0    3  16   0  0.215488    3    0
7463  70   77  39    1  140   16   1   4   14  30  10  0.249077    9    1
7464   3    1   5    0   10    0   0   1    1   1   0  0.242424    1    0
7465  60   49  38    0  188    4   5   3    7  25   3  0.211268    8    1
7466  11    5  14    0   24    1   0   3    0   2   1  0.128571    1    1

           BB%        K%      BB/K       OBP       SLG       OPS       ISO  \
0     0.000000  0.384615  0.000000  0.116279  0.190476  0.306755  0.071429
1     0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
2     0.147465  0.175115  0.842105  0.355140  0.403315  0.758455  0.165746
3     0.043578  0.084862  0.513514  0.339450  0.429975  0.769425  0.130221
4     0.099379  0.254658  0.390244  0.337500  0.489051  0.826551  0.248175
...        ...       ...       ...       ...       ...       ...       ...
7462  0.073846  0.212308  0.347826  0.276923  0.336700  0.613623  0.121212
7463  0.064784  0.232558  0.278571  0.317726  0.442804  0.760530  0.193727
7464  0.128205  0.256410  0.500000  0.342105  0.333333  0.675439  0.090909
7465  0.069470  0.343693  0.202128  0.270221  0.362173  0.632394  0.150905
7466  0.159091  0.272727  0.583333  0.282353  0.171429  0.453782  0.042857

          BABIP     GB/FB       LD%       GB%       FB%     IFFB%     HR/FB  \
0     0.181818  5.000000  0.217391  0.652174  0.130435  0.000000  0.333333
1     0.000000  1.000000  0.000000  0.500000  0.500000  0.000000  0.000000
2     0.264706  1.240000  0.211268  0.436620  0.352113  0.020000  0.140000
3     0.310440  1.626168  0.246649  0.466488  0.286863  0.121495  0.084112
4     0.301075  1.054054  0.224490  0.397959  0.377551  0.108108  0.135135
...        ...       ...       ...       ...       ...       ...       ...
7462  0.265487  0.773196  0.248908  0.327511  0.423581  0.041237  0.041237
7463  0.294737  1.142857  0.172932  0.441103  0.385965  0.090909  0.149351
7464  0.347826  1.571429  0.181818  0.500000  0.318182  0.000000  0.000000
7465  0.296296  1.118644  0.196141  0.424437  0.379421  0.110169  0.144068
7466  0.195652  0.583333  0.155556  0.311111  0.533333  0.083333  0.000000

          IFH%      BUH%      wOBA       wRAA        wRC        Bat       WAR  \
0     0.000000  0.000000  0.129967  -7.613813  -1.766938  -7.717518 -0.076850
1     0.000000  0.000000  0.000000  -0.500539  -0.275659  -0.510371 -0.025590
```

```
2     0.016129  0.000000  0.335944   3.941941  28.341397   4.152935  0.511462
3     0.040230  0.000000  0.333060   6.915596  55.939390   5.395772  0.930997
4     0.076923  0.000000  0.353456   5.177591  23.280414   4.568318  1.489093
...        ...       ...       ...        ...        ...        ...       ...
7462  0.040000  0.000000  0.270277 -10.437673  27.580200  -9.259699  1.078802
7463  0.051136  0.125000  0.329956   9.582577  80.003376   9.723791  3.489527
7464  0.090909  0.000000  0.304744  -0.170604   4.391541  -0.105197  0.257332
7465  0.060606  0.166667  0.276421 -14.862482  49.124524 -11.140251  1.432776
7466  0.071429  0.250000  0.228490  -5.759897   4.534174  -5.989754 -0.429462

          Spd        wRC+       BsR       Def       wSB        Off     Pull%  \
0     0.142857  -27.370685 -0.007843  5.395016 -0.007843  -7.725361  0.312500
1     0.142857 -100.000000  0.000000  0.207501  0.000000  -0.510371  0.000000
2     1.080357  116.424439 -2.715686 -3.321893 -0.115685   1.437249  0.363014
3     1.736382  110.620923 -4.803810 -5.289096 -1.003811   0.591961  0.423592
4     8.036122  124.351495  1.049130  3.438364  0.349129   5.617447  0.353535
...        ...         ...       ...       ...       ...        ...       ...
7462  8.182890   75.099510  5.055838  3.800079  2.671208  -4.203861  0.391304
7463  5.427539  114.116711  1.796802  2.170734  0.916029  11.520593  0.503686
7464  6.500663   97.642603  0.155080  1.139834  0.113183   0.049883  0.375000
7465  6.394459   82.200736  4.285689  2.467187  2.932599  -6.854562  0.466877
7466  4.777140   44.068385  0.156304 -1.256555 -0.170747  -5.833450  0.326531

         Cent%     Oppo%     Soft%      Med%     Hard%         EV         LA  \
0     0.468750  0.218750  0.437500  0.406250  0.156250  84.500738  -9.018391
1     0.500000  0.500000  0.000000  1.000000  0.000000  76.032501  16.334999
2     0.363014  0.273973  0.198630  0.506849  0.294521  87.681620   9.801383
3     0.335121  0.241287  0.201072  0.538874  0.260054  85.400034  10.583399
4     0.343434  0.303030  0.202020  0.505051  0.292929  90.904075  11.568464
...        ...       ...       ...       ...       ...        ...        ...
7462  0.373913  0.234783  0.121739  0.591304  0.286957  87.978473  18.683960
7463  0.299754  0.196560  0.137592  0.535627  0.326781  88.491275  12.305902
7464  0.333333  0.291667  0.083333  0.791667  0.125000  86.866048  14.331496
7465  0.328076  0.205047  0.104101  0.611987  0.283912  89.197045  14.237215
7466  0.326531  0.346939  0.122449  0.632653  0.244898  88.563941  21.244562

      Barrels   Barrel%    maxEV  HardHit  HardHit%  Events    xBA   xSLG  \
0         1.0  0.031250  104.322      7.0  0.218750      32  0.120  0.193
1         0.0  0.000000   82.962      0.0  0.000000       2  0.070  0.089
2         8.0  0.054795  108.544     48.0  0.328767     146  0.243  0.409
3        11.0  0.029491  109.962     95.0  0.254692     373  0.277  0.400
4         2.0  0.020202  110.026     43.0  0.434343      99  0.241  0.347
...       ...       ...      ...      ...       ...     ...    ...    ...
7462     10.0  0.043478  108.147     75.0  0.326087     230  0.238  0.364
7463     34.0  0.083538  111.306    157.0  0.385749     407  0.236  0.414
7464      1.0  0.041667  105.293      7.0  0.291667      24  0.228  0.324
7465     28.0  0.088328  109.136    118.0  0.372240     317  0.199  0.347
7466      2.0  0.040816  108.933     13.0  0.265306      49  0.169  0.250

      xwOBA  O-Swing% (pi)  Z-Swing% (pi)  Swing% (pi)  O-Contact% (pi)  \
0     0.137       0.301887       0.476190     0.424581         0.312500
1     0.070       0.500000       1.000000     0.600000         0.500000
2     0.342       0.129676       0.527619     0.355292         0.615385
3     0.318       0.363023       0.740902     0.544818         0.769517
4     0.312       0.216561       0.601852     0.412226         0.500000
...     ...            ...            ...          ...              ...
```

```
7462  0.294       0.292642       0.639017     0.473179         0.605714
7463  0.317       0.312771       0.660808     0.485802         0.530556
7464  0.302       0.161290       0.505618     0.364238         0.500000
7465  0.271       0.289641       0.703336     0.512895         0.357664
7466  0.271       0.134078       0.617925     0.396419         0.583333

      Z-Contact% (pi)  Contact% (pi)  Zone% (pi)  Pace (pi)
0            0.816667       0.710526    0.703911  22.119048
1            1.000000       0.666667    0.200000  20.250000
2            0.888087       0.844985    0.566955  22.139738
3            0.923379       0.870180    0.481092  22.454822
4            0.861538       0.768061    0.507837  20.627706
...               ...            ...         ...        ...
7462         0.884615       0.802030    0.521217  17.656489
7463         0.839096       0.739209    0.497160  18.721625
7464         0.866667       0.800000    0.589404  17.292035
7465         0.756410       0.652751    0.539659  18.735020
7466         0.763359       0.735484    0.542199  18.495050

[7467 rows x 77 columns]>
```

In [9]:
```
vif_data = pd.DataFrame()
vif_data["Feature"] = features.columns
vif_data["VIF"] = [variance_inflation_factor(features.values, i) for i in range(fea

print("Variance Inflation Factor (VIF):")
print(vif_data)
```

```
c:\Users\nick_\source\repos\BE2100_HR_predictions_ML_project\venv\lib\site-packages
\statsmodels\stats\outliers_influence.py:197: RuntimeWarning: divide by zero encount
ered in scalar divide
  vif = 1. / (1. - r_squared_i)
```

```
Variance Inflation Factor (VIF):
           Feature           VIF
0     season_factor  1.315488e+00
1          tmFactor  1.186864e+00
2            Season  1.566226e+06
3               Age  1.172106e+00
4                 G  2.107677e+01
5                AB  1.982576e+05
6                PA           inf
7                 H  3.086596e+04
8                1B  4.256900e+03
9                2B  2.262900e+02
10               3B  4.486371e+00
11                R  5.094542e+01
12              RBI  3.561279e+01
13               BB           inf
14              IBB  4.105129e+00
15               SO           inf
16              HBP           inf
17               SF  2.494397e+01
18               SH  1.705611e+01
19              GDP  4.649043e+00
20               SB  1.640045e+02
21               CS  6.656914e+01
22              AVG           inf
23              IFH  5.270489e+00
24              BUH  2.363978e+00
25              BB%  1.700148e+01
26               K%  5.249606e+00
27             BB/K  5.301856e+00
28              OBP           inf
29              SLG           inf
30              OPS           inf
31              ISO           inf
32            BABIP  7.478774e+00
33            GB/FB  1.842681e+00
34              LD%  9.945519e+01
35              GB%  2.488037e+02
36              FB%  1.823844e+02
37            IFFB%  1.340331e+00
38            HR/FB  3.735012e+00
39             IFH%  1.320882e+00
40             BUH%  1.660440e+00
41             wOBA  1.163108e+03
42             wRAA  1.623258e+02
43              wRC  5.441271e+03
44              Bat           inf
45              WAR  7.811812e+02
46              Spd  2.640222e+00
47             wRC+  3.505583e+02
48              BsR           inf
49              Def  1.268575e+02
50              wSB  6.430519e+01
51              Off           inf
52            Pull%  5.004000e+14
53            Cent%  4.289143e+14
```

```
54            Oppo%  4.503600e+14
55            Soft%  4.289143e+14
56             Med%  4.740631e+14
57            Hard%  4.289143e+14
58               EV  3.736439e+00
59               LA  6.541375e+00
60          Barrels  2.643796e+01
61          Barrel%  5.265497e+00
62            maxEV  4.398016e+00
63          HardHit  4.684979e+01
64         HardHit%  3.680507e+00
65           Events           inf
66              xBA  4.058077e+01
67             xSLG  6.367011e+01
68            xwOBA  1.408320e+02
69    O-Swing% (pi)  1.121404e+01
70    Z-Swing% (pi)  1.760897e+01
71      Swing% (pi)  3.110378e+01
72  O-Contact% (pi)  6.299387e+00
73  Z-Contact% (pi)  9.586183e+00
74    Contact% (pi)  1.988382e+01
75       Zone% (pi)  3.323649e+00
76        Pace (pi)  1.861410e+00
```

Interpreting VIF data can be done with the following criteria

VIF Value	Interpretation
1	No correlation between the feature and other features.
1 < VIF < 5	Moderate correlation, but not problematic.
5 < VIF < 10	High correlation. This is a warning that multicollinearity might be an issue.
VIF > 10	Very high correlation, indicating significant multicollinearity. This could cause issues with model stability and interpretation.

Using these criteria, I will filter down feature lists for VIF values <=1, <5 and <10, and train a model with each set of features for comparison.

In [10]:
```
# Filter out features with VIF greater than 10
vif_threshold = 10

filtered_features = vif_data[vif_data["VIF"] < vif_threshold]["Feature"].tolist()

# Remove features with high VIF from X
players_filtered = players_df[filtered_features]

print(players_filtered)
```

```
      season_factor  tmFactor   Age  3B  IBB  GDP  IFH  BUH       K%  \
0          0.724469  0.603448  0.76   0    0    1    0    0  0.384615
1          0.724469  0.762931  0.46   0    0    1    0    0  0.000000
2          0.724469  0.806034  0.68   0    1    4    1    0  0.175115
3          0.724469  0.431034  0.76   1    2   19    7    0  0.084862
4          0.724469  0.560345  0.48   4    0    3    3    0  0.254658
...             ...       ...   ...  ..  ...  ...  ...  ...       ...
8495       0.804752  0.683544  0.58   5    0    3    3    0  0.212308
8496       0.804752  0.696203  0.46   1    1   14    9    1  0.232558
8497       0.804752  0.561181  0.60   1    0    1    1    0  0.256410
8498       0.804752  0.827004  0.48   2    0    7    8    1  0.343693
8499       0.804752  0.888905  0.58   0    0    0    1    1  0.272727

          BB/K     BABIP     GB/FB     IFFB%     HR/FB      IFH%      BUH%  \
0     0.000000  0.181818  5.000000  0.000000  0.333333  0.000000  0.000000
1     0.000000  0.000000  1.000000  0.000000  0.000000  0.000000  0.000000
2     0.842105  0.264706  1.240000  0.020000  0.140000  0.016129  0.000000
3     0.513514  0.310440  1.626168  0.121495  0.084112  0.040230  0.000000
4     0.390244  0.301075  1.054054  0.108108  0.135135  0.076923  0.000000
...        ...       ...       ...       ...       ...       ...       ...
8495  0.347826  0.265487  0.773196  0.041237  0.041237  0.040000  0.000000
8496  0.278571  0.294737  1.142857  0.090909  0.149351  0.051136  0.125000
8497  0.500000  0.347826  1.571429  0.000000  0.000000  0.090909  0.000000
8498  0.202128  0.296296  1.118644  0.110169  0.144068  0.060606  0.166667
8499  0.583333  0.195652  0.583333  0.083333  0.000000  0.071429  0.250000

           Spd         EV         LA   Barrel%    maxEV  HardHit%  \
0     0.142857  84.500738  -9.018391  0.031250  104.322  0.218750
1     0.142857  76.032501  16.334999  0.000000   82.962  0.000000
2     1.080357  87.681620   9.801383  0.054795  108.544  0.328767
3     1.736382  85.400034  10.583399  0.029491  109.962  0.254692
4     8.036122  90.904075  11.568464  0.020202  110.026  0.434343
...        ...        ...        ...       ...      ...       ...
8495  8.182890  87.978473  18.683960  0.043478  108.147  0.326087
8496  5.427539  88.491275  12.305902  0.083538  111.306  0.385749
8497  6.500663  86.866048  14.331496  0.041667  105.293  0.291667
8498  6.394459  89.197045  14.237215  0.088328  109.136  0.372240
8499  4.777140  88.563941  21.244562  0.040816  108.933  0.265306

      O-Contact% (pi)  Z-Contact% (pi)  Zone% (pi)  Pace (pi)
0            0.312500         0.816667    0.703911  22.119048
1            0.500000         1.000000    0.200000  20.250000
2            0.615385         0.888087    0.566955  22.139738
3            0.769517         0.923379    0.481092  22.454822
4            0.500000         0.861538    0.507837  20.627706
...               ...              ...         ...        ...
8495         0.605714         0.884615    0.521217  17.656489
8496         0.530556         0.839096    0.497160  18.721625
8497         0.500000         0.866667    0.589404  17.292035
8498         0.357664         0.756410    0.539659  18.735020
8499         0.583333         0.763359    0.542199  18.495050

[8500 rows x 26 columns]
```

This is interesting, when I originally explored the data, I worked with one season as the combined dataset had not been built yet. I also did not have the season factor incuded yet.

This updated dataset produced a filtered list of features with at east 10 more features than when when I previously explored the data and practiced training a model. I can also see some of the columns have NaN values so I will check to see how prevelent they are.

In [11]:
```
# Count how many NaN values there are in each column
nan_counts = players_filtered.isna().sum()

# Print the result
print(nan_counts)
```

```
season_factor        0
tmFactor           170
Age                  0
3B                   0
IBB                  0
GDP                  0
IFH                  0
BUH                  0
K%                   0
BB/K                 0
BABIP                0
GB/FB                0
IFFB%                0
HR/FB                0
IFH%                 0
BUH%                 0
Spd                  0
EV                 576
LA                 576
Barrel%            449
maxEV              576
HardHit%           449
O-Contact% (pi)    656
Z-Contact% (pi)    178
Zone% (pi)           5
Pace (pi)           49
dtype: int64
```

In [12]:
```
players_filtered.dropna(ignore_index = True)
```

Out[12]:

	season_factor	tmFactor	Age	3B	IBB	GDP	IFH	BUH	K%	BB/K	BABI
0	0.724469	0.603448	0.76	0	0	1	0	0	0.384615	0.000000	0.18181
1	0.724469	0.762931	0.46	0	0	1	0	0	0.000000	0.000000	0.00000
2	0.724469	0.806034	0.68	0	1	4	1	0	0.175115	0.842105	0.26470
3	0.724469	0.431034	0.76	1	2	19	7	0	0.084862	0.513514	0.31044
4	0.724469	0.560345	0.48	4	0	3	3	0	0.254658	0.390244	0.30107
...	...	...	...	...	...	...	...	...	...	...	
7463	0.804752	0.683544	0.58	5	0	3	3	0	0.212308	0.347826	0.26548
7464	0.804752	0.696203	0.46	1	1	14	9	1	0.232558	0.278571	0.29473
7465	0.804752	0.561181	0.60	1	0	1	1	0	0.256410	0.500000	0.34782
7466	0.804752	0.827004	0.48	2	0	7	8	1	0.343693	0.202128	0.29629
7467	0.804752	0.888905	0.58	0	0	0	1	1	0.272727	0.583333	0.19565

7468 rows × 26 columns

In [13]:
```
# Extract the column names from the dataframe
column_names = players_filtered.columns.tolist()

with open('../../data/lr_model1_feature_list.txt', 'w') as f:
    for column in column_names:
        f.write(column + '\n')
```

In [14]:
```
vif_threshold = 5

filtered_features = vif_data[vif_data["VIF"] < vif_threshold]["Feature"].tolist()

# Remove features with high VIF from X
players_filtered = players_df[filtered_features]

print(players_filtered)
```

```
      season_factor  tmFactor   Age  3B  IBB  GDP  BUH     GB/FB     IFFB%  \
0          0.724469  0.603448  0.76   0    0    1    0  5.000000  0.000000
1          0.724469  0.762931  0.46   0    0    1    0  1.000000  0.000000
2          0.724469  0.806034  0.68   0    1    4    0  1.240000  0.020000
3          0.724469  0.431034  0.76   1    2   19    0  1.626168  0.121495
4          0.724469  0.560345  0.48   4    0    3    0  1.054054  0.108108
...             ...       ...   ...  ..  ...  ...  ...       ...       ...
8495       0.804752  0.683544  0.58   5    0    3    0  0.773196  0.041237
8496       0.804752  0.696203  0.46   1    1   14    1  1.142857  0.090909
8497       0.804752  0.561181  0.60   1    0    1    0  1.571429  0.000000
8498       0.804752  0.827004  0.48   2    0    7    1  1.118644  0.110169
8499       0.804752  0.888905  0.58   0    0    0    1  0.583333  0.083333

         HR/FB      IFH%      BUH%       Spd         EV    maxEV  HardHit%  \
0     0.333333  0.000000  0.000000  0.142857  84.500738  104.322  0.218750
1     0.000000  0.000000  0.000000  0.142857  76.032501   82.962  0.000000
2     0.140000  0.016129  0.000000  1.080357  87.681620  108.544  0.328767
3     0.084112  0.040230  0.000000  1.736382  85.400034  109.962  0.254692
4     0.135135  0.076923  0.000000  8.036122  90.904075  110.026  0.434343
...        ...       ...       ...       ...        ...      ...       ...
8495  0.041237  0.040000  0.000000  8.182890  87.978473  108.147  0.326087
8496  0.149351  0.051136  0.125000  5.427539  88.491275  111.306  0.385749
8497  0.000000  0.090909  0.000000  6.500663  86.866048  105.293  0.291667
8498  0.144068  0.060606  0.166667  6.394459  89.197045  109.136  0.372240
8499  0.000000  0.071429  0.250000  4.777140  88.563941  108.933  0.265306

      Zone% (pi)  Pace (pi)
0       0.703911  22.119048
1       0.200000  20.250000
2       0.566955  22.139738
3       0.481092  22.454822
4       0.507837  20.627706
...          ...        ...
8495    0.521217  17.656489
8496    0.497160  18.721625
8497    0.589404  17.292035
8498    0.539659  18.735020
8499    0.542199  18.495050

[8500 rows x 18 columns]
```

In [15]:
```
players_filtered.dropna(ignore_index = True, inplace=True)
print(players_filtered)
```

```
      season_factor  tmFactor   Age  3B  IBB  GDP  BUH     GB/FB     IFFB%  \
0          0.724469  0.603448  0.76   0    0    1    0  5.000000  0.000000
1          0.724469  0.762931  0.46   0    0    1    0  1.000000  0.000000
2          0.724469  0.806034  0.68   0    1    4    0  1.240000  0.020000
3          0.724469  0.431034  0.76   1    2   19    0  1.626168  0.121495
4          0.724469  0.560345  0.48   4    0    3    0  1.054054  0.108108
...             ...       ...   ...  ..  ...  ...  ...       ...       ...
7779       0.804752  0.683544  0.58   5    0    3    0  0.773196  0.041237
7780       0.804752  0.696203  0.46   1    1   14    1  1.142857  0.090909
7781       0.804752  0.561181  0.60   1    0    1    0  1.571429  0.000000
7782       0.804752  0.827004  0.48   2    0    7    1  1.118644  0.110169
7783       0.804752  0.888905  0.58   0    0    0    1  0.583333  0.083333

         HR/FB      IFH%      BUH%       Spd         EV    maxEV  HardHit%  \
0     0.333333  0.000000  0.000000  0.142857  84.500738  104.322  0.218750
1     0.000000  0.000000  0.000000  0.142857  76.032501   82.962  0.000000
2     0.140000  0.016129  0.000000  1.080357  87.681620  108.544  0.328767
3     0.084112  0.040230  0.000000  1.736382  85.400034  109.962  0.254692
4     0.135135  0.076923  0.000000  8.036122  90.904075  110.026  0.434343
...        ...       ...       ...       ...        ...      ...       ...
7779  0.041237  0.040000  0.000000  8.182890  87.978473  108.147  0.326087
7780  0.149351  0.051136  0.125000  5.427539  88.491275  111.306  0.385749
7781  0.000000  0.090909  0.000000  6.500663  86.866048  105.293  0.291667
7782  0.144068  0.060606  0.166667  6.394459  89.197045  109.136  0.372240
7783  0.000000  0.071429  0.250000  4.777140  88.563941  108.933  0.265306

      Zone% (pi)  Pace (pi)
0       0.703911  22.119048
1       0.200000  20.250000
2       0.566955  22.139738
3       0.481092  22.454822
4       0.507837  20.627706
...          ...        ...
7779    0.521217  17.656489
7780    0.497160  18.721625
7781    0.589404  17.292035
7782    0.539659  18.735020
7783    0.542199  18.495050

[7784 rows x 18 columns]
C:\Users\nick_\AppData\Local\Temp\ipykernel_13716\1879401192.py:1: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  players_filtered.dropna(ignore_index = True, inplace=True)
```

This new list filtered to VIF values < 5 is a little bigger featurewise than what I ended up with while practicing as well.

In [16]:
```python
# Count how many NaN values there are in each column
nan_counts = players_filtered.isna().sum()

# Print the result
```

```
print(nan_counts)
```

```
season_factor    0
tmFactor         0
Age              0
3B               0
IBB              0
GDP              0
BUH              0
GB/FB            0
IFFB%            0
HR/FB            0
IFH%             0
BUH%             0
Spd              0
EV               0
maxEV            0
HardHit%         0
Zone% (pi)       0
Pace (pi)        0
dtype: int64
```

In [17]:
```
# Extract the column names from the dataframe
column_names = players_filtered.columns.tolist()

with open('../../data/lr_model2_feature_list.txt', 'w') as f:
    for column in column_names:
        f.write(column + '\n')
```

In [18]:
```
vif_threshold = 1

filtered_features = vif_data[vif_data["VIF"] <= vif_threshold]["Feature"].tolist()

# Remove features with high VIF from X
players_filtered = players_df[filtered_features]

print(players_filtered)
```

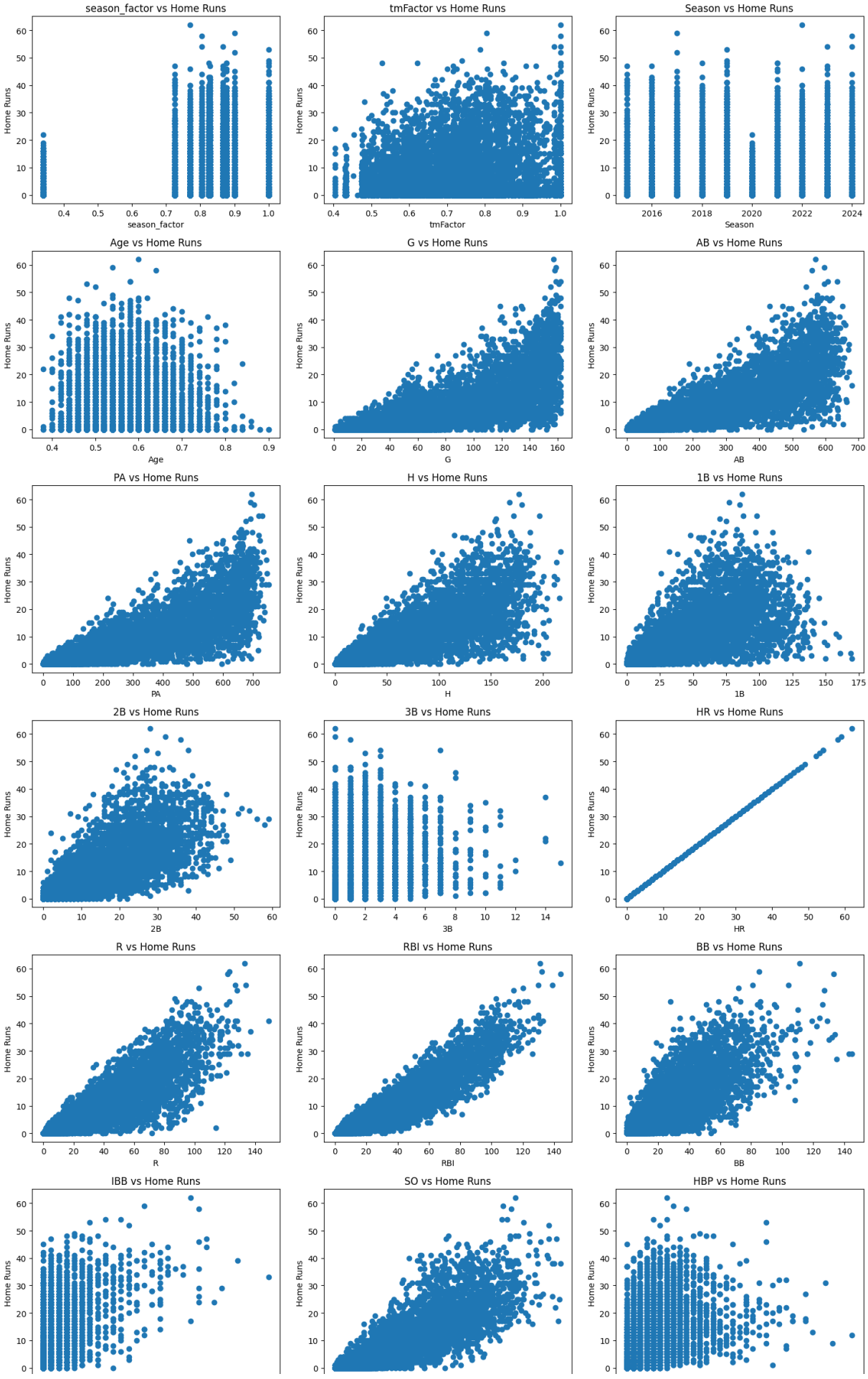
```
Empty DataFrame
Columns: []
Index: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21
, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42
, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63
, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84
, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, ...]

[8500 rows x 0 columns]
```

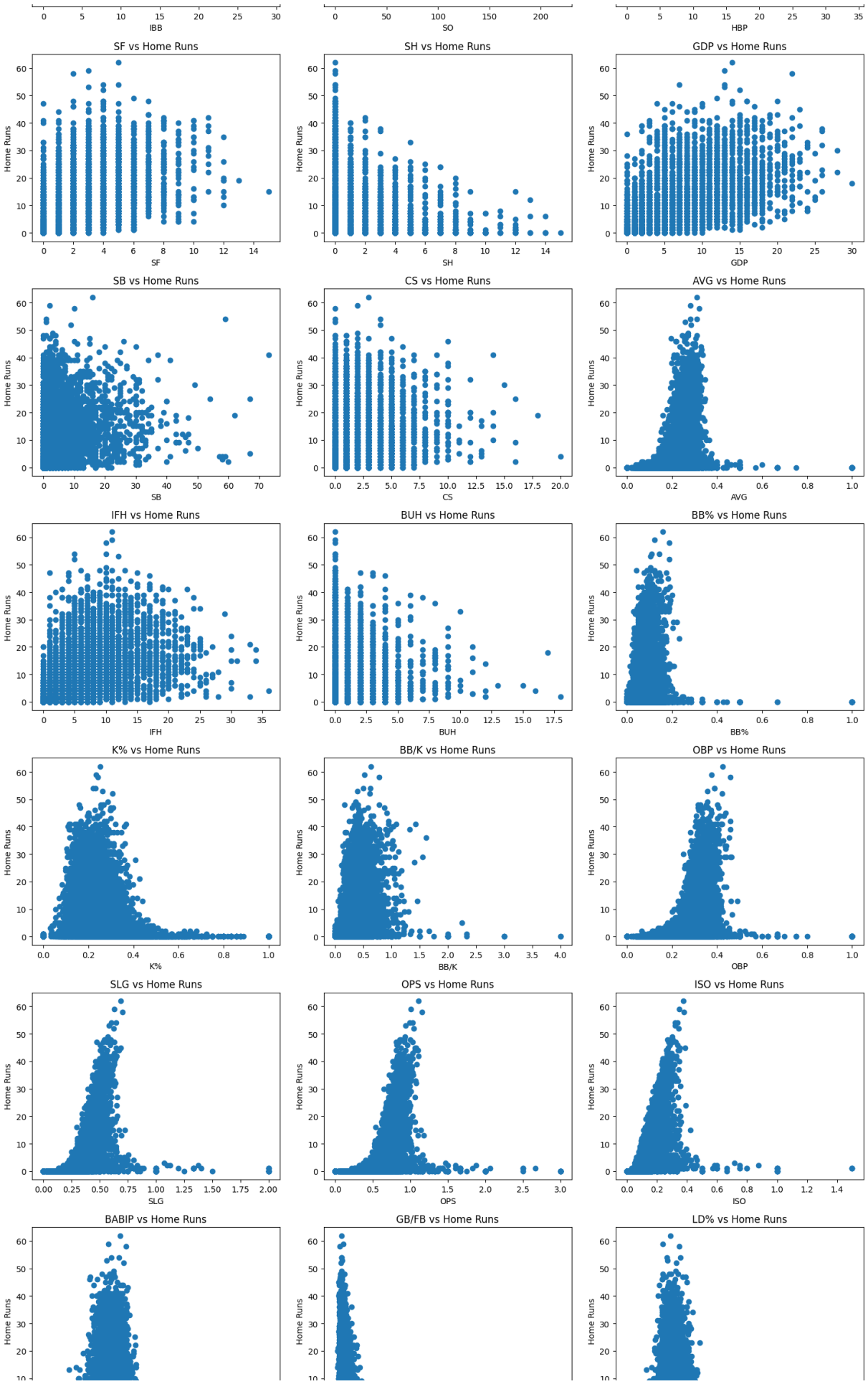
No columns have a vif less than one. So there will be no feature lists with this parameter.

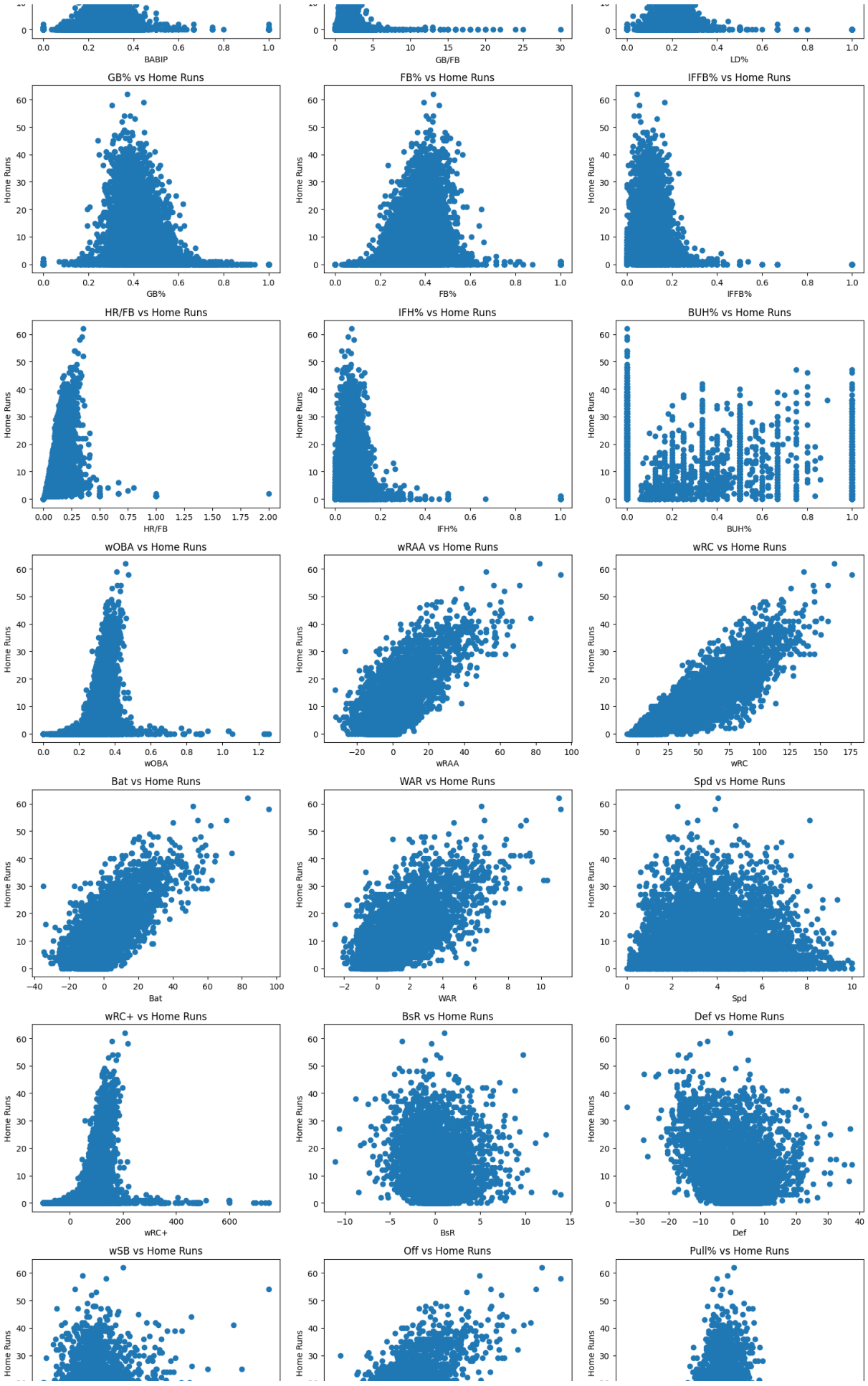
Lastly I will manually pick a feature list based on shapes of scatter plots and intuition based on familiarity with baseball to compare to the other

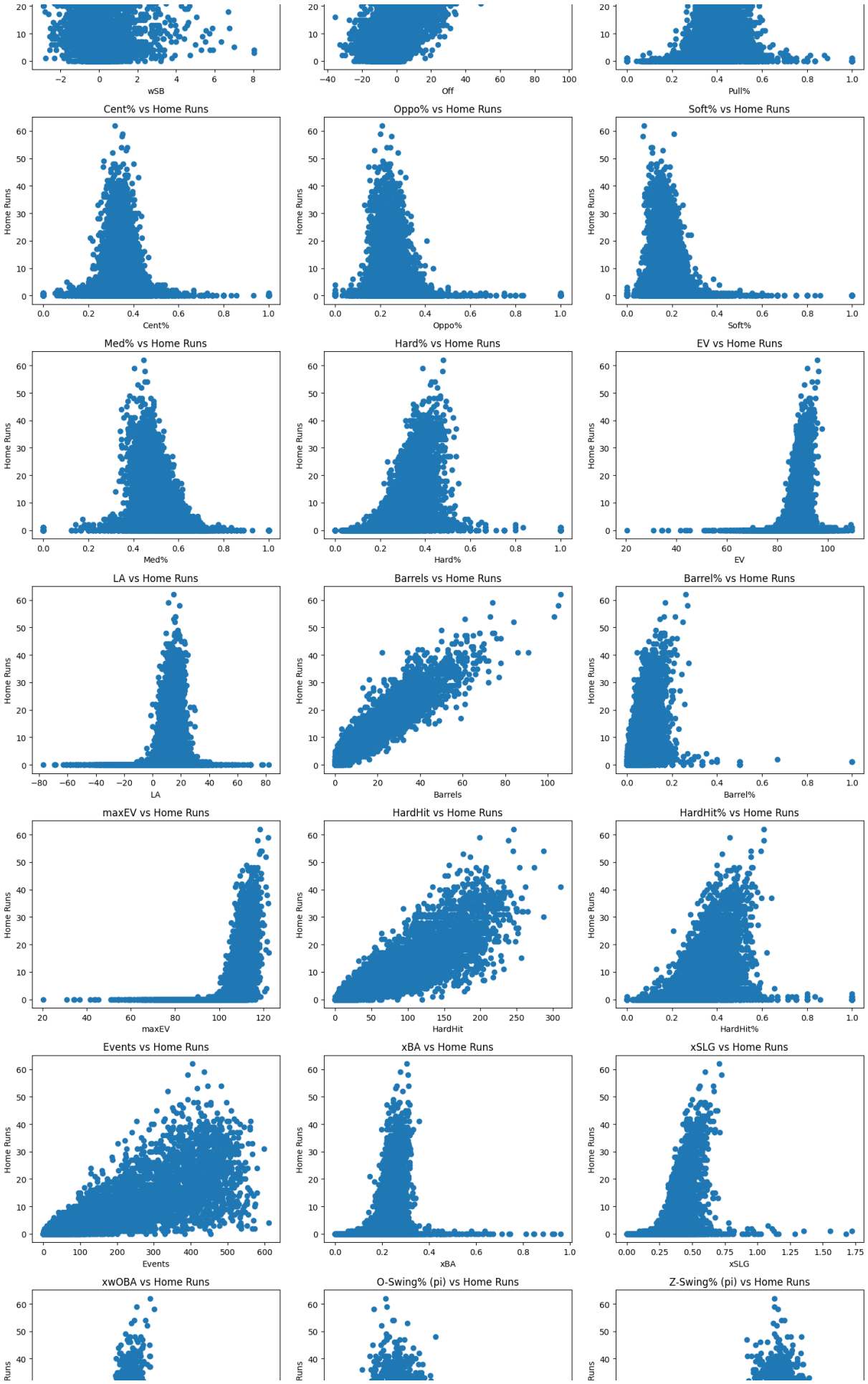
models.

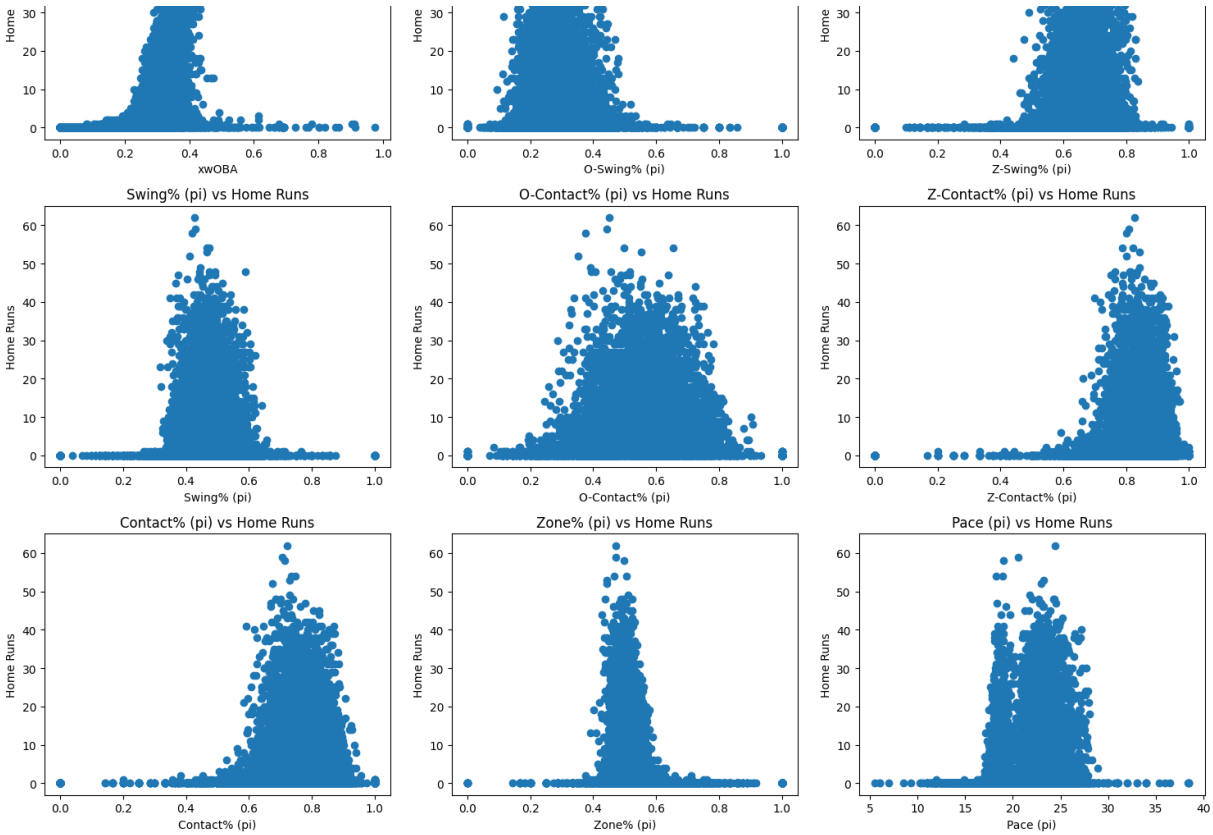
In [19]:
```
features = players_df.columns

n_features = len(features)

# Adjust number of columns per row here
cols = 3
rows = int(np.ceil(n_features / cols))

fig, axes = plt.subplots(nrows=rows, ncols=cols, figsize=(5*cols, 4*rows))
axes = axes.flatten()  # flatten in case it's 2D
y = players_df[target]
for i, feature in enumerate(features):
    x = players_df[feature]
    axes[i].scatter(x, y)


    axes[i].set_title(f"{feature} vs Home Runs")
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel("Home Runs")

# Turn off any extra unused subplots
for j in range(i+1, len(axes)):
    fig.delaxes(axes[j])

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


season_factor vs Home Runs
season_factor
tmFactor vs Home Runs
tmFactor
Season vs Home Runs
Season
Age vs Home Runs
Age
G vs Home Runs
G
AB vs Home Runs
AB
PA vs Home Runs
PA
H vs Home Runs
H
1B vs Home Runs
1B
2B vs Home Runs
2B
3B vs Home Runs
3B
HR vs Home Runs
HR
R vs Home Runs
R
RBI vs Home Runs
RBI
BB vs Home Runs
BB
IBB vs Home Runs
SO vs Home Runs
HBP vs Home Runs
Home Runs


IBB
SO
HBP
SF vs Home Runs
SH vs Home Runs
GDP vs Home Runs
SB vs Home Runs
CS vs Home Runs
AVG vs Home Runs
IFH vs Home Runs
BUH vs Home Runs
BB% vs Home Runs
K% vs Home Runs
BB/K vs Home Runs
OBP vs Home Runs
SLG vs Home Runs
OPS vs Home Runs
ISO vs Home Runs
BABIP vs Home Runs
GB/FB vs Home Runs
LD% vs Home Runs
Home Runs


BABIP
GB/FB
LD%
GB% vs Home Runs
FB% vs Home Runs
IFFB% vs Home Runs
Home Runs
GB%
FB%
IFFB%
HR/FB vs Home Runs
IFH% vs Home Runs
BUH% vs Home Runs
HR/FB
IFH%
BUH%
wOBA vs Home Runs
wRAA vs Home Runs
wRC vs Home Runs
wOBA
wRAA
wRC
Bat vs Home Runs
WAR vs Home Runs
Spd vs Home Runs
Bat
WAR
Spd
wRC+ vs Home Runs
BsR vs Home Runs
Def vs Home Runs
wRC+
BsR
Def
wSB vs Home Runs
Off vs Home Runs
Pull% vs Home Runs

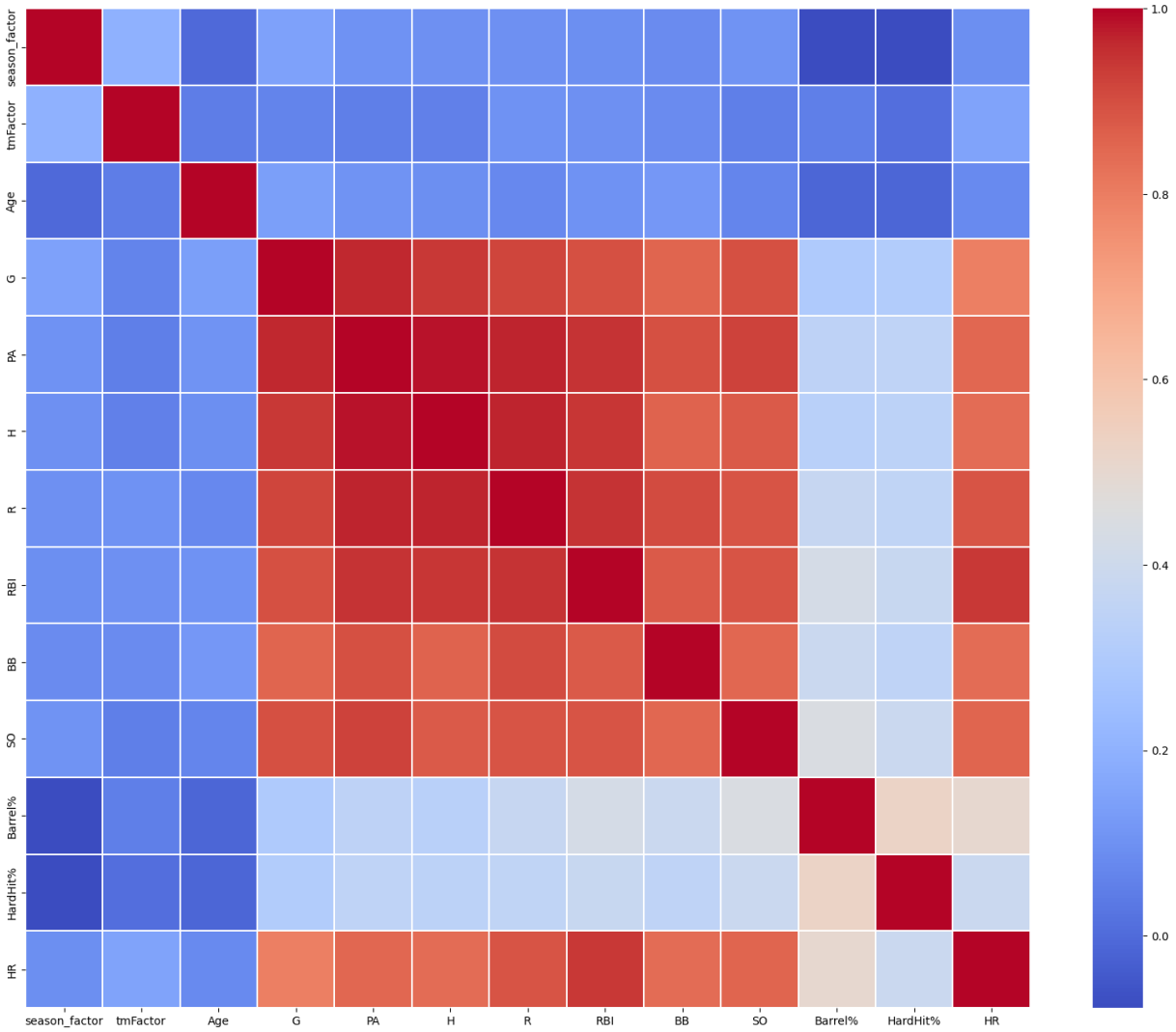

wSB
Off
Pull%
Cent% vs Home Runs
Cent%
Oppo% vs Home Runs
Oppo%
Soft% vs Home Runs
Soft%
Med% vs Home Runs
Med%
Hard% vs Home Runs
Hard%
EV vs Home Runs
EV
LA vs Home Runs
LA
Barrels vs Home Runs
Barrels
Barrel% vs Home Runs
Barrel%
maxEV vs Home Runs
maxEV
HardHit vs Home Runs
HardHit
HardHit% vs Home Runs
HardHit%
Events vs Home Runs
Events
xBA vs Home Runs
xBA
xSLG vs Home Runs
xSLG
xwOBA vs Home Runs
O-Swing% (pi) vs Home Runs
Z-Swing% (pi) vs Home Runs
Home Runs
Runs



Based on the shapes of the scatter plots above, selecting the following 'season_factor', 'tmFactor', 'Age', 'G', 'PA', 'H', 'R', 'RBI', 'BB', 'SO', 'Barrel%', 'HardHit%'

resetting the features dataframe and rerunning the VIF calculations with new manually selected feature list.

In [20]:
```
selected_columns = ['season_factor', 'tmFactor',  'Age', 'G', 'PA', 'H', 'R', 'RBI'
temp_players_df =  players_df[selected_columns].dropna(ignore_index=True)
correlation_matrix = temp_players_df.corr()
plt.figure(figsize=(20,16))
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm', fmt='.2f', linewidths
plt.show()
```



In [21]:
```
target = "HR"
target_corr = correlation_matrix[target].sort_values(ascending=False)

print("Feature Correlation with Target:")
print(target_corr.to_string())
```

```
Feature Correlation with Target:
HR               1.000000
RBI              0.938058
R                0.886952
SO               0.853256
PA               0.849976
H                0.837984
BB               0.837103
G                0.796386
Barrel%          0.502960
HardHit%         0.389474
tmFactor         0.152481
season_factor    0.092800
Age              0.078403
```

now I am going to manually select features to

remove until the remaining calculated VIFs are all in a range I want to try

starting by removing age as it does not seem to have any correlation to HRs hit. I think if you were able to model the data so it tracked each player specifically for multiple seasons, age might be more important.

In [22]:
```
selected_columns = ['season_factor', 'tmFactor', 'G', 'PA', 'H', 'R', 'RBI', 'BB',
features = players_df[selected_columns].dropna(ignore_index=True)

vif_data = pd.DataFrame()
vif_data["Feature"] = features.columns
vif_data["VIF"] = [variance_inflation_factor(features.values, i) for i in range(fea

print("Variance Inflation Factor (VIF):")
print(vif_data)
```

```
Variance Inflation Factor (VIF):
          Feature         VIF
0   season_factor   21.464167
1        tmFactor   20.871046
2               G   41.900047
3              PA  238.552979
4               H  137.827594
5               R   47.694424
6             RBI   22.139597
7              BB   13.548935
8              SO   21.665536
9         Barrel%    3.139142
10       HardHit%    5.770173
```

next going to try removing PA du to the high colinearity

In [23]:
```
selected_columns = ['season_factor', 'tmFactor', 'G', 'H', 'R', 'RBI', 'BB', 'SO',
features = players_df[selected_columns].dropna(ignore_index=True)

vif_data = pd.DataFrame()
vif_data["Feature"] = features.columns
vif_data["VIF"] = [variance_inflation_factor(features.values, i) for i in range(fea

print("Variance Inflation Factor (VIF):")
print(vif_data)
```

```
Variance Inflation Factor (VIF):
         Feature        VIF
0  season_factor  21.443798
1       tmFactor  20.858772
2              G  32.203586
3              H  52.974553
4              R  47.523235
5            RBI  22.136968
6             BB  11.013715
7             SO  15.476471
8        Barrel%   3.117753
9       HardHit%   5.765888
```

Next trying to remove team factor

In [24]:
```
selected_columns = ['season_factor', 'G', 'H', 'R', 'RBI', 'BB', 'SO', 'Barrel%', '
features = players_df[selected_columns].dropna(ignore_index=True)

vif_data = pd.DataFrame()
vif_data["Feature"] = features.columns
vif_data["VIF"] = [variance_inflation_factor(features.values, i) for i in range(fea

print("Variance Inflation Factor (VIF):")
print(vif_data)
```

```
Variance Inflation Factor (VIF):
         Feature        VIF
0  season_factor   5.725218
1              G  31.060645
2              H  51.474321
3              R  46.825087
4            RBI  22.019191
5             BB  10.989323
6             SO  15.065417
7        Barrel%   2.956544
8       HardHit%   5.332124
```

This did not change much, so I will add it back in and try removing hits instead

In [25]:
```
selected_columns = ['season_factor', 'tmFactor', 'G', 'R', 'RBI', 'BB', 'SO', 'Barr
features = players_df[selected_columns].dropna(ignore_index=True)

vif_data = pd.DataFrame()
vif_data["Feature"] = features.columns
vif_data["VIF"] = [variance_inflation_factor(features.values, i) for i in range(fea

print("Variance Inflation Factor (VIF):")
print(vif_data)
```

```
Variance Inflation Factor (VIF):
         Feature        VIF
0  season_factor  21.443777
1       tmFactor  20.531025
2              G  24.017653
3              R  27.110660
4            RBI  19.151925
5             BB  10.163375
6             SO  15.183544
7        Barrel%   3.040916
8       HardHit%   5.689859
```

Removing hits seemed to drop runs VIF considerably. going to drop runs next

In [26]:
```
selected_columns = ['season_factor', 'tmFactor', 'G', 'RBI', 'BB', 'SO', 'Barrel%',
features = players_df[selected_columns].dropna(ignore_index=True)

vif_data = pd.DataFrame()
vif_data["Feature"] = features.columns
vif_data["VIF"] = [variance_inflation_factor(features.values, i) for i in range(fea

print("Variance Inflation Factor (VIF):")
print(vif_data)
```

```
Variance Inflation Factor (VIF):
         Feature        VIF
0  season_factor  21.194453
1       tmFactor  20.489280
2              G  21.409588
3            RBI  13.238839
4             BB   8.549044
5             SO  15.138562
6        Barrel%   3.030780
7       HardHit%   5.689733
```

RBI is directly affected by homeruns, as each homeruns gives a player at least one rbi. I am going to try and combine RBIs and Games and see how that affects the VIF values.

In [27]:
```
features['RBI'] = features['RBI']/features['G']
features.rename(columns={'RBI': 'RBI/G'}, inplace=True)
vif_data["Feature"] = features.columns
vif_data["VIF"] = [variance_inflation_factor(features.values, i) for i in range(fea

print("Variance Inflation Factor (VIF):")
print(vif_data)
```

```
Variance Inflation Factor (VIF):
         Feature        VIF
0  season_factor  20.872755
1       tmFactor  20.977235
2              G  17.867300
3          RBI/G   5.875225
4             BB   7.534300
5             SO  14.586034
6        Barrel%   3.256552
7       HardHit%   5.942025
```

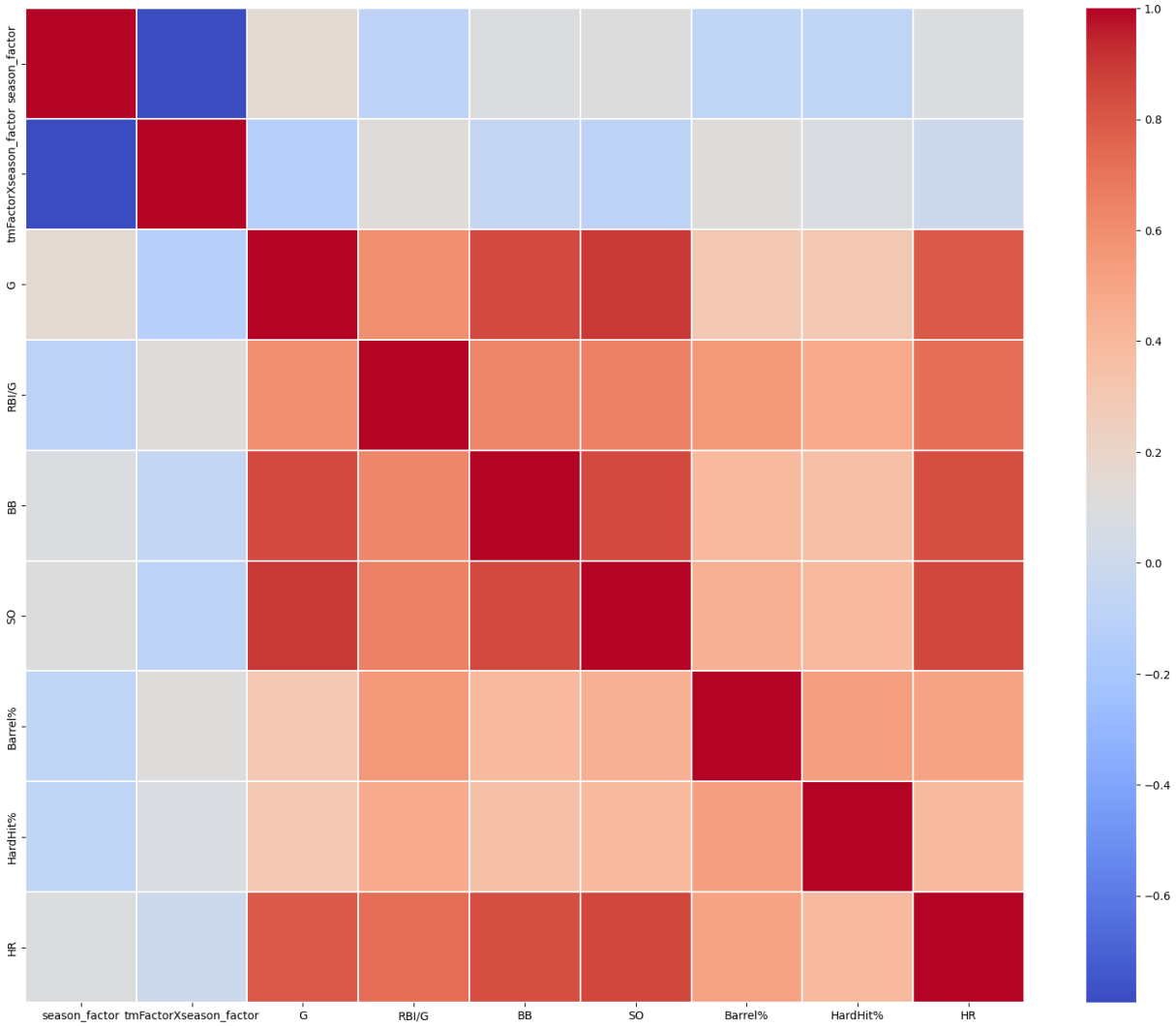
combining those did a lot to drop the VIF in comparison to RBI. I'm going to try and combine tmFactor and Season_factor by multiplying them.

In [28]:
```
features['tmFactor'] = features['tmFactor']/features['season_factor']
features.rename(columns={'tmFactor': 'tmFactorXseason_factor'}, inplace=True)

vif_data["Feature"] = features.columns
vif_data["VIF"] = [variance_inflation_factor(features.values, i) for i in range(fea

print("Variance Inflation Factor (VIF):")
print(vif_data)
```

```
Variance Inflation Factor (VIF):
                  Feature        VIF
0           season_factor   6.880181
1  tmFactorXseason_factor   5.428480
2                       G  17.882291
3                  RBI/G   6.047049
4                      BB   7.551205
5                      SO  14.642628
6                 Barrel%   3.253290
7                HardHit%   6.153749
```

I think this is as close as I want to get. without something to indicate the amount of opportunities a player gets to hit a home rune it feels like this would not work. going to rebuild list with new combined features and run one last correlation matrix.

In [29]:
```
selected_columns = ['season_factor', 'tmFactor', 'G', 'RBI', 'BB', 'SO', 'Barrel%',
temp_players_df =  players_df[selected_columns].dropna(ignore_index=True)

temp_players_df['tmFactor'] = temp_players_df['tmFactor']/temp_players_df['season_f
temp_players_df.rename(columns={'tmFactor': 'tmFactorXseason_factor'}, inplace=True

temp_players_df['RBI'] = temp_players_df['RBI']/temp_players_df['G']
temp_players_df.rename(columns={'RBI': 'RBI/G'}, inplace=True)

correlation_matrix = temp_players_df.corr()
plt.figure(figsize=(20,16))
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm', fmt='.2f', linewidths
plt.show()
```



In [30]:
```
target = "HR"
target_corr = correlation_matrix[target].sort_values(ascending=False)

print("Feature Correlation with Target:")
print(target_corr.to_string())
```

```
Feature Correlation with Target:
HR                          1.000000
SO                          0.853256
BB                          0.837103
G                           0.796386
RBI/G                       0.725833
Barrel%                     0.502960
HardHit%                    0.389474
season_factor               0.092800
tmFactorXseason_factor     -0.005657
```

In [31]:
```
final_features = temp_players_df.drop(columns=['season_factor', 'tmFactorXseason_fa
final_features_list = final_features.columns

vif_data = pd.DataFrame()
vif_data["Feature"] = final_features_list
vif_data["VIF"] = [variance_inflation_factor(final_features.values, i) for i in ran
```

```
print("Variance Inflation Factor (VIF):")
print(vif_data)
```

```
Variance Inflation Factor (VIF):
    Feature        VIF
0         G  14.019825
1     RBI/G   5.708492
2        BB   7.056541
3        SO  13.938472
4   Barrel%   3.249725
5  HardHit%   4.239572
```

In [32]:
```
# Extract the column names from the dataframe
column_names = players_filtered.columns.tolist()

with open('../../data/lr_model_manual_feature_list.txt', 'w') as f:
    for column in column_names:
        f.write(column + '\n')
```