Introduction and EDA

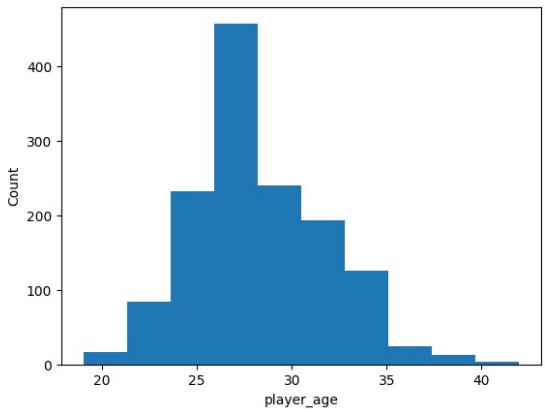
```
'''
The topic for this project will be predicting the number of home runs a player will hit in a season of
MLB. The dataset I am using for this project has been collected from
https://baseballsavant.mlb.com/leaderboard/statcast, and includes a number of
traditional baseball states (batting average, at bats, hits) as well as many newer advanced stats (average exit
velocity, hard hit percentage, launch angle). The data is readily available, and easily downloaded to a .csv file.
For my data, I am usuing statistics gathered during the 2018 through 2023 seasons (omitting shortened 2020 season, 2023
statistics through 06/23/2023). This will be a regression problem, and I plan to use
random forest to boost my regression. The goal of the project would be to predict the number of home runs a player
would hit given his other statistics.
'''

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import math
import pylab
import seaborn as sns
import statsmodels.formula.api as smf
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

'''
Importing data and inspecting data.
'''
df = pd.read_csv('stats.csv')
df.info()
df.describe()
```

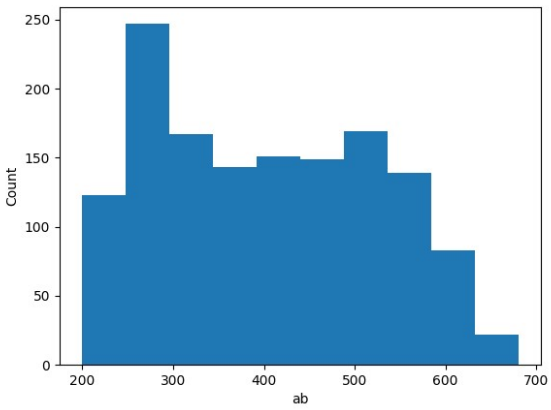
```
>>> df.info()
<class 'pandas.core.frame.DataFrame'>
Index: 1210 entries, 0 to 1390
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   home_run              1210 non-null   int64
 1   player_age            1210 non-null   int64
 2   ab                    1210 non-null   int64
 3   hit                   1210 non-null   int64
 4   single                1210 non-null   int64
 5   double                1210 non-null   int64
 6   triple                1210 non-null   int64
 7   batting_avg           1210 non-null   float64
 8   exit_velocity_avg     1210 non-null   float64
 9   launch_angle_avg      1210 non-null   float64
 10  sweet_spot_percent    1210 non-null   float64
 11  barrel_batted_rate    1210 non-null   float64
 12  solidcontact_percent  1210 non-null   float64
 13  flareburner_percent   1210 non-null   float64
 14  poorlyunder_percent   1210 non-null   float64
 15  poorlytopped_percent  1210 non-null   float64
 16  hard_hit_percent      1210 non-null   float64
 17  avg_best_speed        1210 non-null   float64
dtypes: float64(11), int64(7)
memory usage: 179.6 KB
```

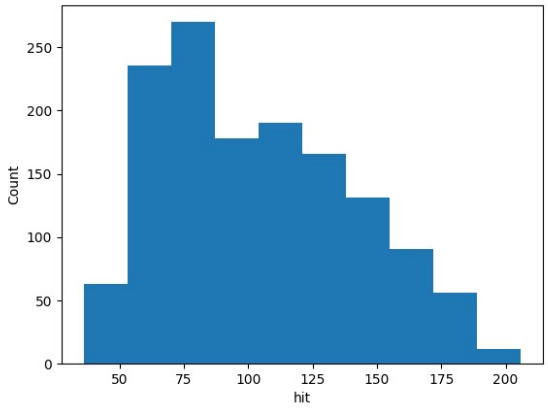
```
>>> df.describe()
          home_run   player_age  ...  hard_hit_percent  avg_best_speed
count  1210.000000  1210.000000  ...       1210.000000     1210.000000
mean     16.628926    28.238843  ...         38.874959       99.673480
std       9.815940     3.643317  ...          7.582779        2.517789
min       0.000000    19.000000  ...          7.300000       89.217314
25%       9.000000    26.000000  ...         34.200000       98.058494
50%      15.000000    28.000000  ...         39.300000       99.754583
75%      22.000000    31.000000  ...         43.700000      101.251303
max      62.000000    42.000000  ...         61.800000      108.657831

[8 rows x 18 columns]
```

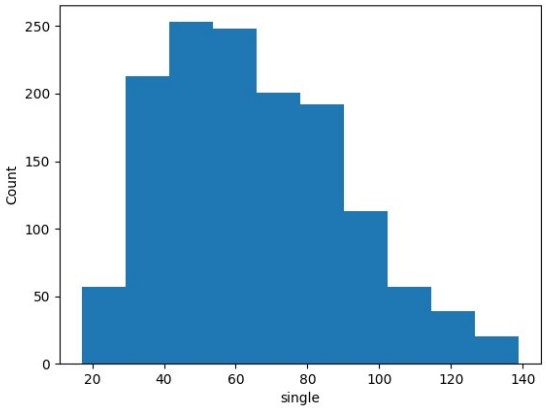
```
'''
Inspection shows that I have 22 columns (features), the majority of which are int
data types or float data types. Two columns are object data type, these are the players names. I have 1897
rows (samples) in most columns. Also reveals four columns that can be removed. player_id is a reference column used
by baseballsavant.com, and year, first_name, and last_name won't be needed.
'''

df = df.drop('year', axis=1)
df = df.drop('player_id', axis=1)
df = df.drop(' first_name', axis=1)
df = df.drop('last_name', axis=1)

'''
Checking for null values.
'''

null_counts = pd.isnull(df).sum()
df = df.dropna()

'''
Checking revealed a very small amount. Since my dataset is large enough, I will
just drop these rows from the dataset.
'''
```

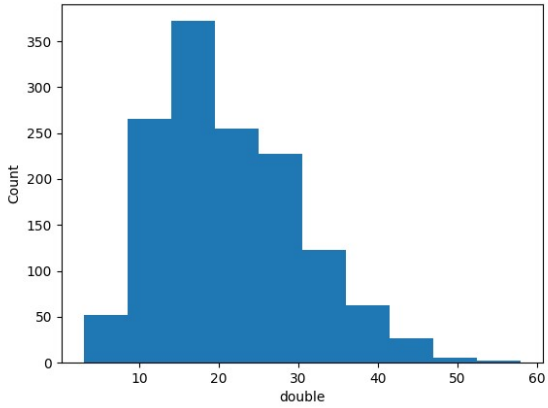
```
'''
Visualizing and describing data. Histogram of each feature created
Moving home runs (my target) to the first column
'''

features = list(df.columns)
neworder = ['home_run', 'player_age', 'ab', 'hit', 'single', 'double', 'triple',
            'batting_avg', 'exit_velocity_avg', 'launch_angle_avg', 'sweet_spot_percent', 'barrel_batted_rate',
            'solidcontact_percent', 'flareburner_percent', 'poorlyunder_percent', 'poorlytopped_percent',
            'hard_hit_percent', 'avg_best_speed']
df=df[neworder]

for item in features:
    plt.hist(df[item])
    plt.xlabel(item)
    plt.ylabel('Count')
    plt.show()

'''
home_run - what I am trying to find
player_age - Age of each player in the given year. Will be interesting to see if correlated with HR
ab - At bats in the season. Should be somewhat correlated with HR since you need to have at bats to hit HR
    Very large percentage of the data appears to be at the very low (<300) end of thd data.
    Will be droping samples with very low 'ab' so they don't distort data.
hit - number of hits in the year
single - number of singles in the year
double - number of doubles in the year
triple - number of triples in the year
batting_avg - number of hits / number of at bats
exit_velocity_avg - average speed of the baseball off of the players bat
launch_angle_avg - angle that the ball leaves the bat from. I expect this to be highly correlated with HR
sweet_spot_percent - percent of batted balls with a launch angle between 8 and 32 degrees. Should be highly
    correlated with HR
barrel_batted_rate - percentage of batted balls with a launch angle and exit velocity combination have led to
    a minimum .500 batting average and 1.500 slugging percentage since Statcast was implemented
    Major League wide in 2015.
solidcontact_percent - percentage of batted balls deemed to have solid contact
flareburner_percent - percentage of batted balls that are burners (hard ground balls) or flares (short pop flies)
poorlyunder_percent - percentage of batted balls that are weak pop ups
poorlytopped_percent - percentage of batted balls that are weak ground balls (impossible to be a HR)
hard_hit_percent - percentage of batted balls with exit velocity over 95 mph
avg_best_speed - average of 50% of players hardest hit balls
'''
```

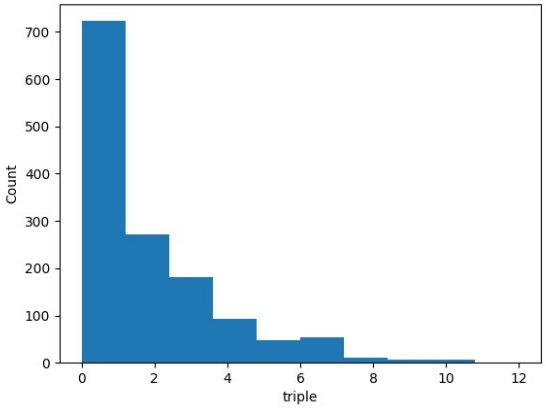


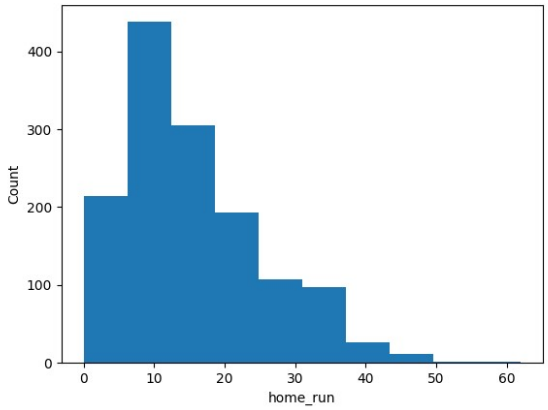


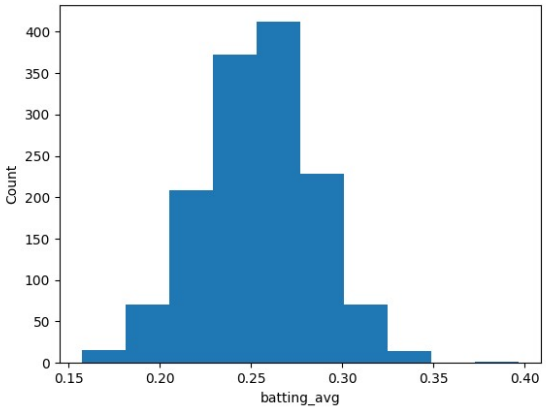

Count
hit
0
50
100
150
200
250
50
75
100
125
150
175
200

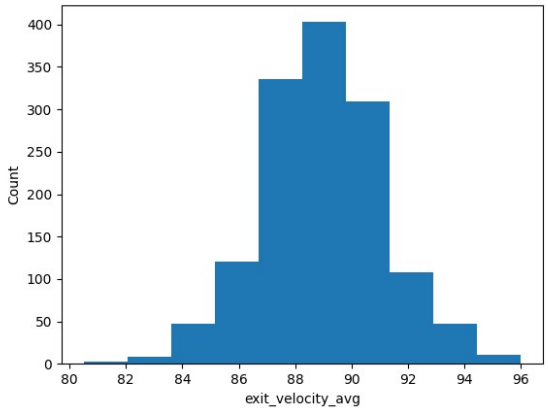

Count
single
0
50
100
150
200
250
20
40
60
80
100
120
140

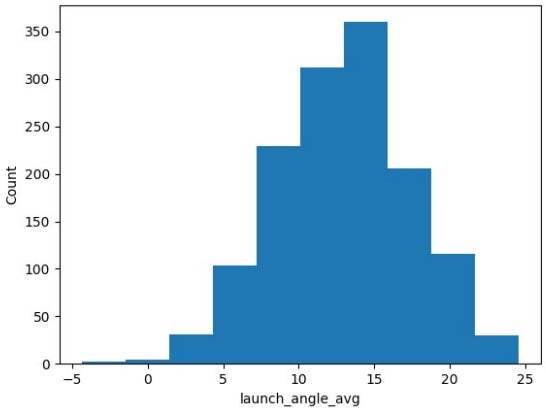

Count
double
0
50
100
150
200
250
300
350
10
20
30
40
50
60

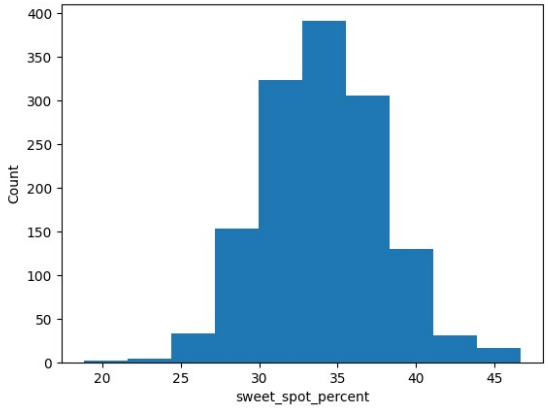

Count
triple
0
100
200
300
400
500
600
700
0
2
4
6
8
10
12

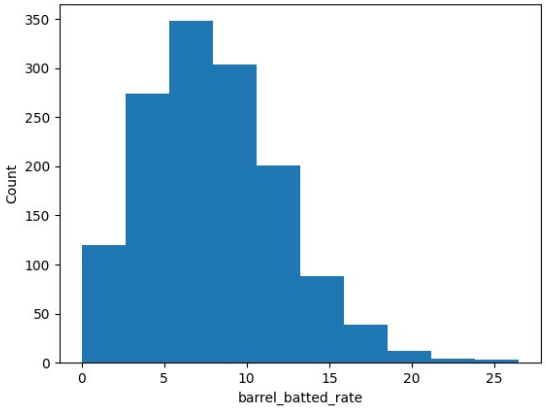

Count
home_run
0
100
200
300
400
0
10
20
30
40
50
60

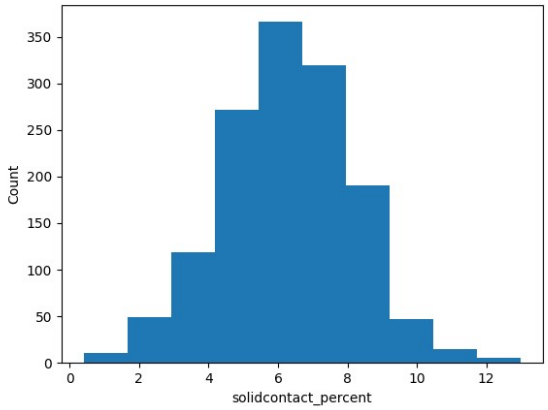

Count
batting_avg
0
50
100
150
200
250
300
350
400
0.15
0.20
0.25
0.30
0.35
0.40

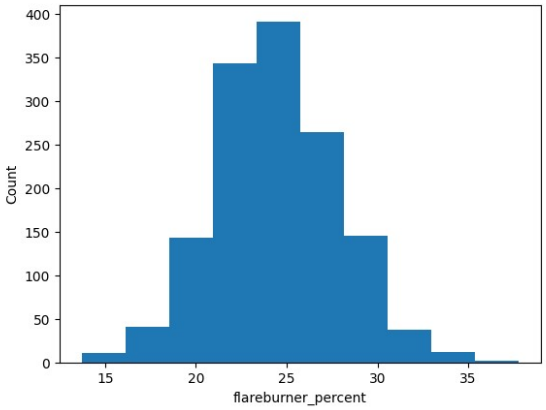

Count
exit_velocity_avg

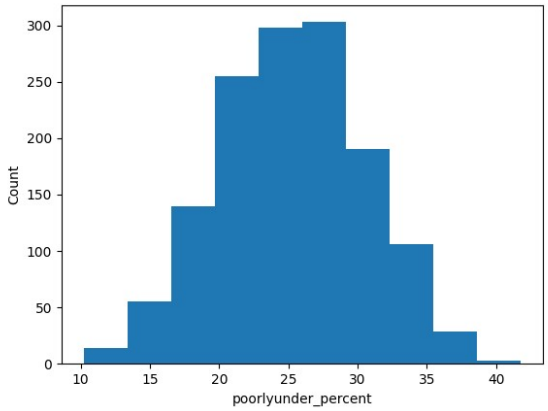

Count
launch_angle_avg

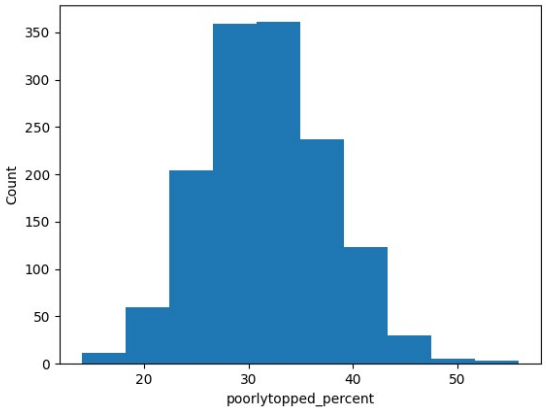

Count
sweet_spot_percent

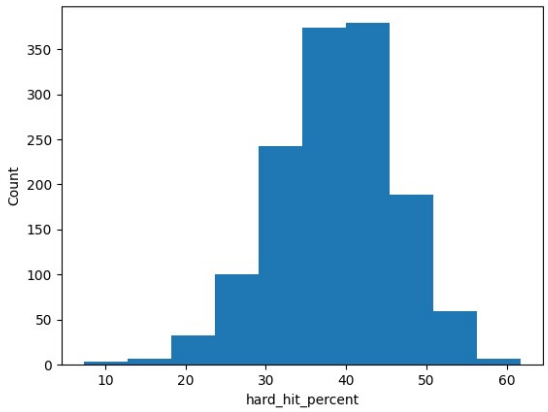

Count
barrel_batted_rate

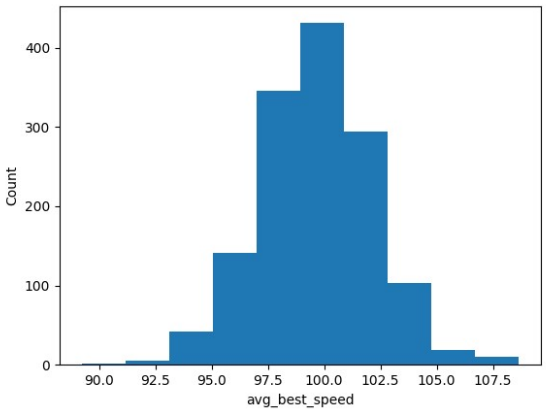

Count
solidcontact_percent


Count
flareburner_percent

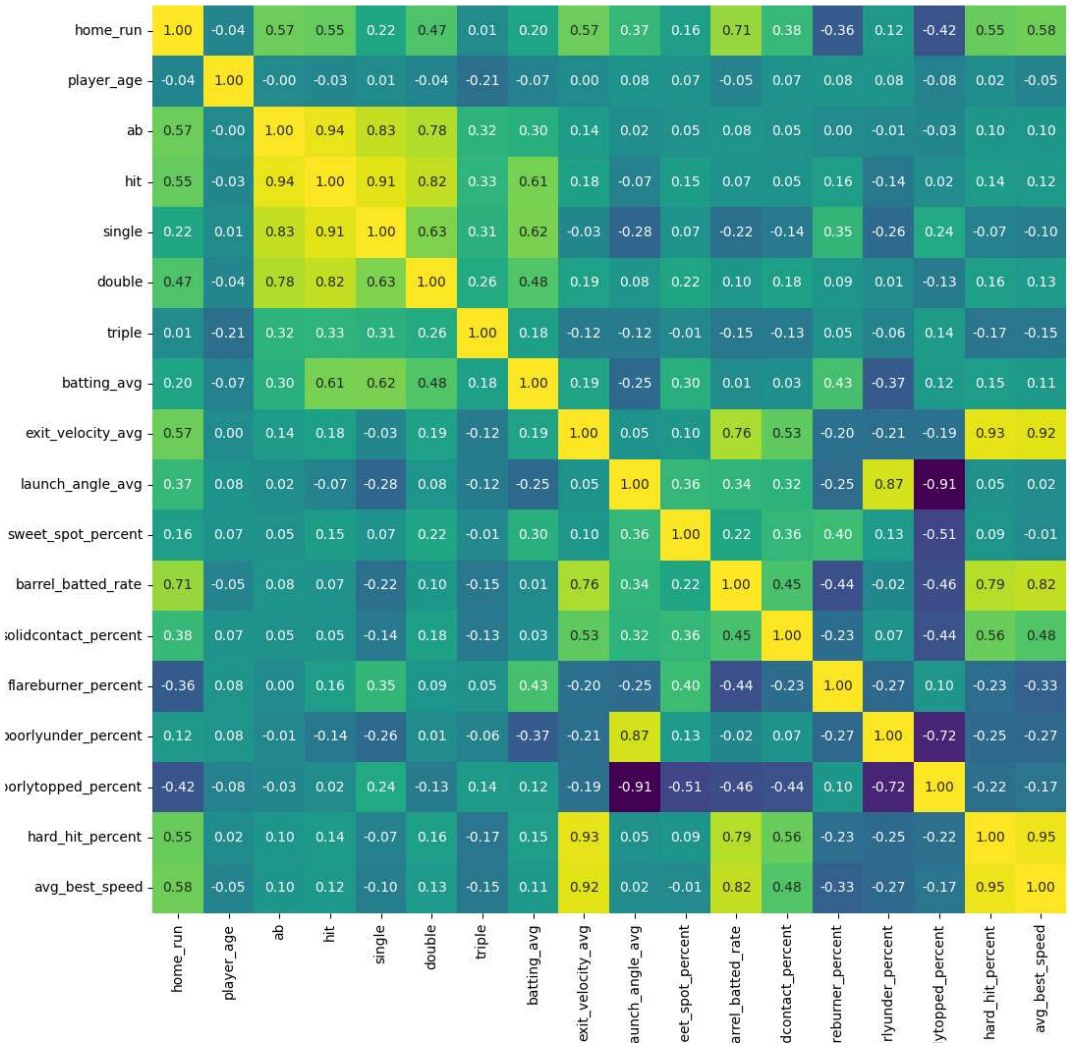

Count
poorlyunder_percent


Count
poorlytopped_percent


Count
hard_hit_percent


Count
avg_best_speed

```
'''
Removing rows of players with less than 260 at bats in a season from the training data. This is a bit arbitrary,
but after 130 at bats you are no longer considered a rookie in MLB, and I needed a cutoff to remove players
with very few at bats whose data could skew the results, so I doubled that number.
After dropping these samples, the dataset now contains 1210 samples
'''

df = df.drop(df[df['ab'] < 260].index)

'''
Making correlation matrix
'''

corr = df.corr()

plt.figure(figsize=(12, 12))
sns.heatmap(corr, cmap="viridis", annot=True, cbar=False, fmt='.2f')
plt.show()
```



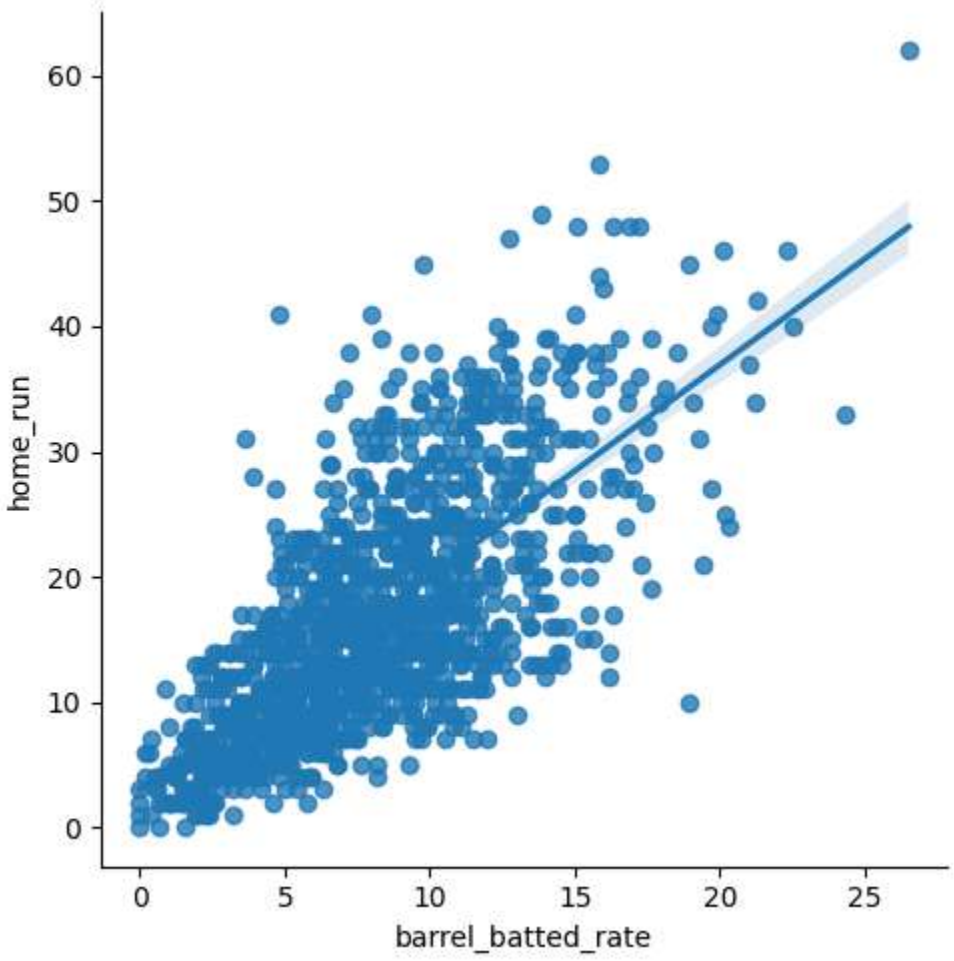
```
'''
Best predictor of home runs based on the correlation matrix is 'barrel_batted_rate', with the next bests being
'avg_best_speed', 'hit', and 'avg_exit_velocity'. 'barrel_batted_rate' is highly correlated with 'avg_best_speed' and
'avg_exit_velocity', so those are not surprising. 'ab' is not correlated strongly with 'barrel_batted_rate'
so that is interesting.
'''
```

Analysis and Results

```
'''
Splitting data to run a simple linear regression model.
'''

X_train, X_test = train_test_split(df, test_size=0.20, random_state=0)
model = smf.ols(formula='home_run ~ barrel_batted_rate', data=df)
res = model.fit()
print(res.summary())

'''
Adj R squared of only 0.501 on the best guess predictor
of 'barrel_batted_rate'
'''
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               home_run   R-squared:                       0.501
Model:                            OLS   Adj. R-squared:                  0.501
Method:                 Least Squares   F-statistic:                     1214.
Date:                Tue, 27 Jun 2023   Prob (F-statistic):          9.86e-185
Time:                        11:34:17   Log-Likelihood:                -4059.1
No. Observations:                1210   AIC:                             8122.
Df Residuals:                    1208   BIC:                             8132.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept          3.0354      0.438      6.929      0.000       2.176       3.895
barrel_batted_rate     1.6948      0.049     34.848      0.000       1.599       1.790
==============================================================================
Omnibus:                       65.401   Durbin-Watson:                   1.688
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               79.840
Skew:                           0.533   Prob(JB):                     4.60e-18
Kurtosis:                       3.669   Cond. No.                         20.0
==============================================================================

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

```
'''
Plot of 'barrel_batted_rate' vs 'home_run' with linear regression line
'''

sns.lmplot(x='barrel_batted_rate',y='home_run', data=df)
plt.show()
```




```
'''
Calculating MSE for model. Creating lists to store future results for later analysis
'''
Models = []
Mean_Squared_Errors = []

y_pred = res.predict(X_test)
y_true = X_test['home_run'].values
MSElinear = mean_squared_error(y_true, y_pred)

Models.append('Linear Regression')
Mean_Squared_Errors.append(MSElinear)

'''
MSE of 53.13
'''
```

```
'''
Testing all other features as predictor. 'barrel_batted_rate' was the best predictor
'''

headers = list(df.columns)
headers = headers[1:]
best_predictor = ''
best_r_squared = 0

for item in headers:
    test = item
    model = smf.ols(formula='home_run ~ ' + test, data=df)
    res = model.fit()
    r_squared = res.rsquared

    if r_squared > best_r_squared:
        best_predictor = item
        best_r_squared = r_squared

print(best_predictor, best_r_squared)
```

```
... print(best_predictor, best_r_squared)
barrel_batted_rate 0.5013218973828548
```

```
'''
Testing polynomial regression up to N=10.
'''
# return updated best_degree and best_r_squared
best_degree = 1

formulaString = 'home_run ~ barrel_batted_rate'
for n in range(2, 11):
    formulaString = formulaString + ' + np.power(barrel_batted_rate,' + str(n) + ')'
    model = smf.ols(formula=formulaString, data=df)
    res = model.fit()
    r_squared = res.rsquared

    if r_squared > best_r_squared:
        best_degree = n
        best_r_squared = r_squared

print(best_degree, best_r_squared)

'''
Best R squared was at degree 10. Adj R2 of 0.507
'''
```

```
... print(best_degree, best_r_squared)
10 0.5076504871250251
```

```
'''
MSE of polynomial regression
'''

model = smf.ols('home_run ~ barrel_batted_rate + np.power(barrel_batted_rate,2) +'
                'np.power(barrel_batted_rate,3) + np.power(barrel_batted_rate,4) +'
                'np.power(barrel_batted_rate,5) + np.power(barrel_batted_rate,6) +'
                'np.power(barrel_batted_rate,7) + np.power(barrel_batted_rate,8) +'
                'np.power(barrel_batted_rate,9) + np.power(barrel_batted_rate,10)', data=df).fit()

y_pred = model.predict(X_test)
y_true = X_test['home_run'].values
MSEpoly = mean_squared_error(y_true, y_pred)

Models.append('Polynomial Regression')
Mean_Squared_Errors.append(MSEpoly)

'''
MSE of 53.568
```

```
'''
Multi-linear regression.
'''

header_string = ''
for index, x in enumerate(headers):
    if index == 0:
        header_string += x
    else:
        header_string += ' + ' + x

model = smf.ols('home_run ~ ' + header_string, data=df).fit()
print(model.summary())

'''
Found a problem right away. Hits - (singles + doubles + triples) = home runs, so
everything else has terrible P values. Dropping those 4 features and re running.
'''
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               home_run   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                5.835e+30
Date:                Tue, 27 Jun 2023   Prob (F-statistic):              0.00
Time:                        11:44:04   Log-Likelihood:                 35808.
No. Observations:                1210   AIC:                         -7.158e+04
Df Residuals:                    1192   BIC:                         -7.149e+04
Df Model:                          17
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
Intercept           1.368e-13    1.6e-13      0.855      0.393   -1.77e-13    4.51e-13
player_age         -1.407e-15   2.88e-16     -4.885      0.000   -1.97e-15   -8.42e-16
ab                 -1.782e-16   8.01e-17     -2.225      0.026   -3.35e-16   -2.11e-17
hit                    1.0000   3.71e-16   2.69e+15      0.000       1.000       1.000
single                -1.0000   2.39e-16  -4.18e+15      0.000      -1.000      -1.000
double                -1.0000   2.94e-16  -3.41e+15      0.000      -1.000      -1.000
triple                -1.0000   5.88e-16   -1.7e+15      0.000      -1.000      -1.000
batting_avg         1.066e-13    1.3e-13      0.822      0.411   -1.48e-13    3.61e-13
exit_velocity_avg   1.357e-15   1.54e-15      0.879      0.380   -1.67e-15    4.39e-15
launch_angle_avg   -8.708e-16   9.34e-16     -0.932      0.351    -2.7e-15    9.62e-16
```

```python
headers.remove('hit')
headers.remove('single')
headers.remove('double')
headers.remove('triple')

header_string = ''
for index, x in enumerate(headers):
    if index == 0:
        header_string += x
    else:
        header_string += ' + ' + x

model = smf.ols('home_run ~ ' + header_string, data=df).fit()
print(model.summary())

'''
After removing hit, single, double, & triple, got a model with an adjusted R quared of 0.812, and 8 significant
features based off p values < 0.05
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               home_run   R-squared:                       0.814
Model:                            OLS   Adj. R-squared:                  0.812
Method:                 Least Squares   F-statistic:                     401.7
Date:                Tue, 27 Jun 2023   Prob (F-statistic):               0.00
Time:                        11:45:44   Log-Likelihood:                -3463.5
No. Observations:                1210   AIC:                             6955.
Df Residuals:                    1196   BIC:                             7026.
Df Model:                          13
Covariance Type:            nonrobust
===========================================================================================
                              coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------------
Intercept                -69.8111     19.428     -3.593      0.000    -107.928     -31.694
player_age                 0.0030      0.035      0.086      0.932      -0.065       0.071
ab                         0.0417      0.001     34.865      0.000       0.039       0.044
batting_avg               56.9721      5.234     10.885      0.000      46.703      67.241
exit_velocity_avg          0.1515      0.191      0.795      0.427      -0.223       0.526
launch_angle_avg           0.6895      0.114      6.032      0.000       0.465       0.914
sweet_spot_percent        -0.2071      0.056     -3.673      0.000      -0.318      -0.096
barrel_batted_rate         1.3036      0.126     10.362      0.000       1.057       1.550
solidcontact_percent       0.2910      0.138      2.106      0.035       0.020       0.562
flareburner_percent       -0.2452      0.102     -2.406      0.016      -0.445      -0.045
poorlyunder_percent       -0.0585      0.104     -0.564      0.573      -0.262       0.145
poorlytopped_percent       0.1316      0.088      1.496      0.135      -0.041       0.304
hard_hit_percent          -0.1497      0.064     -2.324      0.020      -0.276      -0.023
avg_best_speed             0.3561      0.206      1.726      0.085      -0.049       0.761
===========================================================================================
```

```
Multi-model with interactions. Removed insignificant interactions (p > 0.05) prior to running
'''

sig_features = ['ab', 'batting_avg', 'launch_angle_avg', 'sweet_spot_percent', 'barrel_batted_rate',
                'solidcontact_percent', 'flareburner_percent', 'hard_hit_percent']

model_multi = smf.ols('home_run ~ (ab * batting_avg) + (ab * launch_angle_avg) + (ab * sweet_spot_percent) + '
                      '(ab * barrel_batted_rate) + (ab * solidcontact_percent) + (ab * flareburner_percent) + '
                      '(ab * hard_hit_percent) + (batting_avg * launch_angle_avg) + (batting_avg * sweet_spot_percent)'
                      '+ (batting_avg * barrel_batted_rate) + (batting_avg * solidcontact_percent) + '
                      '(batting_avg * flareburner_percent) + (batting_avg * hard_hit_percent) + '
                      '(launch_angle_avg * sweet_spot_percent) + (launch_angle_avg * barrel_batted_rate) + '
                      '(launch_angle_avg * solidcontact_percent) + (launch_angle_avg * flareburner_percent) + '
                      '(launch_angle_avg * hard_hit_percent) + (sweet_spot_percent * barrel_batted_rate) + '
                      '(sweet_spot_percent * solidcontact_percent) + (sweet_spot_percent * flareburner_percent) + '
                      '(sweet_spot_percent * hard_hit_percent) + (barrel_batted_rate * solidcontact_percent) + '
                      '(barrel_batted_rate * flareburner_percent) + (barrel_batted_rate * hard_hit_percent)'
                      '+ (solidcontact_percent * flareburner_percent) + (solidcontact_percent * hard_hit_percent)'
                      '+ (flareburner_percent * hard_hit_percent)', data=df).fit()

print(model_multi.summary())

model_multi = smf.ols('home_run ~ (ab * batting_avg) + (ab * launch_angle_avg) + (ab * sweet_spot_percent) + '
                      '(ab * barrel_batted_rate) + (ab * flareburner_percent) + (batting_avg * launch_angle_avg)'
                      '+ (barrel_batted_rate * hard_hit_percent)',
                      data=df).fit()

print(model_multi.summary())

'''
With interactions achieved adj R2 of 0.854. Removing insignificant interactions and rerunning yields same results
'''
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               home_run   R-squared:                       0.856
Model:                            OLS   Adj. R-squared:                  0.854
Method:                 Least Squares   F-statistic:                     506.6
Date:                Tue, 27 Jun 2023   Prob (F-statistic):               0.00
Time:                        11:46:52   Log-Likelihood:                -3308.4
No. Observations:                1210   AIC:                             6647.
Df Residuals:                    1195   BIC:                             6723.
Df Model:                          14
Covariance Type:            nonrobust
------------------------------------------------------------------------------
```

```
Decided to add 'hit' back in to the model, but leaving hit type (single, double, triple) out to see how
it affects the model
'''

headers.append('hit')

header_string = ''
for index, x in enumerate(headers):
    if index == 0:
        header_string += x
    else:
        header_string += ' + ' + x

model = smf.ols('home_run ~ ' + header_string, data=df).fit()
print(model.summary())

'''
After adding hit back in, adj R2 the same at 0.812. Still have 8
significant features, with hit replacing batting average
'''

sig_features = ['ab', 'hit', 'launch_angle_avg', 'sweet_spot_percent', 'barrel_batted_rate', 'solidcontact_percent',
                'flareburner_percent', 'hard_hit_percent']

model_multi = smf.ols('home_run ~ (ab * hit) + (ab * launch_angle_avg) + (ab * sweet_spot_percent) + '
                      '(ab * barrel_batted_rate) + (ab * solidcontact_percent) + (ab * flareburner_percent) + '
                      '(ab * hard_hit_percent) + (hit * launch_angle_avg) + (hit * sweet_spot_percent) + '
                      '(hit * barrel_batted_rate) + (hit * solidcontact_percent) + (hit * flareburner_percent) + '
                      '(hit * hard_hit_percent) + (launch_angle_avg * sweet_spot_percent) + '
                      '(launch_angle_avg * barrel_batted_rate) + (launch_angle_avg * solidcontact_percent) + '
                      '(launch_angle_avg * flareburner_percent) + (launch_angle_avg * hard_hit_percent)'
                      '+ (sweet_spot_percent * barrel_batted_rate) + (sweet_spot_percent * solidcontact_percent) + '
                      '(sweet_spot_percent * flareburner_percent) + (sweet_spot_percent * hard_hit_percent)'
                      '+ (barrel_batted_rate * solidcontact_percent) + (barrel_batted_rate * flareburner_percent) + '
                      '(barrel_batted_rate * hard_hit_percent) + (solidcontact_percent * flareburner_percent) + '
                      '(solidcontact_percent * hard_hit_percent) + (flareburner_percent * hard_hit_percent)'
                      , data=df).fit()

print(model_multi.summary())

model_multi = smf.ols('home_run ~ (hit * launch_angle_avg) + (barrel_batted_rate * hard_hit_percent)',
                      data=df).fit()

print(model_multi.summary())

'''
After running with model interactions, achieved and adj R2 of 0.854, same as when batting_avg was in place of hits.
Interestingly, there were only two significant interactions in this version of the model, vs 7 in the first version
'''
```

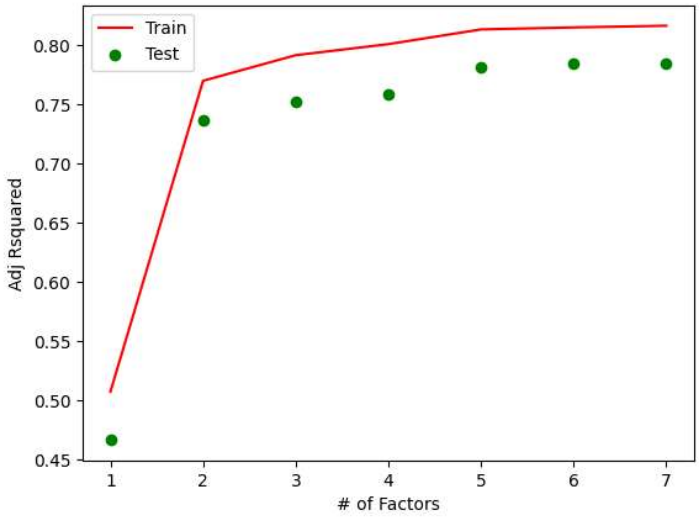
OLS Regression Results

Dep. Variable:	home_run	R-squared:	0.858
Model:	OLS	Adj. R-squared:	0.854
Method:	Least Squares	F-statistic:	197.4
Date:	Tue, 27 Jun 2023	Prob (F-statistic):	0.00
Time:	11:48:17	Log-Likelihood:	-3297.8
No. Observations:	1210	AIC:	6670.
Df Residuals:	1173	BIC:	6858.
Df Model:	36		
Covariance Type:	nonrobust		

```
'''
Forward selection up to k=7.
'''

best = ['',0]
for p in headers:
    model  = smf.ols(formula='home_run~'+p, data=X_train).fit()
    print(p, model.rsquared)
    if model.rsquared>best[1]:
        best = [p, model.rsquared]
print('best:',best)

train_hr1 = smf.ols(formula='home_run ~ barrel_batted_rate', data=X_train).fit()

best = ['',0]
for p in headers:
    model  = smf.ols(formula=train_hr1.model.formula+ '+' + p, data=X_train).fit()
    print(p, model.rsquared)
    if model.rsquared>best[1]:
        best = [p, model.rsquared]
print('best:',best)

train_hr2 = smf.ols(formula=train_hr1.model.formula+ '+' + best[0],data=X_train).fit()
print(train_hr2.model.formula)
train_hr2.rsquared_adj

best = ['',0]
for p in headers:
    model  = smf.ols(formula=train_hr2.model.formula+ '+' + p, data=X_train).fit()
    print(p, model.rsquared)
    if model.rsquared>best[1]:
        best = [p, model.rsquared]
print('best:',best)

train_hr3 = smf.ols(formula=train_hr2.model.formula+ '+' + best[0],data=X_train).fit()
print(train_hr3.model.formula)
train_hr3.rsquared_adj

best = ['',0]
for p in headers:
    model  = smf.ols(formula=train_hr3.model.formula+ '+' + p, data=X_train).fit()
    print(p, model.rsquared)
    if model.rsquared>best[1]:
        best = [p, model.rsquared]
print('best:',best)

train_hr4 = smf.ols(formula=train_hr3.model.formula+ '+' + best[0],data=X_train).fit()
print(train_hr4.model.formula)
train_hr4.rsquared_adj
```

```
403
404 best = ['',0]
405 for p in headers:
406     model  = smf.ols(formula=train_hr4.model.formula+ '+' + p, data=X_train).fit()
407     print(p, model.rsquared)
408     if model.rsquared>best[1]:
409         best = [p, model.rsquared]
410 print('best:',best)
411
412 train_hr5 = smf.ols(formula=train_hr4.model.formula+ '+' + best[0],data=X_train).fit()
413 print(train_hr5.model.formula)
414 train_hr5.rsquared_adj
415
416 best = ['',0]
417 for p in headers:
418     model  = smf.ols(formula=train_hr5.model.formula+ '+' + p, data=X_train).fit()
419     print(p, model.rsquared)
420     if model.rsquared>best[1]:
421         best = [p, model.rsquared]
422 print('best:',best)
423
424 train_hr6 = smf.ols(formula=train_hr5.model.formula+ '+' + best[0],data=X_train).fit()
425 print(train_hr6.model.formula)
426 train_hr6.rsquared_adj
427
428 best = ['',0]
429 for p in headers:
430     model  = smf.ols(formula=train_hr6.model.formula+ '+' + p, data=X_train).fit()
431     print(p, model.rsquared)
432     if model.rsquared>best[1]:
433         best = [p, model.rsquared]
434 print('best:',best)
435
436 train_hr7 = smf.ols(formula=train_hr6.model.formula+ '+' + best[0],data=X_train).fit()
437 print(train_hr7.model.formula)
438 train_hr7.rsquared_adj
439
440 k = [1, 2, 3, 4, 5, 6, 7]
441 test_hr1 = smf.ols(formula=train_hr1.model.formula, data=X_test).fit()
442 test_hr2 = smf.ols(formula=train_hr2.model.formula, data=X_test).fit()
443 test_hr3 = smf.ols(formula=train_hr3.model.formula, data=X_test).fit()
444 test_hr4 = smf.ols(formula=train_hr4.model.formula, data=X_test).fit()
445 test_hr5 = smf.ols(formula=train_hr5.model.formula, data=X_test).fit()
446 test_hr6 = smf.ols(formula=train_hr6.model.formula, data=X_test).fit()
447 test_hr7 = smf.ols(formula=train_hr7.model.formula, data=X_test).fit()
448
449 adjr2_train = [train_hr1.rsquared_adj, train_hr2.rsquared_adj, train_hr3.rsquared_adj, train_hr4.rsquared_adj,
450                train_hr5.rsquared_adj, train_hr6.rsquared_adj, train_hr7.rsquared_adj]
451
452 adjr2_test = [test_hr1.rsquared_adj, test_hr2.rsquared_adj, test_hr3.rsquared_adj, test_hr4.rsquared_adj,
453               test_hr5.rsquared_adj, test_hr6.rsquared_adj, test_hr7.rsquared_adj]
454
```

```
454
455     plt.plot(k, adjr2_train, c='red', label='Train')
456     plt.scatter(k, adjr2_test, c='green', label='Test')
457     plt.xlabel("# of Factors")
458     plt.ylabel("Adj Rsquared")
459     plt.legend()
460     plt.show()
461
462     '''
463     Adjusted R2 sharply increases for testing and training data from k=1 to k=2, but then starts to level
464     off quickly. k=5 is likely the best solution, as afterwards there was no meaningful increase in adj R2.
465     '''
466
467     '''
468     Predicting home runs on the test data, and calculating MSE to compare future models
469     '''
470     y_pred = test_hr5.predict(X_test)
471     y_true = X_test['home_run'].values
472     MSEmultilinear = mean_squared_error(y_true, y_pred)
473
474     Models.append('Multi Linear Regression')
475     Mean_Squared_Errors.append(MSEmultilinear)
476
477     '''
478     MSE values of 21.367 for test data
479     '''
```



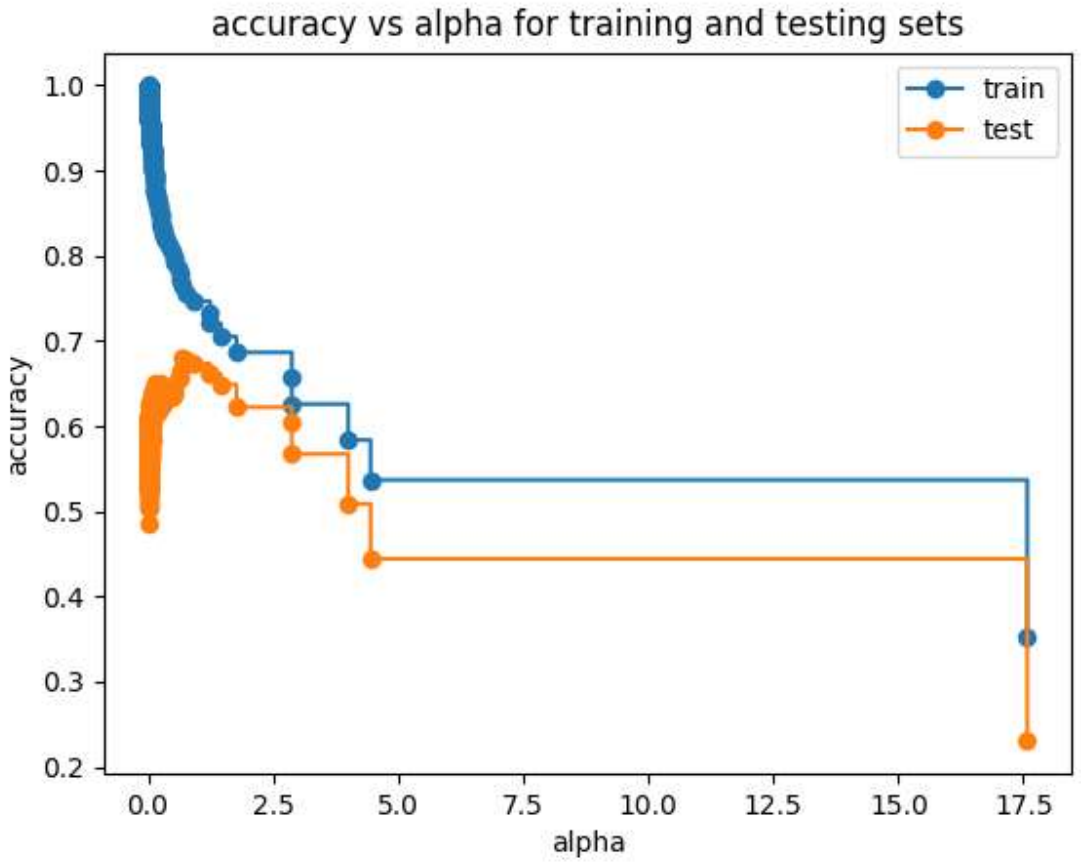
```python
from math import exp
import numpy as np
import pandas as pd
import sklearn
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor

from EDA import df

'''
Create data sets for decision tree regressor
'''

y = df['home_run'].values
X = df.drop('home_run', axis=1).values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .2, random_state=0)

'''
Creating a DecisionTreeRegressor and seeing how it performs
'''

dt = DecisionTreeRegressor().fit(X_train, y_train,)
y_pred_DTR = dt.predict(X_test)
MSEdt = mean_squared_error(y_test, y_pred_DTR)

Models.append('Decision Tree Regressor')
Mean_Squared_Errors.append(MSEdt)

'''
MSE of 45.194 with standard DecisionTreeRegressor. Running GridSearchCV to find best parameters for DTR
'''

rf = DecisionTreeRegressor()
parameters = {'max_depth':[3,5,7,10], 'min_samples_leaf':[1,2,4,6,8,10]}
clf = GridSearchCV(rf, parameters)
clf.fit(X_train, y_train)

best_estimator = clf.best_estimator_
clf.best_score_

'''
Best estimator is DTR with max_depth=7 and min_samples_leaf=8
'''
```

```
>>> best_estimator
DecisionTreeRegressor(max_depth=7, min_samples_leaf=8)
>>> clf.best_score_
0.7317336845749212
```

```
rf = best_estimator
y_pred = rf.predict(X_test)
MSEbestDTR = mean_squared_error(y_test, y_pred)

Models.append('Best Decision Tree Regressor')
Mean_Squared_Errors.append(MSEbestDTR)

'''
MSE = 32.736 when using best_estimator
'''

```

```
'''
Pruning with ccp_alpha
'''

dt = DecisionTreeRegressor().fit(X_train, y_train)

path = dt.cost_complexity_pruning_path(X_train,y_train) #post pruning
ccp_alphas, impurities = path.ccp_alphas, path.impurities

clfs = [] # VECTOR CONTAINING CLASSIFIERS FOR DIFFERENT ALPHAS

for ccp_alpha in ccp_alphas:
    clf = DecisionTreeRegressor(ccp_alpha = ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)

print("Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
      clfs[-1].tree_.node_count, ccp_alphas[-1]))

train_scores = []
test_scores = []

clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]

train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]

max_test_score_index = test_scores.index(max(test_scores))
best_alpha = ccp_alphas[max_test_score_index]
best_test = test_scores[max_test_score_index]
best_train = train_scores[max_test_score_index]

print('alpha', best_alpha)
print('test', best_test)
print('train', best_train)

fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, train_scores, marker='o', label="train",
        drawstyle="steps-post")
ax.plot(ccp_alphas, test_scores, marker='o', label="test",
        drawstyle="steps-post")
ax.legend()
plt.show()
```

```
>>> print('alpha', best_alpha)
... print('test', best_test)
... print('train', best_train)
alpha 0.6977238173879772
test 0.6802645248724117
train 0.7642581660091694
```




```
109
110     clf = DecisionTreeRegressor(ccp_alpha = best_alpha, max_depth=7, min_samples_leaf=8).fit(X_train, y_train)
111     y_pred = clf.predict(X_test)
112     MSEprunedDTR = mean_squared_error(y_test, y_pred)
113
114     Models.append('Pruned Decision Tree Regressor')
115     Mean_Squared_Errors.append(MSEprunedDTR)
116
117     '''
118     MSE = 31.87 after pruning with optimal parameters
119     '''
120
```

```
'''
Beginning with RandomForestRegressor, which randomly samples the data and features.
'''

regr = RandomForestRegressor(max_depth=7, min_samples_leaf=8).fit(X_train, y_train)
y_pred = regr.predict(X_test)
MSErf = mean_squared_error(y_test, y_pred)

Models.append('Random Forest Regressor')
Mean_Squared_Errors.append(MSErf)

'''
MSE = 22.553
'''

'''
Using GridSearchCV to tests for optimal parameters
'''

rf = RandomForestRegressor()
parameters = {'max_depth':[3,5,7,10], 'min_samples_leaf':[1,2,4,6,8,10], 'n_estimators':[10,50,100,200],
              'max_samples':[None, 1, 300, 600]}
clf = GridSearchCV(rf, parameters)
clf.fit(X_train, y_train)

best_estimator = clf.best_estimator_
```

```
>>> best_estimator
RandomForestRegressor(max_depth=10, max_samples=600, min_samples_leaf=2,
                      n_estimators=200)
```

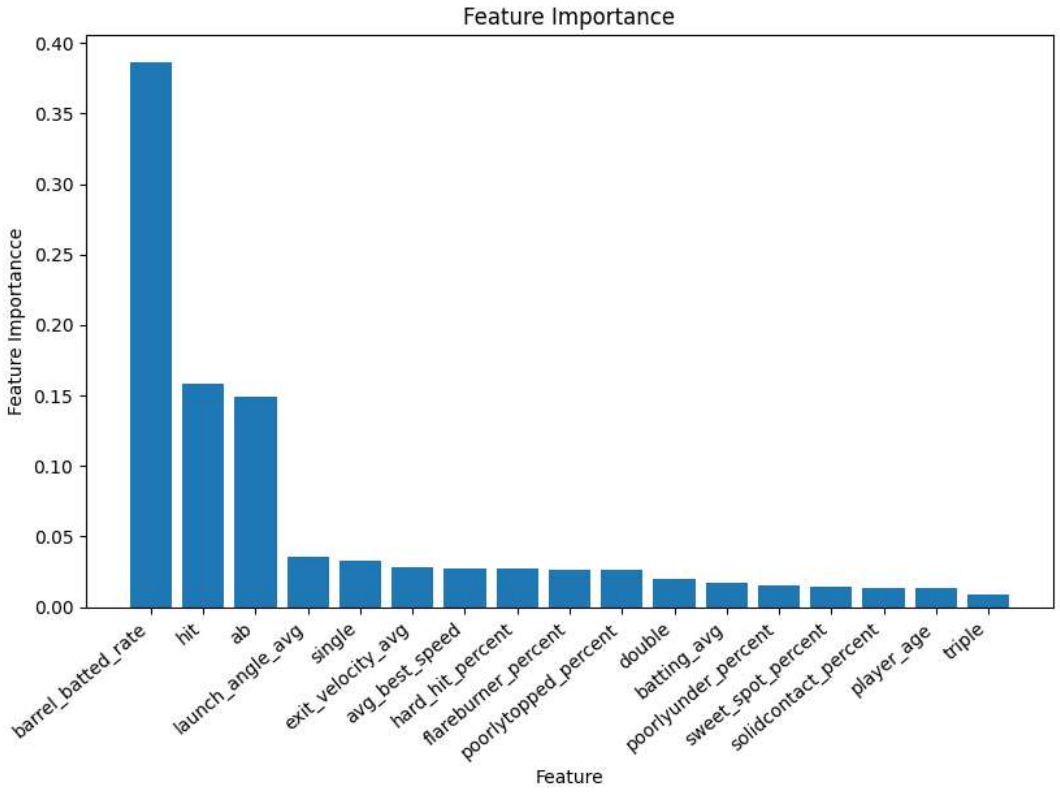
```
'''
Best estimator shows that the optimal parameters for RandomForestRegressor are max_depth=10, min_samples_leaf=2,
max_samples=600, n_estimators=200
'''

regr = RandomForestRegressor(max_depth=10, min_samples_leaf=2, max_samples=600, n_estimators=200).fit(X_train, y_train)
y_pred_regr = regr.predict(X_test)
MSEbestrf = mean_squared_error(y_test, y_pred_regr)

Models.append('Best Random Forest Regressor')
Mean_Squared_Errors.append(MSEbestrf)

'''
After running RandomForestRegressor with optimal paramters, MSE down to 21.681
'''
```

```
'''
Boosting the ensemble with AdaBoost. Starting with standard AdaBoost then GridSearch for optimal parameters
'''

regr_ada = AdaBoostRegressor().fit(X_train, y_train)
y_pred = regr_ada.predict(X_test)
MSEada = mean_squared_error(y_test, y_pred)

Models.append('AdaBoost')
Mean_Squared_Errors.append(MSEada)

'''
Initial adaBoost has MSE of 24.72. GridSearchCV for best params
'''

rf = AdaBoostRegressor()
parameters = {'n_estimators':[50,100], 'learning_rate':[1,5,10], 'loss':['linear','square','exponential']}
clf = GridSearchCV(rf, parameters)
clf.fit(X_train, y_train)

best_estimator = clf.best_estimator_
```

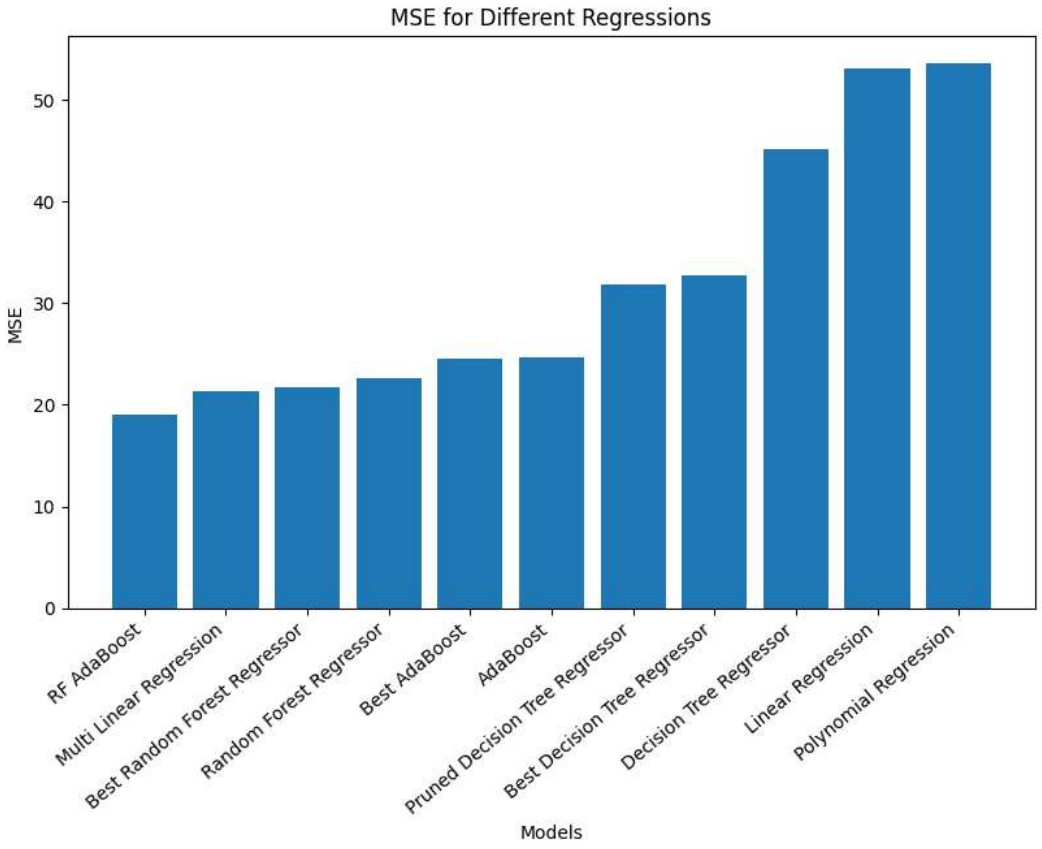
```
>>> best_estimator
AdaBoostRegressor(learning_rate=1, loss='square')
```

```

regr_ada = AdaBoostRegressor(learning_rate=1, n_estimators=100).fit(X_train, y_train)
y_pred = regr_ada.predict(X_test)
MSEbestada = mean_squared_error(y_test, y_pred)

Models.append('Best AdaBoost')
Mean_Squared_Errors.append(MSEbestada)

'''
MSE = 24.474 with optimal parameters
'''

'''
AdaBoost with optimnal RandomForest parameters
'''

regr_ada_rf = AdaBoostRegressor(estimator=RandomForestRegressor(max_depth=10, min_samples_leaf=2, max_samples=600,
                                                                n_estimators=200), n_estimators=100,
                                                                loss='square').fit(X_train, y_train)
y_pred = regr_ada_rf.predict(X_test)
MSErfada = mean_squared_error(y_test, y_pred)

Models.append('RF AdaBoost')
Mean_Squared_Errors.append(MSErfada)

'''
MSE = 19.068 with optimal rf parameters
'''
```

```
features_importance = []
features = list(df.columns)
features.pop(0)


for index, item in enumerate(features):
    features_importance.append([features[index], regr_ada_rf.feature_importances_[index]])

features_importance.sort(key = lambda x: x[1], reverse=True)

'''
Most important features to AdaBoost were 'barrel_batted_rate', 'hit', and 'ab'. Everything else much lower
'''

x = features
y = regr_ada_rf.feature_importances_
sorted_data = sorted(zip(x,y), key=lambda x: x[1], reverse=True)
x_values, y_values = zip(*sorted_data)

fig, ax = plt.subplots(figsize=(9.6,7.2))
ax.set_xlabel("Feature")
ax.set_ylabel("Feature Importancce")
ax.set_title("Feature Importance")
ax.bar(x_values, y_values)
ax.set_xticklabels(x_values, rotation=40, ha='right')
plt.subplots_adjust(bottom=0.27)
plt.show()
```



```
'''
Compare all different regressions
'''
x = Models
y = Mean_Squared_Errors
sorted_data = sorted(zip(x,y), key=lambda x: x[1])
x_values, y_values = zip(*sorted_data)

fig, ax = plt.subplots(figsize=(9.6,7.2))
ax.set_xlabel("Models")
ax.set_ylabel("MSE")
ax.set_title("MSE for Different Regressions")
ax.bar(x_values, y_values)
ax.set_xticklabels(x_values, rotation=40, ha='right')
plt.subplots_adjust(bottom=0.27)
plt.show()
```



Discussion/Conclusion

```
'''
Discussion and conclusion - Based on the handful of regression performed on my data, a RandomForestRegressor with
AdaBoost was the most effective model. This was not surprising, as RF generally out performs linear regression and
decision trees. It was interesting that multi linear regression performed better than decision tree regression, this
may have been related to that fact that while I had a high number of features, only a small number were atctually
important to the regression. My initial dataset did not work for a regression because there was a combination of four
variables that perfectly calculated my target, so I had an R2 of 1.0 and none of the other features were important. But
I was able to easily adjust for this by removing those features from the dataset. To improve on this analysis, you
could add many more features. With the addition of Statcast to MLB, there are now hundreds of advanced statistics that
are automatically tracked for every game, and I sampled just a handful that I thought would be significant. That being
said, after achieving not very good results with any of my regressions (high MSE with all of them), I do think that
modeling home runs from other statistics may not be a wortwhile venture.
'''
```