The below predictor model uses the RandomForestRegressor. I have chosen randomforestregressor because:
1. It is known to avoid overfitting as it makes use of ensemble learning.
2. The predictor model hence can converge and generalize more.
3. It is also known for its act of handling msising values in the dataset.
4. The most important reason why I used randomforest was because this algorithm is known to be less sensitive to outliers compared to the other regression algorithms because the averaging of predictions from multiple trees helps reduce the impact of outliers on the final predictions.
I have also performed hyper parameter tuning with:
1. n_estimators
2. max_depth of the tree
3. min_sample_split and
4. the max_sample_split.
This helps in determining the best combination of the hyperparameters. After obtaining the best hyperparameters, we train the Random Forest model with those hyperparameters and evaluate its performance on the training data.
The best parameters are : 'max_depth': 5, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 100
From the result predicted, you can see that the accuracy on the training set is 0.90% and the overall RMSE is around 0.45 which indicates the model has perfomeed quite well without overfitting.

```
#importing the libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
import seaborn as sns
#loading the baseball dataset
df=pd.read_csv('baseball.csv')
df.head()
W R AB H 2B 3B HR BB SO SB RA ER ERA
CG SHO \
0 95 724 5575 1497 300 42 139 383 973 104 641 601 3.73
2 8
1 83 696 5467 1349 277 44 156 439 1264 70 700 653 4.07
2 12
2 81 669 5439 1395 303 29 141 533 1157 86 640 584 3.67
11 10
3 76 622 5533 1381 260 27 136 404 1231 68 701 643 3.98
7 9
4 74 689 5605 1515 289 49 151 455 1259 83 803 746 4.64
7 12
SV E
0 56 88
1 45 86
2 38 79
```

3 37 101
4 35 86
Data Preprocessing
#This gives the information about the data types, column names, nonnull values and the memory usage.
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 17 columns):
Column Non-Null Count Dtype
--- ------ -------------- -----
0 W 30 non-null int64
1 R 30 non-null int64
2 AB 30 non-null int64
3 H 30 non-null int64
4 2B 30 non-null int64
5 3B 30 non-null int64
6 HR 30 non-null int64
7 BB 30 non-null int64
8 SO 30 non-null int64
9 SB 30 non-null int64
10 RA 30 non-null int64
11 ER 30 non-null int64
12 ERA 30 non-null float64
13 CG 30 non-null int64
14 SHO 30 non-null int64
15 SV 30 non-null int64
16 E 30 non-null int64
dtypes: float64(1), int64(16)
memory usage: 4.1 KB
#statistical information of the dataset
df.describe()
W R AB H 2B \
count 30.000000 30.000000 30.000000 30.000000 30.000000
mean 80.966667 688.233333 5516.266667 1403.533333 274.733333
std 10.453455 58.761754 70.467372 57.140923 18.095405
min 63.000000 573.000000 5385.000000 1324.000000 236.000000
25% 74.000000 651.250000 5464.000000 1363.000000 262.250000
50% 81.000000 689.000000 5510.000000 1382.500000 275.500000
75% 87.750000 718.250000 5570.000000 1451.500000 288.750000
max 100.000000 891.000000 5649.000000 1515.000000 308.000000
3B HR BB SO SB
RA \
count 30.000000 30.000000 30.000000 30.00000 30.000000
30.000000
mean 31.300000 163.633333 469.100000 1248.20000 83.500000
688.233333
std 10.452355 31.823309 57.053725 103.75947 22.815225
72.108005
min 13.000000 100.000000 375.000000 973.00000 44.000000
525.000000
25% 23.000000 140.250000 428.250000 1157.50000 69.000000
636.250000
50% 31.000000 158.500000 473.000000 1261.50000 83.500000
695.500000

```
75% 39.000000 177.000000 501.250000 1311.50000 96.500000
732.500000
max 49.000000 232.000000 570.000000 1518.00000 134.000000
844.000000
ER ERA CG SHO SV
E
count 30.000000 30.000000 30.000000 30.000000 30.000000
30.000000
mean 635.833333 3.956333 3.466667 11.300000 43.066667
94.333333
std 70.140786 0.454089 2.763473 4.120177 7.869335
13.958889
min 478.000000 2.940000 0.000000 4.000000 28.000000
75.000000
25% 587.250000 3.682500 1.000000 9.000000 37.250000
86.000000
50% 644.500000 4.025000 3.000000 12.000000 42.000000
91.000000
75% 679.250000 4.220000 5.750000 13.000000 46.750000
96.750000
max 799.000000 5.040000 11.000000 21.000000 62.000000
126.000000
#to check any missing values. The dataset contains no missing values.
df.isnull().all()
W False
R False
AB False
H False
2B False
3B False
HR False
BB False
SO False
SB False
RA False
ER False
ERA False
CG False
SHO False
SV False
E False
dtype: bool
#visualising the correlation matrix for the 16 attributes
correlation_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
#splitting the attributes into feature and target variables.
X = df.drop("W", axis=1)
y = df["W"]
#splitting the attributes into training and testing sets. The split
ratio is 70% and 30%. 70% contains the training set and 30% contains
the test set.
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```
test_size=0.3, random_state=42)
X_train.head()
R AB H 2B 3B HR BB SO SB RA ER ERA CG
SHO \
0 724 5575 1497 300 42 139 383 973 104 641 601 3.73 2
8
4 689 5605 1515 289 49 151 455 1259 83 803 746 4.64 7
12
16 697 5631 1462 292 27 140 461 1322 98 596 532 3.21 0
13
5 891 5509 1480 308 17 232 570 1151 88 670 609 3.80 7
10
13 656 5544 1379 262 22 198 478 1336 69 726 677 4.16 6
12
SV E
0 56 88
4 35 86
16 54 122
5 34 88
13 45 94
X_test
R AB H 2B 3B HR BB SO SB RA ER ERA CG
SHO \
27 720 5649 1494 289 48 154 490 1312 132 713 659 4.04 1
12
15 647 5484 1386 288 39 137 506 1267 69 525 478 2.94 1
15
23 573 5420 1361 251 18 100 471 1107 69 760 698 4.41 3
10
17 689 5491 1341 272 30 171 567 1518 95 608 546 3.36 6
21
8 644 5485 1383 278 32 167 436 1310 87 642 604 3.74 1
12
9 748 5640 1495 294 33 161 478 1148 71 753 694 4.31 3
10
28 650 5457 1324 260 36 148 426 1327 82 731 655 4.09 1
6
24 626 5529 1374 272 37 130 387 1274 88 809 749 4.69 1
7
12 661 5417 1331 243 21 176 435 1150 52 675 630 3.94 2
12
SV E
27 44 86
15 62 96
23 44 90
17 48 111
8 60 95
9 40 97
28 41 92
24 35 117
12 46 93
#standardization using the 'standardscaler' of scikit-learn. This
ensures that all the features have similar scales and follow a
standard distribution.
#This helps in facilitating the models and increase the efficiency.
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
#model initialisation
model = RandomForestRegressor()
#fitting the training feature and target variable
model.fit(X_train_scaled, y_train)
RandomForestRegressor()
```

Hyperparameter Tuning using GridSearch

```
param_grid = {
'n_estimators': [100, 200], # Number of trees in the forest
'max_depth': [None, 5], # Maximum depth of each tree
'min_samples_split': [2, 5], # Minimum number of samples required
to split an internal node
'min_samples_leaf': [1, 2], # Minimum number of samples required
to be at a leaf node
}
grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
cv=5)
grid_search.fit(X_train_scaled, y_train)
GridSearchCV(cv=5, estimator=RandomForestRegressor(),
param_grid={'max_depth': [None, 5], 'min_samples_leaf':
[1, 2],
'min_samples_split': [2, 5],
'n_estimators': [100, 200]})
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)
Best Hyperparameters: {'max_depth': 5, 'min_samples_leaf': 2,
'min_samples_split': 5, 'n_estimators': 100}
# Train the Random Forest model with the best hyperparameters
rf_best = RandomForestClassifier(**best_params)
rf_best.fit(X_train_scaled, y_train)
RandomForestClassifier(max_depth=5, min_samples_leaf=2,
min_samples_split=5)
# Make predictions on the training data
y_pred = rf_best.predict(X_train_scaled)
accuracy = accuracy_score(y_train, y_pred)
print("Training Accuracy:", accuracy)
Training Accuracy: 0.9047619047619048
```

Predicting the Winners

```
y_pred = model.predict(X_test_scaled)
print("2015 WINNERS: ")
print(*y_pred)
2015 WINNERS:
82.06 85.65 74.49 87.69 84.96 78.33 75.91 69.11 80.86
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse/100)
Mean Squared Error: 0.4898806666666664
```