고교야구선수 성적 예측 모델 개발

-앙상블 기법(LightGBM, XGBoost)을 중심으로-

컴퓨터교육과 2020311252 최재석
컴퓨터교육과 2022315923 배서현
컴퓨터교육과 2022315947 최태림

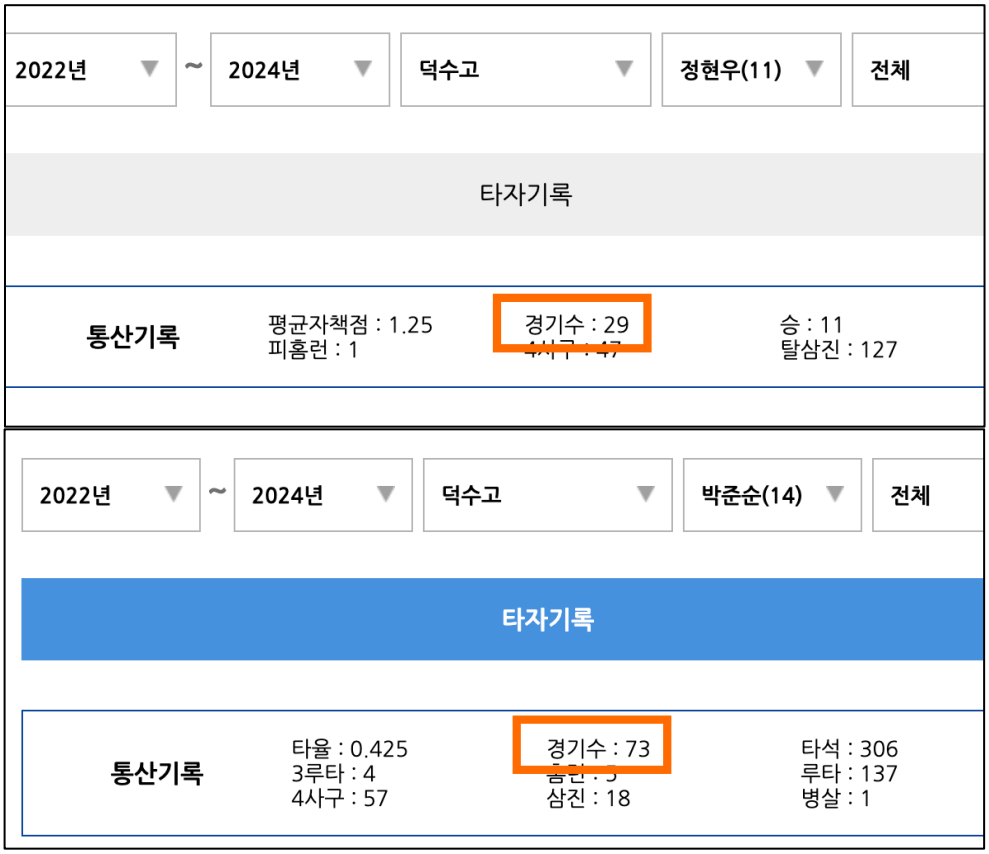
목 차

I. 문제 정의

본 프로젝트는 앙상블 기법을 활용한 고교야구선수의 성적 예측 모델을 개발하는 것을 목적으로 한다. 주제 선정 배경은 다음과 같다.

II. 배경

프로야구단이 매년 신인 선수를 선발하는 과정은 구단의 미래를 결정 짓는 매우 중요한 결정이자 행사이다. 따라서 구단은 매년 팀의 방향성과 비전에 알맞은 최선의 선택을 하기 위해 아마추어 선수들을 면밀히 관찰한다. 하지만 주말리그와 토너먼트 대회로 구성된 아마추어 야구의 특성상 선수의 실력을 판단할 수 있는 경기 표본이 부족하다는 문제점이 존재한다. 아래 사진은 2025 년 신인선수 드래프트에서 선발된 110 명의 선수 중 각각 전체 1 번과 전체 6 번으로 선발된 정현우(키움, 투수)와 박준순(두산, 야수)의 고교야구 통산 기록이다.



최상위권 순서에 선발된 두 선수의 경우에도 3 년간 누적된 경기 수가 투수인 정현우 선수는 30 경기, 야수인 박준순 선수는 73 경기에 불과하다. 프로야구의 한 시즌이 144 경기로 진행되는 것을 고려하면 이는 턱없이 부족한 표본이라 할 수 있다. 이마저도 소속 학교가 토너먼트 대회에서 조기 탈락한다면 표본 수는 더욱 줄어들게 된다.

이러한 배경으로부터 발생되는 문제는 스카우터와 학교 및 학생선수의 입장에서 정리해볼 수 있다.

[스카우터 입장에서의 문제]

1. 선수를 판단할 표본이 부족한 상태에서 선수에 대해 더 면밀히 관찰하기 어렵다.
2. 이 선수가 1 년 144 경기인 한 시즌을 온전히 치를 수 있을지 검증하기 어렵다.
3. 모든 선수를 면밀히 살피기 어렵고 임팩트를 보여주는 '최대어' 중심으로 살펴보게 된다.

[학생선수 및 학교 입장에서의 문제]
1. 학교가 대회에서 조기탈락 할 경우 본인을 어필할 기회가 외부 요인에 의해 줄어든다.
2. 스카우터에게 강한 인상을 남겨주기 위해 투수는 빠른 구속, 야수는 강한 파워에 집중한다.
3. 2 번으로 인해 학생 선수들이 부상 당하는 경우가 잦아졌다.
4. 3 번으로 인해 부상 공백이 생긴 학교팀을 운영하는데 어려움을 겪는다.

특히나 학생선수, 특히 투수는 더 빠른 구속을 던져 임팩트를 남기기 위해 본인의 신체 능력 범위를 벗어난 과훈련을 진행하여 부상을 입는 경우가 잦아졌다. 이는 최근 아마추어 야구 선수의 UCL(팔꿈치 내측부 인대) 부상이 늘어나며 이를 회복하기 위한 수술과 유급이 늘어난 것에서 읽어낼 수 있다 (Kriz et al, 2022). 아마추어 야구선수의 과도한 구속 증가로 인한 부상 증가는 비단 한국 뿐만 아니라 미국에서도 부각되는 문제점이라는 점에서 예의주시해야 할 안건임에 틀림없다.

따라서 본 프로젝트는 위와 같은 한계를 극복할 수 있는 보조 도구를 개발하는데 있어 그 의미를 갖는다. 해당 모델을 통해서 학생선수의 현재 성적 지표를 가지고 미래 성적을 예측하고, 스카우터가 이 수치를 활용하여 학생선수를 평가하는데 적용하는 것을 목표로 한다.

III. 데이터 설명

데이터는 KBSA(대한야구소프트볼협회)에서 제공하는 경기 기록을 직접 수집해 생성했다. 2020 년부터 2024 년 5 년간 프로구단에 지명된 선수들의 아마추어 시절 마지막 1 년분 성적을 수집했다. 데이터는 수치 데이터로, 투수의 경우 평균자책점, 경기수, 승, 패, 이닝, 타자, 피안타, 피홈런, 4 사구, 탈삼진, 실점, 자책점, 승률, WHIP(이닝당 주자 허용률)의 총 14 개 열을 지니며, 총 1359 행으로 구성되었다.

각각의 열에 대한 의미는 다음과 같다.

용어	설명
평균자책점	투수가 9 이닝 동안 허용한 자책점의 평균. 투수의 경기력과 안정성을 평가하는 중요한 지표이며, 낮을 수록 좋은 성적을 의미한다.
경기수	해당 투수가 등판한 경기의 수. 투수가 얼마나 자주 경기에 기용되었는지를 보여준다.
승, 패	투수가 해당 경기에서 승리 또는 패배를 기록했는지를 보여준다.
이닝	투수가 던진 이닝 수를 나타낸다. 1 이닝은 3 개의 아웃을 잡는 것을 의미한다.
타자	투수가 상대한 타자의 총 수를 의미한다
피안타, 피홈런	투수가 상대 타자에게 허용한 안타의 수와 홈런의 수를 의미한다. 두 지표는 투수의 상대 타선에 대한 억제력을 나타낸다

4 사구	투수가 상대 타자에게 허용한 볼넷(4 구)과 사구(몸에 맞는 공)의 합계이다. 이 수치가 높은 것은 투수의 제구력이 불안정함을 내포한다
탈삼진	투수가 상대 타자를 삼진으로 처리한 횟수이다. 수치가 높을 수록 투수가 타자를 압도하는 능력이 뛰어남을 의미한다
실점	투수가 경기 중 허용한 총 득점을 의미한다. 자책점 뿐만 아니라 수비 실책 등 모든 원인으로 인해 상대 팀에 허용된 득점을 포함한다
자책점	투수가 허용한 실점 중 수비 실책 등과 무관하게 투수의 책임으로 기록된 득점을 의미한다
승률	투수의 승리 비율을 의미한다
WHIP	이닝 당 허용한 출루 수를 나타낸다. (볼넷 + 피안타) / 이닝 수로 계산된다. 투수의 안정성과 제구력을 평가하는 지표이며, 낮을 수록 좋은 성적이다

야수의 경우 타율, 경기수, 타석, 타수, 득점, 총안타, 2 루타, 3 루타, 홈런, 루타, 타점, 도루, 희타, 희비, 4 사구, 삼진, 병살, 장타율, 출루율, OPS 의 총 20 개 열을 지니며, 총 1290 행으로 구성되었다.
각각의 열에 대한 의미는 다음과 같다.

용어	설명
타율	안타의 수를 타수로 나눈 값. 타자가 얼마나 자주 안타를 치는지를 나타낸다.
경기수	해당 야수가 등판한 경기의 수. 얼마나 자주 경기에 기용되었는지를 보여준다
타석, 타수	타석은 타자가 타석에 들어선 횟수를 나타낸다. 볼넷, 몸에 맞는 공, 희생타 등을 모두 포함한다. 타수는 타자가 정식으로 타격한 횟수만을 나타낸다. 볼넷, 몸에 맞는 공, 희생타 등은 제외된다.
득점	타자가 홈을 밟아 기록한 득점의 수를 의미한다
총 안타	타자가 기록한 안타의 총 수. 단타, 2 루타, 3 루타, 홈런을 모두 포함한다
2 루타, 3 루타, 홈런	각각 타자가 한번에 2 루까지 진루한 안타의 수/한번에 3 루까지 진루한 안타의 수/직접 홈까지 돌아온 안타의 수를 의미한다
루타	타자가 기록한 총 누적 베이스 수를 의미한다. 단타는 1, 2 루타는 2, 3 루타는 3, 홈런은 4 베이스로 계산한다.
타점	타자가 타격을 통해 주자를 홈으로 불러들인 횟수를 의미한다. 팀 공격에서의 기여도를 평가하는 지표로 활용된다
도루	타자가 투수의 투구 도중 다음 베이스로 도달한 횟수를 의미한다. 타자의 주루 능력을 평가하는 지표로 활용된다
희타, 희비	희타는 희생번트로, 주자를 진루 시키거나 득점 시키기 위해 타자가 희생한 경우를 의미한다. 희비는 희생 플라이로, 이를 통해 주자를 득점하게 한 경우를 의미한다

4 사구	타자가 볼넷(4 구) 혹은 사구(몸에 맞는 공)을 통해 출루한 횟수의 합계이다
삼진	타자가 삼진으로 아웃 된 횟수를 의미한다. 타자의 약점을 평가하는 지표로 활용된다.
병살	타자가 친 공으로 인해 두 명의 주자가 아웃 된 경우를 의미한다
장타율	타자가 기록한 총 루타 수를 타수로 나눈 값. 타격의 위력을 평가하는 지표이다
출루율	타자가 출루한 비율을 나타낸다. (안타+볼넷+몸에 맞는 공) / (타수+볼넷+몸에 맞는 공+희생 플라이)로 계산된다
OPS	출루율과 장타율의 합으로 계산된다. 타자의 공격 생산성을 종합적으로 평가하는 지표이다

IV. EDA

엑셀 파일의 '2020-2025_fielder', '2020-2025_pitcher' 시트에서 데이터를 불러오고, 선수의 이름과 경기 및 대회명이 담긴 player_id, game_tag 열을 제거한 수치형 데이터에 대해 EDA 를 실시했다. IQR(사분위 범위)를 계산하여 이상치를 탐지하고 각 열의 기초 통계량 및 이상치 개수를 출력했다.

1. 2020-2025_fielder 데이터 EDA 결과

1290 개의 행과 22 개의 열이 있으며, 대부분의 데이터가 결측치 없이 잘 채워져 있다.

```
--- Sheet: 2020-2025_fielder ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1290 entries, 0 to 1289
Data columns (total 22 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   player_id  1290 non-null   int64
 1   game_tag   1290 non-null   object
 2   타율         1290 non-null   float64
 3   경기수        1290 non-null   int64
 4   타석         1290 non-null   int64
 5   타수         1290 non-null   int64
 6   득점         1290 non-null   int64
 7   총안타        1290 non-null   int64
 8   2루타        1290 non-null   int64
 9   3루타        1290 non-null   int64
 10  홈런         1290 non-null   int64
 11  루타         1290 non-null   int64
 12  타점         1290 non-null   int64
 13  도루         1290 non-null   int64
 14  희타         1289 non-null   float64
 15  희비         1290 non-null   int64
 16  4사구        1290 non-null   int64
 17  삼진         1289 non-null   float64
 18  병살         1290 non-null   int64
 19  장타율        1290 non-null   float64
 20  출루율        1290 non-null   float64
 21  OPS        1290 non-null   float64
dtypes: float64(6), int64(15), object(1)
memory usage: 221.8+ KB
None
```

총 1290 개의 샘플로 구성되어 있으며, 타율의 평균은 0.31, OPS 의 평균은 0.90 이다.

```
Basic Statistics:
        player_id          타율         경기수          타석          타수  ₩
count  1290.000000  1290.000000  1290.000000  1290.000000  1290.000000
mean    136.584496     0.313967     4.008527    17.470543    13.780620
std      69.667922     0.171035     2.445830    11.044356     8.664012
min       1.000000     0.000000     0.000000     1.000000     0.000000
25%      81.250000     0.208750     2.000000     8.000000     6.000000
50%     141.000000     0.316000     4.000000    17.000000    13.000000
75%     197.000000     0.417000     6.000000    26.000000    20.000000
max     248.000000     1.000000    34.000000   157.000000   113.000000

              득점         총안타          2루타          3루타          홈런  ...  ₩
count  1290.000000  1290.000000  1290.000000  1290.000000  1290.000000  ...
mean      3.569767     4.578295     0.946512     0.262016     0.314729  ...
std       3.383919     3.733208     1.174235     0.599611     0.714594  ...
min       0.000000     0.000000     0.000000     0.000000     0.000000  ...
25%       1.000000     2.000000     0.000000     0.000000     0.000000  ...
50%       3.000000     4.000000     1.000000     0.000000     0.000000  ...
75%       5.000000     7.000000     2.000000     0.000000     0.000000  ...
max      49.000000    50.000000     8.000000     5.000000     9.000000  ...

              타점          도루          희타          희비       4사구  ₩
count  1290.000000  1290.000000  1289.000000  1290.000000  1290.000000
mean      3.100000     1.442636     0.225756     0.265891     3.144961
std       3.190378     2.269003     0.548364     0.545664     2.856796
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       1.000000     0.000000     0.000000     0.000000     1.000000
50%       2.000000     1.000000     0.000000     0.000000     2.000000
75%       5.000000     2.000000     0.000000     0.000000     5.000000
max      33.000000    22.000000     4.000000     3.000000    37.000000

              삼진          병살         장타율         출루율          OPS
count  1289.000000  1290.000000  1290.000000  1290.000000
mean      2.369279     0.189147     0.472709     0.429911     0.902619
std       2.112927     0.454141     0.308616     0.167271     0.440400
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       1.000000     0.000000     0.268500     0.333000     0.629000
50%       2.000000     0.000000     0.438000     0.438000     0.875000
75%       4.000000     0.000000     0.636000     0.524000     1.156000
max      12.000000     4.000000     2.500000     1.000000     3.500000

[8 rows x 21 columns]
```

타율과 OPS 에서는 각각 20 개와 21 개의 이상치가 탐지되었으며, 특히 홈런(291 개)과 3 루타(261 개)에서 많은 이상치가 발견되었다. 다만 주말리그 및 대회의 경기수가 모두 10 경기 내외여서 홈런과 3 루타 모두 1 개 이상 기록하기 어렵다는 점을 고려해야 한다. 즉 대부분의 데이터가 홈런과 3 루타는 그 값으로 0 을 지니고 있으므로, 이상치들에 대해 특별한 처리는 진행하지 않았다.

```
[8 rows x 21 columns]

Outliers detected:
player_id      0
타율            20
경기수             1
타석              1
타수              1
득점             27
총안타            14
2루타             9
3루타           261
홈런            291
루타             27
타점             25
도루             73
희타            227
희비            285
4사구             8
삼진             16
병살            215
장타율            26
출루율            63
OPS            21
dtype: int64
```

2. 2020-2025_pitcher 데이터 EDA 결과

```
--- Sheet: 2020-2025_pitcher ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1359 entries, 0 to 1358
Data columns (total 32 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   player_id     1359 non-null   int64
 1   game_tag      1359 non-null   object
 2   평균자책점         1359 non-null   float64
 3   경기수           1359 non-null   int64
 4   승             1359 non-null   int64
 5   패             1359 non-null   float64
 6   이닝            1359 non-null   float64
 7   타자            1359 non-null   int64
 8   피안타           1358 non-null   float64
 9   피홈런           1359 non-null   int64
 10  4사구           1359 non-null   int64
 11  탈삼진           1359 non-null   int64
 12  실점            1358 non-null   float64
 13  자책점           1359 non-null   int64
 14  승률            1359 non-null   float64
 15  WHIP          1359 non-null   float64
 16  Unnamed: 16  0 non-null      float64
 17  Unnamed: 17  0 non-null      float64
 18  Unnamed: 18  0 non-null      float64
 19  Unnamed: 19  0 non-null      float64
 20  Unnamed: 20  0 non-null      float64
 21  Unnamed: 21  0 non-null      float64
 22  Unnamed: 22  0 non-null      float64
 23  Unnamed: 23  0 non-null      float64
 24  Unnamed: 24  0 non-null      float64
 25  Unnamed: 25  0 non-null      float64
 26  Unnamed: 26  0 non-null      float64
 27  Unnamed: 27  0 non-null      float64
 28  Unnamed: 28  0 non-null      float64
 29  Unnamed: 29  0 non-null      float64
 30  Unnamed: 30  235 non-null    float64
 31  Unnamed: 31  235 non-null    float64
dtypes: float64(23), int64(8), object(1)
memory usage: 339.9+ KB
None
```

총 1359개의 샘플로 구성되어 있으며, 평균자책점의 평균은 3.67, WHIP의 평균은 1.32이다. 이닝과 탈삼진의 중앙값은 각각 6.1과 7로 확인되었다.

```
Basic Statistics:
         player_id       평균자책점          경기수           승           패  \
count  1359.000000  1359.000000  1359.000000  1359.000000  1359.000000
mean    157.109639     3.665145     2.629875     0.646063     0.345401
std      80.628389     5.950048     1.549105     0.995112     0.719469
min       1.000000     0.000000     0.000000     0.000000     0.000000
25%      94.000000     0.000000     1.000000     0.000000     0.000000
50%     158.000000     2.140000     2.000000     0.000000     0.000000
75%     227.000000     4.500000     4.000000     1.000000     1.000000
max     291.000000    72.000000     9.000000    19.000000    17.000000

              이닝           타자          피안타          피홈런         4사구  \
count  1359.000000  1359.000000  1358.000000  1359.000000  1359.000000
mean      7.800442    33.590140     6.028719     0.157469     4.055923
std       5.993456    23.902735     4.973021     0.446143     4.512603
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       3.200000    16.000000     2.000000     0.000000     2.000000
50%       6.100000    28.000000     5.000000     0.000000     3.000000
75%      11.100000    47.000000     8.000000     0.000000     5.500000
max      45.000000   178.000000    39.000000     5.000000   113.000000

              탈삼진          실점          자책점          승률         WHIP
count  1359.000000  1358.000000  1359.000000  1359.000000  1359.000000
mean      8.948492     3.252577     2.380427     0.393960     1.316946
std       7.423066     3.059995     2.471981     0.460686     0.899990
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       3.000000     1.000000     0.000000     0.000000     0.810000
50%       7.000000     3.000000     2.000000     0.000000     1.120000
75%      13.000000     5.000000     3.000000     1.000000     1.590000
max      58.000000    18.000000    15.000000     1.000000     9.000000
```

평균자책점에서 69 개의 이상치가 발견되었으며, 피홈런(179 개)과 WHIP(72 개)에서는 이상치가 조금 더 많이 탐지되었다. 다만 피홈런은 타자의 홈런 기록과 유사하게 대부분의 데이터가 0 이므로 특별한 처리를 진행하진 않았다. 그 외 WHIP 의 경우 계산식이 (볼넷 +

피안타) / (이닝 수)인데, 투수가 등판하여 제구력 난조 등의 사유로 해결한 이닝이 적은 경우 가파르게 증가할 수 있으나 이 또한 선수의 기록이라 판단하여 제거하지 않았다.

```
Outliers detected:
player_id      0
평균자책점         69
경기수             1
승                63
패                 6
이닝               34
타자               34
피안타              42
피홈런             179
4사구              68
탈삼진              29
실점               35
자책점              65
승률                0
WHIP             72
dtype: int64
```

V. 모델 선택 근거

해당 모델은 앙상블 기법(Random Forest, LightGBM, XGBoost)과 단순 회귀 기법(선형회귀)을 선택해 지도학습을 진행했으며, 검증 결과 XGBoost 알고리즘을 적용한 모델이 가장 성능이 뛰어남을 확인했다. 각 모델을 선택한 근거로는 크게 1. 앙상블 기법과 회귀 기법의 비교 2. 앙상블 기법간 비교를 들 수 있다. 각 근거에 대한 구체적 설명은 다음과 같다.

첫번째로 앙상블 기법과 회귀 기법 간의 비교이다. 해당 데이터는 수치형 데이터이자 정형 데이터이고 모델의 목적이 수치 데이터인 성적 예측이므로 단순 선형회귀가 효과적일 것이라는 가설을 세웠다. 이에 덧붙여 데이터의 피처가 많으니 각 피처 간에 비선형 패턴이 존재할 것이므로 앙상블 기법을 동시에 검증하겠다는 계획을 세웠다.

두번째로는 앙상블 기법 간 비교이다. Random Forest 는 앙상블 기법의 대표적인 방안 중 하나로 배깅의 이점을 지니며 변수를 임의로 선택해 예측력을 높이며, XGBoost 는 과적합 방지와 비선형 패턴 감지에 장점이 있지만 학습 속도가 느릴 수 있다는 단점을 지니며, LightGBM 은 학습속도가 빠르다는 장점이 있지만 과적합이 발생할 수 있다는 단점을 지닌다. 따라서 본 프로젝트에서는 어떤 앙상블 기법이 우위를 갖는지 확인하고자 했다.

VI. 모델 구현

본 챕터에서는 이 프로젝트에서 활용한 주요 방법론들에 대해 소개하고자 한다.

1. 야수 피처 및 레이블 선택

야수의 피처는 타율, 경기수, 타석, 타수, 득점, 총안타, 2 루타, 3 루타, 홈런, 루타, 타점, 도루, 희타, 희비, 4 사구, 삼진, 병살, 장타율, 출루율, OPS 의 총 20 개 열로 구성되어 있다. 이 중 타율, 득점, 홈런, 타점, 4 사구, OPS 를 선택했다. 특히나 출루율과 장타율을 제외했다. 해당 모델은 타자의 생산성을 나타내는 OPS 를 예측하는 모델로 설계하고자 했는데, OPS 는 출루율과 장타율의 합으로 나타내지기 때문에 피처들 간 다중 공선성 문제를 일으킬 가능성이 있다 판단하여 제외했다.

```
# 필요한 열 선택 (예: player_id, game_id, 성적 지표 등)
data_selected = data[['player_id', 'game_tag', '타율', '득점', '홈런', '타점', '4사구', '출루율', '장타율', 'OPS']]

# 데이터 변환 (예: player_id와 game_id를 기준으로 새로운 데이터 프레임 생성)
data_transformed = {
    'player_id': [],
    'game_tag': [],
    '타율': [],
    '득점': [],
    '홈런': [],
    '타점': [],
    '4사구': [],
    'OPS': []
}
```

야수 레이블의 경우 최근 타자의 생산성을 평가하기에 유용하다 여겨지는 OPS 으로 선정했다.

2. 투수 피처 및 레이블 선택

투수의 피처는 평균자책점, 경기수, 승, 패, 이닝, 타자, 피안타, 피홈런, 4 사구, 탈삼진, 실점, 자책점, 승률, WHIP(이닝당 주자 허용률)의 14 개 열로 구성되어 있다. 이 중 평균자책점, 피안타, 4 사구, 탈삼진을 선택하여 데이터로 활용했다. 이 성적들은 외부의 영향을 덜 받고 투수 본연의 실력을 평가하기에 용이하다고 판단되어 선택했다.

```
# 필요한 열 선택 (예: player_id, game_id, 성적 지표 등)
data_selected = data[['player_id', 'game_tag', '평균자책점', '피안타', '4사구', '탈삼진', 'WHIP']]

# 데이터 변환 (예: player_id와 game_id를 기준으로 새로운 데이터 프레임 생성)
data_transformed = {
    'player_id': [],
    'game_tag': [],
    '평균자책점': [],
    '피안타': [],
    '4사구': [],
    '탈삼진': [],
    'WHIP': []
}
```

투수 레이블의 경우 투수의 능력을 종합적으로 판단할 수 있고, 여러 기록이 누적되어 계산되는 평균자책점으로 선택했다.

3. 데이터 증강

데이터를 직접 수집하긴 했으나 그 수가 다른 여타 데이터셋에 비해 적다는 단점이 존재했다. 따라서 적절한 방법을 활용하여 데이터를 증강할 필요가 있다 판단했다. 선수들의 성적 지표가 주로 주말리그(전반기), 토너먼트 대회(주말리그 왕중왕전), 주말리그(후반기), 토너먼트 대회(주말리그 왕중왕전)의 흐름으로 진행되는 것을 고려할 때 선수들의 성적은 시계열 데이터의 성질을 지니고 있다고 볼 수 있다.

즉 성적 변동을 활용하면 시즌의 흐름, 선수의 기량 변화, 슬럼프 등의 요소 등을 내포한 증강 데이터를 형성할 수 있게 된다. 해당 프로젝트에선 성적들의 표준편차를 계산하여 노이즈를 포함한 데이터를 추가했다.

```
player_stats_std = data_selected.groupby('player_id').agg({
    '평균자책점': 'std',
    '피안타': 'std',
    '탈삼진': 'std',
    '4 사구': 'std',
```

```
    'WHIP': 'std',
}).reset_index()

# 컬럼 이름 변경
player_stats_std.columns = ['player_id', '평균자책점_표준편차',
'피안타_표준편차', '탈삼진_표준편차', '4 사구_표준편차','WHIP_표준편차']

# 결과 확인
print(player_stats_std)
```

해당 코드를 활용하여 선수 개별 id 를 기준으로 해당 선수 데이터들의 표준편차를 계산한다.

```
# 노이즈 추가 함수
def augment_data_with_fluctuation(row, std_multipliers,
noise_level=0.1):
    augmented_row = row.copy()
    for col, std_col in std_multipliers.items():
        noise = np.random.normal(0, noise_level * row[std_col])
        augmented_row[col] += noise
    return augmented_row
```

augment_data_with_fluctuation 함수를 통해 데이터를 증강한다. 데이터 증강을 위해 앞서 계산된 선수의 성적 표준편차를 활용해 정규분포를 따르는 랜덤 노이즈를 생성하고, 이를 더하여 증강 데이터를 생성하게 된다. 이후 이 과정을 4 번 반복하여 원본 데이터 5 배 크기의 추가 데이터셋을 형성하게 된다. 이렇게 얻어진 약 6500 행의 데이터를 모델 학습에 사용했다.

4. 데이터 학습 및 평가 - 야수

아래는 각 모델 학습 및 평가 지표 출력 결과이다. 각 모델 구현 코드 전문은 Appendix A.에 수록되었다.

1) Random Forest 결과

```
Mean Squared Error: 0.008380862802255917
R^2 Score: 0.95825330102502
Fitting 3 folds for each of 108 candidates, totalling 324 fits
Best Mean Squared Error: 0.008324121780100531
Best R^2 Score: 0.9585359390334619
```

2) LightGBM 결과

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.004174 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2352
[LightGBM] [Info] Number of data points in the train set: 5160, number of used features: 11
[LightGBM] [Info] Start training from score 0.900570
Mean Squared Error: 0.012176088773516894
R^2 Score: 0.9391780498439708
Fitting 3 folds for each of 27 candidates, totalling 81 fits
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001174 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2352
[LightGBM] [Info] Number of data points in the train set: 5160, number of used features: 11
[LightGBM] [Info] Start training from score 0.900570
Best Mean Squared Error: 0.006825965413335846
```

```
Best R^2 Score: 0.9659029647484431
['best_lgb_model.pkl']
```

3) XGBoost 결과

```
Mean Squared Error: 0.008838086442135764
R^2 Score: 0.9558520257976901
Fitting 3 folds for each of 27 candidates, totalling 81 fits
Best Mean Squared Error: 0.007473204443997614
Best R^2 Score: 0.9626698789198008
['best_xgb_model.pkl']
```

4) Linear Regression 결과

```
Mean Squared Error: 0.03833990910494065
R^2 Score: 0.8084846387093273
['linear_regression_model.pkl']
```

위와 같이 모델을 설계하고 데이터 학습 및 평가를 진행했으며, 그 결과를 표로 정리하면 다음과 같다.

	Mean Square Error (MSE)	R^2 Score
Random Forest	0.0084	0.958
LightGBM	0.0068	0.966
XGBoost	0.0074	0.963
Linear Regression	0.0383	0.808

Table 1. 야수 성적 예측 모델 MSE, R^2 Score 비교 분석

5. 데이터 학습 및 평가 - 투수

1) Random Forest 결과

```
Mean Squared Error: 1.5632976601362136
R^2 Score: 0.9568878933292769
Fitting 3 folds for each of 108 candidates, totalling 324 fits
Best Mean Squared Error: 1.5706841266400193
Best R^2 Score: 0.9566841917950444
['best_model_rf.pkl']
```

2) LightGBM 결과

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000631 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2184
[LightGBM] [Info] Number of data points in the train set: 5436, number of used features: 9
[LightGBM] [Info] Start training from score 3.681200
Mean Squared Error: 2.586324431665214
R^2 Score: 0.9286751988272454
Fitting 3 folds for each of 27 candidates, totalling 81 fits
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000559 seconds.
You can set `force_col_wise=true` to remove the overhead.
```

```
[LightGBM] [Info] Total Bins 2184
[LightGBM] [Info] Number of data points in the train set: 5436, number of used features: 9
[LightGBM] [Info] Start training from score 3.681200
Best Mean Squared Error: 1.4103718620296362
Best R^2 Score: 0.9611052304933981
['best_model_lgb.pkl']
```

3) XGBoost 결과

```
Mean Squared Error: 1.5474994239396078
R^2 Score: 0.9573235718705331
Fitting 3 folds for each of 27 candidates, totalling 81 fits
Best Mean Squared Error: 1.2589076566023158
Best R^2 Score: 0.9652822603372284
['best_model_xgb.pkl']
```

4) Linear Regression 결과

```
Mean Squared Error: 11.778397389080487
R^2 Score: 0.6751792460279384
['linear_regression_model.pkl']
```

위와 같이 모델을 설계하고 데이터 학습 및 평가를 진행했으며, 그 결과를 표로 정리하면 다음과 같다.

	Mean Square Error (MSE)	R^2 Score
Random Forest	1.5706	0.957
LightGBM	1.4103	0.961
XGBoost	1.2589	0.965
Linear Regression	11.7783	0.675

Table 2. 투수 성적 예측 모델 MSE, R^2 Score 비교 분석

6. 잔차 분석을 통한 결과 검증

위와 같이 모델을 설계하고 학습한 결과 R^2 score 의 경우 LightGBM > XGBoost > Random Forest > Linear Regression 순으로 LightGBM 모델이 가장 높았고, MSE 의 경우 0.0383 > 0.0084 > 0.0074 > 0.0068 순으로 LightGBM 이 가장 낮았다.

본 프로젝트에서는 이에 그치지 않고 잔차 분석을 추가로 수행했다. 야구 데이터의 경우 타율과 OPS 등의 지표는 소수점 단위로 측정이 되고, 안타와 홈런 등의 지표는 상대적으로 작은 자연수 범위를 갖는다는 특징이 있다. 이러한 데이터 특성으로 인해 분산이 작고 비교적 일관된 값이 나타날 가능성이 큰데, 이러한 특성 때문에 모델 성능 평가 시 데이터의 통계적 특성에 의한 우연한 좋은 결과가 나타날 수 있다는 가설을 세웠다. 이를 검증하고자 잔차를 계산하고 그 분포를 히스토그램으로 시각화 했다.

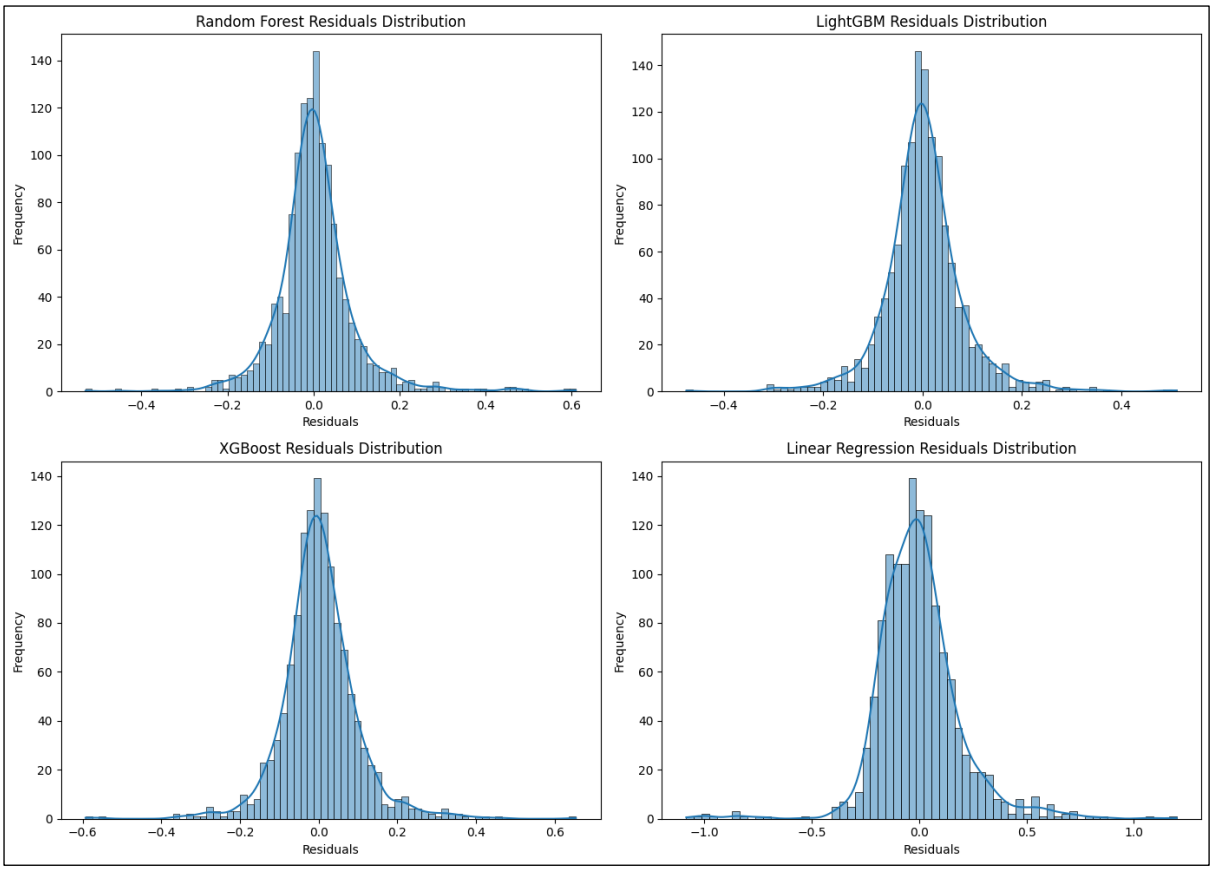


Figure 1-야수 모델 잔차 분석

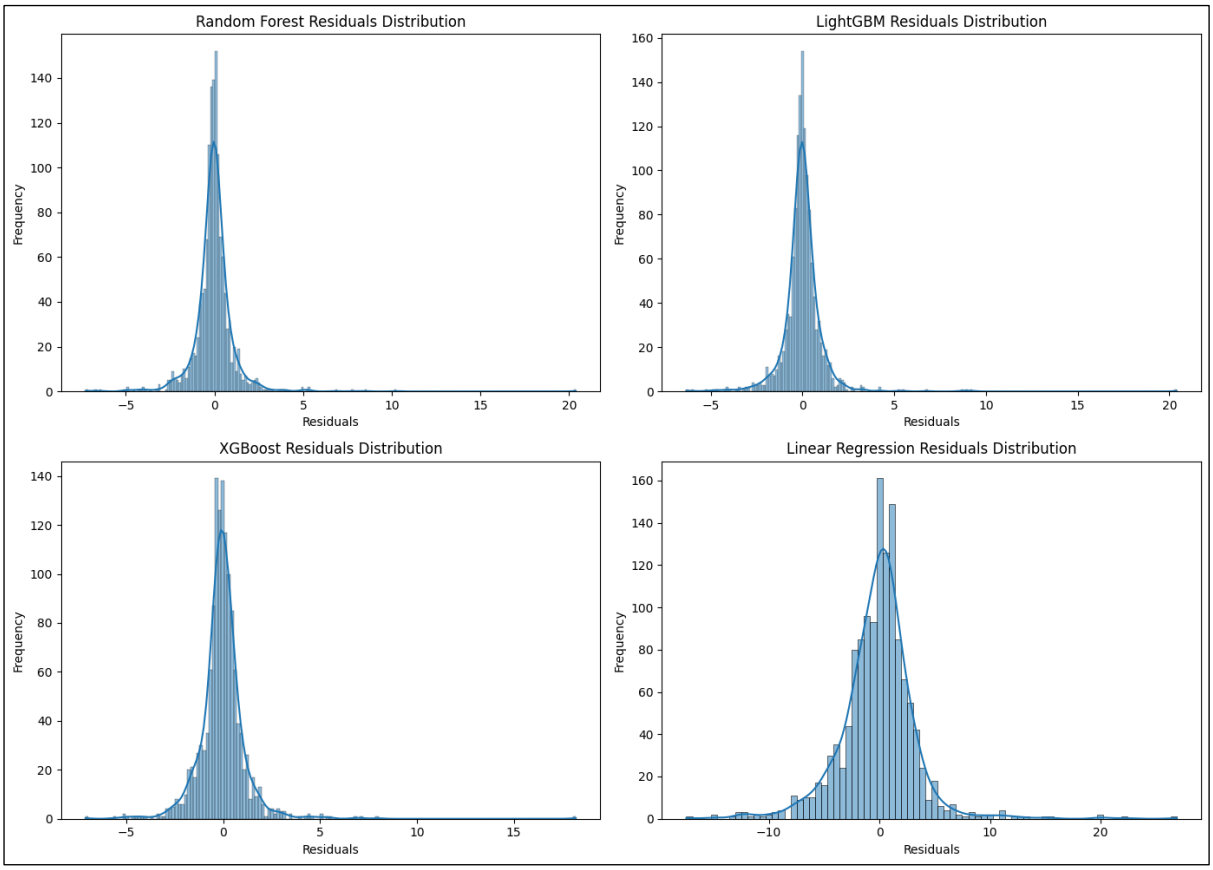


Figure 2 - 투수 모델 잔차 분석

해당 잔차 분석으로 도출한 결론은 다음과 같다.

1. 앙상블 기법의 경우 잔차 또한 정규분포를 따르며 평균이 0 에 가깝게 계산되었다.
이는 두 모델 모두 데이터를 고르게 반영하고 있으며 그 구조를 적절히 학습했음을 의미한다.
2. 두 앙상블 기법 중에서는 XGBoost 모델이 LightGBM 모델보다 더 좋은 성능을 보였다.
3. 선형 회귀의 경우 잔차 분석 결과 분산이 앙상블 기법보다 컸다. 따라서 앙상블 기법을 적용한 모델이 더욱 데이터의 패턴을 잘 분석했다 해석할 수 있다.

Ⅶ. 결론

1. 주요 성과

위와 같은 일련의 탐구 과정을 통해 해당 모델은 고교야구 선수의 성적을 예측하는데 활용할 수 있음을 확인했다. 해당 모델들은 야수의 경우 OPS, 투수의 경우 평균자책점을 예측하는 데 있어 그 예측 수치가 신뢰할만한 수준이었다. 앙상블 기법은 선형회귀보다 우수한 성능을 보여주어 해당 데이터 패턴을 적절히 학습했음을 보여주었다. 앙상블 기법 중에서는 LightGBM 과 XGBoost 가 좋은 성능을 보여주었다.
이 모델들은 단기적으로는 스카우터의 선수 가치 판단을 돕는 도구로서, 장기적으로는 부족한 경기 표본으로 인한 판단의 어려움을 해소하고 학생 선수들의 부담감을 덜어줄 수 있는 장치가 될 것이라 기대한다.

2. 모델의 강점 및 약점

해당 모델의 강점과 약점을 정리하면 다음과 같다.

〈강점〉
1. 특정 레이블에 대한 예측 능력이 우수하다.
2. 모델 학습의 결과의 분산이 작아 신뢰도가 높다.
3. 도메인 지식을 추가한다면 실제로도 활용 가능하다.

〈약점〉
1. 야구는 성적 지표로만 설명하기엔 어려움이 있는 스포츠다.
2. 특정 한 예측 성적만을 가지고 선수를 고르기엔 한계점이 존재한다.
3. 데이터 증강 과정을 거치긴 했으나 그럼에도 데이터가 충분하다 말하기에 어렵다.

3. 추후 연구 방향 및 개선 방안

해당 프로젝트는 이후 다음과 같은 연구 방향으로 나아간다면 더욱 활용도가 높아질 것으로 예상된다.
첫번째는 데이터 확충이다. 이 프로젝트는 확실한 명단 확보가 가능하고 프로에 지명 받을 만한 실력이 갖추어진 프로야구 지명 선수들의 데이터를 확보했다. 다만 이 같은 경우 실력이 뛰어난 선수 위주로 데이터가 수집되므로 편향된 데이터를 학습할 수 있다. 따라서 실력이 뛰어난 선수와 그렇지 못한 선수 모두의 데이터를 수집하여 학습시킨다면 일반성을 확보할 수 있을 것으로 기대한다.
두번째는 데이터 증강 방법 개선이다. 이 프로젝트에서는 슬럼프, 타격 감각 등락이 성적 변동에 반영되었을 것이라는 가설을 세웠고, 이를 바탕으로 표준편차를 사용해 가상 데이터를

생성해 데이터를 증강했다. 다만 표준편차를 사용하는 방법 외에도 데이터 속 패턴을 유지하면서 신뢰도 있는 데이터를 증강할 수 있는 방법이 있을 것으로 추측한다. 이 방법을 연구하여 적용한다면 더 많은 데이터를 통한 모델 학습이 가능할 것이다.
마지막으로는 현장의 도메인 지식과의 결합이다. 이 프로젝트에서는 사후적인 성적 데이터를 학습 데이터로 활용했다. 더 다각적인 분석을 위해선 타구 발사각, 투구 RPM 등 퍼포먼스에 대한 데이터, 훈련에 대하는 태도, 20-80 스케일 평가 등의 데이터까지도 활용할 수 있을 것이다. 현장의 지도자들, 실제 신인 선발 업무를 담당하는 스카우터와의 협력을 통해 도메인 정보를 추가한다면 더욱 활용도 높은 모델을 개발할 수 있을 것이라 예측한다.

<참고문헌>

Kriz PK, Staffa SJ, Kriz JP, DeFroda S. Ulnar Collateral Ligament Tear in Elite Baseball Pitchers: Are High School Showcase Exposures Associated With Injury? The American Journal of Sports Medicine. 2022;50(11):3073-3082.
doi:10.1177/03635465221113859

Appendix
Code

1. 야수 모델 구현 코드

〈Random Forest〉

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
import joblib

# 데이터 로드
data = augmented_df

# 필요 없는 열 제거 (예: 'game_tag')
data = data.drop(columns=['game_tag', 'player_id'])

# 결측치 처리 (필요한 경우)
data = data.fillna(data.mean())

# 특성(X)와 타깃(y) 분리
X = data.drop(columns=['OPS'])
y = data['OPS']

# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# 모델 초기화 및 학습
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# 모델 평가
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R^2 Score: {r2}')

# 모델 튜닝 (선택 사항)
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

```
grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
cv=3, n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)
best_model_rf = grid_search.best_estimator_

# 최적의 모델 평가
y_pred_best = best_model_rf.predict(X_test)
mse_best = mean_squared_error(y_test, y_pred_best)
r2_best = r2_score(y_test, y_pred_best)
print(f'Best Mean Squared Error: {mse_best}')
print(f'Best R^2 Score: {r2_best}')

# 모델 저장
joblib.dump(best_model_rf, 'best_model_rf.pkl')
```

〈LightGBM〉

```
# 데이터 로드
data = augmented_df

# 필요 없는 열 제거 (예: 'game_tag')
data = data.drop(columns=['game_tag', 'player_id'])

# 결측치 처리 (필요한 경우)
data = data.fillna(data.mean())

# 특성(X)와 타깃(y) 분리
X = data.drop(columns=['OPS'])
y = data['OPS']

# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# LightGBM 모델 초기화 및 학습
model = lgb.LGBMRegressor(random_state=42)
model.fit(X_train, y_train)

# 모델 평가
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R^2 Score: {r2}')
```

```
# 모델 튜닝
param_grid = {
    'num_leaves': [31, 50, 100],
    'learning_rate': [0.01, 0.05, 0.1],
    'n_estimators': [100, 200, 500]
}
grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
cv=3, n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_

# 최적의 모델 평가
y_pred_best = best_model.predict(X_test)
mse_best = mean_squared_error(y_test, y_pred_best)
r2_best = r2_score(y_test, y_pred_best)
print(f'Best Mean Squared Error: {mse_best}')
print(f'Best R^2 Score: {r2_best}')

# 모델 저장
joblib.dump(best_model, 'best_lgb_model.pkl')
```

〈XGBoost〉

```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
import joblib

# 데이터 로드
data = augmented_df

# 필요 없는 열 제거 (예: 'game_tag')
data = data.drop(columns=['game_tag', 'player_id'])

# 결측치 처리 (필요한 경우)
data = data.fillna(data.mean())

# 특성(X)와 타깃(y) 분리
X = data.drop(columns=['OPS'])
y = data['OPS']

# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

```
# XGBoost 모델 초기화 및 학습
model = xgb.XGBRegressor(random_state=42)
model.fit(X_train, y_train)

# 모델 평가
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R^2 Score: {r2}')

# 모델 튜닝
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.05, 0.1],
    'n_estimators': [100, 200, 500]
}
grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
cv=3, n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_

# 최적의 모델 평가
y_pred_best = best_model.predict(X_test)
mse_best = mean_squared_error(y_test, y_pred_best)
r2_best = r2_score(y_test, y_pred_best)
print(f'Best Mean Squared Error: {mse_best}')
print(f'Best R^2 Score: {r2_best}')

# 모델 저장
joblib.dump(best_model, 'best_xgb_model.pkl')
```

〈Linear Regression〉

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import joblib

# 데이터 로드
data = augmented_df

# 필요 없는 열 제거 (예: 'game_tag')
data = data.drop(columns=['player_id', 'game_tag'])

# 결측치 처리 (필요한 경우)
```

```
data = data.fillna(data.mean())

# 특성(X)와 타깃(y) 분리
X = data.drop(columns=['OPS'])
y = data['OPS']

# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# 선형 회귀 모델 초기화 및 학습
model = LinearRegression()
model.fit(X_train, y_train)

# 모델 평가
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R^2 Score: {r2}')

# 모델 저장
joblib.dump(model, 'linear_regression_model.pkl')
```

2. 투수 모델 구현 코드

〈Random Forest〉

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
import joblib

# 데이터 로드
data = augmented_df

# 필요 없는 열 제거 (예: 'game_tag')
data = data.drop(columns=['player_id', 'game_tag'])

# 결측치 처리 (필요한 경우)
data = data.fillna(data.mean())

# 특성(X)와 타깃(y) 분리
X = data.drop(columns=['평균자책점'])
y = data['평균자책점']
```

```
# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# 모델 초기화 및 학습
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# 모델 평가
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R^2 Score: {r2}')

# 모델 튜닝 (선택 사항)
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
cv=3, n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)
best_model_rf = grid_search.best_estimator_

# 최적의 모델 평가
y_pred_best = best_model_rf.predict(X_test)
mse_best = mean_squared_error(y_test, y_pred_best)
r2_best = r2_score(y_test, y_pred_best)
print(f'Best Mean Squared Error: {mse_best}')
print(f'Best R^2 Score: {r2_best}')

# 모델 저장
joblib.dump(best_model, 'best_model_rf.pkl')
```

〈LightGBM〉

```
import lightgbm as lgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
import joblib
# 데이터 로드
```

```
data = augmented_df

# 필요 없는 열 제거 (예: 'game_tag')
data = data.drop(columns=['game_tag', 'player_id'])

# 결측치 처리 (필요한 경우)
data = data.fillna(data.mean())

# 특성(X)와 타깃(y) 분리
X = data.drop(columns=['평균자책점'])
y = data['평균자책점']

# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# LightGBM 모델 초기화 및 학습
model = lgb.LGBMRegressor(random_state=42)
model.fit(X_train, y_train)

# 모델 평가
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R^2 Score: {r2}')

# 모델 튜닝
param_grid = {
    'num_leaves': [31, 50, 100],
    'learning_rate': [0.01, 0.05, 0.1],
    'n_estimators': [100, 200, 500]
}
grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
cv=3, n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)
best_model_lgb = grid_search.best_estimator_

# 최적의 모델 평가
y_pred_best = best_model_lgb.predict(X_test)
mse_best = mean_squared_error(y_test, y_pred_best)
r2_best = r2_score(y_test, y_pred_best)
print(f'Best Mean Squared Error: {mse_best}')
print(f'Best R^2 Score: {r2_best}')
```

```
# 모델 저장
joblib.dump(best_model, 'best_model_lgb.pkl')
```

〈XGBoost〉

```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
import joblib

# 데이터 로드
data = augmented_df

# 필요 없는 열 제거 (예: 'game_tag')
data = data.drop(columns=['player_id', 'game_tag'])

# 결측치 처리 (필요한 경우)
data = data.fillna(data.mean())

# 특성(X)와 타깃(y) 분리
X = data.drop(columns=['평균자책점'])
y = data['평균자책점']

# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# XGBoost 모델 초기화 및 학습
model = xgb.XGBRegressor(random_state=42)
model.fit(X_train, y_train)

# 모델 평가
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R^2 Score: {r2}')

# 모델 튜닝
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.05, 0.1],
    'n_estimators': [100, 200, 500]
}
```

```
grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
cv=3, n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)
best_model_xgb = grid_search.best_estimator_

# 최적의 모델 평가
y_pred_best = best_model_xgb.predict(X_test)
mse_best = mean_squared_error(y_test, y_pred_best)
r2_best = r2_score(y_test, y_pred_best)
print(f'Best Mean Squared Error: {mse_best}')
print(f'Best R^2 Score: {r2_best}')

# 모델 저장
joblib.dump(best_model, 'best_model_xgb.pkl')
```

〈Linear Regression〉

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import joblib

# 데이터 로드
data = augmented_df

# 필요 없는 열 제거 (예: 'game_tag')
data = data.drop(columns=['player_id', 'game_tag'])

# 결측치 처리 (필요한 경우)
data = data.fillna(data.mean())

# 특성(X)와 타깃(y) 분리
X = data.drop(columns=['평균자책점'])
y = data['평균자책점']

# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# 선형 회귀 모델 초기화 및 학습
model_lr = LinearRegression()
model_lr.fit(X_train, y_train)

# 모델 평가
y_pred = model_lr.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
```

```
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R^2 Score: {r2}')

# 모델 저장
joblib.dump(model_lr, 'linear_regression_model.pkl')
```