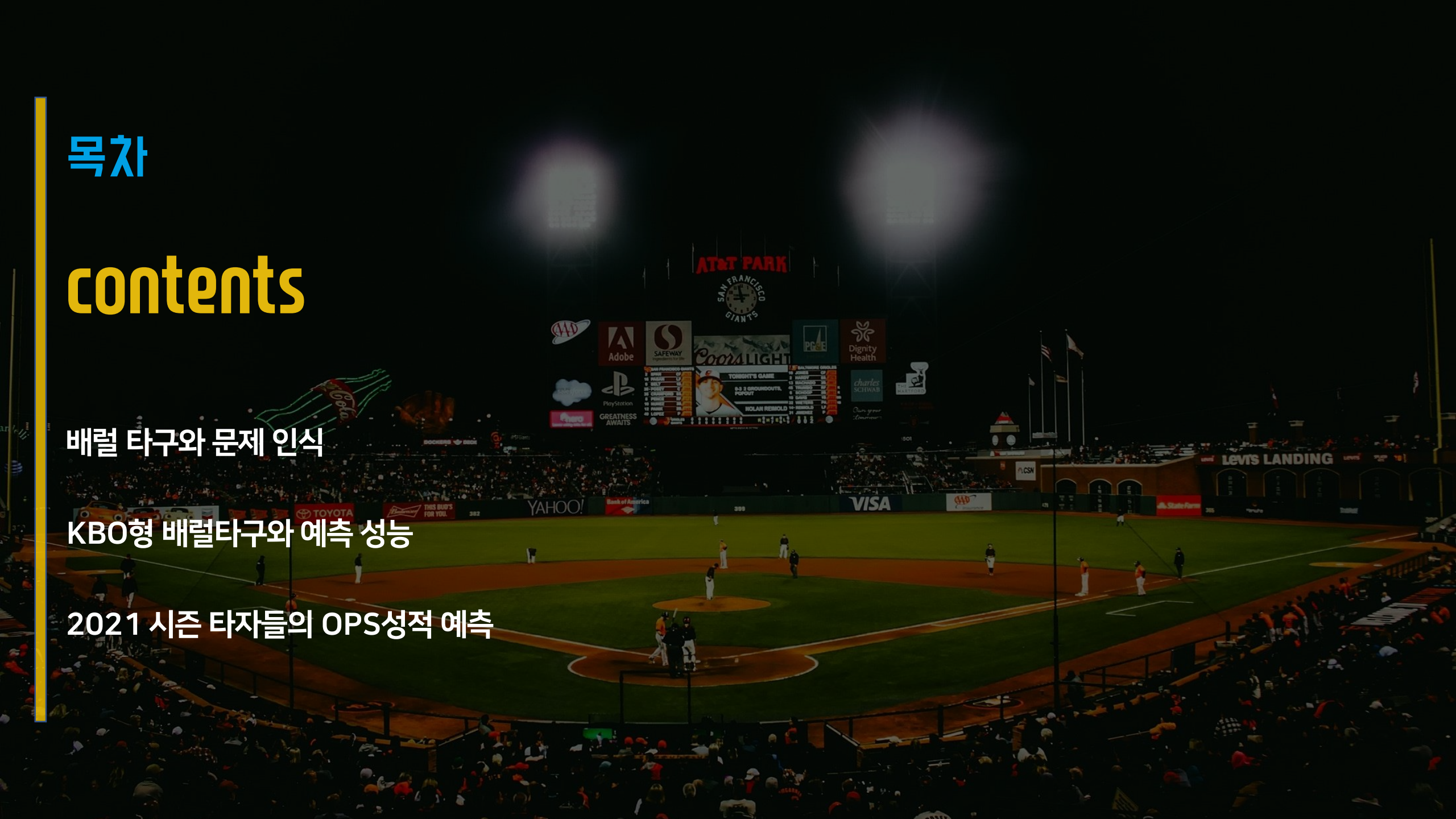
KBO를 위한

배럴타구와 타자 OPS예측모델

TEAM : 참외맛 인생
김기훈-rlgnsno35@naver.com
김지우-rlawldn112@naver.com

목차

contents



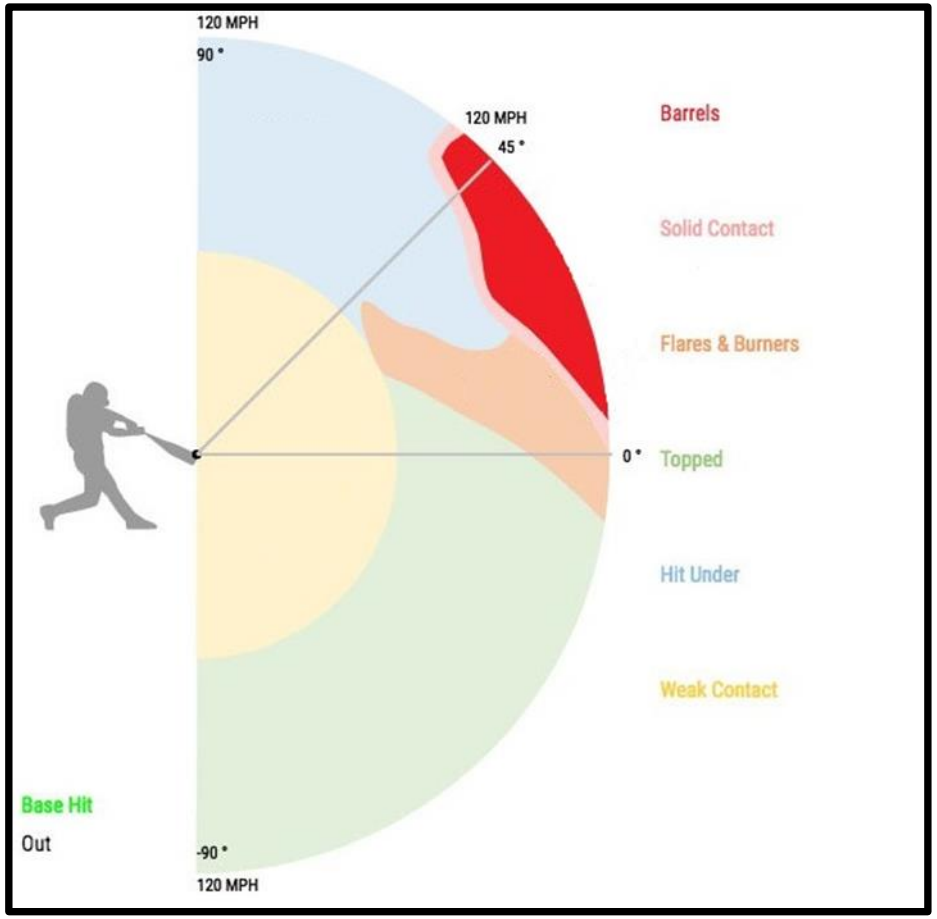
배럴타구란?

배럴타구의 정의

조건1: 발사각 26~30도, 타구속도 98 마일 ⬆

조건2: 100마일 이상 ,
타구속도 1마일 증가 시
, 상하 2~3도 씩 증가

기대효과: 타율 최소 0.500, 장타율 1.500



배럴타구 = 생산성을 높여주는 '좋은 타구'

사진 출처:Beyond the Barrel: An Introduction to Ideal Contact Rate - Pitcher List

배럴타구란?

배럴타구의 효과

오타니 2021 성적

타율: 0.269

장타율:0.644

OPS:1.008

타석당 배럴타구 비율: 25.1%(ML1위)

타율로만으로는 개인의 기여도나 생산성을 판단할 수 없다.
팀 득점 생산에 얼마나 기여할 수 있는지
보여주는 배럴타구로 선수의 폼과 생산성을 판단할 수 있다.

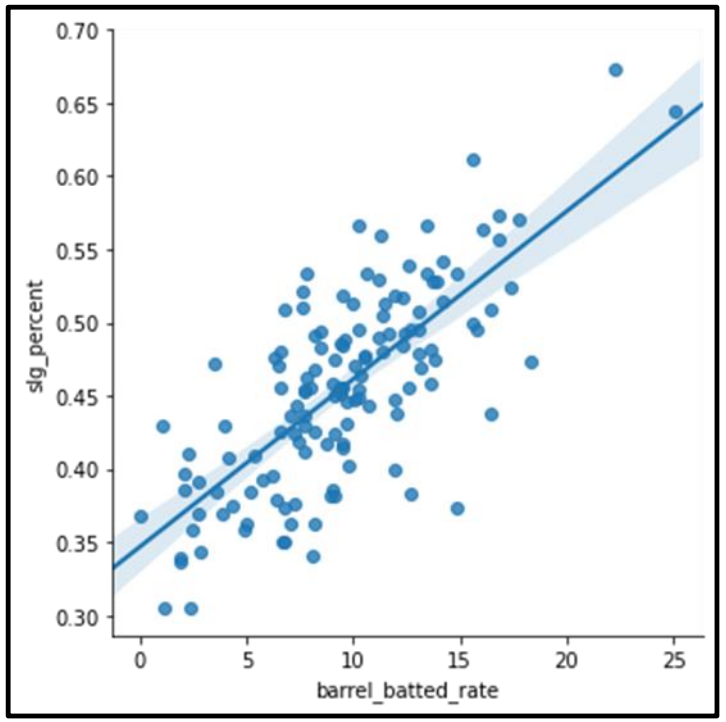
배럴타구란?

배럴타구와 장타율

배럴타구와 장타율은 0.7389의 상관관계

배럴타구 비율이 증가할 수록,

선수의 장타율도 증가한다.



<3년간 ML의 장타율과 배럴타구의 상관관계>

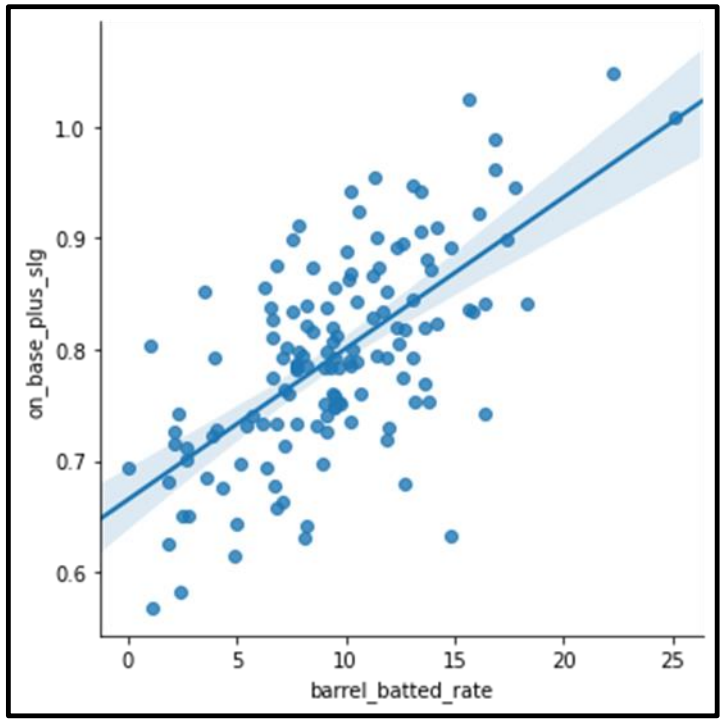
배럴타구란?

배럴타구와 OPS

배럴타구와 OPS는 0.6475의 상관관계

배럴타구 비율이 증가할 수록,

선수의 OPS도 증가한다.



<3년간 ML의 OPS와 배럴타구의 상관관계>

->이는 배럴타구가 생산성을 향상시킨다는 것을 보여준다.

<kbo와 mlb의 타구속도, 발사각 비교>

MLB		KBO	
3년평균타구속도	3년평균타구각도	3년평균타구속도	3년평균타구각도
142.6km/h	12.6	134km/h	17
3년 평균배럴비율: 7.87%		3년 평균배럴배율:1.18%	

KBO는 MLB에 비해 타구각도가 높지만, 타구속도가 느려 배럴타구 발생비율이 현저히 낮다.

	MLB		KBO	
	출루율	장타율	출루율	장타율
2019	0.32	0.418	0.337	0.385
2020	0.32	0.435	0.349	0.409

MLB출처:https://baseballsavant.mlb.com/
KBO출처:https://www.koreabaseball.com/Default.aspx

문제인식

작은 발생 비율

메이저리그 기준 배럴을 KBO에 그대로 가져오기엔 무리가 있다.
생산되는 타구가 다르기 때문이다.
과연 우리는 발생하지 않는 배럴타구를 의미있는 기록으로 볼 수 있을까?
따라서, 이번 대회의 목표대로 새로운 배럴 기준을 설정해야 한다

문제인식

와닿지 않는 단위체계

우리나라 단위체계는 km를 반영,

배럴기준은 , mile을 단위로 삼는다.

Km로 보여주는 기록속에서

팬들과 현장관계자들은

배럴타구 기준을 편하게 여길 수 있을까?

<중계화면 캡쳐>

중계화면에서도, 우리는 km로 된 단위를 본다.

출처: 네이버스포츠_크보연구소 최정 홈런 모음 영상

좋은타구란?

새로운 배럴기준

이에 따라

새로운 배럴기준을 제시해

배럴타구가 기대하는 생산성을 유지하며

높은 발생비율

Km로 단위 변경을 이뤄낼 것입니다.

<NEW-BARREL 기준>

1. 타구 속도가 143~160km인경우
- 타구 속도가 143km이상에 발사각이 26~30도 인 경우 배럴타구
- 타구속도가 3km 빨라질 때마다, 발사각 상한선은 위로 4km 증가, 하한선은 아래로 3km 감소

2. 타구속도가 161km이상인 경우
- 타구속도가 3km 빨라질 때마다, 발사각 상한선은 위로 3km 증가, 하한선은 아래로 2km 감소

해당 기준은 KM 단위로 되어 있어 즉각 확인이 용이하다.

<생산성과, 발생빈도>

1. 타석 당 배럴 타구 발생 비율 :10.6%
2. 타구 발생시 기대 타율: 0.658
3. 타구 발생시 기대 장타율 :1.887

-> 종전 기준 배럴타구 발생비율은 1.18%
->해당 생산성 모두 배럴 타구의 정의상
기대 타율, 및 장타율을 상회(각각 0.500,1.500)

좋은타구란?

보장된 생산성과
높아진 발생빈도

.

새로운 배럴기준을 적용하면

KBO환경에서

배럴타구에서 기대되는

최소 타율 0.500이상과

최소 장타율 1.500이상이 보장된다.

또한 발생빈도 역시 상승한다.

	각도 17도 이하 타구	속도 134km/h이하 타구
타율	0.443	0.221
장타율	0.513	0.245

1루타	17343
2루타	2781
3루타	184
내야안타(1루타)	1190
땅볼아웃	18386
번트아웃	4
번트안타	5
병살타	2156
삼중살타	2
야수선택	31
직선타	2416
파울플라이	16
플라이	1150
홈런	11
희생번트	24
희생플라이	85

<각도 17도 이하 타구의 분포>

1루타	7245
2루타	775
3루타	50
내야안타(1루타)	616
땅볼아웃	9026
번트아웃	6
번트안타	6
병살타	876
야수선택	20
인필드플라이	219
직선타	915
파울플라이	2683
플라이	13563
홈런	4
희생번트	27
희생플라이	408

<속도 134km/h 이하 타구의 분포>

기준선정 이유

평균 분석

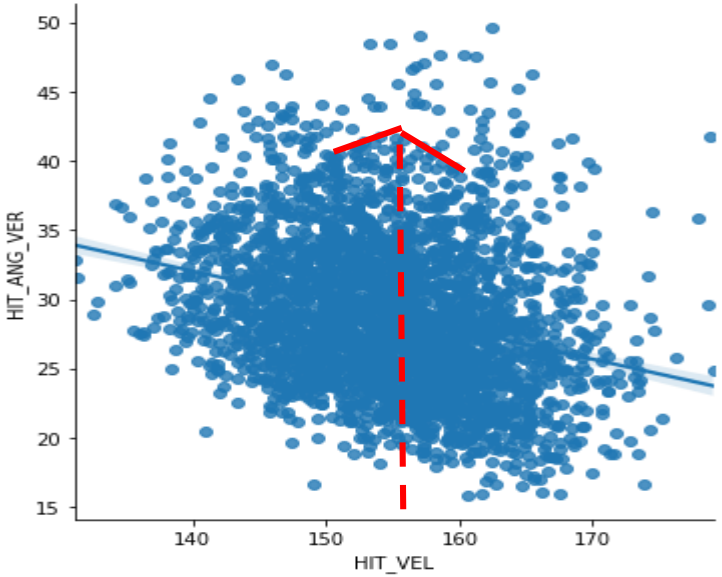
평균적인 타구속도에서는 플라이가 많이 나온다는 것을 알 수 있다.
즉, 각도가 안 좋더라도 속도가 빠르면 수비를 뚫을 수 있지만, 각도가 좋더라도 속도가 빠르지 않으면 수비를 뚫을 수 없다.

->생산성 높은타구의 발생비율을 높이려면 평균이상의 타구 속도를 기준으로, ML 발사각기준보다 폭넓게 타구들을 포함해야한다.

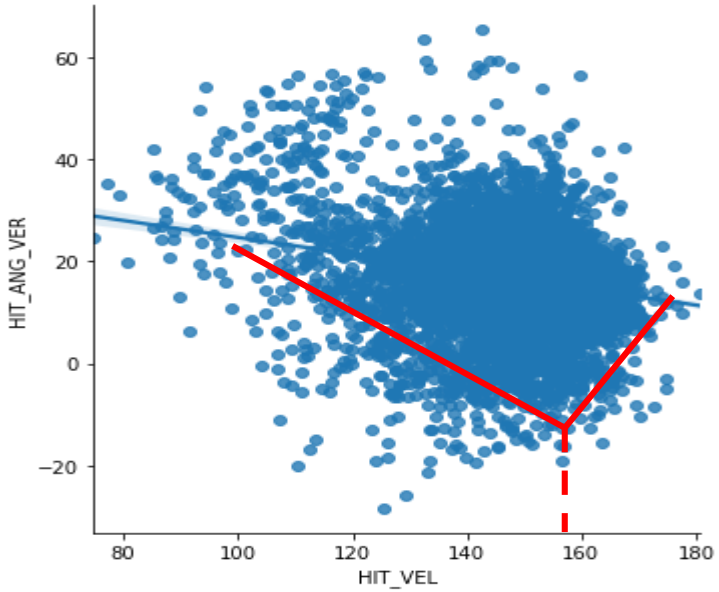
->최저타구속도 기준 143의 이유

WHY?

장타 분석



<3년간 홈런타구의 분포>
평균적으로 28도에서 154.8km/h로 날아갔다.
140km이상, 15도 이상은 되야 홈런이 나오는 것을 볼 수 있다.

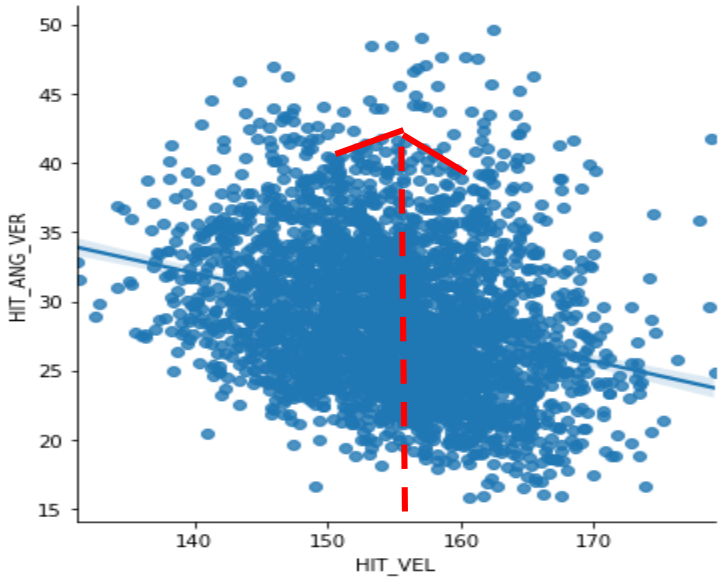


<3년간 2루타,3루타의 분포>
평균적으로 17도에서 146.6km/h로 날아갔다.
120km이상, -10도 이상부터 분포가 많아지는 것을 볼 수 있다.

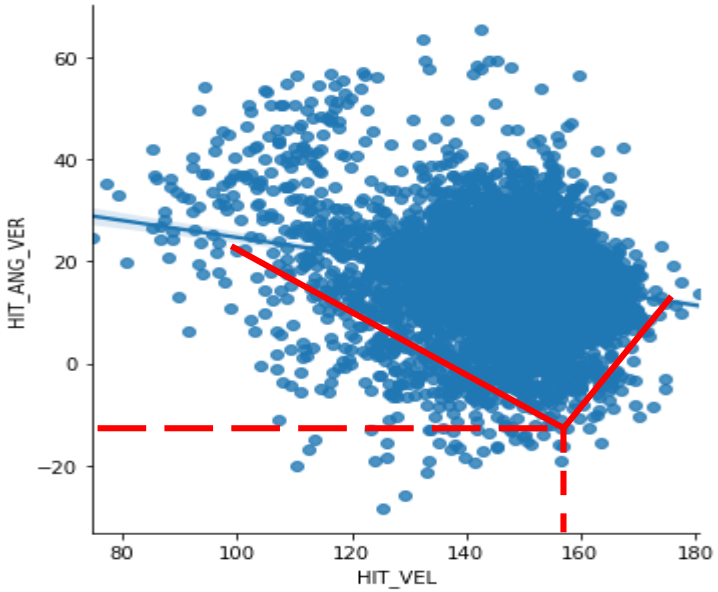
타구분포를 보면
최대 각도가 80도를 넘지 않는다.
또한 발사각 0도 아래로 내려가면
발생빈도가 급격히 줄어든다.

WHY?

장타 분석



<3년간 홈런타구의 분포>
홈런타구의 산포도를 통해 ,
발사각 155KM 부근에서,
40도 이상에서 발생한 홈런 분포가
줄어든다



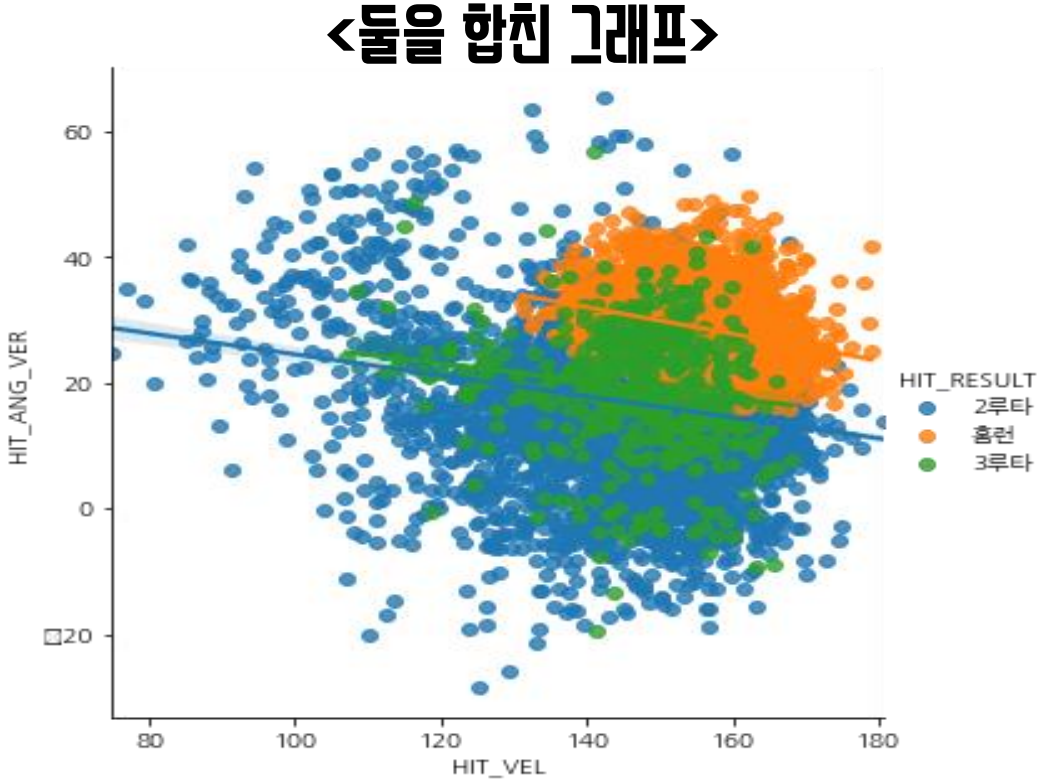
<3년간 2루타,3루타의 분포>
2,3루타의 분포를 통해
타구속도 159~160KM 부근부터
-10보다 더 낮은 발사각에서 발생한
2,3루타 분포가 줄어든다.

홈런은 타구속도 155KM, 발사각40도
2,3루타는 타구속도 159KM,발사각 -10도
해당지점보다 높거나 낮은 발사각인 경우
타구의 발생빈도가 현격히 줄어든다.

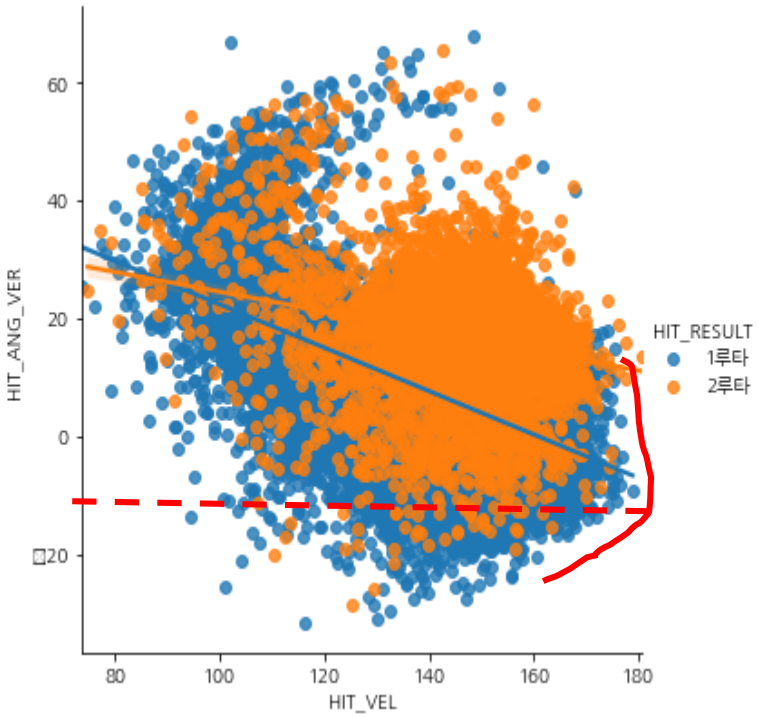
따라서 배럴기준은,
해당지점까지 빠르게
발사각 상하한선을 바꿔야한다.
->따라서 160이하인 경우
타구속도 3KM증가마다,
발사각 상한선은
4KM,
발사각 하한선은
3KM씩 변화한다.

WHY?

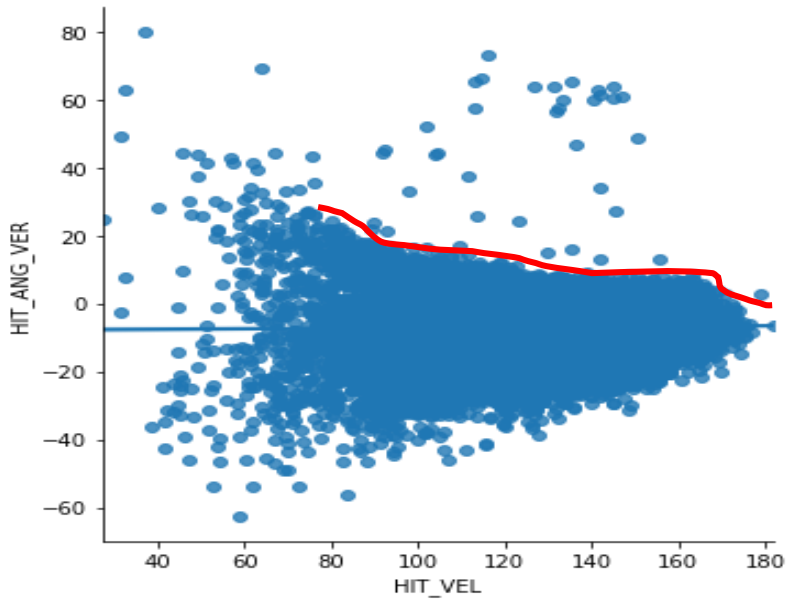
장타 분석



배럴 타구 기준은
해당 타구들을 최대한 포함 해야한다.



<3년간 1,2루타의 분포>
평균적으로 10.05도에서
140.2km/h로 날아갔다.



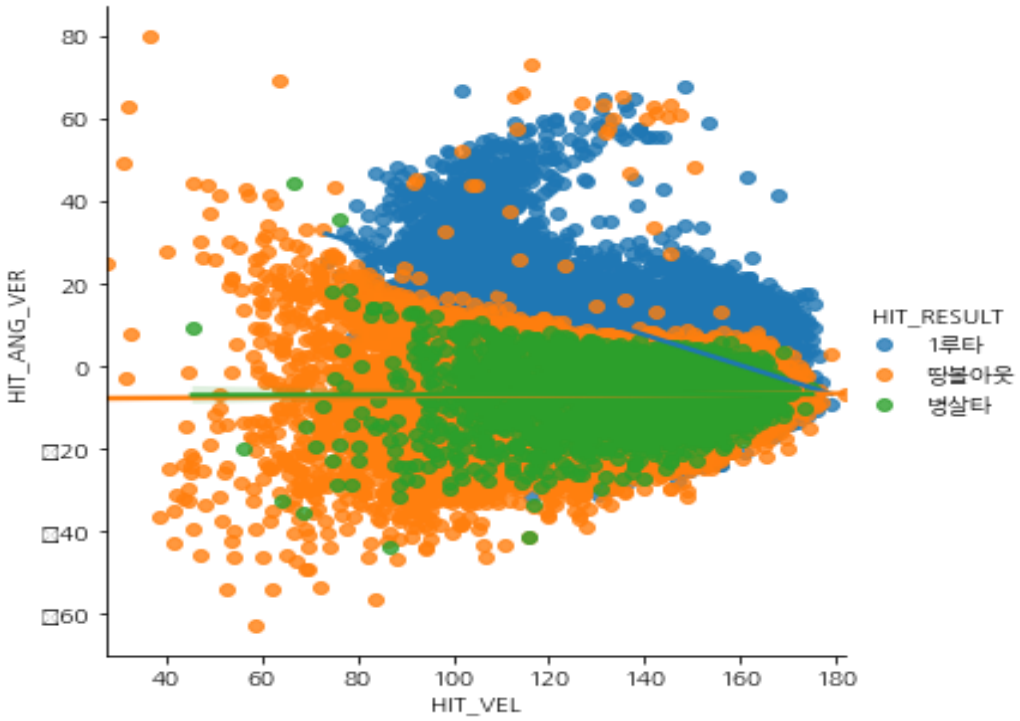
<3년간 땅볼아웃, 병살타의 분포>
평균적으로 -6.88도에서
131.19km/h로 날아갔다.

WHY?

안타와 아웃의 차이

발사각이 0도 보다 낮은 경우
땅볼 아웃이 될 가능성이
비약적으로 높아지지만,
발사각 -10도 부근의 타구라도,
타구 속도가 160보다 빠르다면,
내야를 뚫는 타구를 만들 가능성이 높다.

->따라서 160이상의 타구에서
발사각 하한선이 빠르게 낮아지지 않도록
조절해야한다.
->아웃인 타구를 최대한 배제하면서,
내야를 뚫는 타구를 포함해야하기 때문.



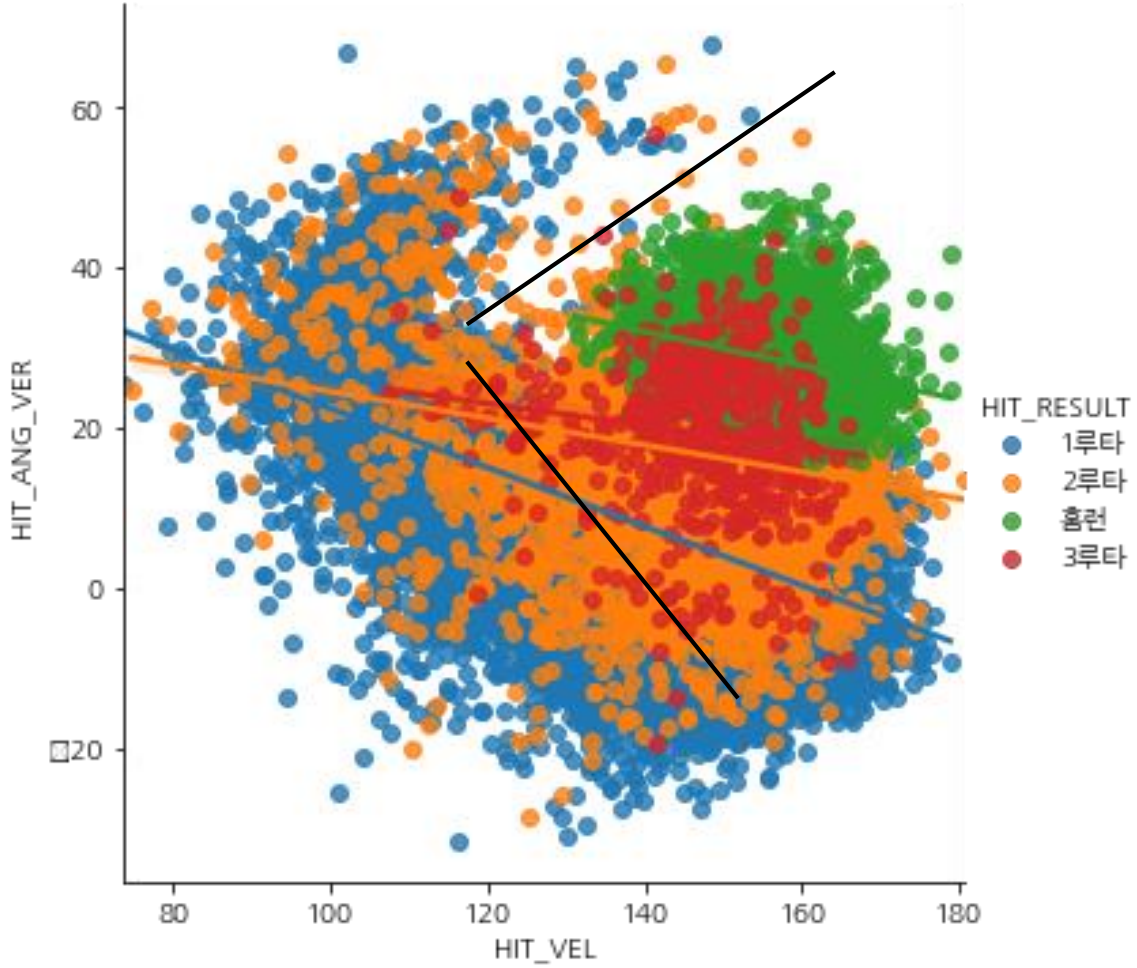
둘을 합친 그래프.
각도의 중요성을 알 수 있다.

WHY?

안타와 아웃의 차이

타구속도는 비슷하나
각도에서 큰 차이를 보인다.

각도기준의 하한선 조정이 필요하다.
이를 바탕으로 161KM이상의 타구부터
상한선은 3도씩,
하한선은 2도씩 변화한다.



<전체 타구 분포>
두 검은 선 사이의 타구가
배럴이라 할 수 있다.

WHY?

새로운 배럴기준

장타를 최대한 많이
포함하면서도 적정 비율
(메이저리그 3년 기준 7.8%)을
넘어서는 것이 중요하다.

<배럴기준코드>

```
#새로 만든 배럴기준
def barrel(df):
    if (df['HIT_VEL']>=143)&(df['HIT_ANG_VER']<=30)&(df['HIT_ANG_VER']>=26):
        return 1
    elif (df['HIT_VEL']>=146)&(df['HIT_ANG_VER']<=34)&(df['HIT_ANG_VER']>=23):
        return  1
    elif (df['HIT_VEL']>=149)&(df['HIT_ANG_VER']<=38)&(df['HIT_ANG_VER']>=20):
        return  1
    elif(df['HIT_VEL']>=152)&(df['HIT_ANG_VER']<=42)&(df['HIT_ANG_VER']>=17):
        return  1
    elif(df['HIT_VEL']>=155)&(df['HIT_ANG_VER']<=46)&(df['HIT_ANG_VER']>=14):
        return  1
    elif(df['HIT_VEL']>=158)&(df['HIT_ANG_VER']<=50)&(df['HIT_ANG_VER']>=11):
        return  1
    elif(df['HIT_VEL']>=161)&(df['HIT_ANG_VER']<=54)&(df['HIT_ANG_VER']>=8):
        return 1
    elif(df['HIT_VEL']>=164)&(df['HIT_ANG_VER']<=57)&(df['HIT_ANG_VER']>=6):
        return 1
    elif (df['HIT_VEL']>=167)&(df['HIT_ANG_VER']<=60)&(df['HIT_ANG_VER']>=4):
        return 1
    elif (df['HIT_VEL']>=170)&(df['HIT_ANG_VER']<=63)&(df['HIT_ANG_VER']>=2):
        return 1
    elif (df['HIT_VEL']>=173)&(df['HIT_ANG_VER']<=66)&(df['HIT_ANG_VER']>=0):
        return 1
    elif (df['HIT_VEL']>=176)&(df['HIT_ANG_VER']<=69)&(df['HIT_ANG_VER']>=-2):
        return 1
    elif (df['HIT_VEL']>=179)&(df['HIT_ANG_VER']<=72)&(df['HIT_ANG_VER']>=-4):
        return 1
    elif (df['HIT_VEL']>=182)&(df['HIT_ANG_VER']<=75)&(df['HIT_ANG_VER']>= -6):
        return  1
    else:
        return 0
```

+4
-3

+3
-2

좋은타구란?

새로운 배럴기준

이에 따라

새로운 배럴기준을 제시해

배럴타구가 기대하는 생산성을 유지하며

높은 발생비율

Km로 단위 변경을 이뤄낼 것입니다.

문제인식

18년도 데이터의 특이성

반발계수가 0.4134~0.4374인 공인구를 사용한 2018 시즌은 무려 1,756개의 홈런을 기록하였습니다.
이렇게 높은 반발력을 가진 야구공은 홈런이 많이 생산되지만 투수의 피로도를 높이고 공격 시간이 길어지는 등의 부작용도 있습니다. 이에 위원회가 공인구의 반발계수를 0.4034~0.4234로 하향 조정한 전례도 있습니다. 그 결과 다음 해인 2019시즌은 총 1,014개 홈런으로 전년 대비 무려 42%가 급감하며 타자들이 고전하는 양상을 보이기도 했습니다.1)

	출루율	장타율	OPS
2018	0.353	0.455	0.808
2019	0.337	0.385	0.722
2020	0.349	0.409	0.758

18년도는 지나친 타고투저이다.
이에 따라 혹자는 18년도 데이터를 포함해 진행하는 것에 의문을 제기할 수 있다.

1)https://m.blog.naver.com/with_msip/221948610273

2)https://www.koreabaseball.com/Default.aspx

	출루율	장타율	OPS	새로운 배럴타구비율(%)
2018	0.353	0.455	0.808	12%
2019	0.337	0.385	0.722	9.6%
2020	0.349	0.409	0.758	10.6%

<전체 18~20시즌 데이터>

	장타율	증감율(%)	새로운 배럴타구비율(%)	증감율(%)
2018	0.455		12	
2019	0.385	-15%	9.6	-20%
2020	0.409	+6.2%	10.6	+10.4%

<18~20시즌 장타율과 배럴타구 비율의 증감비율>

문제인식

2018년 데이터의 배럴비율

- 19시즌 타고투저로 인한 공인구의 반발계수 조정이 있었는데 이로 인해 배럴타구 비율과 장타율이 크게 변화 했다.

그러나
공인구에 관한 논란은
매 시즌 존재했고,
매 시즌 장타율의 증감과,배럴타구비율은
같은 방향으로 증감을 만들었다.

따라서 타구와 결과 양상이
함께간다면,
18시즌 데이터를 배제할 필요가 없다

EDA+배럴기준

사용데이터

사용한 데이터는 다음과 같다

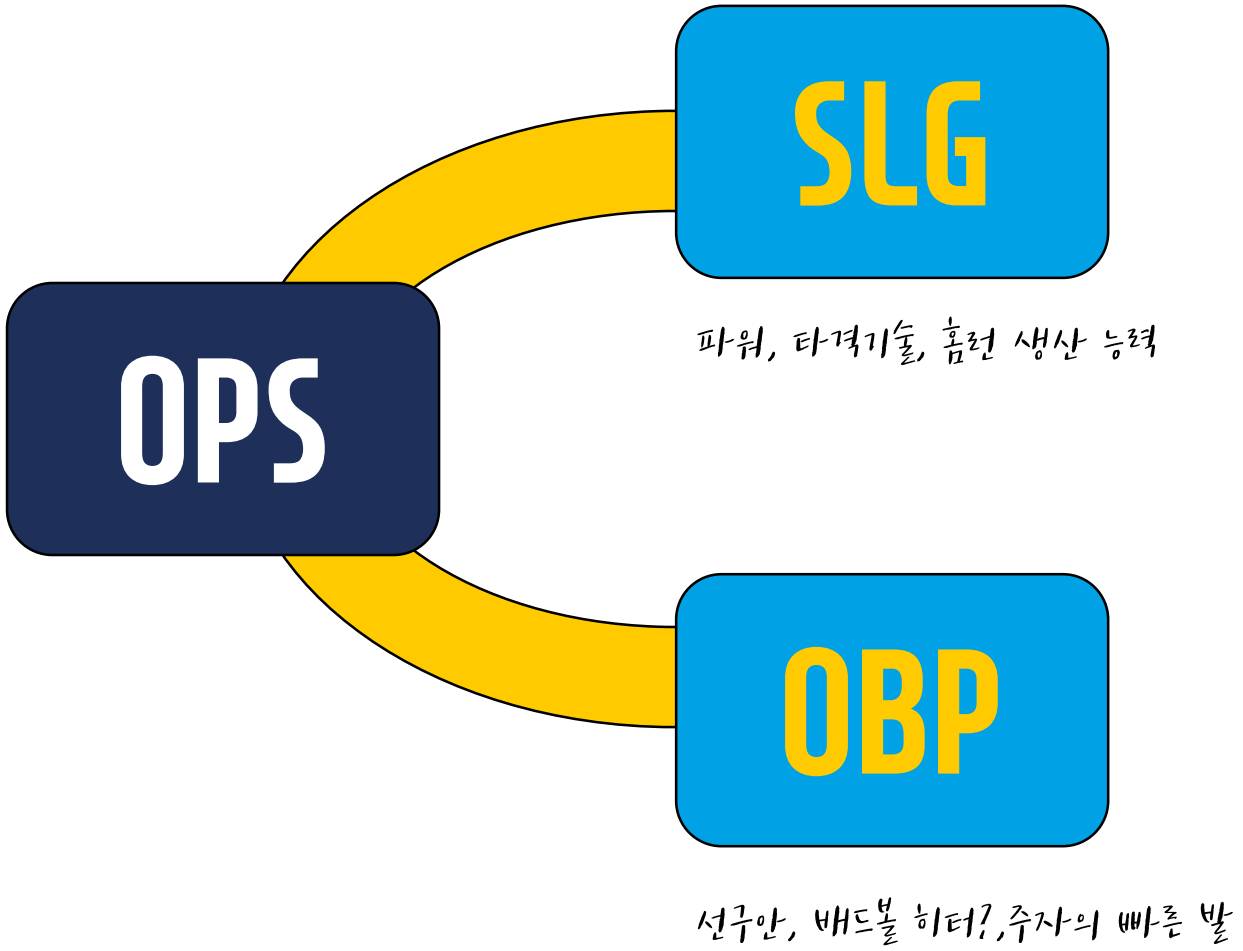
4_years_records:
2021 빅콘테스트_데이터분석분야_챔피언리그_스포츠테크_HTS_2018
2021 빅콘테스트_데이터분석분야_챔피언리그_스포츠테크_HTS_2019
2021 빅콘테스트_데이터분석분야_챔피언리그_스포츠테크_HTS_2020
2021 빅콘테스트_데이터분석분야_챔피언리그_스포츠테크_HTS_2021
4개의 데이터를 하나의 파일로 만든 것이 4_years_records이다.

mlb::
baseballsavant에서 제공한 3년간(2018~2020) 메이저리그 타자들 (100타석 이상)의 SLG, OBP, OPS, BARREL%이다.
https://baseballsavant.mlb.com/leaderboard/custom?year=2020,2019,2018&type=batter&filter=&sort=1&sortDir=desc&min=50&selections=slg_percent,on_base_percent,on_base_plus_slg,barrel_batted_rate,&chart=false&x=slg_percent&y=slg_percent&r=no&chartType=beeswarm

OPS 예측 모델링

- OPS는 장타율과 출루율이 합쳐진 개념
- 장타율과 출루율은 독립 변수가 다름

따라서 두가지 모델을 만든 후
합쳐야 한다!



OPS 예측 모델

독립변수의 조건

학습 데이터는 18~20 시즌 데이터

테스트 데이터는 21 시즌 데이터다.

조건을 같게 하기 위해서

독립변수로 **'비율 스탯'**을 사용 해야 한다.

〈LG 김현수〉 스탯비교

	안타	타율	장타율	루타
2020	181	0.331	0.523	286
2021	77	0.288	0.468	125
비율	2.35	1.15	1.18	2.28

예측에 활용할 데이터는 시즌이 마감되지 않았기 때문에,
누적 스탯은 스케일 차이가 많이 난다.
따라서 종속변수의 스케일차이를 만들지 않기 위해
독립변수를 비율 스탯으로 활용해야 한다.

OPS 예측 모델

종속 변수의 특징

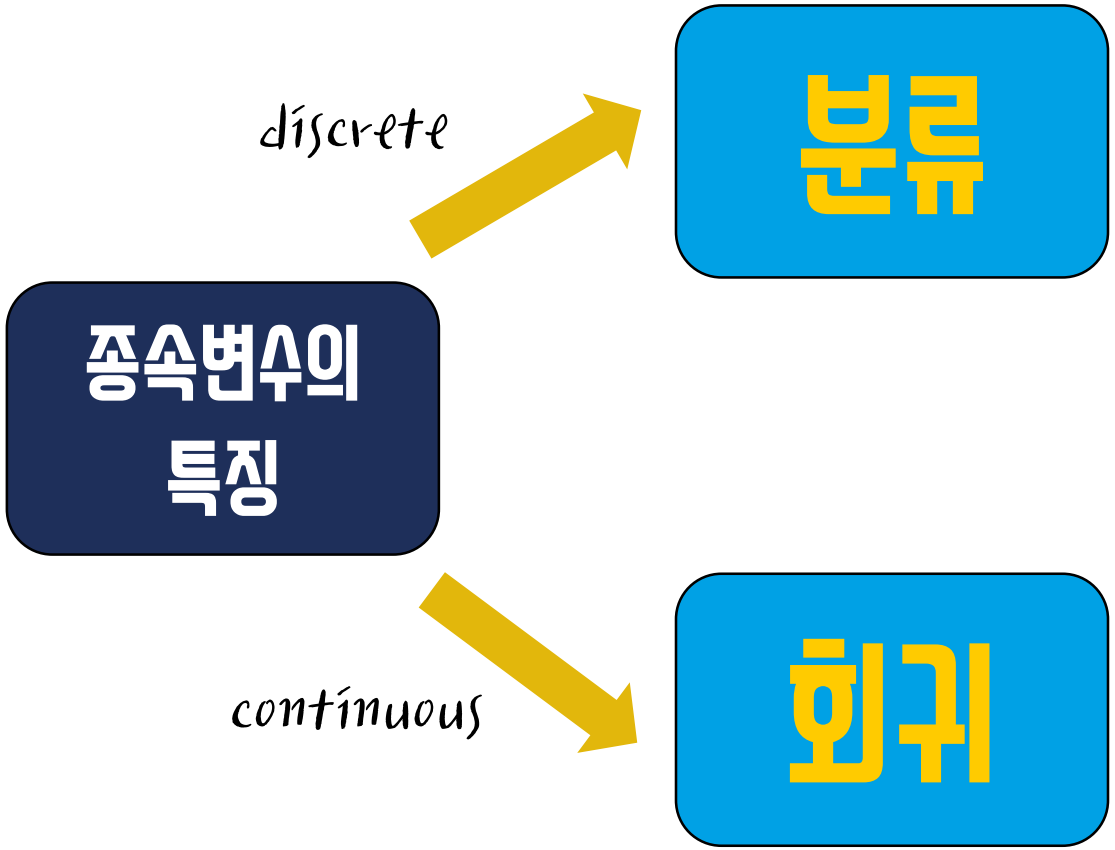
ML 알고리즘은 종속 변수의 특징에 따라

적용하는 알고리즘이 다르다.

종속변수가 이산형이라면 분류

연속형이라면 회귀 알고리즘을 사용한다.

〈ML 알고리즘의 선택 기준〉



OBP와 SLG 모두 연속형이기 때문에 회귀 모델을 사용할 예정

OPS 예측 모델

종속변수의 특징

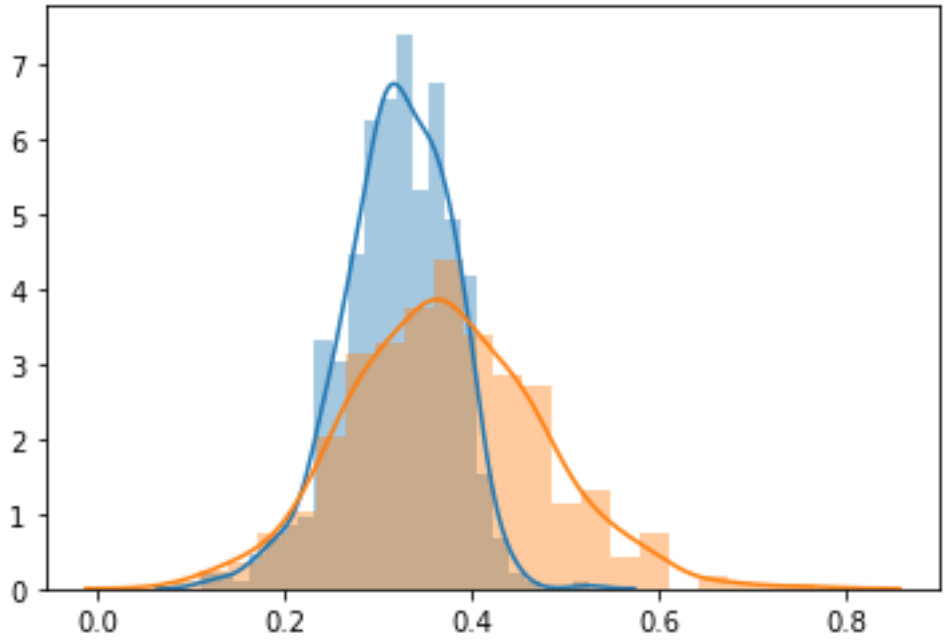
학습 데이터의 종속변수 분포를 파악해보자

출루율은 -0.409

장타율은 0.235의 왜도를 갖는다.

겉보기로는 종모양에 가까운 분포로 보인다.

〈종속변수의 분포〉



1. 파란선이 OPB, 주황선이 SLG로 연속임을 확인 할 수 있다.
2. 두 변수 모두 각각 어느 정도 왜도는 가지고 있지만, 모두 0~1사이의 값이며, 로그변환이나,standardscaler사용시 scale에 미칠 영향을 고려 했을 때, *추가적인 feature scaling은 하지 않는다.*

OPS 예측 모델

학습 데이터의 특징

학습 데이터는 18~20시즌 데이터를 사용

동일한 선수라도,

시즌별 준비 상태가 다르기 때문에,

시즌이 다르면 다른 데이터로 판단

〈학습 데이터 in Pandas DataFrame〉

	PCODE	BA	SLG	BARREL%	OBP	BB%	KK%	HR%	iso	2B%	3B%	BB/KK
0	60100	0.243	0.414	8.284024	0.313609	0.071006	0.213018	0.017751	0.171	0.082840	0.005917	0.333333
3	60343	0.216	0.389	6.321839	0.264368	0.045977	0.281609	0.045977	0.173	0.022989	0.000000	0.163265
5	60523	0.305	0.494	10.309278	0.360825	0.082474	0.201031	0.036082	0.189	0.046392	0.005155	0.410256
6	60558	0.254	0.421	5.882353	0.307487	0.064171	0.245989	0.032086	0.167	0.042781	0.005348	0.260870
7	60566	0.251	0.262	0.000000	0.279412	0.039216	0.073529	0.000000	0.011	0.009804	0.000000	0.533333
...	...	...	...	...	...	...	...	...	...	...	...	...
273	79334	0.129	0.129	2.702703	0.243243	0.108108	0.189189	0.000000	0.000	0.000000	0.000000	0.571429
274	79365	0.250	0.426	10.752688	0.336022	0.102151	0.182796	0.032258	0.176	0.053763	0.000000	0.558824
275	79402	0.304	0.401	2.760085	0.394904	0.116773	0.116773	0.010616	0.097	0.038217	0.006369	1.000000
276	79456	0.309	0.363	1.038062	0.359862	0.065744	0.096886	0.000000	0.054	0.034602	0.006920	0.678571
277	79608	0.286	0.413	5.434783	0.347826	0.076087	0.102174	0.017391	0.127	0.060870	0.000000	0.744681

613 rows × 12 columns

데이터 양이 613개 밖에 되지 않는다?

OPS 예측 모델

작은 데이터셋의 특징

데이터 셋이 작으면
과적합(overfitting)이 발생할 수 있다.
이번 모델링은 이를 보완하기 위해
복잡한 알고리즘을 지양하고,
최대한 **단순화** 시켜야 한다.

과적합이란?

과적합(overfitting)이란
데이터가 학습데이터만 맞춘 학습이 이루어져,
새로운 데이터를 접했을 때,
그 예측력이 떨어지는 현상을 말한다.
다항회귀나, max_depth가 깊은 모델에서
주로 발생한다.

참조:Kdnuggets – 5ways to Deal with Lack of Data in Machine Learning

https://www.kdnuggets.com/2019/06/5-ways-lack-data-machine-learning.html

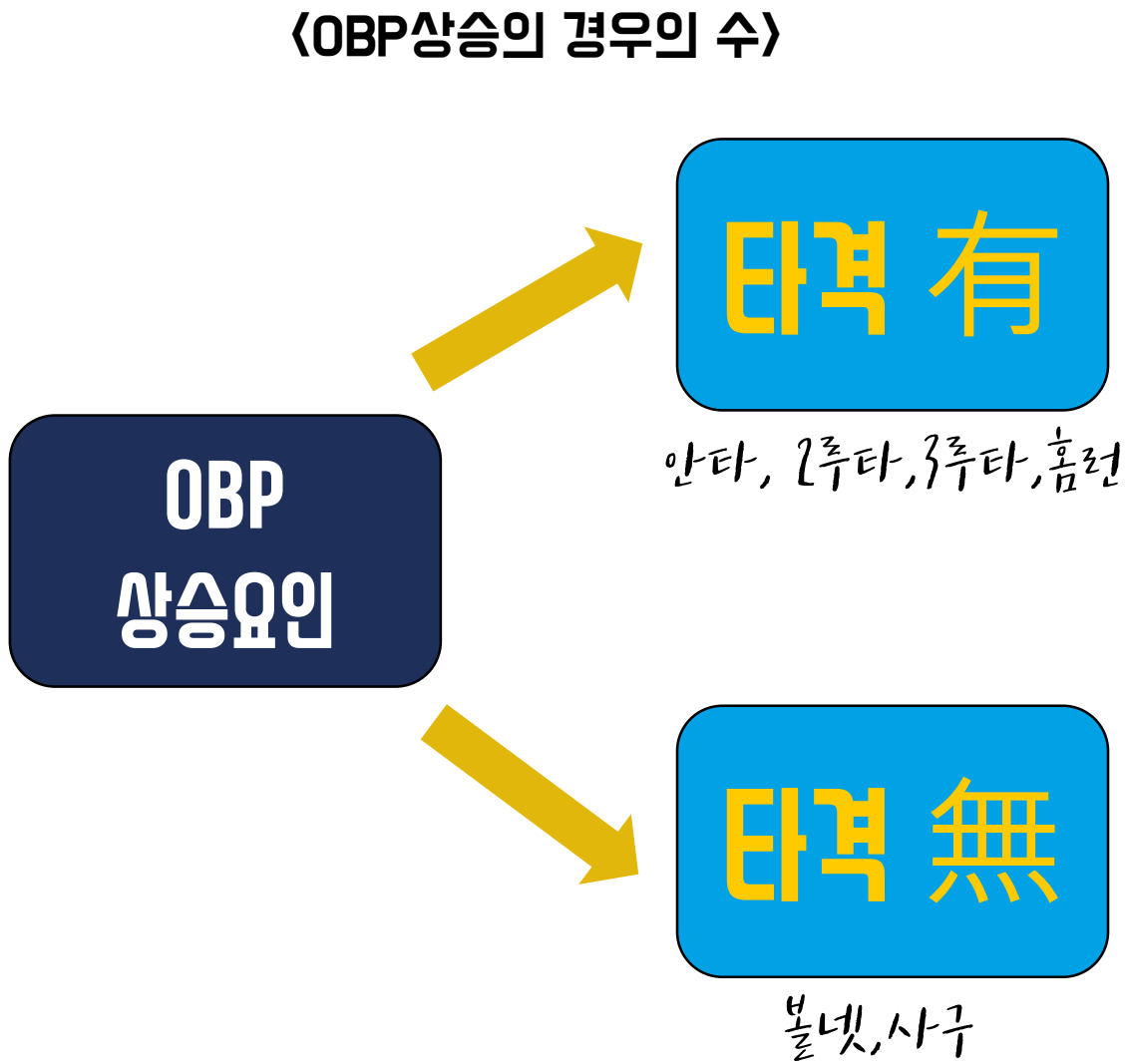
OBP 예측 모델

OBP 독립변수

출루율을 높이는

독립(원인)변수는 무엇이 있을까?

->경우의 수를 나눠서 생각해 보자.



OBP 예측 모델

OBP 독립변수

어떤 독립변수를 활용할지 판단하기 위해
비율 스탯 간 상관관계를 정리해 보았다.

BA = 타율

SLG = 장타율

BARREL% = 타석당 배럴 비율

BB% = 타석당 볼넷 비율

KK% = 타석당 삼진비율

ISO = 순장타율

2B% = 타석당 2루타 비율

3B% = 타석당 3루타 비율

BB/KK = 볼넷 삼진 비율

〈OBP와의 상관계수〉

비율스탯	상관계수
BA	0.881
SLG	0.787
BARREL%	0.420
BB%	0.558
KK%	-0.442
ISO	0.514
2B%	0.366
3B%	0.073
BB/KK	0.493

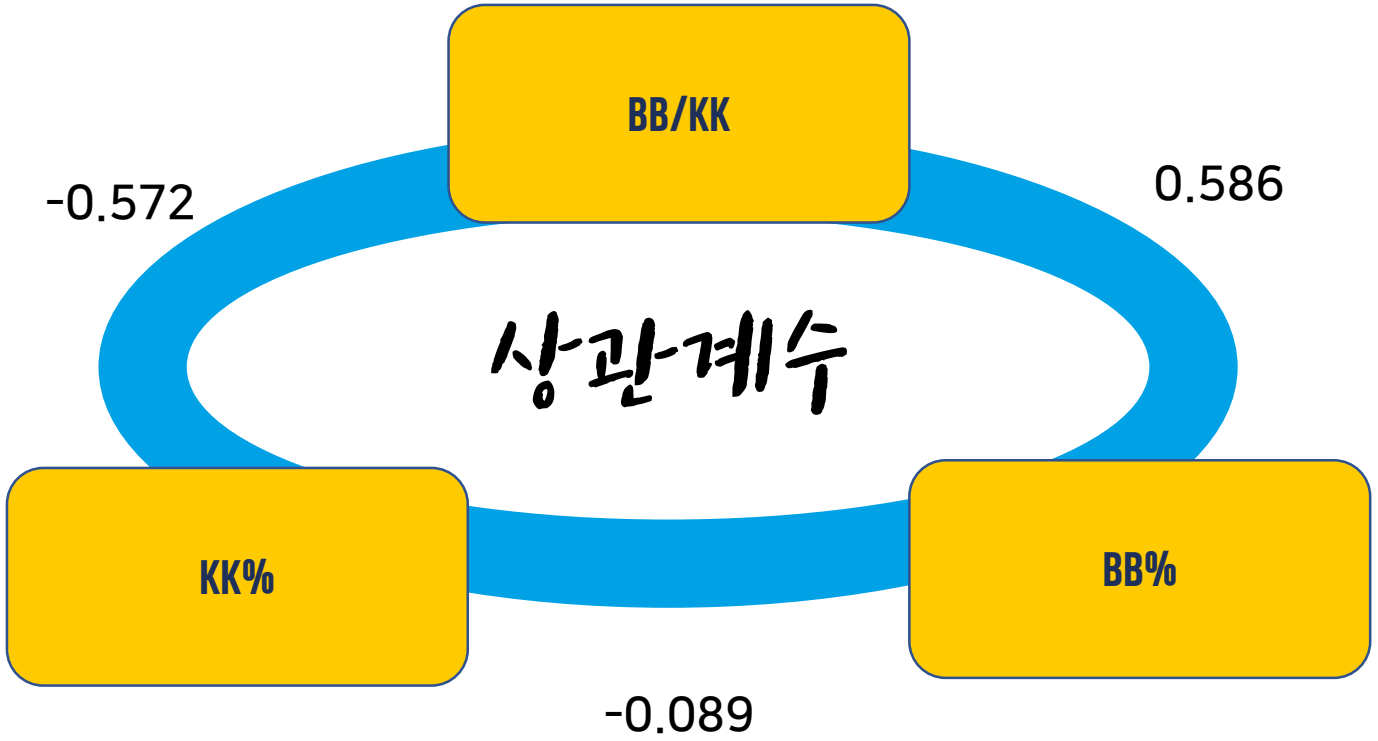
BA, BB%, KK%, BB/KK, BARREL% 가 OBP와 상관관계를 유의미하게 갖는다.

따라서 독립변수의 후보로 5가지 FEATURE를 선정했다.

OBP 예측 모델

다중 공선성 점검

다중 공선성이란, 독립 변수들 간의 상관관계가 높을 때,
회귀계수의 분산이 증가하고,
회귀계수 추정치가 불안해져 발생하는 문제이다.



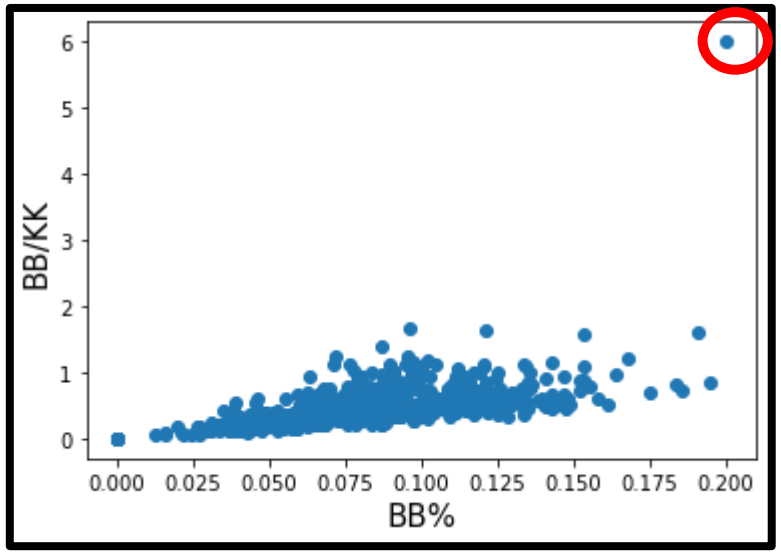
KK%와 BB%는 BB/KK 와 각각 -0.572, 0.586으로
OBP와 갖는 상관관계보다 더 큰 상관 관계를 갖는다.

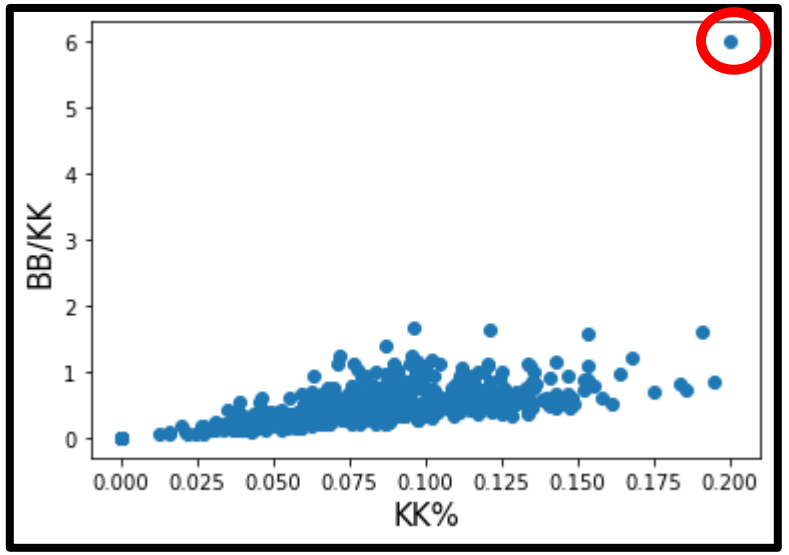
따라서 BB/KK를 독립변수로 사용하지 않는 것으로,
다중 공선성을 방지할 것이다.

OBP 예측 모델

다중 공선성 점검

다중 공선성이란, 독립 변수들 간의 상관관계가 높을 때,
회귀계수의 분산이 증가하고,
회귀계수 추정치가 불안해져 발생하는 문제이다.





산포도를 통해
BB/KK와
나머지 두 독립변수의
선형성을 확인하려 하였는데,

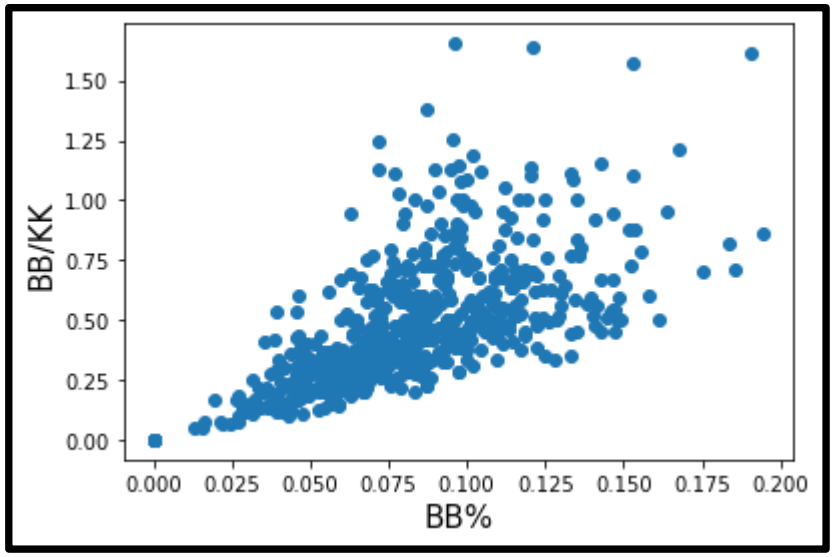
이상치 하나를 발견

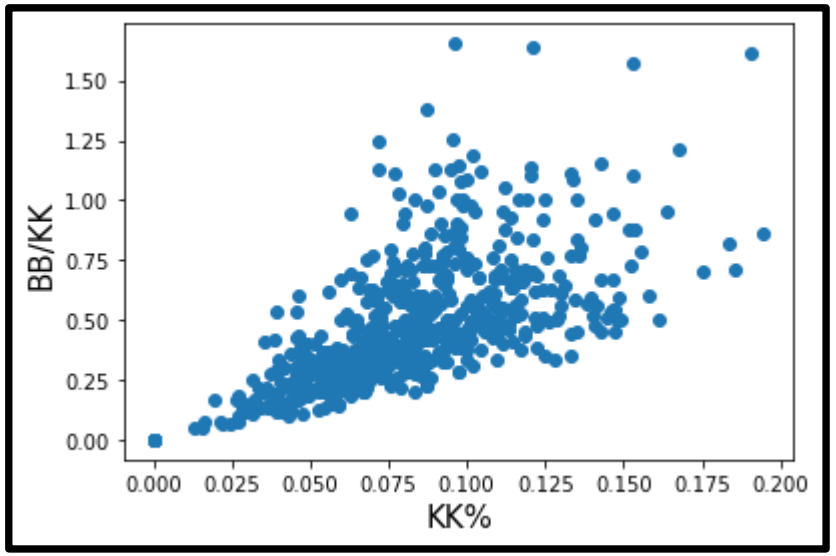
이를 제거 해 보았다.

OBP 예측 모델

다중 공선성 점검

다중 공선성이란, 독립 변수들 간의 상관관계가 높을 때,
회귀계수의 분산이 증가하고,
회귀계수 추정치가 불안해져 발생하는 문제이다.





이상치를 제거 하자,
두 변수 사이의
선형성이 더 명확해 졌고,
상관관계 역시 각각
0.649
-0.672로 증가 했다.

OBP 예측 모델

전과 같은 정보들을 종합해보았을 때,
LINEAR REGRESSION를 바탕으로
OBP예측 모델을 만들었다.

OBP-MODEL

모델: LINEAR REGRESSION

독립변수:BA, BARREL%,BB%,KK%
종속변수:OBP
결과예측에 사용되는 데이터:2021 전반기 데이터
예측된 결과:OBP_hat

OBP 예측 모델

Why LR?

많은 회귀 알고리즘 중에

ridge를 선택한 이유는

과적합과 정확도를 동시에 잡고자 하는

합리적인 선택이기 때문이다.

X_train = 학습 데이터 독립변수

y_train = 학습 데이터 종속변수

X_test = 테스트 데이터 독립변수

y_test = 테스트 데이터 종속변수

Base-line

```
#LinearRegression, Ridge, Lasso,RandomForest,XGB,lightGMB 학습, 예측 , 평가
lr_reg = LinearRegression()
lr_reg.fit(X_train, y_train)
ridge_reg = Ridge()
ridge_reg.fit(X_train, y_train)
lasso_reg = Lasso()
lasso_reg.fit(X_train, y_train)
rf_reg = RandomForestRegressor(n_estimators=100, max_depth = 5)
rf_reg.fit(X_train, y_train)
xgb_reg =XGBRegressor(n_estimators = 100, learning_rate=0.05,max_depth=6,
                      )
xgb_reg.fit(X_train, y_train)
lgbm_reg = LGBMRegressor(n_estimators=100, learning_rate=0.05, num_leaves=6,
                         subsample=0.6, colsample_bytree=0.4, reg_lambda=10, n_jobs=-1)
lgbm_reg.fit(X_train, y_train)
```

LinearRegression, Ridge, Lasso,
RandomForestRegressor, XGB, lightGBM 총 6개의 모델에
X_train과 y_test을 학습 시켰다.

OBP 예측 모델

TREE_BASED

트리 기반 모델의 경우

Max_depth가 커지거나,

모델이 복잡해질 경우

과적합 문제가 존재한다.

또한 lgbm의 경우 보통

데이터 크기가

10000개 이상인 경우에 주로 사용되기에,

과적합 이슈가 부각될 수 밖에 없다.

->이상치까지 학습하기 때문이다.

<base-line metrics>

```
LinearRegression  RMSE: 0.010654
Ridge  RMSE: 0.029354
Lasso  RMSE: 0.06191
RandomForestRegressor  RMSE: 0.013974
XGBRegressor  RMSE: 0.014482
LGBMRegressor  RMSE: 0.024751


LinearRegression  MAE: 0.007521
Ridge  MAE: 0.022751
Lasso  MAE: 0.049305
RandomForestRegressor  MAE: 0.010064
XGBRegressor  MAE: 0.010375
LGBMRegressor  MAE: 0.018573


LinearRegression  RSQUARE: 0.96756
Ridge  RSQUARE: 0.753758
Lasso  RSQUARE: -0.095322
RandomForestRegressor  RSQUARE: 0.944198
XGBRegressor  RSQUARE: 0.940063
LGBMRegressor  RSQUARE: 0.824936
```

RandomForest은
튜닝이 필요없는 Linear Regression에 비해
성능이 좋지 않았다.
튜닝할 수록 과적합 이슈가 있을 수 있기에, 사전에 배제할 것
Lasso와 LGBM은 각각 Ridge와, XGB와 비교했을 때,
성능이 좋지 않아 배제한다.

OBP 예측 모델

PARAMETER TUNNING

Ridge와 XGB에 대한
parameter tunning 진행

GridSearchCV를 활용

```
ridge_params = {'alpha':[0.05,0.1,1,5,8,10,12,15,20]}
```

→ 파라미터 후보를 담은 딕셔너리

```
def print_best_params_r2(model,params):
    grid_model = GridSearchCV(model, param_grid=params,
                              scoring='r2', cv=5)
    grid_model.fit(X_features, y_target)
    r2 = grid_model.best_score_
    print('{0} 5 CV시 최적 평균 R2 값: {1}, 최적 alpha:{2}'.format(model.__class__.__name__,
                                                        np.round(r2, 4), grid_model.best_params_))



def print_best_params_mae(model,params):
    grid_model = GridSearchCV(model, param_grid=params,
                              scoring='neg_mean_absolute_error', cv=5)
    grid_model.fit(X_features, y_target)
    mae = -1 * grid_model.best_score_
    print('{0} 5 CV시 최적 평균 MAE 값: {1}, 최적 alpha:{2}'.format(model.__class__.__name__,
                                                        np.round(mae, 4), grid_model.best_params_))
```

딕셔너리에 담은 파라미터 중
최적의 파라미터를 찾기 위한 함수를 만들었고, 해당함수는
최적 R2값과 MAE값 그리고 최적 alpha값(학습률)을 반환한다.

OBP 예측 모델

PARAMETER TUNNING

Ridge에 대한
parameter tunning 진행

GridSearchCV를 활용

<릿지 파라미터 후보군>

```
ridge_params = {'alpha':[0.05,0.07]}
print_best_params_r2(ridge_reg, ridge_params)
print_best_params_mae(ridge_reg, ridge_params)
ridge_params = {'alpha':[0.03,0.05]}
print_best_params_r2(ridge_reg, ridge_params)
print_best_params_mae(ridge_reg, ridge_params)
ridge_params = {'alpha':[0.01,0.03,0.05]}
print_best_params_r2(ridge_reg, ridge_params)
print_best_params_mae(ridge_reg, ridge_params)
```

->딕셔너리에 담은 파라미터 값들을 바꿔가며,
최적값을 탐색

OBP 예측 모델

PARAMETER TUNNING

우선 Ridge에 대한
parameter tunning 진행

GridSearchCV를 활용

<릿지 튜닝 결과>

```
Ridge 5 CV시 최적 평균 R2 값: 0.9613, 최적 parameter:{'alpha': 0.05}
Ridge 5 CV시 최적 평균 MAE 값: 0.008, 최적 parameter:{'alpha': 0.05}
Ridge 5 CV시 최적 평균 R2 값: 0.9634, 최적 parameter:{'alpha': 0.03}
Ridge 5 CV시 최적 평균 MAE 값: 0.0078, 최적 parameter:{'alpha': 0.03}
Ridge 5 CV시 최적 평균 R2 값: 0.9646, 최적 parameter:{'alpha': 0.01}
Ridge 5 CV시 최적 평균 MAE 값: 0.0077, 최적 parameter:{'alpha': 0.01}
```

파라미터 리스트에 더 낮은 학습율을 포함시킬 때마다 최적값 조정 확인
그러므로 Ridge 모델에 대한 학습율을 0.01로 설정할 예정.

OBP 예측 모델

PARAMETER TUNNING

XGB모델은 과적합규제 기능이 있다.
기본 성능이 좋은 만큼
과적합 문제만 해결 할 수 있다면
충분히 활용가능하다.

XGB에 대한
parameter tunning 진행

GridSearchCV를 활용할 예정.

<XGB 규제를 하기 위한 방법>

1.learning_rate(학습률) 값을 낮춘다.(0.01~0.1범위에서),
학습률을 낮출 경우, n_estimators를 높인다.

2. max_depth값을 낮춘다. (default = 6)

3. min_child_weight 값을 높인다.

4.Gamma값을 높인다.

5.Subsample과 colsample_bytree를 조정한다.

OBP 예측 모델

PARAMETER TUNNING

XGB모델은 과적합규제 기능이 있다.
기본 성능이 좋은 만큼
과적합 문제만 해결 할 수 있다면
충분히 활용가능하다.

XGB에 대한
parameter tunning 진행

GridSearchCV를 활용할 예정.

<XGB 규제를 하기 위한 방법>

1. 기본적으로 learning_rate는 0.01로 설정할 예정.

2.이에 맞춰, n_estimator를 [500,700,1000] 사이에서 정할 것(default = 100)

3. max_depth값은 5로 할 것(default =6)

4.Subsample과 colsample_bytree 조정은 feature가 많을 때 효과적이기 때문에, 이번 튜닝에서 제외할 예정

OPS 예측 모델

PARAMETER TUNNING

XGB모델은 과적합규제 기능이 있다.
기본 성능이 좋은 만큼
과적합 문제만 해결 할 수 있다면
충분히 활용 가능하다.

XGB에 대한
parameter tunning 진행

GridSearchCV를 활용할 예정.

<XGB 튜닝을 위한 파라미터들>

```
#파라미터 튜닝 XGB
xgb_params ={'n_estimators':[500,700,1000],
             'learning_rate':[0.01],
             'max_depth' : [5]}
```

릿지를 튜닝할 때 사용했던 함수를 바탕으로 XGB를 사용할 때,
최적의 파라미터가 무엇인지 알아볼 예정,
앞에서 언급했듯, 학습율과 max_depth 는 고정하고,
시행횟수의 최적 값을 찾아 볼 예정

OBP 예측 모델

PARAMETER TUNNING

XGB모델은 과적합규제 기능이 있다.
기본 성능이 좋은 만큼
과적합 문제만 해결 할 수 있다면
충분히 활용가능하다.

XGB에 대한
parameter tunning 진행

GridSearchCV를 활용할 예정.

<최적의 파라미터>

```
XGBRegressor 5 CV시 최적 평균 R2 값: 0.9431, 최적 parameter:{'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 700}
XGBRegressor 5 CV시 최적 평균 MAE 값: 0.0095, 최적 parameter:{'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 700}
```

-> GridSearchCV결과
n_estimators는 700이 적절하다.

여기서 알 수 있는 것은
n_estimators를 올리는 것 보다 learning_rate를 내리는 것이 과적합에 효과적이라는 것.

OBP 예측 모델

Cross_val_score

새로운 데이터에도 일정한 결과가
나올 수 있는지에 대해 확인하기 위해
교차 검증을 시도해 보았다.
Cv = 5개로 나누었고, 각각의 성능을
리스트로 반환 후,
Metrics 별로 **평균**을 반환해 보았다.

〈cross_val_score〉 결과 반환 함수〉

```python
#cv를 나눠서 학습했을때, cv별 metric을 판단해보기 위한 함수
def get_avg_r2_cv(models):

    for model in models:
        #분할하지 않고 전체 데이터로 cross_val_score()수행, 모델 별 CV R2값과 평균 R2 출력
        r2_list = cross_val_score(model, X_features, y_target, scoring = 'r2', cv =5)

        r2_avg = np.mean(r2_list)
        print('₩n{0} CV R2 값 리스트 :{1}'.format(model.__class__.__name__, np.round(r2_list,3)))
        print('{0}CV 평균 R2 값: {1}'.format(model.__class__.__name__, np.round(r2_avg,3)))

def get_avg_mae_cv(models):

    for model in models:
        #분할하지 않고 전체 데이터로 cross_val_score()수행, 모델 별 CV MAE값과 평균 MAE 출력
        mae_list = -cross_val_score(model, X_features, y_target, scoring = 'neg_mean_absolute_error', cv =5)

        mae_avg = np.mean(mae_list)
        print('₩n{0} CV MAE 값 리스트 :{1}'.format(model.__class__.__name__, np.round(mae_list,3)))
        print('{0}CV 평균 MAE 값: {1}'.format(model.__class__.__name__, np.round(mae_avg,3)))
```

OBP 예측 모델

Cross_val_score

새로운 데이터에도 일정한 결과가
나올 수 있는지에 대해 확인하기 위해
교차 검증을 시도해 보았다.
Cv = 5개로 나누었고, 각각의 성능을
리스트로 반환 후,
Metrics 별로 평균을 반환해 보았다.

<결과>

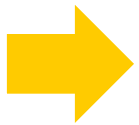
```
LinearRegression CV R2 값 리스트 :[0.959 0.964 0.972 0.96  0.968]
LinearRegressionCV 평균 R2 값: 0.965

Ridge CV R2 값 리스트 :[0.96  0.964 0.972 0.958 0.969]
RidgeCV 평균 R2 값: 0.965

XGBRegressor CV R2 값 리스트 :[0.946 0.926 0.959 0.931 0.954]
XGBRegressorCV 평균 R2 값: 0.943

LinearRegression CV MAE 값 리스트 :[0.008 0.008 0.007 0.008 0.007]
LinearRegressionCV 평균 MAE 값: 0.008

Ridge CV MAE 값 리스트 :[0.008 0.008 0.007 0.008 0.007]
RidgeCV 평균 MAE 값: 0.008

XGBRegressor CV MAE 값 리스트 :[0.009 0.01  0.009 0.01  0.009]
XGBRegressorCV 평균 MAE 값: 0.01
```

Linear Regression 과 Ridge(alpha =0.01) 모두
리스트 안의 metrics 들이 대체로 비슷하며, 평균 역시 같다.
따라서 두 모델의 새로운 데이터에 대한 학습 성능은 비슷한 수준이라고 볼 수 있다.
한편 XGB의 경우 학습 성능은 다소 떨어지나, BASE_LINE에서
과적합우려가 있었던 만큼 튜닝이 다소 성공적이었다고 볼 수 있다.

OBP 예측 모델

Why LR?

Ridge와 XGB를 튜닝한 후
Linear Regression 과 비교해보니,
MAE 성능은 상대적으로 좋아졌고,
Base-line과 비교해서
과적합양상에서 멀어진 것을 확인



〈OBP FINAL〉

```
LinearRegression  MAE: 0.007521
Ridge  MAE: 0.007538
XGBRegressor  MAE: 0.009855


LinearRegression  RSQUARE: 0.96756
Ridge  RSQUARE: 0.967969
XGBRegressor  RSQUARE: 0.945422
```

그러나 과적합 양상에서 멀어졌지만,
성능 역시 유의미한 차이가 발생해
XGB는 배제
남은 모델중 성능이 가장 좋은 LR로 선택

OBP 예측 모델

예측을 위한 데이터

학습된 XGB모델을 바탕으로
2021년 데이터를 학습 시켜
선수들의 시즌 최종 성적을 예측

<결과 예측을 위한 함수>

```
#21시즌 데이터 대입
def get_test_OBP(df=None):
    df_copy = df.copy()
    y_target = df_copy['OBP']
    X_features = df_copy.drop(['PCODE','HR%','iso','SLG','OBP','3B%','2B%','BB/KK'], axis =1)

    return y_target, X_features

y_target, X_features = get_test_OBP(mid_preprocessed_2021)

mid_preprocessed_2021['OBP_hat']= xgb_reg.predict(X_features)
mid_preprocessed_2021
```

결과는 장타율과 함께 공개 합니다.

SLG 예측 모델

SLG 독립변수

SLG 역시 과적합 방지를 위해

최소한의 feature로

독립변수를 구성하려고 함

〈SLG와의 상관계수〉

비율스탯	상관계수
BA	0.805
OBP	0.778
BARREL%	0.713
BB%	0.245
KK%	-0.271
ISO	0.891
2B%	0.494
3B%	0.076
BB/KK	0.224

SLG 예측 모델

전과 같은 정보들을 종합해보았을 때,

XGB를 바탕으로

OBP예측 모델을 만들었다.

SLG-MODEL

모델: XGB
Parameter:[n_estimator = 3000,
learning_rate=0.01, max_depth = 3,
min_child_weight=2

독립변수:iso, BA, BARREL%
종속변수:SLG
결과예측에 사용되는 데이터:2021 전반기 데이터
예측된 결과:SLG_hat

SLG 예측 모델

Why XGB?

많고 많은 회귀 알고리즘 중에

XGB를 선택한 이유는

과적합과 정확도를 동시에 잡고자 하는

합리적인 선택이기 때문이다.

X_train = 학습 데이터 독립변수

y_train = 학습 데이터 종속변수

Base-line

```
lr_reg = LinearRegression()
lr_reg.fit(X_train, y_train)
ridge_reg = Ridge()
ridge_reg.fit(X_train, y_train)
lasso_reg = Lasso()
lasso_reg.fit(X_train, y_train)
rf_reg = RandomForestRegressor(n_estimators=1500, max_depth = 5)
rf_reg.fit(X_train, y_train)
xgb_reg =XGBRegressor(n_estimators = 1500, learning_rate=0.05,max_depth=5,
                      colsample_bytree=0.5, subsample=0.8)
xgb_reg.fit(X_train, y_train)
lgbm_reg = LGBMRegressor(n_estimators=1000, learning_rate=0.05, num_leaves=4,
                         subsample=0.6, colsample_bytree=0.4, reg_lambda=10, n_jobs=-1)
lgbm_reg.fit(X_train, y_train)
```

LinearRegression, Ridge, Lasso,
RandomForestRegressor, XGB, lightGBM 총 6개의 모델에
X_train과 y_train을 학습 시켰다. Test_size=0.2로 설정

SLG 예측 모델

BASE LINE METRICS

장타율의 경우

Iso와의 높은 상관관계(0.889)로 인해

LinearRegression이

과적합 상태로 보인다.

Base-line

```
LinearRegression  RMSE: 0.0
Ridge  RMSE: 0.035792
Lasso  RMSE: 0.108928
RandomForestRegressor  RMSE: 0.018128
XGBRegressor  RMSE: 0.020576
LGBMRegressor  RMSE: 0.048828


LinearRegression  MAE: 0.0
Ridge  MAE: 0.025847
Lasso  MAE: 0.084117
RandomForestRegressor  MAE: 0.009656
XGBRegressor  MAE: 0.013255
LGBMRegressor  MAE: 0.035509


LinearRegression  RSQUARE: 1.0
Ridge  RSQUARE: 0.891691
Lasso  RSQUARE: -0.003165
RandomForestRegressor  RSQUARE: 0.972215
XGBRegressor  RSQUARE: 0.964206
LGBMRegressor  RSQUARE: 0.798431
```

장타율의 경우
linear regression과 random forest는 과적합 문제로 배제하고,
Lasso와, LGBM은 기본 성능이 상대적으로 모자라 배제한다.
XGB와 Ridge를 과적한 규제와, 1푼이내 오차를 목표로 튜닝할 것이다.

SLG 예측 모델

PARAMETER TUNNUNG

RIDGE의 경우

GridSearchCV를 포함한 함수를 통해

최적의 alpha값을 찾아보았다.

<Ridge-parameter-tunning>

```
ridge_params = {'alpha':[1,5,8,10,12,15,20]}
print_best_params_r2(ridge_reg, ridge_params)
print_best_params_mae(ridge_reg, ridge_params)
print('₩n')
ridge_params = {'alpha':[0.1,1,5,8,10,12,15,20]}
print_best_params_r2(ridge_reg, ridge_params)
print_best_params_mae(ridge_reg, ridge_params)
print('₩n')
ridge_params = {'alpha':[0.05,0.1,1,5,8,10,12,15,20]}
print_best_params_r2(ridge_reg, ridge_params)
print_best_params_mae(ridge_reg, ridge_params)
print('₩n')
ridge_params = {'alpha':[0.03,0.05,0.1,1,5,8,10,12,15,20]}
print_best_params_r2(ridge_reg, ridge_params)
print_best_params_mae(ridge_reg, ridge_params)
print('₩n')
ridge_params = {'alpha':[0.01,0.03,0.05,0.1,1,5,8,10,12,15,20]}
print_best_params_r2(ridge_reg, ridge_params)
print_best_params_mae(ridge_reg, ridge_params)
```

출루율과 마찬가지로 학습률을 낮추는 것이 정확도 상승에 영향을 주는지 확인해보고자 함

SLG 예측 모델

PARAMETER TUNNUNG

RIDGE의 경우

GridSearchCV를 포함한 함수를 통해

최적의 alpha값을 찾아보았다.

<ridge 튜닝 결과>

```
Ridge 5 CV시 최적 평균 R2 값: 0.9892, 최적 parameter:{'alpha': 0.2}
Ridge 5 CV시 최적 평균 MAE 값: 0.008, 최적 parameter:{'alpha': 0.2}


Ridge 5 CV시 최적 평균 R2 값: 0.9934, 최적 parameter:{'alpha': 0.15}
Ridge 5 CV시 최적 평균 MAE 값: 0.0063, 최적 parameter:{'alpha': 0.15}


Ridge 5 CV시 최적 평균 R2 값: 0.9968, 최적 parameter:{'alpha': 0.1}
Ridge 5 CV시 최적 평균 MAE 값: 0.0043, 최적 parameter:{'alpha': 0.1}


Ridge 5 CV시 최적 평균 R2 값: 0.9991, 최적 parameter:{'alpha': 0.05}
Ridge 5 CV시 최적 평균 MAE 값: 0.0023, 최적 parameter:{'alpha': 0.05}


Ridge 5 CV시 최적 평균 R2 값: 0.9997, 최적 parameter:{'alpha': 0.03}
Ridge 5 CV시 최적 평균 MAE 값: 0.0014, 최적 parameter:{'alpha': 0.03}


Ridge 5 CV시 최적 평균 R2 값: 1.0, 최적 parameter:{'alpha': 0.01}
Ridge 5 CV시 최적 평균 MAE 값: 0.0005, 최적 parameter:{'alpha': 0.01}
```

출루율과 마찬가지로 학습율이 작아질 수록 높은 성능을 보여주지만, R2값으로 보아 0.01에는 과적합이 된 것으로 보인다.
따라서 성능이 0.005에 진입하는 최고 학습율인 0.1으로 Ridge를 학습 시키는 것으로 한다.

SLG 예측 모델

PARAMETER TUNNING

XGB모델은 과적합규제 기능이 있다.
기본 성능이 좋은 만큼
과적합 문제만 해결 할 수 있다면
충분히 활용가능하다.

XGB에 대한
parameter tunning 진행

GridSearchCV를 활용할 예정.

<XGB 규제를 하기 위한 방법>

1. 기본적으로 learning_rate는 0.01로 설정할 예정. 이에 맞춰, n_estimator는 1000으로 올릴 것(default = 100)

2. max_depth값은 [4,5]를 후보로 올려볼것

3. min_child_weight 은 과하게 높일 경우 과소적합이 있을 수 있어서 [1,2]중을 후보로 올려볼 것.

4.Subsample과 colsample_bytree 조정은 feature가 많을 때 효과적이기 때문에, 이번 튜닝에서 제외할 예정

SLG 예측 모델

PARAMETER TUNNING

XGB는 장타율의 경우

기본적으로 과적합양상이

출루율에 비해 심할 것으로 보여

전반적으로 n_estimators를

높인 상태로 튜닝을 시작한다.

<튜닝과정>

```
xgb_params ={'n_estimators':[2500,3000],
             'learning_rate':[0.01],
            'min_child_weight':[1,2],
             'max_depth' : [3,4]}
```

기본조건

```
xgb_params ={'n_estimators':[2500,3000],
             'learning_rate':[0.01],
            'min_child_weight':[2,3],
             'max_depth' : [3,4]}
```

Min_child_weight를 [2,3]으로 바꿈

```
xgb_params ={'n_estimators':[3000,3500],
             'learning_rate':[0.01],
            'min_child_weight':[2,3],
             'max_depth' : [3,4]}
```

n_estimaor를 [3000,3500]으로 바꿈

```
xgb_params ={'n_estimators':[3000,3500],
             'learning_rate':[0.01],
            'min_child_weight':[1,2],
             'max_depth' : [3,4]}
```

Min_child_weight를 [1,2]으로 바꿈

SLG 예측 모델

PARAMETER TUNNING

XGB는 장타율의 경우

기본적으로 과적합양상이

출루율에 비해 심할 것으로 보여

전반적으로 n_estimators를

높인 상태로 튜닝을 시작한다.

<튜닝결과>

```
XGBRegressor 5 CV시 최적 평균 R2 값: 0.9887, 최적 parameter:{'learning_rate': 0.01, 'max_depth': 3, 'min_child_weight': 2, 'n_estimators': 3000}
XGBRegressor 5 CV시 최적 평균 MAE 값: 0.0069, 최적 parameter:{'learning_rate': 0.01, 'max_depth': 3, 'min_child_weight': 2, 'n_estimators': 3000}
```


```
XGBRegressor 5 CV시 최적 평균 R2 값: 0.9887, 최적 parameter:{'learning_rate': 0.01, 'max_depth': 3, 'min_child_weight': 2, 'n_estimators': 3000}
XGBRegressor 5 CV시 최적 평균 MAE 값: 0.0069, 최적 parameter:{'learning_rate': 0.01, 'max_depth': 3, 'min_child_weight': 2, 'n_estimators': 3000}
```


```
XGBRegressor 5 CV시 최적 평균 R2 값: 0.9895, 최적 parameter:{'learning_rate': 0.01, 'max_depth': 3, 'min_child_weight': 2, 'n_estimators': 3500}
XGBRegressor 5 CV시 최적 평균 MAE 값: 0.0065, 최적 parameter:{'learning_rate': 0.01, 'max_depth': 3, 'min_child_weight': 2, 'n_estimators': 3500}
```


```
XGBRegressor 5 CV시 최적 평균 R2 값: 0.9895, 최적 parameter:{'learning_rate': 0.01, 'max_depth': 3, 'min_child_weight': 2, 'n_estimators': 3500}
XGBRegressor 5 CV시 최적 평균 MAE 값: 0.0065, 최적 parameter:{'learning_rate': 0.01, 'max_depth': 3, 'min_child_weight': 2, 'n_estimators': 3500}
```


min_child_weight는 3으로 올라가지 않는다.
과적합 양상 때문인지, max_depth는 어떤 조합에서도 3이 나온다.
그러나 n_estimator는 3500일때 과적합 양상이 심화된다.
따라서 XGB로 튜닝을 하면 learning_rate=0.01, max_depth=3,
Min_child_weight = 2, n_estimator = 3000으로 한다.

SLG 예측 모델

PARAMETER TUNNING

이유1:과적합을 잡은 XGB

<튜닝 후 결과>

```
Ridge   MAE: 0.004496
XGBRegressor   MAE: 0.005477


Ridge   RSQUARE: 0.996685
XGBRegressor   RSQUARE: 0.982504
```

이유1:데이터의 개수를 고려 해 보았을때, 과적합이 우려되기에, 성능보다, 다른 데이터에서도 예측 가능한 모델이 좋은 모델이라고 판단

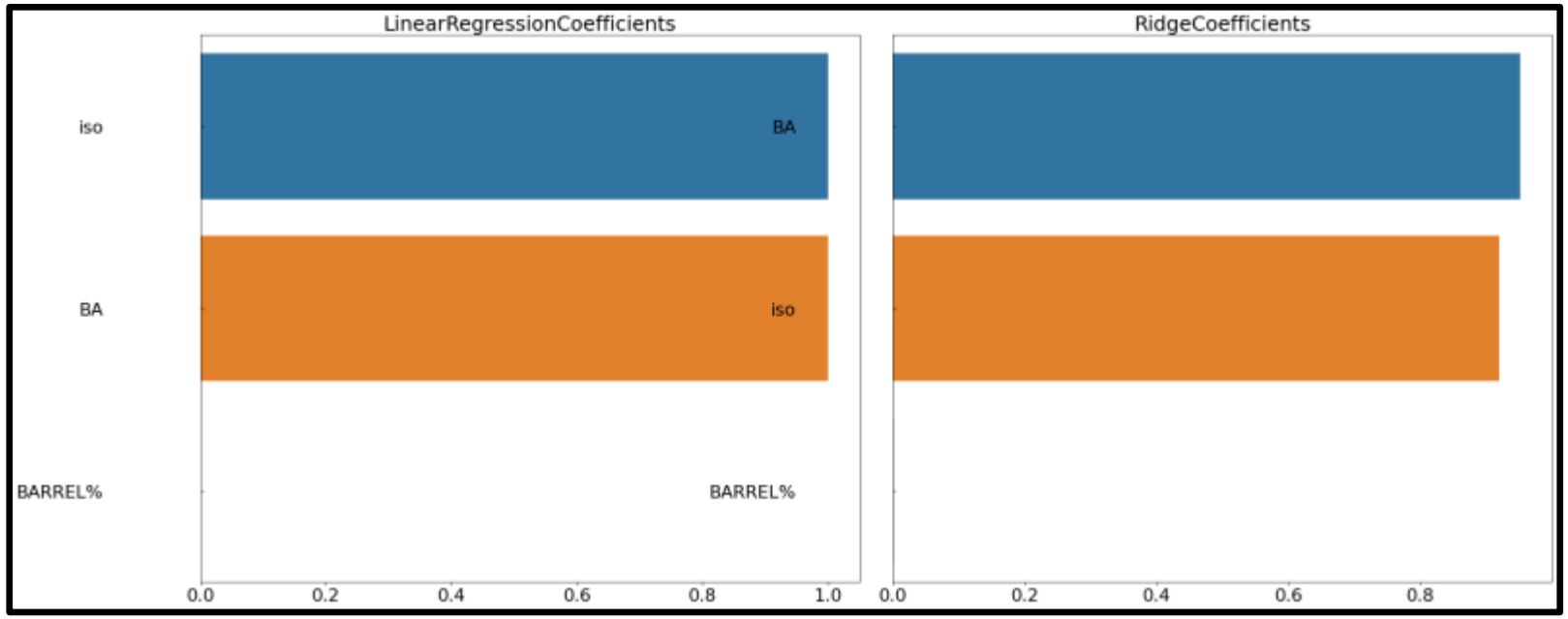
SLG 예측 모델

PARAMETER TUNNING

이유2: 그나마 BARREL%을

고려하는 모델이기 때문.

<회귀 계수 시각화>



두 모델은 사실 상 두가지 회귀계수만 고려한다.
LR = [1, -0, 1] -> BA, BARREL%, ISO 순
Ridge = [0.938,0.002,0.997] -> BA, BARREL%, ISO 순

SLG 예측 모델

PARAMETER TUNNING

이유2: 그나마 BARREL%을

고려하는 모델이기 때문.

한편 XGB모델을 사용했을때,
회귀계수는 알수 없지만,

Feature_importance는
각각 0.285,0.0017,0.714로
(BA,BARREL%,ISO순)

생산성을 의미할 수 있는,
BARREL%를 과적합 규제하에서
최대한 반영할 수 있는 모델임을 알 수 있다.

다른 모델의 회귀계수를 중요도로 가정하고
100분위화시킨다면 배럴 비율은
0.001에도 미치지 못함.

OPS 예측 모델

OPS 예측 결과

위의 OBP,SLG모델에 따라

선수들의 성적을 예측해본 결과는

다음과 같다.

<예측 결과>

	PCODE	OBP_hat	SLG_hat	OPS_hat
151	67341	0.446	0.510	0.956
156	67872	0.365	0.473	0.838
158	68050	0.512	0.542	1.054
200	75847	0.413	0.591	1.003
202	76232	0.449	0.680	1.129
205	76290	0.389	0.474	0.863
219	78224	0.403	0.516	0.919
222	78513	0.416	0.445	0.861
230	79192	0.380	0.536	0.916
231	79215	0.407	0.440	0.848

OPS 예측 모델

사용데이터

다음과 같은 데이터들을 사용했다.

players_2018:
2021 빅콘테스트_데이터분석분야_챔피언리그_스포츠테크_HTS_2018
->사용하기 편하게 이름만 바꿈, 연도별로 바꿈

records_2018:
2021 빅콘테스트_데이터분석분야_챔피언리그_스포츠테크_선수_2018

->사용하기 편하게 이름만 바꿈, 연도별로 바꿈

stats_2018
2021 빅콘테스트_데이터분석분야_챔피언리그_스포츠테크_타자기본_2018

->사용하기 편하게 이름만 바꿈, 연도별로 바꿈

->기타 크롤링 데이터 사용안함