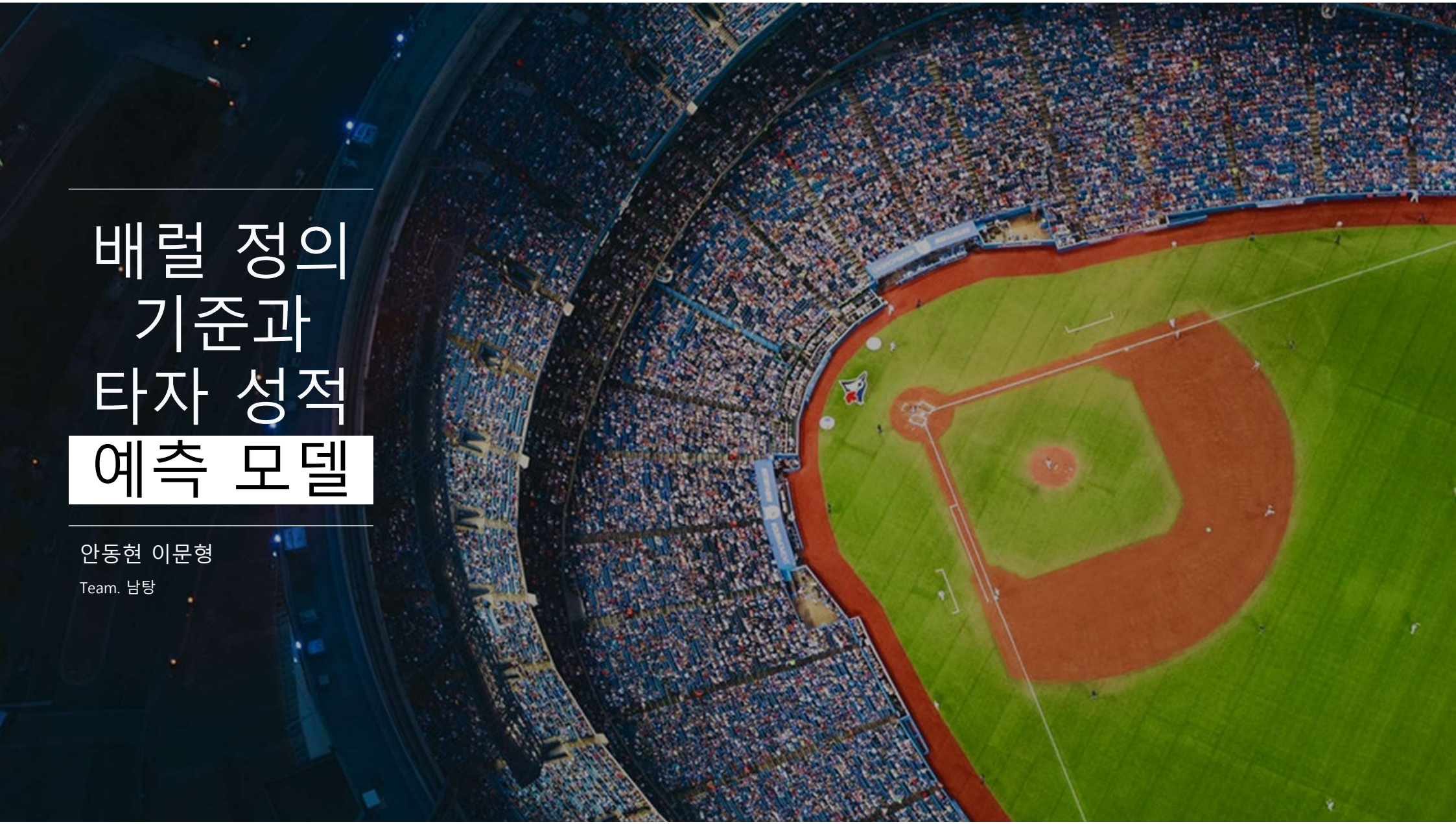


배럴 정의 기준과 타자 성적 예측 모델

안동현 이문형

Team. 남탕

1루 **배럴의 정의**

- 필요성
- 데이터 탐색
- 배럴 기준

2루 **모델링을 위한 데이터 탐색**

- 데이터 수집
- 전처리 및 EDA
- 특성 공학

3루 **타자 성적 예측 모델**

- 모델 탐색
- 모델 채택

1루 배럴의 정의

- 필요성

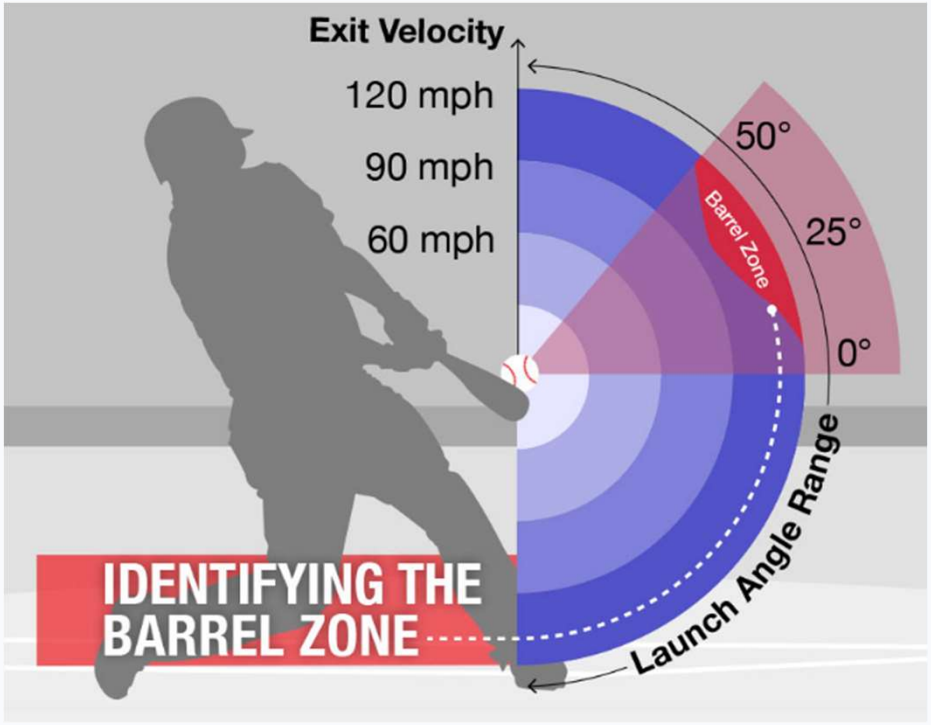
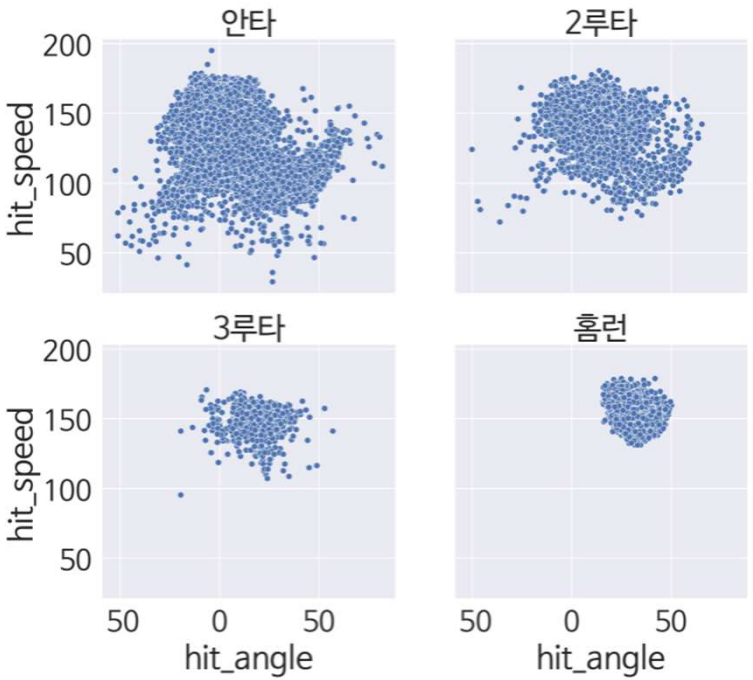


사진 출처. MLB.COM
(http://m.mlb.com/glossary/statcast/barrel/)

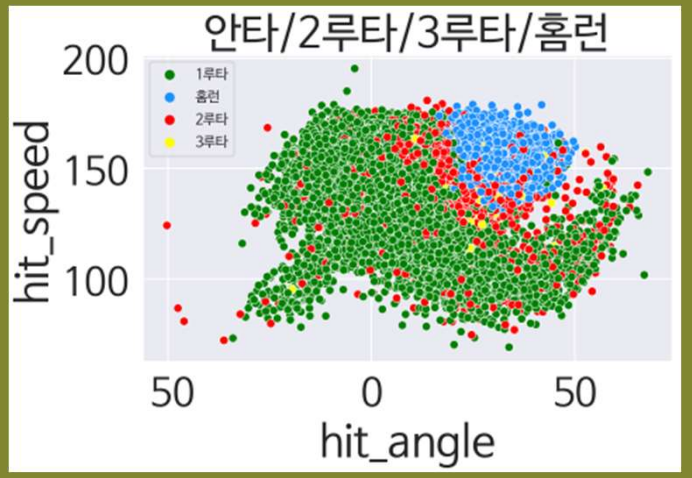
- 타구 속도와 발사각의 일정 군집에서 **타율 0.500, 장타율 1.500 이상을 기록하는 타구**
- 최소 98마일 이상 (발사각 26~30도)
- **타구 속도가 증가할수록 발사각의 범위는 넓어짐**
- 100마일 이상부터는 1마일당 2~3도가 증가함
- 하지만 이는 미국 MLB 기준
- **공인구, 구장, 선수의 수비 능력 등 KBO와 MLB간의 차이는 명확하기 때문에 KBO만의 배럴 기준을 세울 필요가 있음**

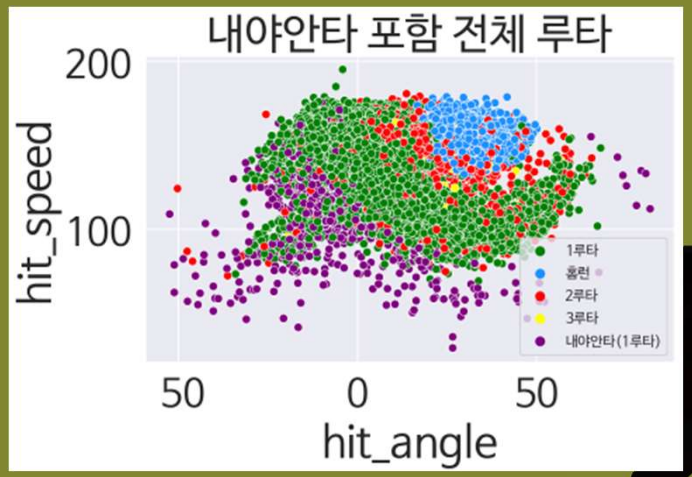
1루 배럴의 정의

- 데이터 탐색



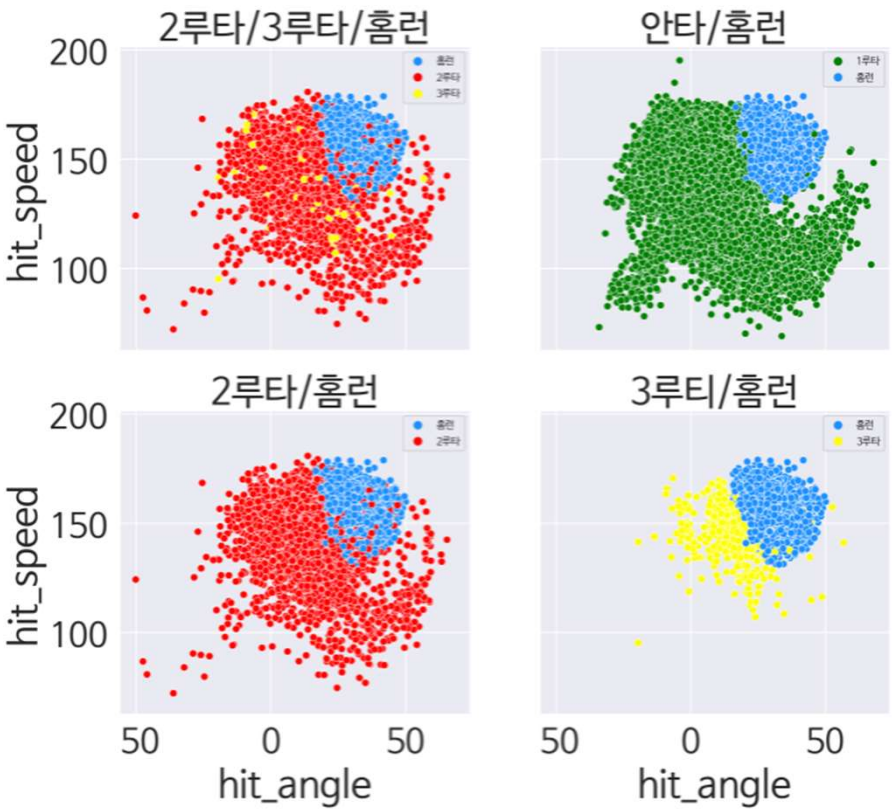
- 장타일수록 **특정 발사각과 타구속도로 수렴**하는 것을 확인할 수 있음

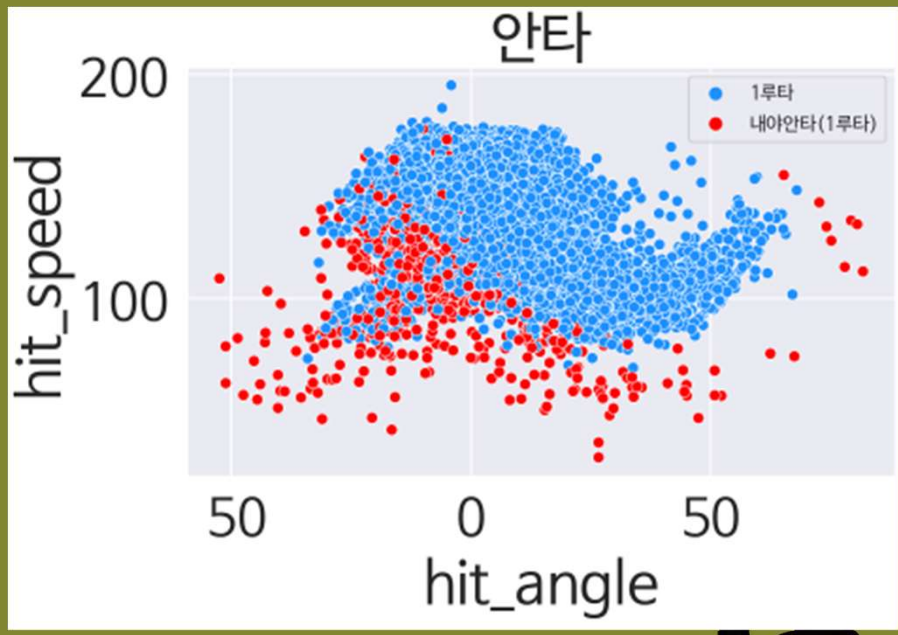




1루 배럴의 정의

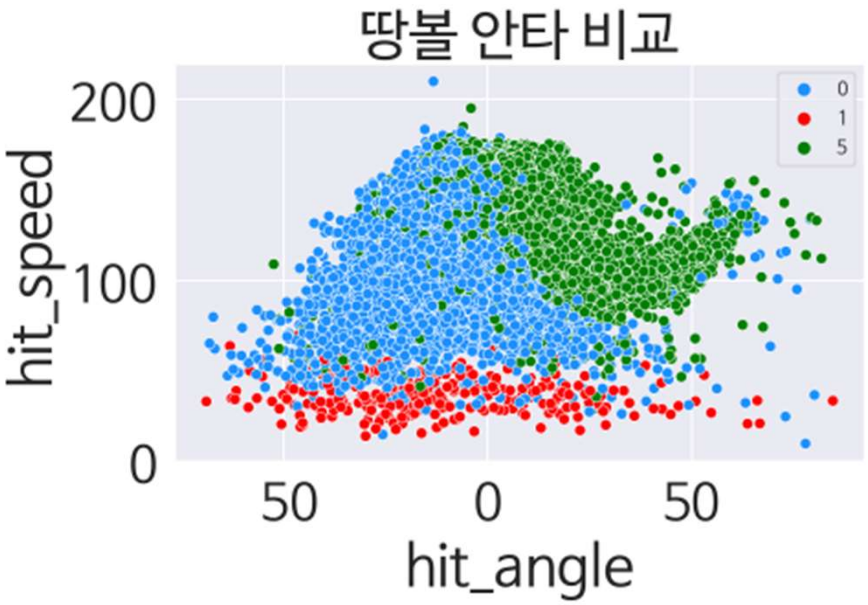
- 데이터 탐색



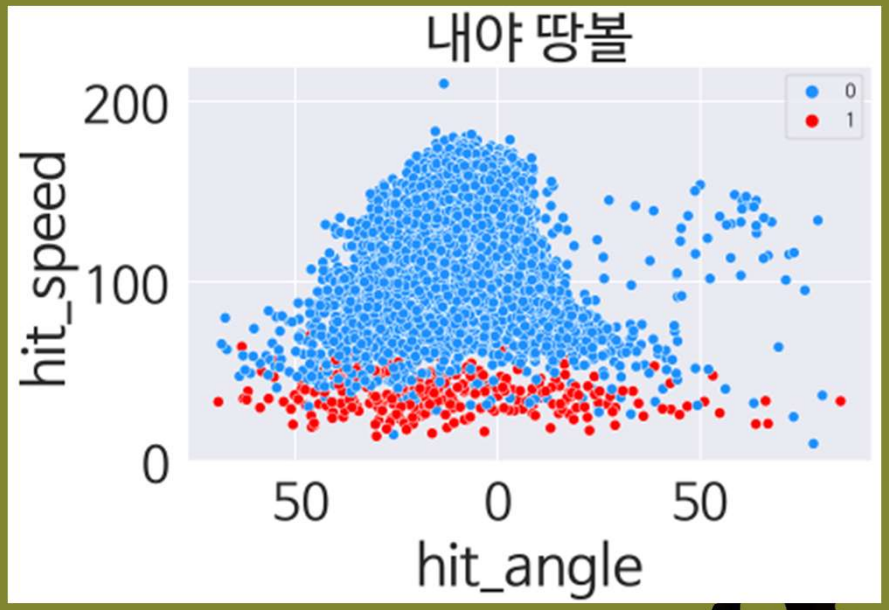


1루 배럴의 정의

- 데이터 탐색

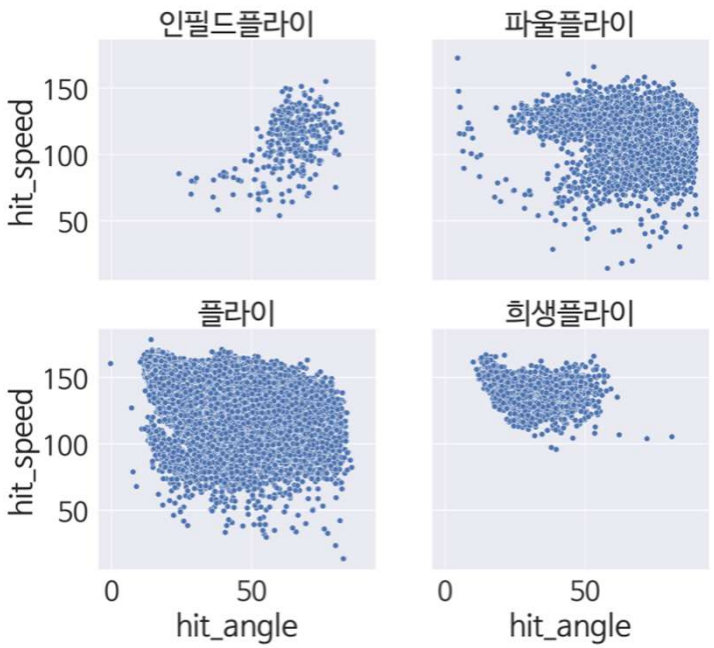


- **땅볼(0, 파란색)**은 일정 발사각과 속도에서 명확하게 **안타(5, 초록색)와 구분되는 구간**이 있음을 확인 할 수 있음

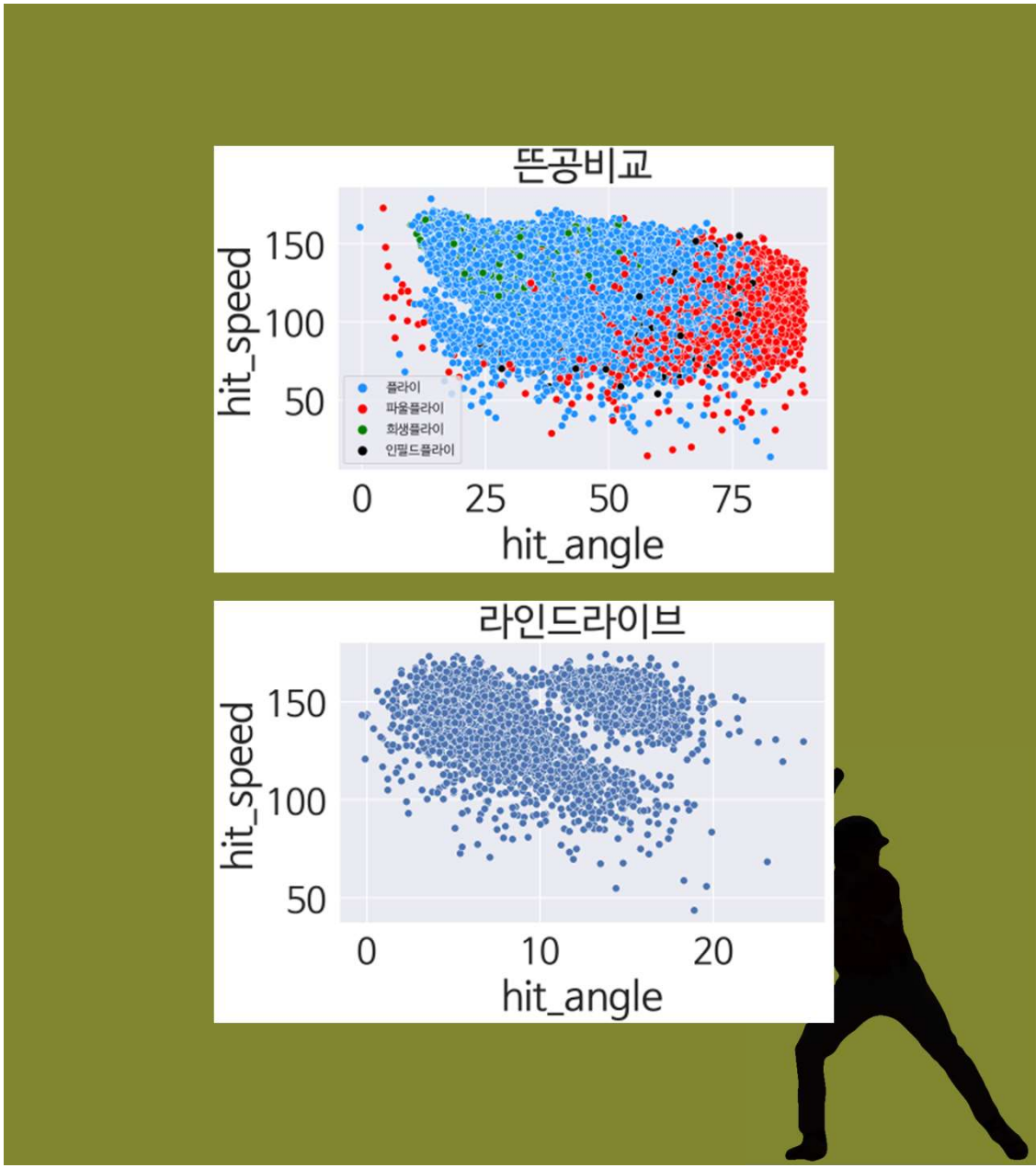


1루 배럴의 정의

- 데이터 탐색

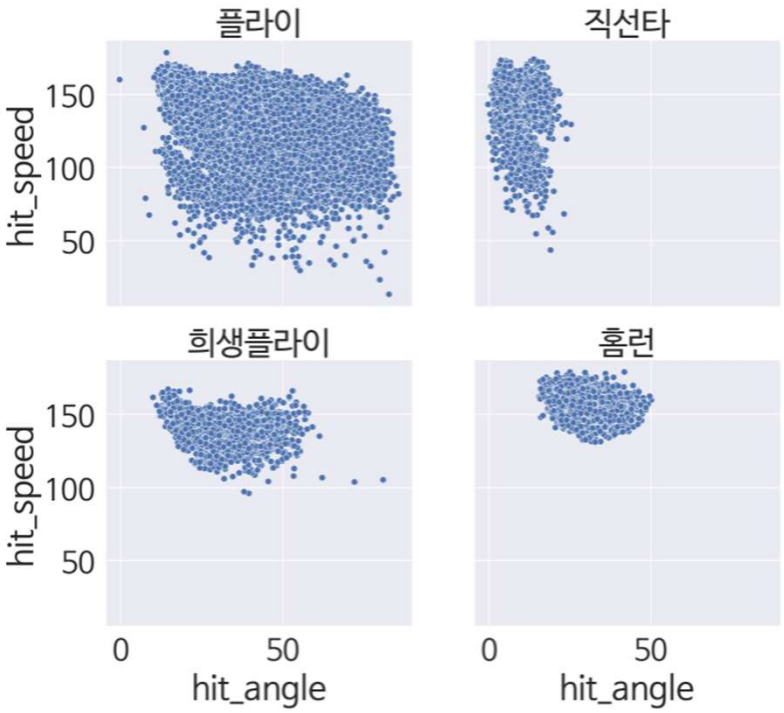


- 뜬공은 전반적으로 모든 영역에 분포하고 있음
- 인필드 플라이와 파울 플라이는 발사각이 대체적으로 높게 형성됨

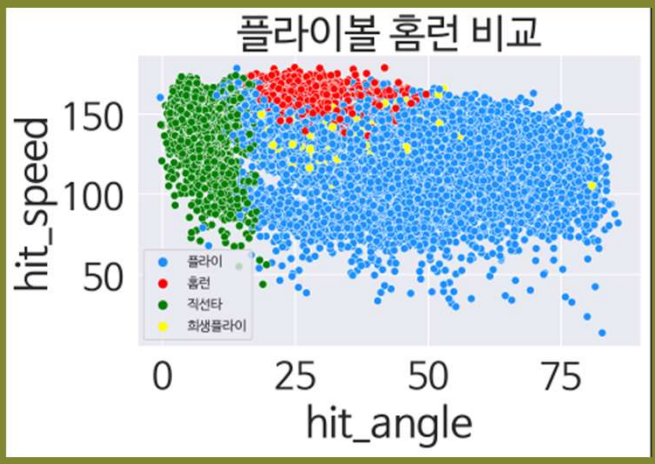


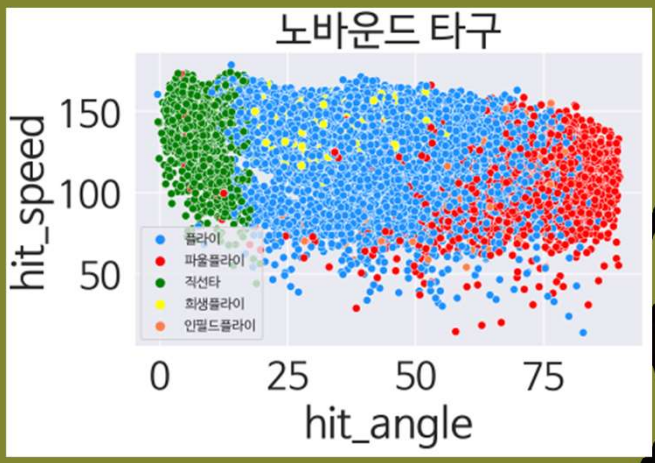
1루 배럴의 정의

- 데이터 탐색



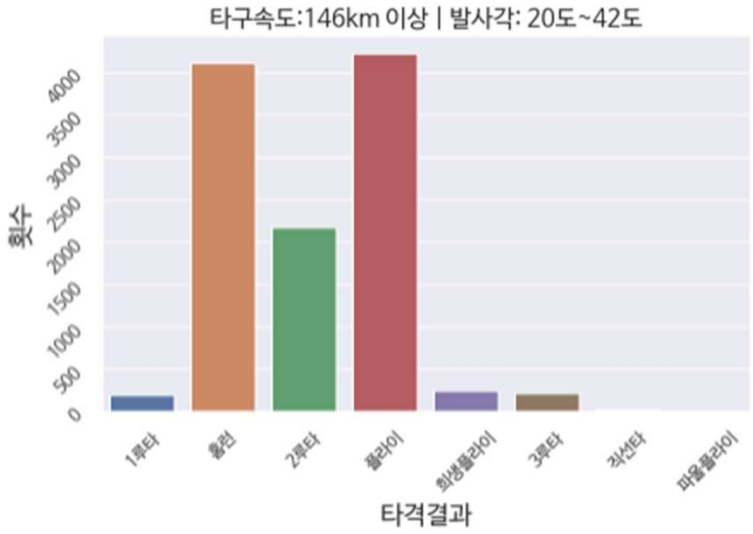
- 라인드라이브는 확실하게 발사각이 낮고 타구 속도가 빠름
- **희생타는 홈런과 비슷한 영역에 분포**하고 있다는 점을 확인





1루 배럴의 정의

- 배럴 기준



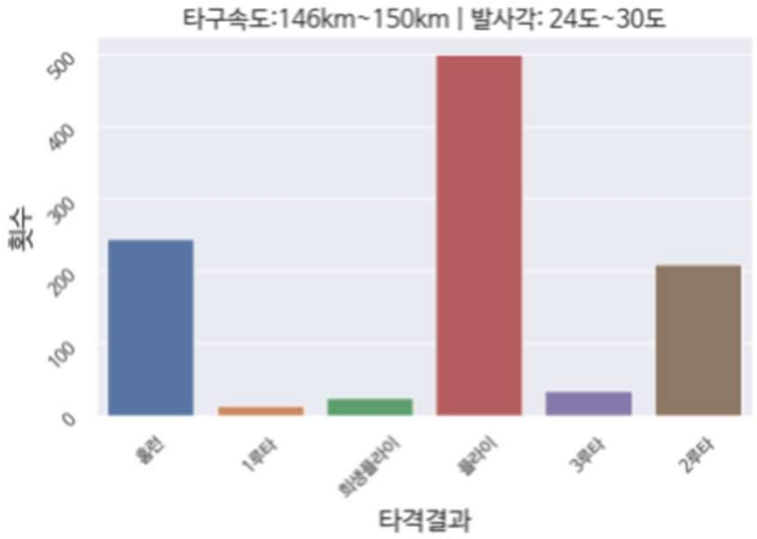

```
[159] barrel5 = hit[(hit.hit_speed >= 146) & (hit.hit_angle >= 18) & (hit.hit_angle <= 44)]
      h_result5 = barrel5.hit_result.value_counts()
      barrel5.hit_result.unique()

      array(['1루타', '홈런', '2루타', '플라이', '희생플라이', '3루타', '직선타', '파울플라이'],
            dtype=object)

      avg =(h_result5['1루타'] + h_result5['2루타'] + h_result5['3루타'] + h_result5['홈런']) / (h_result5.sum() - h_result5['희생플라이'])
      slg = (h_result5['1루타'] + h_result5['2루타']*2 + h_result5['3루타']*3 + h_result5['홈런']*4)/(h_result5.sum() - h_result5['희생플라이'])
      print('최저속도:',146,' 최저/최고 발사각:', 20, 42 ," 타율:",avg," 장타율", slg)

      최저속도: 146  최저/최고 발사각: 20 42  타율: 0.611385685520776l   장타율 1.9771187991945818
```

- 1차적으로 정의된 배럴

140km이상으로 잡아도 배럴 기준에 부합하는 타율과 장타율은 충족하지만 더 **섬세한 정의가 불가**하여 **146km 이상**으로 기준을 잡음

1루 배럴의 정의

- 배럴 기준



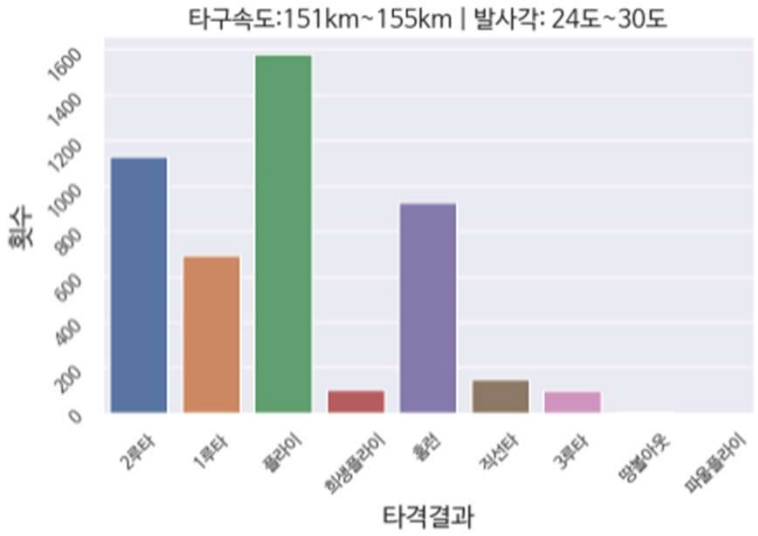
- 세부기준1

146km 이상 150km 이하의 타구에서는
발사각 **24도와 30도 사이**가 적절한 것으로 나타남

```
[ ]  barrel5_1 = hit[(hit.hit_speed >= 146) &(hit.hit_speed <= 150) & (hit.hit_angle >= 24) & (hit.hit_angle <= 30)]
     h_result5_1 = barrel5_1.hit_result.value_counts()
     avg =(h_result5_1['1루타'] + h_result5_1['2루타'] + h_result5_1['3루타'] + h_result5_1['홈런']) / (h_result5_1.sum() - h_result5_1['희생플라이'])
     slg = (h_result5_1['1루타'] + h_result5_1['2루타']*2 + h_result5_1['3루타']*3 + h_result5_1['홈런']*4)/(h_result5_1.sum() - h_result5_1['희생플라이'])
     print('최저/최고 속도:',146,150,' 최저/최고 발사각:', 24,30 ," 타율:",avg,"  장타율",  slg)

     최저/최고 속도: 146 150  최저/최고 발사각: 24 30  타율: 0.50199203187251   장타율 1.5129482071713147
```

1루 배럴의 정의

- 배럴 기준



- 세부기준2

151km 이상 155km 이하의 타구에서는
발사각이 기하급수적으로 증가함

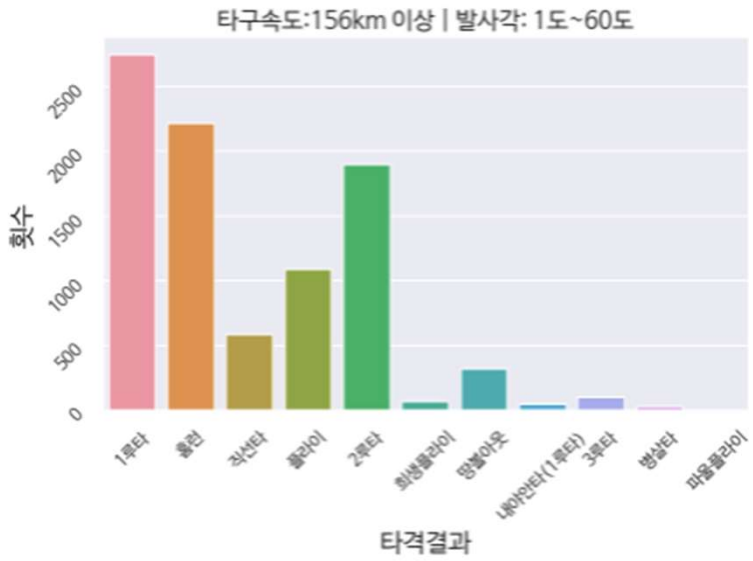
발사각 영역이 **10도와 44도로 증가**하며 타구 속도의 중요성을 시사함

```
barrel5_2 = hit[(hit.hit_speed >= 151) &(hit.hit_speed <= 155) & (hit.hit_angle >= 10) & (hit.hit_angle <= 50)]
h_result5_2 = barrel5_2.hit_result.value_counts()
avg =(h_result5_2['1루타'] + h_result5_2['2루타'] + h_result5_2['3루타'] + h_result5_2['홈런']) / (h_result5_2.sum() - h_result5_2['희생플라이'])
slg = (h_result5_2['1루타'] + h_result5_2['2루타']*2 + h_result5_2['3루타']*3 + h_result5_2['홈런']*4)/(h_result5_2.sum() - h_result5_2['희생플라이'])
print('최저/최고 속도:',151,155,' 최저/최고 발사각:', 10,44 ," 타율:",avg,"  장타율",  slg)
```

```
최저/최고 속도: 151 155  최저/최고 발사각: 10 44  타율: 0.6222707423580786  장타율 1.5192139737991266
```

1루 배럴의 정의

- 배럴 기준




```
[184] barrel5_3 = hit[(hit.hit_speed >= 156) & (hit.hit_angle >= 1) & (hit.hit_angle <= 60)]
      h_result5_3 = barrel5_3.hit_result.value_counts()
      avg =(h_result5_3['1루타'] + h_result5_3['2루타'] + h_result5_3['3루타'] + h_result5_3['홈런']) / (h_result5_3.sum() - h_result5_3['희생플라이'])
      slg = (h_result5_3['1루타'] + h_result5_3['2루타']*2 + h_result5_3['3루타']*3 + h_result5_3['홈런']*4)/(h_result5_3.sum() - h_result5_3['희생플라이'])
      print('최저 속도:',156,' 최저/최고 발사각:', 1,60 ," 타율:",avg,"  장타율",  slg)

      최저 속도: 156  최저/최고 발사각: 1 60  타율: 0.7707133362871068   장타율 1.7399202481169693
```

- 세부기준3

156km 이상 타구에서는 **발사각의 의미가 없어짐** 1도와 60도 사이도 넉넉하게 기준 충족시킴

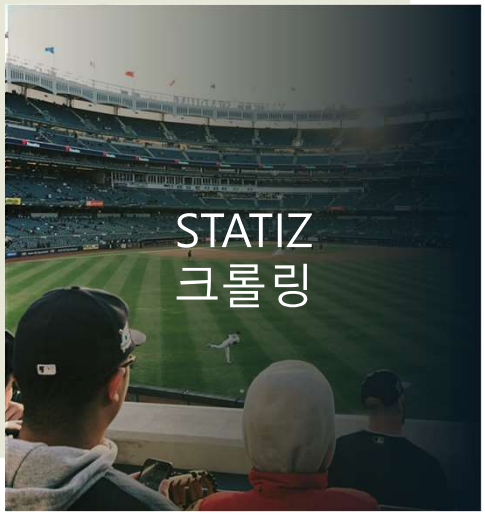
사실상 **타구 속도가 가장 중요**하다는 점을 시사하며 메이저리그의 타구 속도와 비교해서 볼 필요가 있음

타율과 장타율이 기준이기 때문에 안타의 영향이 큰 것으로 보임

배럴 판단 기준이 **생산성이 높은 타구라면 적절한 기준**이지만 **장타 본질의 특성에 맞춰져 있다면 기준을 바꿀** 필요가 있음

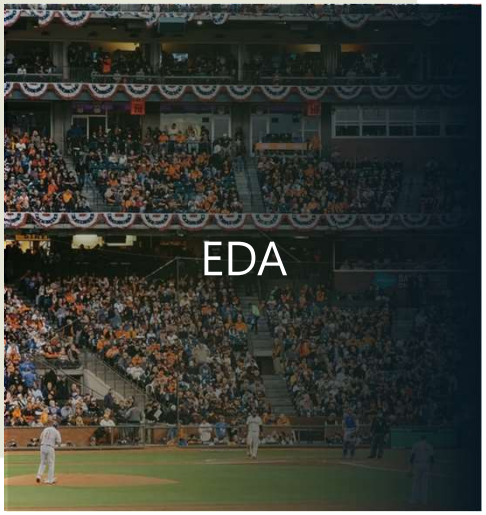
2루 | **모델링을 위한 데이터 탐색**

관전문화의 장점 3가지



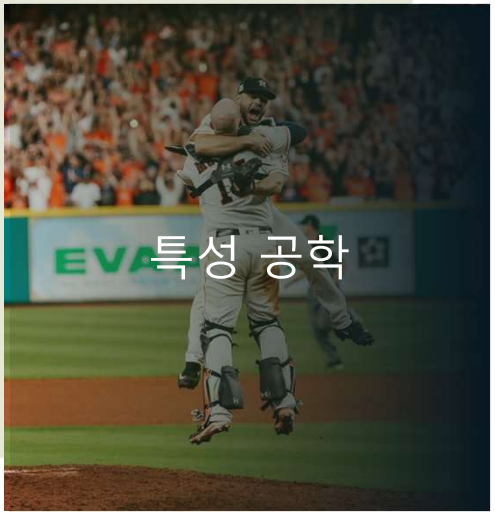
01 타격 지표 데이터 보완

Statiz 사이트에서 2018년도부터 2021년도 까지의 성적을 크롤링



02 유의미한 특성 탐색

가공한 데이터의 특성들과 OPS의 연관성을 탐색



03 모델링을 위한 특성 핸들링

성적 누적 칼럼
N시즌 직전 데이터 칼럼 추가

2루 모델링을 위한 데이터 탐색

- 데이터 수집

- 주어진 Player2018~2021의 타격 지표가 부족하다고 판단
- STATIZ 사이트(http://www.statiz.co.kr/main.php)에서 웹크롤링 실행

```
for i in range(15):
    # 2018년 부터 2021년 까지 statiz에 기록된 선수를 필터링(1415명)
    url = 'http://www.statiz.co.kr/stat.php?mid=stat&re=0&ys=2018&ye=2021&sn=100&pa={}'.format(i*100)

    driver.get(url)
    driver.implicitly_wait(5)

    html = driver.find_element_by_xpath('//*[@id="mytable"]/tbody').get_attribute("innerHTML") #기록 table를 str형태로 저장
    soup = BeautifulSoup(html, 'html.parser') #str 객체를 BeautifulSoup 객체로 변경

    temp = [i.text.strip() for i in soup.findAll("tr")] #tr 태그에서, text만 저장하기
    temp = pd.Series(temp) #list 객체에서 series 객체로 변경

    #'순'이나 'W'로 시작하는 row 제거
    # 즉, 선수별 기록만 남기고, index를 reset 해주기
    temp = temp[~temp.str.match("[순W]")].reset_index(drop=True)

    temp = temp.apply(lambda x: pd.Series(x.split(' '))) #띄어쓰기 기준으로 나눠서 dataframe으로 변경

    #선수 팀 정보 이후 첫번째 기록과는 space 하나로 구분, 그 이후로는 space 두개로 구분이 되어 있음
    #그래서 space 하나로 구분을 시키면, 빈 column들이 존재 하는데, 해당 column를 제거
    temp = temp.replace('', np.nan).dropna(axis=1)

    #WAR 정보가 들어간 column이 2개 있다. (index가 1인 column과, 제일 마지막 column)
    #그 중에서 index가 1인 column 제거
    temp = temp.drop(1, axis=1)

    #선수 이름 앞의 숫자 제거
    temp[0] = temp[0].str.replace("^\d+", '')

    # page별 완성된 dataframe을 계속해서 result에 추가 시켜주기
    if i == 0:
        result = temp
    else:
        result = result.append(temp)
        result = result.reset_index(drop=True)

    print(i, "완료")

#column 명 정보 저장
columns = ['선수'] + [i.text for i in soup.findAll("tr")[0].findAll("th")][4:-3] + ['타율', '출루', '장타', 'OPS', 'wOBA', 'wRC+', 'WAR+','WPA']

#column 명 추가
result.columns = columns

#webdriver 종료
driver.close()

print("최종 완료")
result.to_csv(drive_path+'record_2018_2021.csv',encoding='euc-kr')
```

2루 모델링을 위한 데이터 탐색

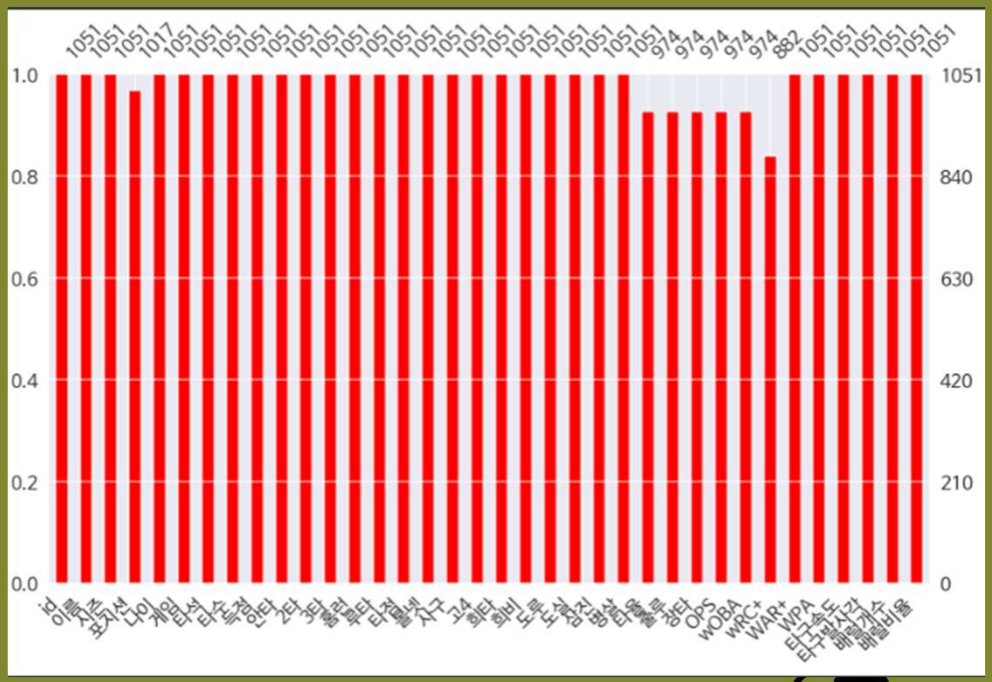
- 전처리 및 EDA

배럴 관련 지표 추출

```
#타구 데이터에서 배럴 관련 지표를 뽑아내기 위한 함수 선언
def hitdata_extract(data):

    # 평균 타구속도 및 발사각 추출
    result = data.groupby('PCODE').mean()[['HIT_VEL','HIT_ANG_VER']]
    result['PCODE'] = result.index

    # 타격을 한 전체 타석 수를 알기 위한 count 추출
    b = data.groupby('PCODE').count().iloc[:,1]
    result['ab'] = b

    # 배럴 타구 개수 추출
    temp = data[((data.HIT_VEL >= 146) & (data.HIT_ANG_VER >= 24) & (data.HIT_ANG_VER <= 30)) |
                ((data.HIT_VEL >= 151) &(data.HIT_VEL <= 155) & (data.HIT_ANG_VER >= 10) & (data.HIT_ANG_VER <= 50)) |
                ((data.HIT_VEL >= 156) & (data.HIT_ANG_VER >= 1) & (data.HIT_ANG_VER <= 59))]
    c = temp.groupby('PCODE').count().iloc[:,1]
    result['barrel'] = c

    #결측치는 0으로 처리
    result.fillna(0,inplace=True)

    #배럴 비율 변수
    result.columns = ['hit_speed','hit_angle','PCODE','ab','br']
    result['br_ratio'] = result['br']/result['ab']

    #추후 데이터 조인을 위한 시즌 구분이 필요 -> 시즌 데이터 만들기
    result['GYEAR'] = data.GYEAR[0]

    result.reset_index(drop = True,inplace=True)

    return result
```

```
[ ] h_ext1= hitdata_extract(hit1)
    h_ext2= hitdata_extract(hit2)
    h_ext3= hitdata_extract(hit3)
    h_ext4= hitdata_extract(hit4)
    h_ext = pd.concat([h_ext1,h_ext2,h_ext3,h_ext4],axis=0)
```

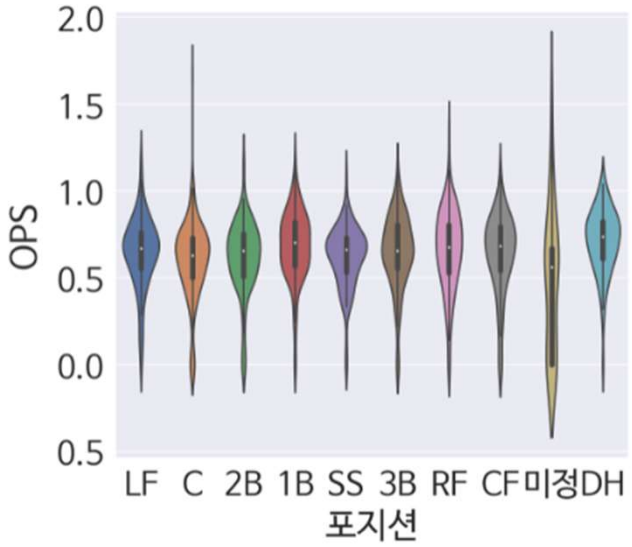
- 앞서 정한 배럴의 기준에 따라 **배럴 타구 횟수**, **배럴 타구 비율**, **평균 발사각**, **평균 속도**를 타구 데이터로부터 추출함 -> **배럴데이터**



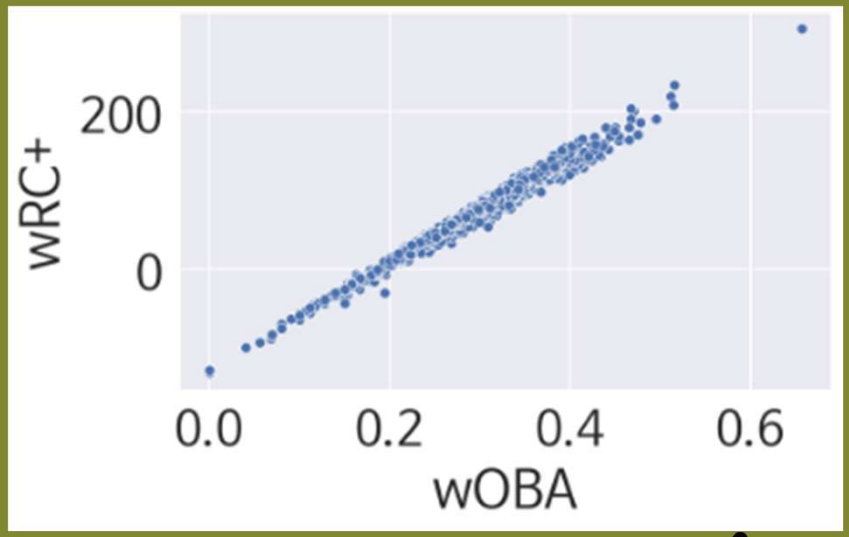
- 크롤링한 데이터에 Player 데이터의 PCODE를 할당 후 배럴 추출 데이터와 병합
- 결측치는 계산식을 사용하여 채우거나 대체 가능한 지표를 사용함

2루 모델링을 위한 데이터 탐색

- 전처리 및 EDA



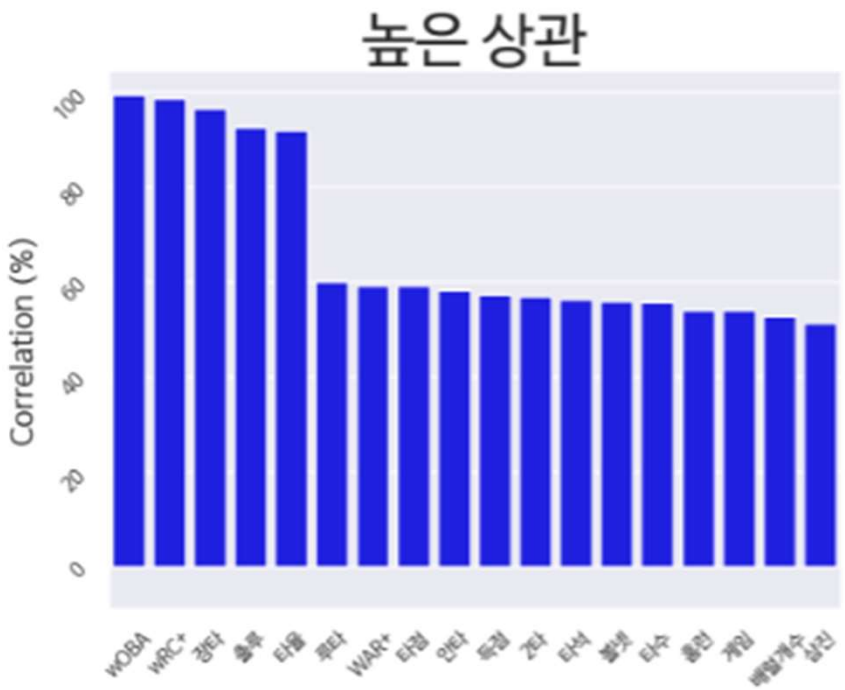
- 포지션별 OPS 차이는 크게 유의미하게 나타나지 않음
- 미정(트레이드 직후, 누락, 참여 게임 수가 적은 경우)집단은 분산이 큼



- **wRC+ 지표는** 계산식으로 결측치를 채우기 힘든 지표였으나 **wOBA와 상관관계가 99프로**로 나타나서 제거함

2루 모델링을 위한 데이터 탐색

- 전처리 및 EDA

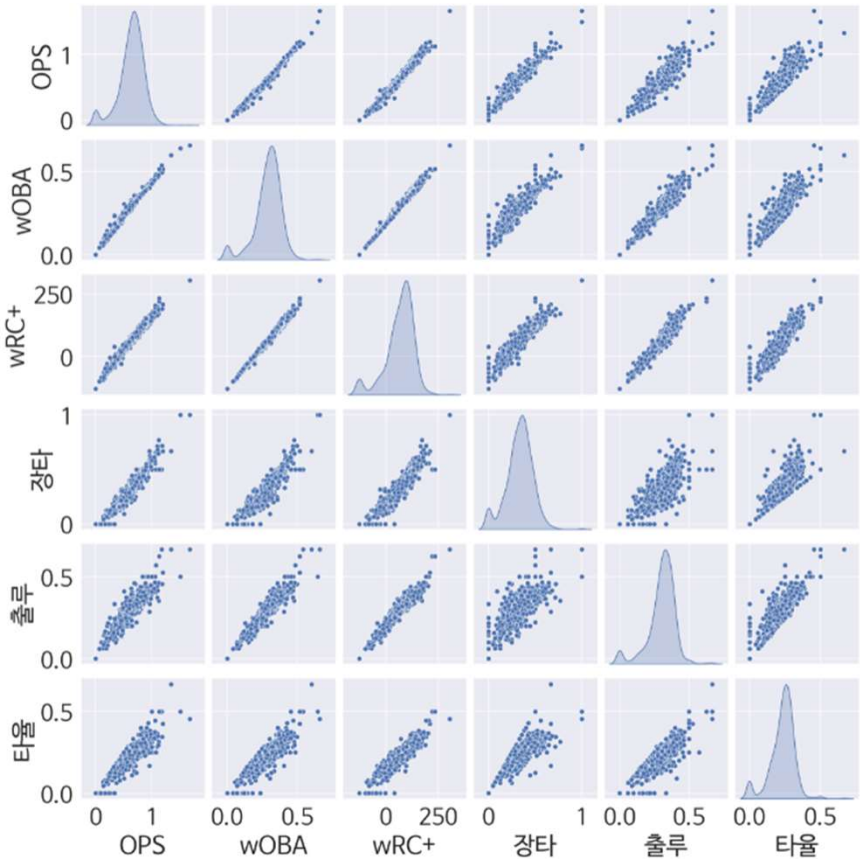


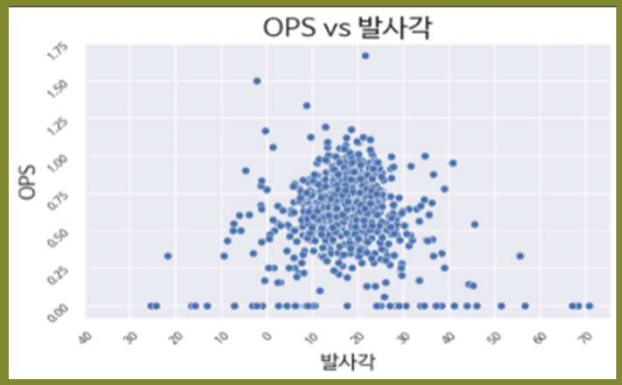
- OPS와 상관관계 0.5 기준으로 나눔

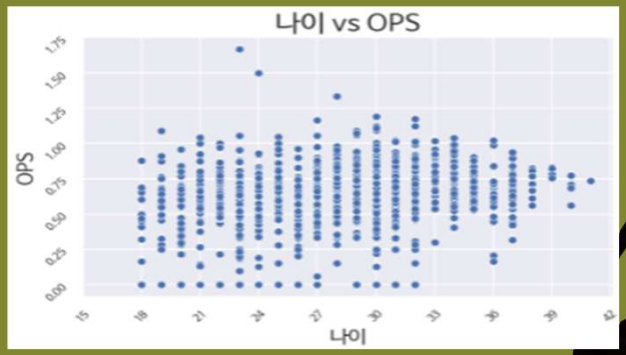
- 타구 발사각이 클 수록 뜬공 타구가 많이 나오기 때문에 음의 상관이 나온 것으로 보임

2루 모델링을 위한 데이터 탐색

- 전처리 및 EDA







- 전반적으로 연속형 변수가 많아서 **산점도를 통해 OPS와 변수의 관계 혹은 변수간의 관계를 확인**함

2루 모델링을 위한 데이터 탐색

- 특성공학

```
[5]  bb = pd.read_csv(drive_path + '최종사용데이터.csv', encoding = 'euc-kr')
     bb.drop('Unnamed: 0',inplace=True, axis=1)
     bb['id'] = bb['id'].astype('str')
     bb['1타'] = bb['안타']- bb['2타'] - bb['3타'] - bb['홈런']

[6]  bbfe = bb[bb.시즌<2021]
     bbfe.shape

     (799, 37)

[7]  # 칼럼 정리
     info = ['id','이름','나이','포지션','시즌']
     stats = ['게임', '타석', '타수', '안타','1타', '2타', '3타','홈런', '루타','득점','타점',
              '볼넷', '사구', '고4', '희타', '희비', '도루', '도실', '삼진', '병살',
              '타구속도', '타구발사각','배럴개수', '배럴비율',
              '타율', 'wOBA', 'WAR+', 'WPA','출루', '장타', 'OPS']
     cum_stats = [i + '_cum' for i in stats]
     lag1_stats = [i + '_lag1' for i in stats]
     lag2_stats = [i + '_lag2' for i in stats]
```

- 1루타 변수를 추가
- 특성 선택과 칼럼 추가 작업을 위한 칼럼 정리

과거 성적 누적(평균) 칼럼 생성

```
[8]  def get_cumulative(player_id, season, stat=stats ,df=bbfe):
         try:
             result_cum = df[(df['id']==player_id) & (df['시즌'] < season)][stat]
             result_cum = result_cum.mean()

         except:
             result_cum = [np.nan for i in stat]
             result_cum = pd.Series(result_cum)

         return result_cum

     bbfe[cum_stats] = bbfe.apply(lambda x: get_cumulative(x['id'],x['시즌']),axis=1)
```

- **과거 누적 데이터 칼럼 생성을 위한 함수 선언**
- 과거 누적 성적으로 특성으로 사용

2루 모델링을 위한 데이터 탐색

- 특성공학

n시즌 과거 칼럼 생성

```
def get_past(player_id, season, lag, stat=stats, df=bbfe):
  try:
    result_lag = df[(df['id'] == player_id) & (df['시즌'] == season - lag)][stat].values[0]
    result_lag = pd.Series(result_lag)
  except:
    result_lag = [np.nan for i in stat]
    result_lag = pd.Series(result_lag)

  return result_lag
```

```
# 1시즌 과거 칼럼 추가
bbfe[lag1_stats] = bbfe.apply(lambda x: get_past(x['id'],x['시즌'],1),axis=1)
# 2시즌 과거 칼럼 추가
bbfe[lag2_stats] = bbfe.apply(lambda x: get_past(x['id'],x['시즌'],2),axis=1)
```

- **1시즌, 2시즌(n시즌) 과거 칼럼 생성을 위한 함수 선언**
- 과거의 데이터를 특성으로 사용할 예정

Train Test Set

```
[15]
    #Train
    train3_1 = bb[bb.시즌<2020]
    # 과거 누적 평균
    train3_1[cum_stats] = train3_1.apply(lambda x: get_cumulative(x['id'],x['시즌']),axis=1)
    # 1시즌 과거 칼럼 추가
    train3_1[lag1_stats] = train3_1.apply(lambda x: get_past(x['id'],x['시즌'],1),axis=1)
    # 정답
    train_ans = bb[bb.시즌 == 2020][['id','출루','장타','OPS']]
    train_ans.columns = ['id','출루_미래','장타_미래','OPS_미래']
    train3_1 = pd.merge(train3_1,train_ans,how='inner',on = 'id')
    train3_1 = train3_1[train3_1['시즌']==2019]

    #Test
    train3_2 = bb[bb.시즌<2021]
    # 과거 누적 평균
    train3_2[cum_stats] = train3_2.apply(lambda x: get_cumulative(x['id'],x['시즌']),axis=1)
    # 1시즌 과거 칼럼 추가
    train3_2[lag1_stats] = train3_2.apply(lambda x: get_past(x['id'],x['시즌'],1),axis=1)
    # 정답
    train_ans = bb[bb.시즌 == 2021][['id','출루','장타','OPS']]
    train_ans.columns = ['id','출루_미래','장타_미래','OPS_미래']
    train3_2 = pd.merge(train3_2,train_ans,how='inner',on = 'id')
    train3_2 = train3_2[train3_2['시즌']==2020]
```

예측데이터

```
test3_2 = bb[bb.시즌 <= 2021]
# 과거 누적 평균
test3_2[cum_stats] = test3_2.apply(lambda x: get_cumulative(x['id'],x['시즌']),axis=1)
# 1시즌 과거 칼럼 추가
test3_2[lag1_stats] = test3_2.apply(lambda x: get_past(x['id'],x['시즌'],1),axis=1)
test3_2 = test3_2[test3_2.시즌 == 2021]
```

```
train3_1 = train3_1.fillna(-1)
train3_2 = train3_2.fillna(-1)
test3_2 = test3_2.fillna(-1)
```

- 학습, 평가, 예측을 위한 데이터 셋 설정

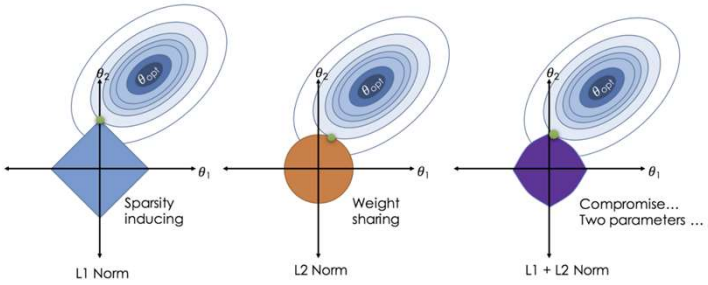
3루 **타자 성적 예측 모델**

- 모델 탐색

XGBoost

LightGBM

Random Forest



XGBoost, LightGBM, Lasso, Ridge, ElasticNet, KNN,
Support Vector Machine
모델 별로 학습, 평가를 진행

학습 데이터: 2019 시즌 데이터 size=(194x102)

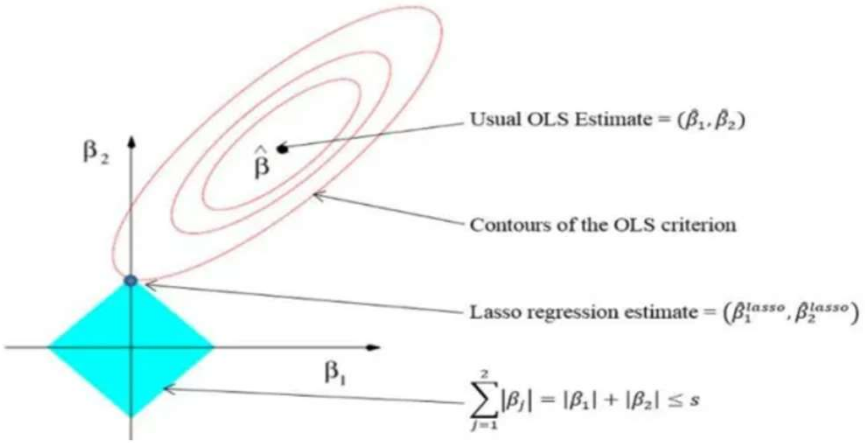
Feature			Target
2019 성적	2019 누적 성적	2018 성적 (1년 전)	2020 출루율 장타율 OPS

평가 데이터: 2020 시즌 데이터 size=(200x102)

Feature			Target
2020 성적	2020 누적 성적	2019 성적 (1년 전)	2021 전반기 출루율 장타율 OPS

3루 타자 성적 예측 모델

- 모델 채택



LASSO 모델 채택

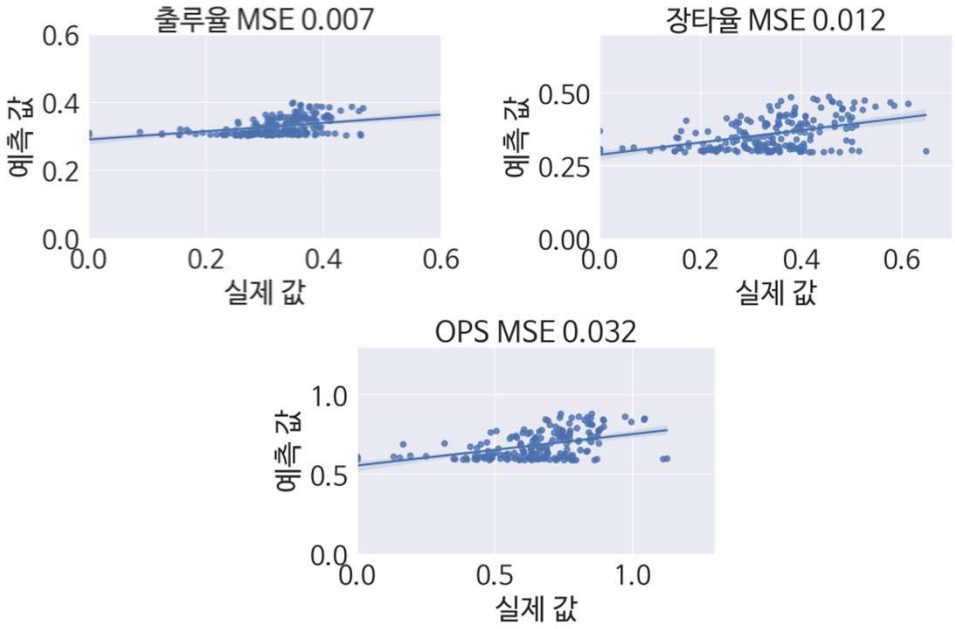
- **학습 데이터가 행에 비해 열이 더 많고, 다중 공선성과 과적합을 해소하는 것이 주요 이슈**
- Ridge 모델은 계수를 완전히 제거하지 않고 0에 가깝게 패널티 부여
- LASSO 모델은 **불필요한 칼럼의 계수를 0으로 만들 수 있기 때문에**(Xu et.al, 2010) 다중 공선성의 리스크가 큰 본 과제에 적합하다고 판단

예측 데이터: 2021 시즌 전반기 데이터 size=(236x99)

Feature			Target
2021 전반기 성적	2021 누적 성적	2020 성적 (1년 전)	없음

3루 타자 성적 예측 모델

- 모델 채택



- OPS = 장타율 + 출루율이라고 볼 수 있음
- 장타율과 출루율의 MSE가 OPS의 MSE보다 낮기 때문에 출루율과 장타율을 각각 추정하는 것이 좋다고 판
- **추정한 출루율과 장타율을 통해 OPS를 추정함**

예측 결과

	PCODE	OPS	장타율	출루율
0	67341	0.815	0.441	0.374
1	67872	0.800	0.435	0.365
2	68050	0.838	0.451	0.387
3	75847	0.813	0.441	0.372
4	76232	0.817	0.444	0.373
5	76290	0.808	0.437	0.371
6	78224	0.807	0.442	0.365
7	78513	0.810	0.446	0.364
8	79192	0.758	0.417	0.341
9	79215	0.772	0.422	0.350

References

<크롤링 데이터>
스탯티즈 기록실 데이터 (2018 ~ 2021. 9. 12) http://www.statiz.co.kr/main.php

<사진 출처>
웹사이트 MLB.COM(2019 . 9. 15)
(http://m.mlb.com/glossary/statcast/barrel/)

<참고 논문>
Xu, H., Caramanis, C., & Mannor, S. (2010). Robust regression and lasso. *IEEE Transactions on Information Theory*, *56*(7), 3561-3574.


감사합니다
Thank you for listening