

2020 SUMMER ESC

빅콘테스트 퓨처스리그 최종발표

7조 곽현지 김채형 박상재 오탁환 이승준



CONTENTS

01

Review

02

EDA

03

Prediction

04

Conclusion

01

Review

“팀 기록을 이용해서 예측”

VS

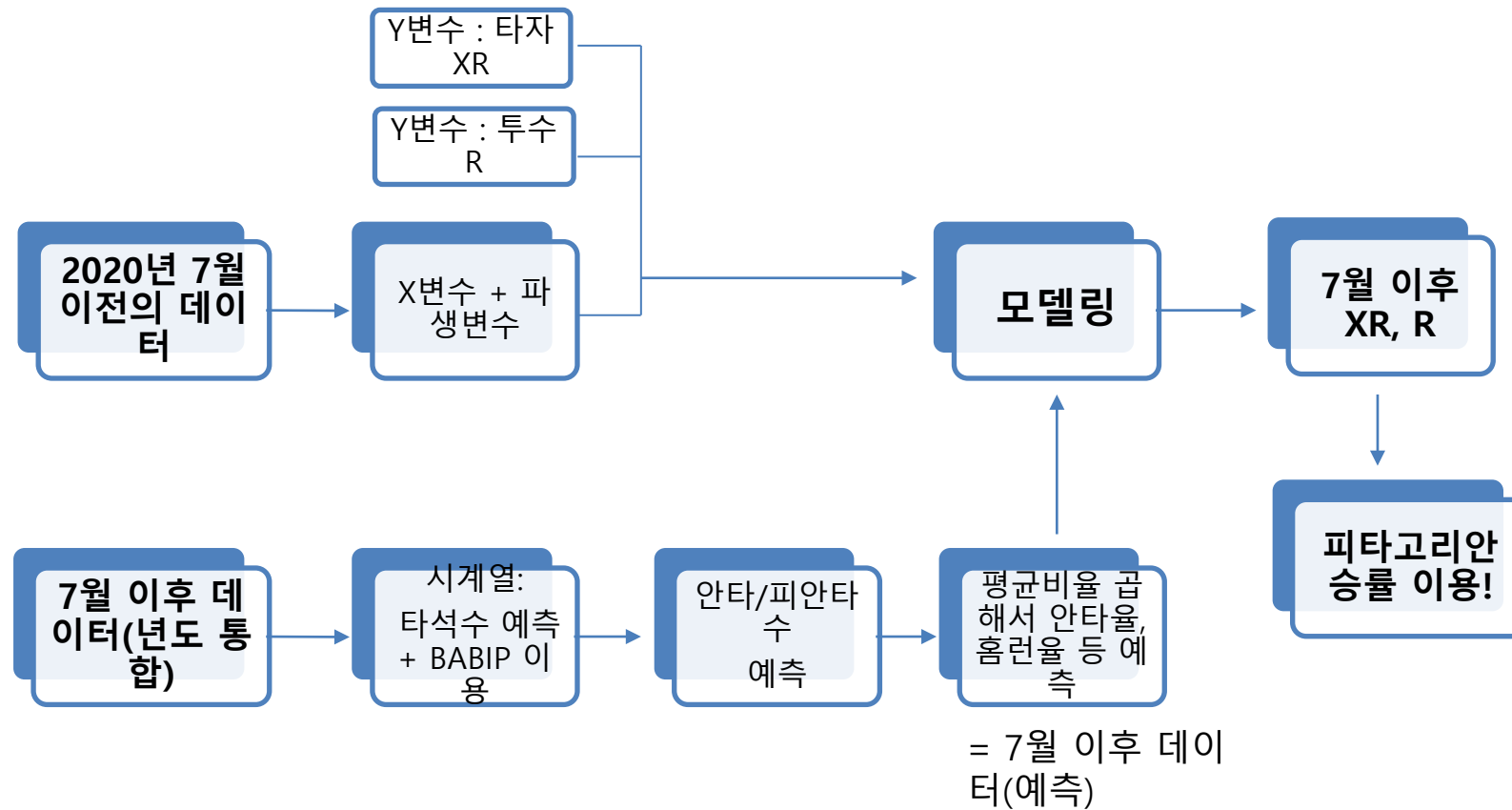
“선수 개개인 성적의 합으로 예측”

“팀 기록을 이용해서 예측”

VS

“ 선수 개개인 성적의 합으로 예측 ”

데이터 분석 흐름도



02

EDA

새로운 변수 생성

투수



PA, BK, R, AVG, OBP, SLG, OPS, KBB, H2_rate,
H3_rate, HR_rate, BABIP

타자



PA, AVG, OBP, SLG, OPS, XR, SB_rate, H2_rate,
H3_rate, BB_rate, HR_rate, BABIP

새로운 변수 생성

출루율

```
df['OBP'] = (df['HIT'] + df['BB'] + df['HP']) / (df['AB'] + df['BB'] + df['HP'] + df['SF'])
```

```
df[['HIT', 'BB', 'HP', 'AB', 'SF', 'OBP']]
```

타율

```
df['AVG'] = df['HIT'] / df['AB']
```

```
df[['AB', 'HIT', 'AVG']]
```

```
> pitcher_all <- read.csv('C:/Users/seungjun/Desktop/baseball/Baseball_ChilliShrimp/data/pitcher_all.csv',header=T)
> names(pitcher_all)
 [1] "x"          "year"       "month"      "T_ID"       "P_ID"       "START_CK"   "RELIEF_CK"  "CG_CK"      "HOLD"
[10] "INN2"       "BF"         "PA"         "AB"         "HIT"        "H2"         "H3"         "HR"         "SB"
[19] "CS"         "SH"         "SF"         "BB"         "HP"         "KK"         "GD"         "WP"
[28] "BK"         "ERR"        "R"          "ER"         "VS_T_ID_HH" "VS_T_ID_HT" "VS_T_ID_KT" "VS_T_ID_LG" "VS_T_ID_LT"
[37] "VS_T_ID_NC" "VS_T_ID_OB" "VS_T_ID_SK" "VS_T_ID_SS" "VS_T_ID_WO" "TB_SC_B"    "TB_SC_T"    "WLS_D"      "WLS_L"
[46] "WLS_ND"     "WLS_S"     "WLS_W"     "AVG"        "OBP"        "SLG"        "OPS"        "KBB"        "BABIP"
```

다음

```
df['AVG'] = df['HIT'] / df['AB']
```

```
df[['AB', 'HIT', 'AVG']]
```

	AB	HIT	AVG
0	12	2	0.166667
1	9	3	0.333333
2	0	0	NaN
3	0	0	NaN
4	31	6	0.193548
...



선수 스탯을
비율 스탯으로 바꾸는 과정에서
NaN, INF 문제

	PA	AVG
2	0	NaN
3	1	NaN
14	0	NaN
17	0	NaN
18	0	NaN
...
6882	0	NaN
6885	0	NaN
6907	0	NaN
6920	0	NaN
6928	0	NaN

709 rows × 2 columns

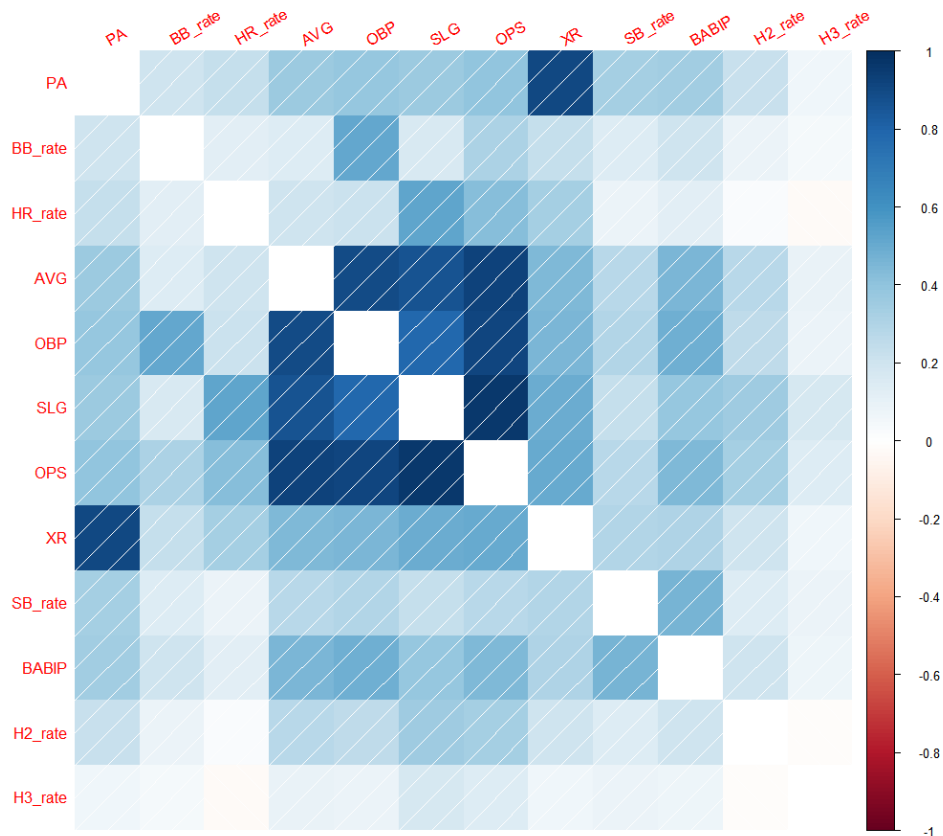
```
df2[['PA']].max()
```

```
PA      3
dtype: int64
```



**NaN, INF 의 경우
타석수가 0이거나 매우 작은 수
0으로 바꾸어도 큰 영향 X**

타자의 변수 선택

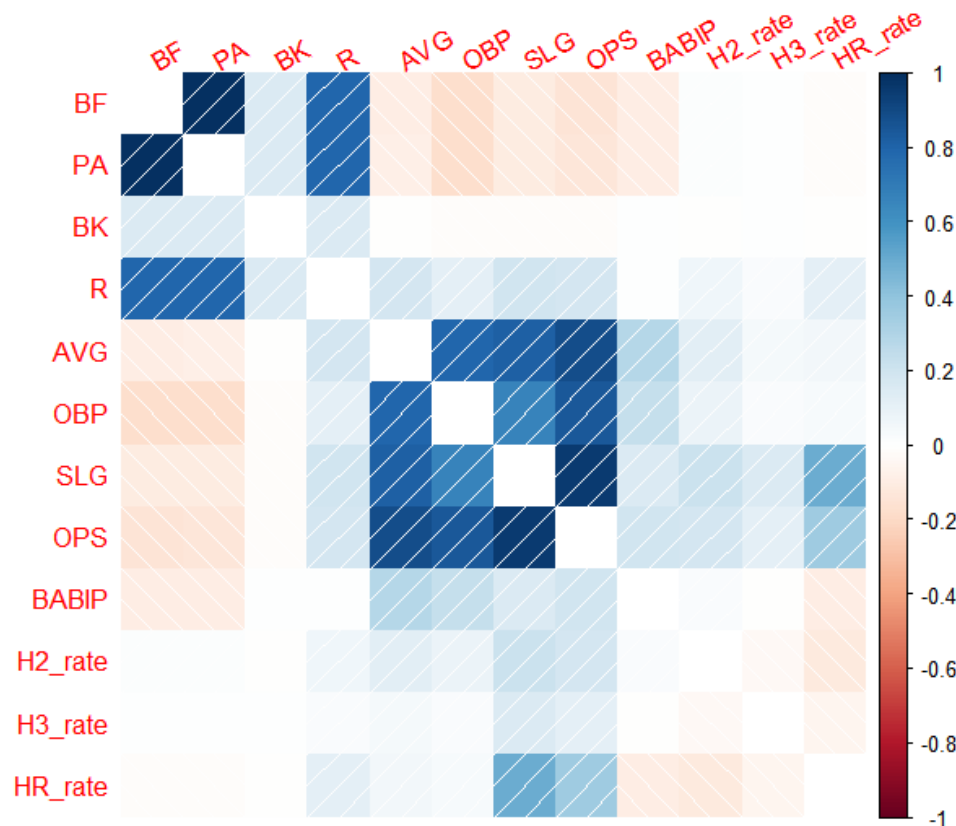


**파생변수를 만드는데 사용
된 변수 제거**

**각 변수에 대한 상관관계
고려해서 변수 선택**

**최종 변수로는
타석수, 볼넷율, 2루타율, 3
루타율, 홈런율, AVG, OBP,
SLG, OPS, XR, 도루율,
BABIP, 그 외 더미변수**

투수의 변수 선택



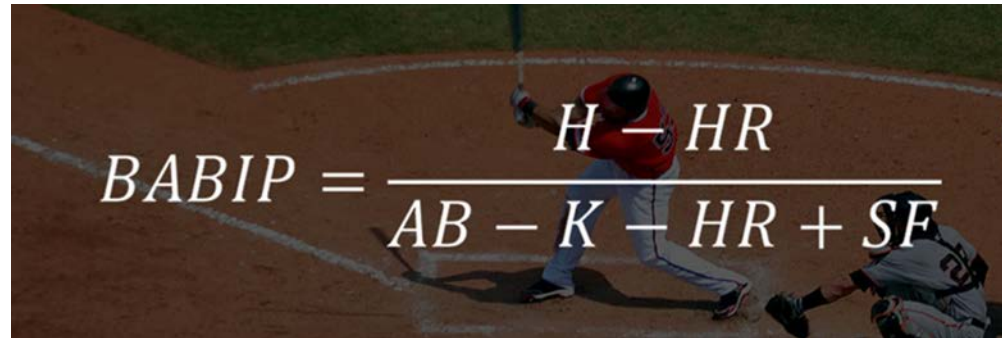
**파생변수를 만드는데 사용
된 변수 제거**

**각 변수에 대한 상관관계
고려해서 변수 선택**

최종 변수로는
BF, PA, BK, R, AVG, OBP,
SLG, OPS, BABIP, 2루타율,
3루타율, 홈런율, 그 외 더
미변수들

03

Prediction



$$BABIP = \frac{H - HR}{AB - K - HR + SF}$$

이름	2008 타율	2008 BABip	통산 BABip	2009 타율
최정	0.328	0.358	0.299	0.265
조성환	0.327	0.369	0.307	0.294
박재홍	0.318	0.341	0.307	0.27
이용규	0.312	0.342	0.317	0.266
안치용	0.295	0.348	0.171	0.237
브룸바	0.293	0.341	0.328	0.245
박기혁	0.291	0.33	0.283	0.217

(타율 20위 이내 선수 중)


$$TOTAL\ BABIP = \frac{Inplay\ hit(\sim 20.07.19) + x}{Inplay(\sim 20.07.19) + Predicted\ inplay}$$

시계열 분석을 통해 2020년 7월 17일 이후 선수 별 잔여 타석을 구하자!

예측한 타석에 해당 선수의 평균 인플레이 타구 비율을 곱해 2020년 7월 19일 이후 predicted inplay를 구하자!

위의 방정식을 풀어 미지수를 구하자!

PROPHET 을 이용한 잔여 타석 예측



Forecasting at scale.

Prophet is a forecasting procedure implemented in R and Python. It is fast and provides completely automated forecasts that can be tuned by hand by data scientists and analysts.

[INSTALL PROPHET](#)
[GET STARTED IN R](#)
[GET STARTED IN PYTHON](#)
[READ THE PAPER](#)

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

Prophet is [open source software](#) released by Facebook's [Core Data Science team](#). It is available for download on [CRAN](#) and [PyPI](#).

<p>Accurate and fast.</p> <p>Prophet is used in many applications across Facebook for producing reliable forecasts for planning and goal setting. We've found it to perform better than any other approach in the majority of cases. We fit models in Stan so that you get forecasts in just a few seconds.</p>	<p>Fully automatic.</p> <p>Get a reasonable forecast on messy data with no manual effort. Prophet is robust to outliers, missing data, and dramatic changes in your time series.</p>	<p>Tunable forecasts.</p> <p>The Prophet procedure includes many possibilities for users to tweak and adjust forecasts. You can use human-interpretable parameters to improve your forecast by adding your domain knowledge.</p>	<p>Available in R or Python.</p> <p>We've implemented the Prophet procedure in R and Python, but they share the same underlying Stan code for fitting. Use whatever language you're comfortable with to get forecasts.</p>
--	---	---	---

The logo for Facebook's Prophet forecasting tool. It consists of the word "PROPHET" in a white, sans-serif font, centered within a dark blue rectangular box. The letter "O" is replaced by a white circular icon containing a dot and a curved arrow, suggesting a cycle or trend.

$$y(t) = g(t) + s(t) + h(t) + \epsilon_i$$

$g(t)$: 'Trend' 주기적이지 않는 변화

$s(t)$: 'Seasonality' 주기적으로 나타나는 패턴

$h(t)$: 'Holiday' 휴일과 같이 불규칙한 이벤트

e : 정규분포를 따르는 잔차

통산 BABIP를 이용해 구한 인플레이 안타수(피안타수)

Out[26]:

P_ID	year	month	PA	inplay_H
50054	2020	7	7	2
		8	27	6
		9	32	7
		10	29	7
50066	2020	7	0	0
		8	0	0
		9	0	0
		10	0	0
50150	2020	7	8	3
		8	13	4

타자

Out[33]:

P_ID	T_ID	year	month	PA	inplay_H
50030	KT	2020	7	40	9
			8	64	15
			9	60	14
			10	53	12
50036	KT	2020	7	4	0
			8	16	3
			9	18	3
			10	16	3
50040	KT	2020	7	82	18
			8	160	36

투수

타자 예측 테이블

	PA	BB_rate	HR_rate	AVG	OBP	SLG	OPS	SB_rate	BABIP	H2_rate	...
0	7	0.113208	0.000000	0.322188	0.398922	0.402736	0.801657	0.037736	0.315789	0.250000	...
1	27	0.113208	0.000000	0.250591	0.335430	0.313239	0.648669	0.037736	0.315789	0.250000	...
2	32	0.113208	0.000000	0.246676	0.331958	0.308344	0.640302	0.037736	0.315789	0.250000	...
3	29	0.113208	0.000000	0.272194	0.354587	0.340242	0.694829	0.037736	0.315789	0.250000	...
4	0	0.074074	0.166667	0.000000	0.000000	0.000000	0.000000	0.000000	0.277778	0.333333	...
5	0	0.074074	0.166667	0.000000	0.000000	0.000000	0.000000	0.000000	0.277778	0.333333	...
6	0	0.074074	0.166667	0.000000	0.000000	0.000000	0.000000	0.000000	0.277778	0.333333	...
7	0	0.074074	0.166667	0.000000	0.000000	0.000000	0.000000	0.000000	0.277778	0.333333	...
8	8	0.000000	0.000000	0.375000	0.375000	0.482143	0.857143	0.142857	0.411765	0.285714	...
9	13	0.000000	0.000000	0.307692	0.307692	0.395604	0.703297	0.142857	0.411765	0.285714	...

10 rows × 28 columns

투수 예측 테이블

	BF	PA	BK	AVG	OBP	SLG	OPS	BABIP	H2_rate	H3_rate	...
0	148.992248	40	0.0	0.266774	0.323613	0.387652	0.711265	0.316832	0.214286	0.000000	...
1	238.387597	64	0.0	0.277889	0.333867	0.403805	0.737672	0.316832	0.214286	0.000000	...
2	223.488372	60	0.0	0.276654	0.332728	0.402010	0.734738	0.316832	0.214286	0.000000	...
3	197.414729	53	0.0	0.268452	0.325161	0.390090	0.715251	0.316832	0.214286	0.000000	...
4	13.212121	4	0.0	0.000000	0.242424	0.000000	0.242424	0.400000	0.111111	0.111111	...
5	52.848485	16	0.0	0.278438	0.453362	0.414342	0.867703	0.400000	0.111111	0.111111	...
6	59.454545	18	0.0	0.247500	0.429924	0.368304	0.798228	0.400000	0.111111	0.111111	...
7	52.848485	16	0.0	0.278438	0.453362	0.414342	0.867703	0.400000	0.111111	0.111111	...
8	320.279898	82	0.0	0.259224	0.312002	0.373553	0.685556	0.329787	0.176471	0.009804	...
9	624.936387	160	0.0	0.265705	0.318021	0.382892	0.700913	0.329787	0.176471	0.009804	...

10 rows × 28 columns

LGBM 모델링

1-1) LGBM Modeling

```
[14] 1 from sklearn.model_selection import train_test_split
      2
      3 X = batter.drop(["Unnamed: 0", "XR"], axis = 1)
      4 Y = batter['XR']
```

```
[15] 1 X['T_ID'] = X['T_ID'].astype('category')
      2 X['P_ID'] = X['P_ID'].astype('category')
      3 X['BAT_ORDER'] = X['BAT_ORDER'].astype('category')
```

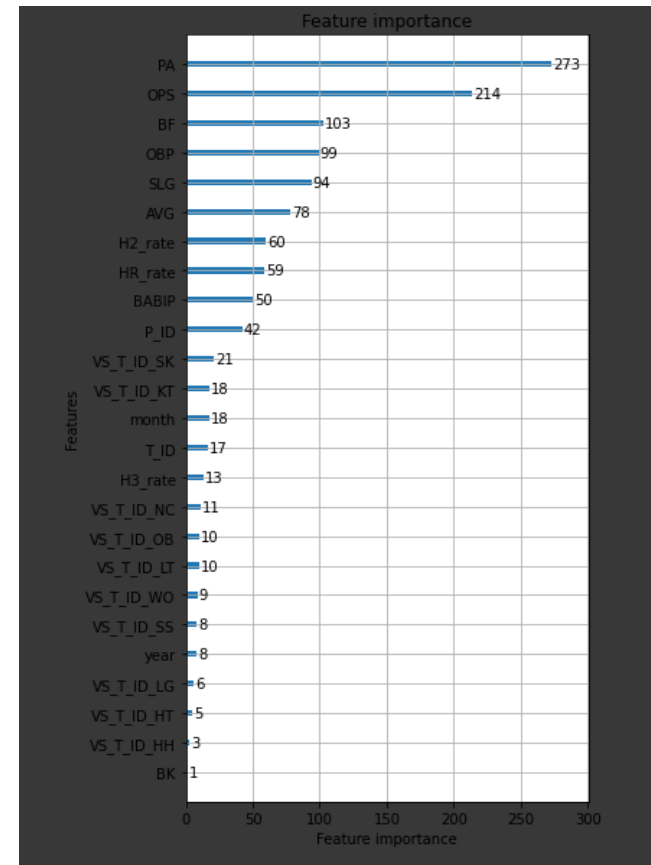
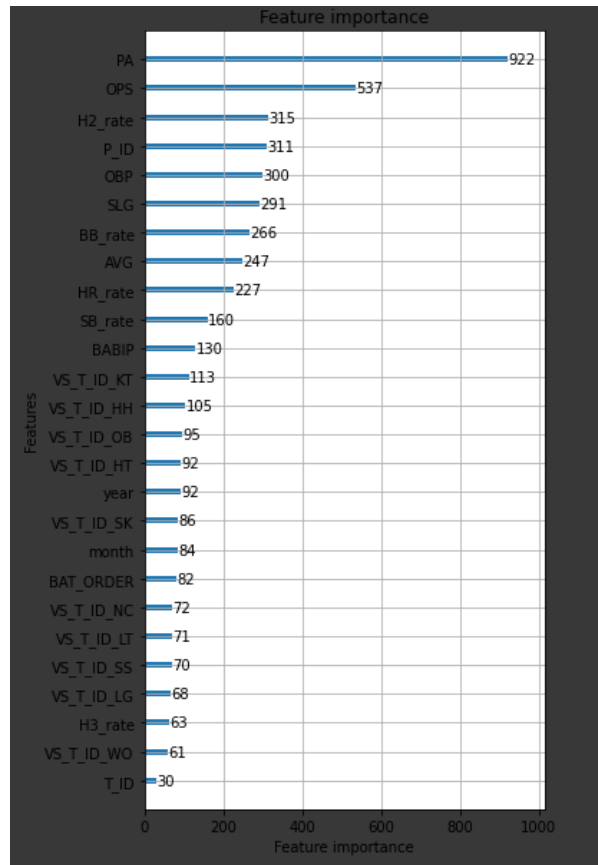
```
[16] 1 x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state = 99)
```

```
[17] 1 lgbm = LGBMRegressor(n_estimators=200, random_state=99)
```

```
[18] 1 from sklearn.model_selection import GridSearchCV
      2 from sklearn.metrics import mean_squared_error
```

```
[19] 1 params = {'max_depth': [10, 15, 20, 25, 30],
      2         'min_child_samples': [20, 40, 60, 80, 100],
      3         'subsample': [0.8, 1]}
      4
      5 grid = GridSearchCV(lgbm, param_grid=params)
      6 grid.fit(x_train, y_train, early_stopping_rounds=100, eval_metric='MSE',
      7         eval_set=[(x_train, y_train), (x_test, y_test)])
```

Feature Importance



승률 예측

```
[93] 1 r_by_t = pitcher[['T_ID', 'R']]
      2 r_by_t = r_by_t.groupby(['T_ID']).sum()
      3 r_by_t.sort_values(by = 'R', ascending=False)
      4 r_by_t = r_by_t * medxr / ((3367 + 3261)/2 )
```

```
[96] 1 rn = r_by_t ** 1.83
```

```
[97] 1 xrn = xr_by_t ** 1.83
```

```
[105] 1 rn['WIN'] = rn['R']
        2 xrn['WIN'] = xrn['XR']
        3
        4 rn = rn[['WIN']]
        5 xrn = xrn[['WIN']]
```

```
1 xrn / (xrn + rn)
```

	WIN
T_ID	
HH	0.549349
HT	0.464127
KT	0.347003
LG	0.534306
LT	0.436549
NC	0.573297
OB	0.481133
SK	0.569646
SS	0.283805
WO	0.550113

피타고리안 승률(PE)을 통한 승률 계산

$$P = \frac{W^n}{W^n + L^n}$$

W는 득점, L은 실점

n에는 보통 2를 넣어주나 정확도를 높이기 위해 1.83을 대입하기도 한다.

득점과 실점을 기준으로 만들어진 지표이므로 득실점이 많이 나고 경기 수가 많을수록 신뢰도가 좋다.

04

Conclusion

향후 계획 – 추가

- 팀 간 선수 트레이드 반영 추가
- 상대 팀에 따른 성적 변동 반영 추가
- 다양한 모델링을 통한 최적의 모델 선택
- 시계열 예측에 RNN을 적용 할 수 있는 방안 모색

향후 계획 – 보완

- 투수의 경우 MSE가 높아서 변수 수정 필요
- R0이 크게 나오는 부분을 보완할 필요가 있다
 - XR과 팀 득점 간의 차이가 커서 타자 예측 변수 수정필요
- 투수의 R 과대평가 된 것 조정 필요

**Thank
you**