머신러닝을 이용한 2022년 KBO 리그 우승팀 예측

2조 김바램
이종승
이주연

Table of Contents

1

프로젝트 목표

프로젝트 플랜

STEP 1	STEP 2	STEP 3	STEP 4
2022년도 야구 우승팀은 어느 구단이 될까?	STATIZ와 KBO 공식사이트를 통해 자료조사를 하고 데이터를 분석해보면 어떨까?	기존 데이터 지도학습 후 현재 성적과 비교 했을 때 얼마나 유의미한 값을 도출할 수 있을까?	그래서 2022년 포스트시즌에 진출하는 팀은 어느 구단이 될까?

2

기획의도 및 배경

신문기사

Sports & Data Analysis

예전의 '명감독'은 잊어라, 데이터가 우승을 이끈다

286호 (2019년 12월 Issue 1)

Article at a Glance

이미 십수 년 전에 미국에서 본격화한 후 한국 프로야구에도 도입되기 시작한 데이터 분석 야구와 전략 수립 방법론은 스포츠계, 야구계를 통째로 바꾸고 있다. 다음은 한 야구 전문가가 데이터 중심 구단을 만들기 위해 제시한 조언 네 가지다. 구단이 아니라 '기업'이라고 생각해도 완벽하게 맞아떨어진다.

1. 데이터 자체는 아무것도 바꿀 수 없다. 데이터를 다룰 줄 아는 '사람'이 필요하다.
2. 올바른 데이터 시스템 구축이 필요하다. 인프라가 먼저다.
3. 데이터를 통해 측정하고 싶은 게 무엇인지 설정하고 어떤 통찰력을 얻을지 연구해야 한다.
4. 데이터를 믿는 문화가 바탕에 깔려 있어야 한다.

데이터 분석과 활용은 이처럼 야구에서는 이제 선택이 아닌 필수가 됐다. 데이터를 활용할지 말지는 더 이상 선택의 문제가 아니다. 오늘날 야구에서 데이터는 전통적인 훈련, 스카우트, 전력 분석처럼 야구의 일부로 자연스럽게 자리 잡았다. 대량의 데이터를 처리하고, 통계 모델을 만들고, 이를 기반으로 미래를 위해 선수를 스카우트하고, 현장과 프런트의 의사결정에 활용하는 것은 이제 모든 프로구단이 당연히 해야 할 일상적이고도 주요한 업무다. 팀 내부에 쌓여 있는 데이터를 어떻게 분석해 어떻게 활용하느냐가 중요해졌기 때문에 삼성 라이온즈 내부에서 전력분석원을 하며 모든 데이터를 활용하고 분석했던 허삼영이 감독이 되는 것은 사실 자연스러운 흐름이었다. 삼성 라이온즈의 결정에 대해서 필자가 고개를 끄덕였던 이유다.

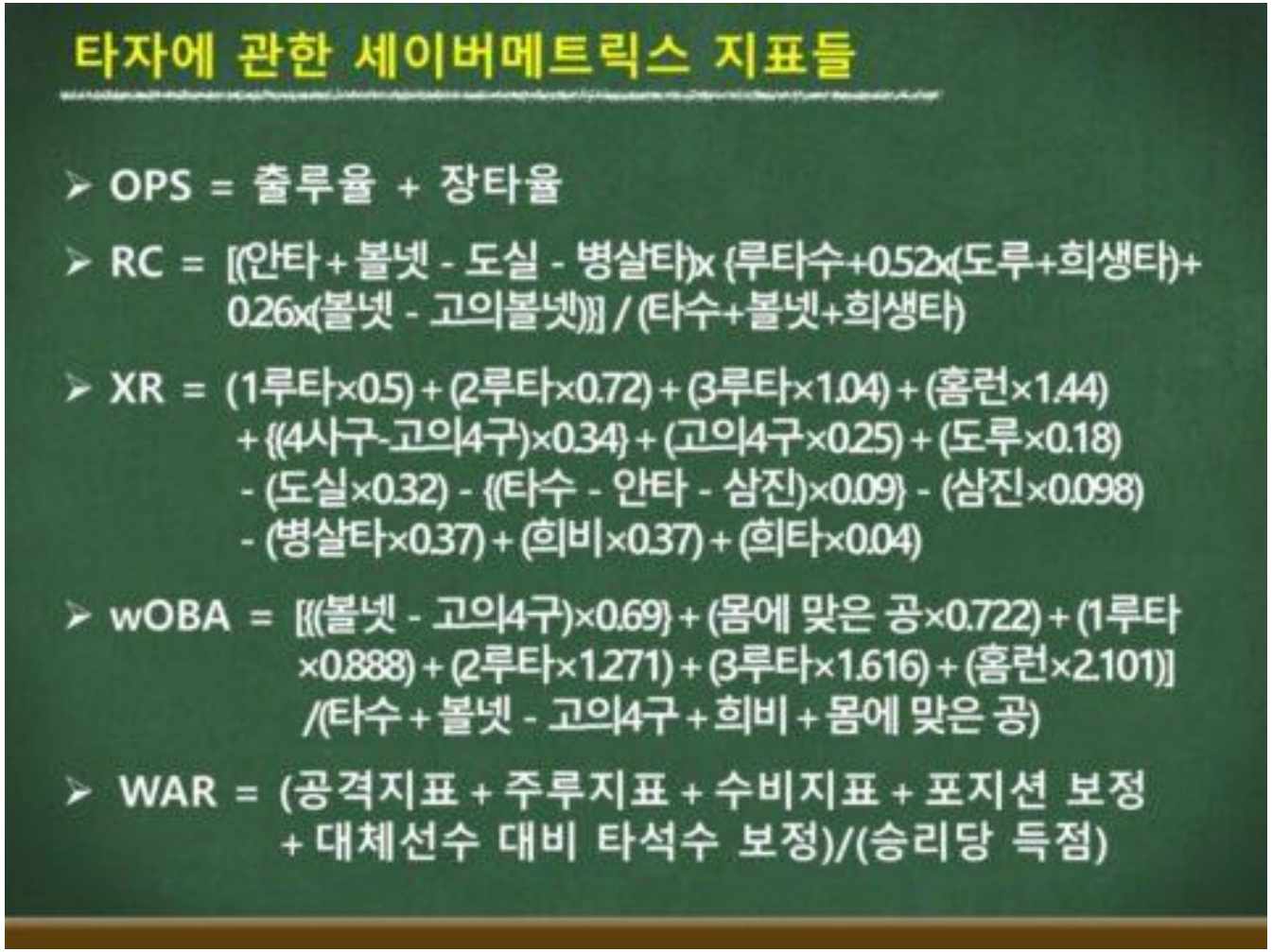
신문기사

서두에서부터 언급했듯 한국프로야구에서도 비슷한 현상이 벌어지고 있다. 올 시즌 순위표를 보면 데이터 분석의 중요성을 간파하고 일찌감치 팀을 꾸린 구단들이 상위권을 차지했지만 데이터 파트가 아예 없거나 역량이 떨어지는 팀들은 하위권으로 추락한 것을 확인할 수 있다. 뛰어난 데이터 분석 역량을 보유한 구단은 SK 와이번스다. 선수 영입부터 훈련까지 구단 운영의 모든 분야에 데이터를 활용하며, 분석 파트의 능력도 뛰어나다. 키움 히어로즈 역시 김치현 단장이 국제전략팀장 시절부터 기반을 닦아 놓은 분석 역량을 토대로 지금의 성과를 만들고 있다. 팀 창단 때부터 데이터 중심 조직을 만드는 데 공을 들였던 NC 다이노스는 '비선(수)출(신)' 데이터 팀장에게 스카우트 역할까지 맡기는 파격을 시도했다.

아직 데이터 분석 파트가 없는 팀들의 움직임도 눈에 띈다. 올 정규 시즌이 끝난 스토브리그를 이끌고 있는 롯데 자이언츠는 신임 단장으로 국내 프로야구 경험이 없는 시카고 컵스 스카우터 출신 성민규를 임명했다. 그는 데이터 이해에 뛰어난 감독과 외국인 선수 스카우트, 2군 육성에 주력하고 장기적 계획을 세우고 팀을 만들겠다고 선언했다. 성 단장 말대로 되기 위해서는 우선 데이터에 대한 분석 능력과 롯데만의 팀 철학이 있어야 한다. 또한 단장에 뒤이어 임명돼야 할 감독의 캐릭터는 분명하다. 스타플레이어 출신이거나 우승 경험 조건이 중요한 것이 아닌 코치들과 화합하고, 데이터 분석에 대한 이해와 실행 능력이 있는 운영 책임자여야 한다. 코치 경험이 필수적인 것이 아니고, 이름값이 필요한 것도 아니다. 그가 확보한 인력과 데이터 안에서 능력을 최대한 끌어낼 사람이면 되는 것이다. 더군다나 성 단장이 데이터 전문가이니 함께 데이터에 대한 이해를 같이할 사람이면 된다. 성민규 단장을 선임한 롯데 자이언츠의 변화를 주목할 필요가 있다. 성 단장은 취임 기자간담회 당시 "한국에서 흔히 데이터 팀이라 부르는 R&D(Research and Development) 팀의 역량이 중요하다"며 구단의 데이터 분석 역량을 대폭 강화하겠다는 뜻을 밝혔다. 이

출처: 장원석 책임연구원, '예전의 '명감독'은 잊어라, 데이터가 우승을 이끈다', DBR donga기술, 2019년 12월 286호

세이버메트릭스



기록의 스포츠

[세이버 메트릭스]

세이버메트릭스는 야구를 객관적으로 이해하려는 시도"라고 설명한다.

선수와 팀의 능력을 객관화하기 위한 다양한 데이터 분석적 시도가 이뤄지고 있음

출처: 김홍재 칼럼니스트, '야구는 숫자놀음? "승부를 지배하는 숫자들"", 과학기술, 2021년 5월 25일자.

프로야구 FA 시장

고비용 선수 속출하는 프로야구 FA 시장

원소속(포지션)	선수명(출생연도)	시즌 성적	계약 현황
KT(포수)	장성우(1990)	127경기 0.231 14홈런 OPS 0.711	잔류·4년 총액 42억원
LG(외야수)	김현수(1988)	140경기 0.285 17홈런 96타점(8위)	잔류·4+2년 총액 115억원
두산(외야수)	김재환(1988)	137경기 0.274 27홈런(8위) 102타점(2위)	잔류·4년 총액 115억원
두산(외야수)	박건우(1990)	126경기 0.325(5위) 149안타 63타점	NC 이적·6년 총액 100억원
삼성(투수)	백정현(1987)	27경기 14승(4위) ERA 2.63(2위)	잔류·4년 총액 38억원
삼성(외야수)	박해민(1990)	127경기 0.291 132안타 36도루(3위)	LG 이적·4년 총액 60억원
한화(포수)	최재훈(1989)	116경기 0.275 44타점 WAR 3.87	잔류·5년 총액 54억원
NC(외야수)	나성범(1989)	144경기 0.281 33홈런 101타점	기아 이적·6년 총액 150억원

- 선수의 시즌 성적(능력)에 따라 막대한 규모의 자금이 오가는 KBO FA 시장

참고논문

논문 17-09-01-03 한국IT정책경영학회 논문지 '17-02 Vol.9 No.01

기계학습을 활용한 프로야구 승부예측에 관한 연구

노언석*, 최재현**

A Study on the Prediction of Professional Baseball Game Result Using Machine Learning

Eon-Seok Roh*, Jae-Hyun Choi**

요 약

본 논문에서는 KBO(Korea Baseball Organization) 프로야구경기의 승패 예측을 위해 선수들이 기록한 날짜별 데이터를 기반으로 인공신경망을 이용하여 경기를 예측하는 모델을 제시한다. 야구 경기 9이닝 중 선발 투수가 차지하는 비중이 높음을 고려하여 선발투수의 세부 기록과 나머지 투수들의 기록을 분리하여 적용하여 2014년부터 2016년 까지의 데이터를 활용하여 실험 하였다.

한국빅데이터학회지
제5권 제2호, 2020, pp. 77-84 https://doi.org/10.36498/kbigdt.2020.5.2.77

인공지능 모델에 따른 한국 프로야구의 승패 예측 분석에 관한 연구

A Study on the Win-Loss Prediction Analysis of Korean Professional Baseball by Artificial Intelligence Model

김태훈[1] · 임성원[1] · 고진광[1†] · 이재학[2]
순천대학교 컴퓨터공학과[1], 송원대학교 전기전자공학과[2]

요 약

본 연구에서는 인공지능 모델에 따른 한국 프로야구의 승패 예측 분석에 관한 연구를 했다. 승리할 팀과 해당 팀의 최종 리그 순위를 예측했고, 사용자의 편의를 위해 웹사이트도 구축했다. 각 1 · 3 · 5이닝 별로 가장 정확도가 높으면서도 오차가 적은 모델을 최적 모델로 선정해 승 · 패 결과를 예측했고, 이를 토대로 순위표를 작성했다. 결과표는 2020년 개막인 5월 5일부터 8월 30일까지의 예측 결과를 바탕으로 작성했다. 기아타이거즈가 아닌 다른 구단끼리의 경기는 실제 결과를 사용했다. 머신러닝 모델은 KNN과 AdaBoost가 최적 모델로 선정되었으며, 실제 순위와 비교해 본 결과, 경기가 진행될수록, 예측 결과의 순위 오차가 점점 작아지는 것을 확인했다. 딥러닝 모델은 89%의 정확도를 기록했고, 머신러닝 모델과 마찬가지로 경기를 진행할수록 예측 결과 순위 오차가 작아지는 것을 확인했다. 실험 결과는 한국 프로야구 승 · 패 결과 예측뿐 아니라 다양한 분야에서 사용할 수 있을 것으로 사료된다. 방송국에서 야구 경기를 중계하는 중 이닝별로 인공지능 알고리즘이 예상한 승 · 패 여부를 중계화면에 띄울 수 있다. 시청자들에게 새로운 흥미를 일으킬 수 있을 것이고, 나아가 구단의 감독들이 이닝마다 데이터를 분석해 경기 중 유동적으로 승리하기 위한 전략을 세울 수 있을 것으로 기대된다.

■ 중심어 : 머신러닝 모델(KNN과 AdaBoost), 딥러닝 모델, 프로야구, 승패 예측

데이터 수집 방법

1 STATIZ

2 KBO 공식사이트

3

변수 설정 및 데이터 수집

구단별 No.

```
# 구단별 No

# 1. kt wiz
# 2. 두산 베어스
# 3. 삼성 라이온즈
# 4. LG 트윈스
# 5. 키움히어로즈
# 6. SSG 랜더스
# 7. NC 다이노스
# 8. 롯데 자이언츠
# 9. 기아 타이거즈
# 10. 한화 이글스
```

- 구단별 No.
  - 2021년 순위기준으로 Numbering

- 구단별 특이사항
  - NC 다이노스 : 2011년 창단/ 2013년부터 경기 참여.
  - kt wiz : 2013년 창단/ 2015년부터 경기 참여.
  - 키움 히어로즈 : 2019년부터 변경됨 (전: 넥센 히어로즈)
  - SSG 랜더스: 2022년부터 변경됨 (전: SK 와이번스)

변수 설명

ITEM			Data Type	Comment
구단		team	int(1~10)	
년도		year	int(2015~2022)	
월		month	int(3~10)	
승률		pov	float(0~1)	- 승수/(승수+패수)
타자	출루율 on-base percentage	obp	float(0~1)	- 타석에 나왔을 때 아웃을 당하지 않고 주자로 살아남는 확률 - (안타+볼넷+몸에 맞은 공) / (타수+볼넷+몸에 맞은 공+희생플라이) - 베이스에서 많이 살아남는다는 증거가 되기 때문에 중요한 요인으로 선정
	장타율 Slugging Percentage	slg	float(0~1)	- '총 누타수/타수.' - 좀 더 정확하게는'(단타 수[1] + 2루타 수*2 + 3루타 수*3 + 홈런 수*4)/타수.'
	타율 Batting Average	ba	float	- 안타/타수 - 득점과 직결되는 변수이므로 가장 중요한 요인이기에 선정
투수	평균자책점 ERA	era	float	- 야구에서 투수가 한 게임(9이닝) 당 내준 평균 자책점 - 타자의 타율과 마찬가지로 투수의 역량을 재는 가장 유명하고 고전적이어서 상징성이 있는 비율 스탯으로, 숫자가 낮을수록 좋은 것 - 투수로서의 능력을 직접적으로 보여주는 지표(실점과도 연관)로 이해할 수 있음
	WHIP	whip	float	- (피안타 + 볼넷) / 이닝 (고의사구는 포함시키며 몸에 맞는 볼은 포함하지 않음) - 투수의 안정감을 측정하는 지표 (낮을 수록 좋음) - 볼넷 : 타자가 타석에서 4개의 볼 카운트를 얻어내 1루로 나가는 것 이닝 : 야구 또는 소프트볼에서 양팀이 한 번씩의 공격을 주고 받는 단위.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 534 entries, 0 to 533
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   team    534 non-null    int64
 1   year    534 non-null    int64
 2   month   534 non-null    int64
 3   pov     534 non-null    float64
 4   obp     534 non-null    float64
 5   slg     534 non-null    float64
 6   ba      534 non-null    float64
 7   era     534 non-null    float64
 8   whip    534 non-null    float64
dtypes: float64(6), int64(3)
memory usage: 37.7 KB
```

Grade 기준

grade 1~4 기준

```
# grade 1~4 기준 (Grade 4 = 0.60 이상 , Grade 3 = 0.50 이상 0.60 미만, Grade 2 = 0.40 이상 0.50 미만, Grade 1 = 0.40 미만)
bb['grade'] = bb['pov'].apply(lambda pov: get_grade_1(pov))
bb
```

✓ 0.6s

	team	year	month	pov	obp	slg	ba	era	whip	grade
0	1	2015	3	0.000	0.403	0.412	0.284	8.28	1.92	1
1	1	2015	4	0.136	0.290	0.295	0.208	5.52	1.79	1
2	1	2015	5	0.259	0.342	0.356	0.263	5.72	1.58	1
3	1	2015	6	0.478	0.353	0.456	0.290	5.93	1.68	2
4	1	2015	7	0.444	0.339	0.431	0.274	5.73	1.60	2
...	...	...	...	...	...	...	...	...	...	...
529	10	2021	8	0.500	0.331	0.331	0.231	4.33	1.46	3
530	10	2021	9	0.381	0.361	0.373	0.263	4.92	1.38	1
531	10	2021	10	0.278	0.318	0.310	0.219	4.18	1.47	1
532	10	2022	4	0.360	0.309	0.309	0.225	4.00	1.40	1
533	10	2022	5	0.385	0.318	0.387	0.251	6.51	1.67	1

534 rows × 10 columns

- 승률별로 등급을 나누어 높은 grade를 가질수록 우승할 수 있는 확률이 높은팀으로 선정
- 등급으로 라벨을 정한 이유 : 이전 데이터를 고려해봤을 때 반드시 승률이 높다고 해서 우승팀이 된 것이 아니기 때문에 승률로 예측을 하는 것 보다는 승률의 범위를 설정하여 우승팀의 예측 확률을 좀 더 높이는 방향을 고민해서 해당 라벨로 결정

Correlation (상관관계)

```
corr = bb.corr()
print(corr)
```

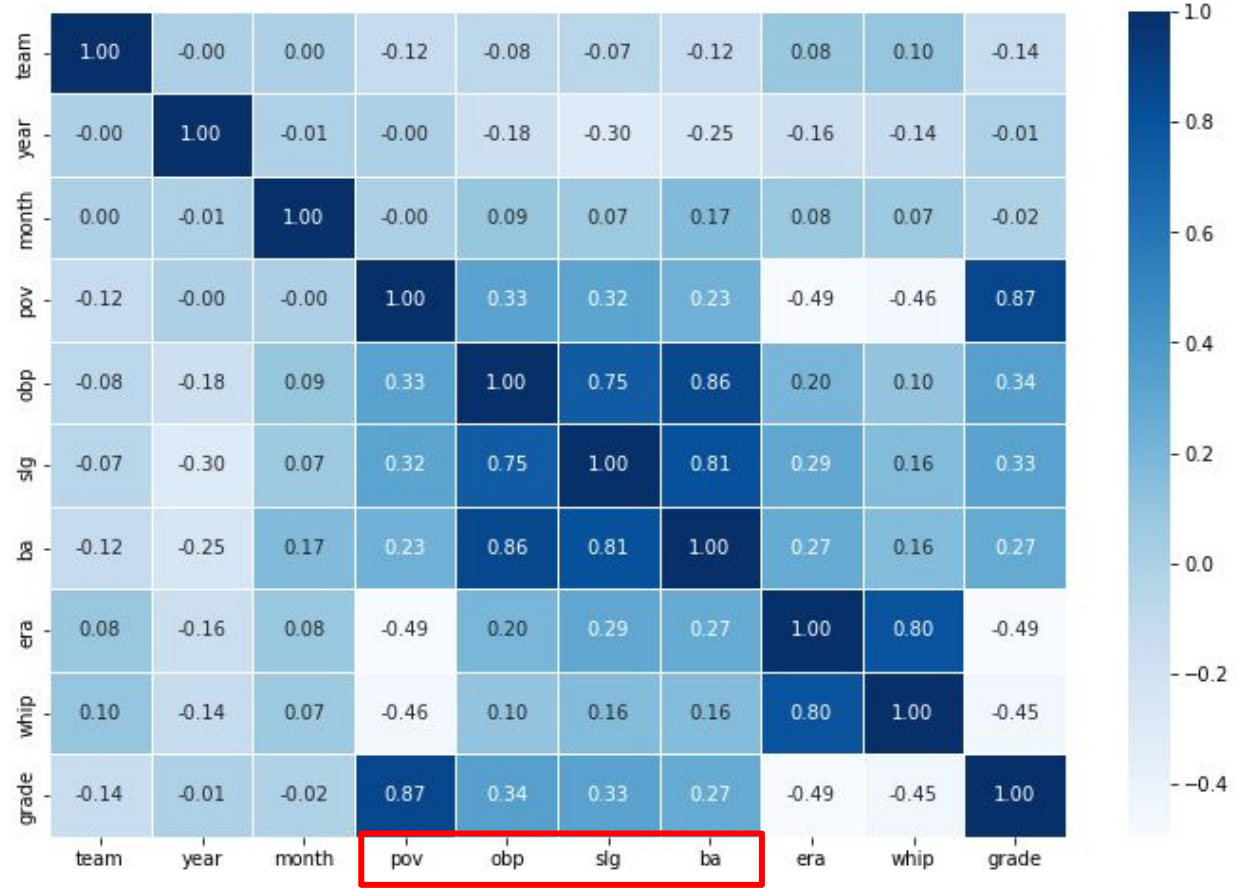
```
              team          year         month       pov       obp       slg  #
team   1.000000e+00 -1.216931e-13  1.684128e-17 -0.115621 -0.078452 -0.068407
year  -1.216931e-13  1.000000e+00 -1.199418e-02 -0.001751 -0.179386 -0.299121
month  1.684128e-17 -1.199418e-02  1.000000e+00 -0.004201  0.094359  0.069708
pov   -1.156206e-01 -1.750906e-03 -4.201248e-03  1.000000  0.332913  0.316072
obp   -7.845232e-02 -1.793857e-01  9.435866e-02  0.332913  1.000000  0.747138
slg   -6.840687e-02 -2.991214e-01  6.970795e-02  0.316072  0.747138  1.000000
ba    -1.170678e-01 -2.545450e-01  1.695144e-01  0.233210  0.862917  0.812916
era    8.328565e-02 -1.628113e-01  7.797397e-02 -0.493769  0.195104  0.294633
whip   9.793962e-02 -1.351297e-01  6.655080e-02 -0.455063  0.102360  0.160535
grade -1.434019e-01 -1.462872e-02 -1.736043e-02  0.870339  0.342944  0.331619

              ba       era      whip     grade
team   -0.117068  0.083286  0.097940 -0.143402
year   -0.254545 -0.162811 -0.135130 -0.014629
month   0.169514  0.077974  0.066551 -0.017360
pov     0.233210 -0.493769 -0.455063  0.870339
obp     0.862917  0.195104  0.102360  0.342944
slg     0.812916  0.294633  0.160535  0.331619
ba      1.000000  0.274560  0.164612  0.273100
era     0.274560  1.000000  0.798332 -0.494847
whip    0.164612  0.798332  1.000000 -0.449930
grade   0.273100 -0.494847 -0.449930  1.000000
```

- 컬럼간의 상관관계를 통해 서로 얼마나 연관도가 있는지 파악하기 위해서 해당 조건을 확인함

Heatmap (히트맵)

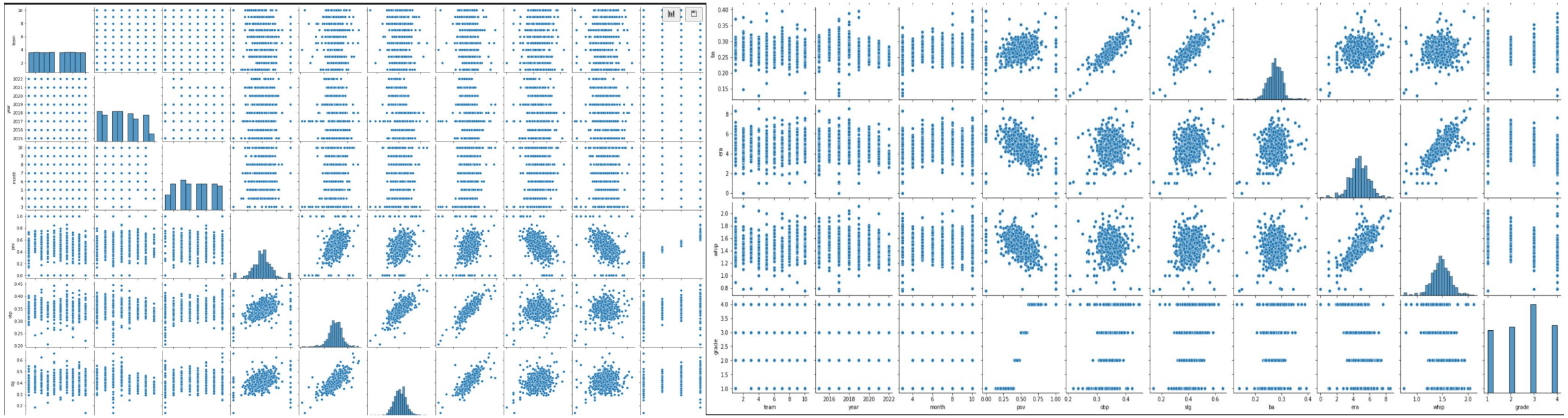
```
plt.figure(figsize=(12,8))
sns.heatmap(data = bb.corr(), annot=True, fmt = '.2f', linewidths =.5, cmap='Blues');
```



- grade와 양의 상관관계를 가지는 항목은 **pov(승률), obp(출루율), slg(장타율), ba(타율)**로 나타남.

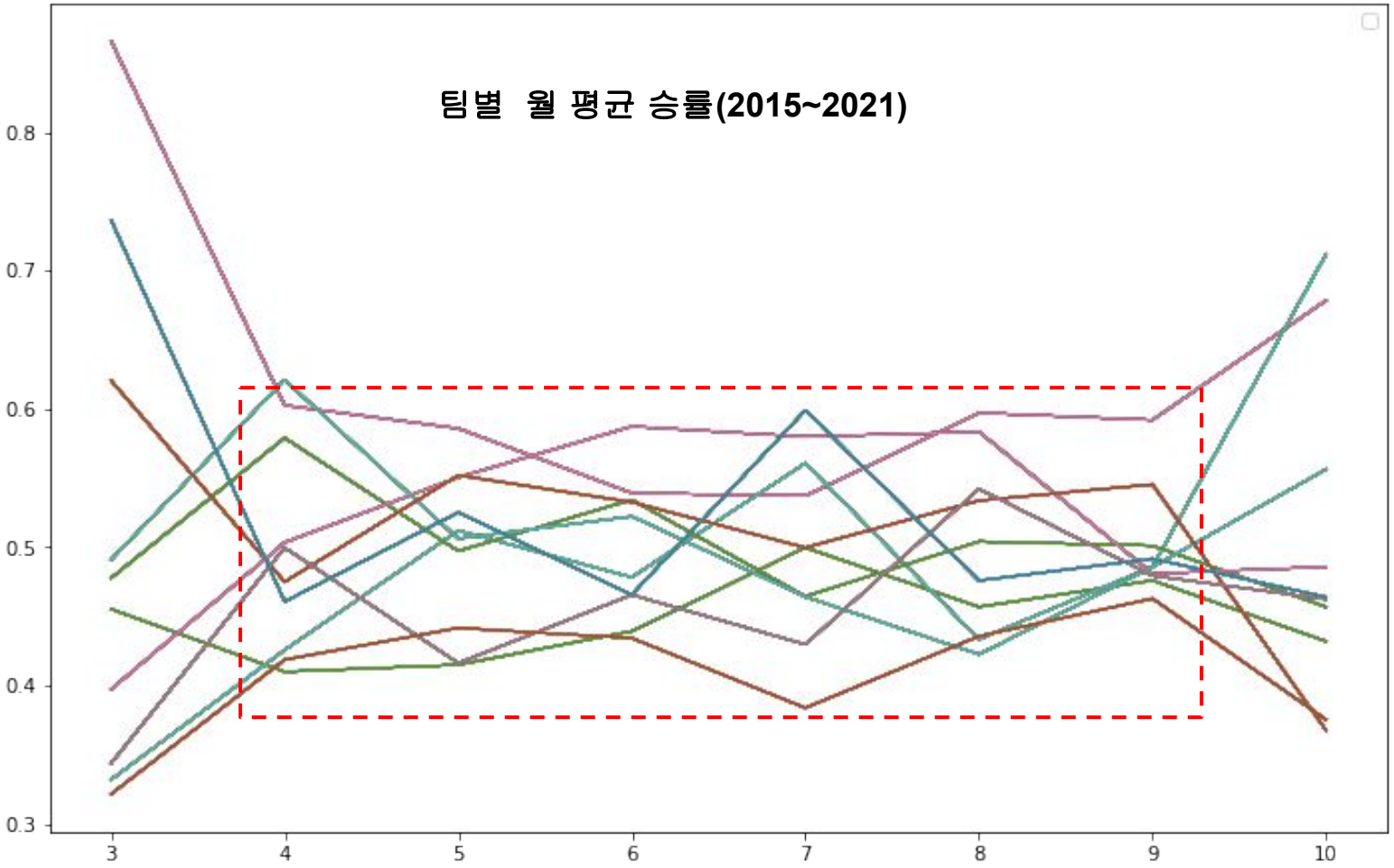
Pairplot (관계 그래프)

```
plt.figure(figsize=(12,6))
sns.pairplot(data=bb);
```



- 특정 컬럼의 경우 (ex) team, year, month)의 경우에는 연관도가 다른 컬럼에 비해 상대적으로 적은편이다.

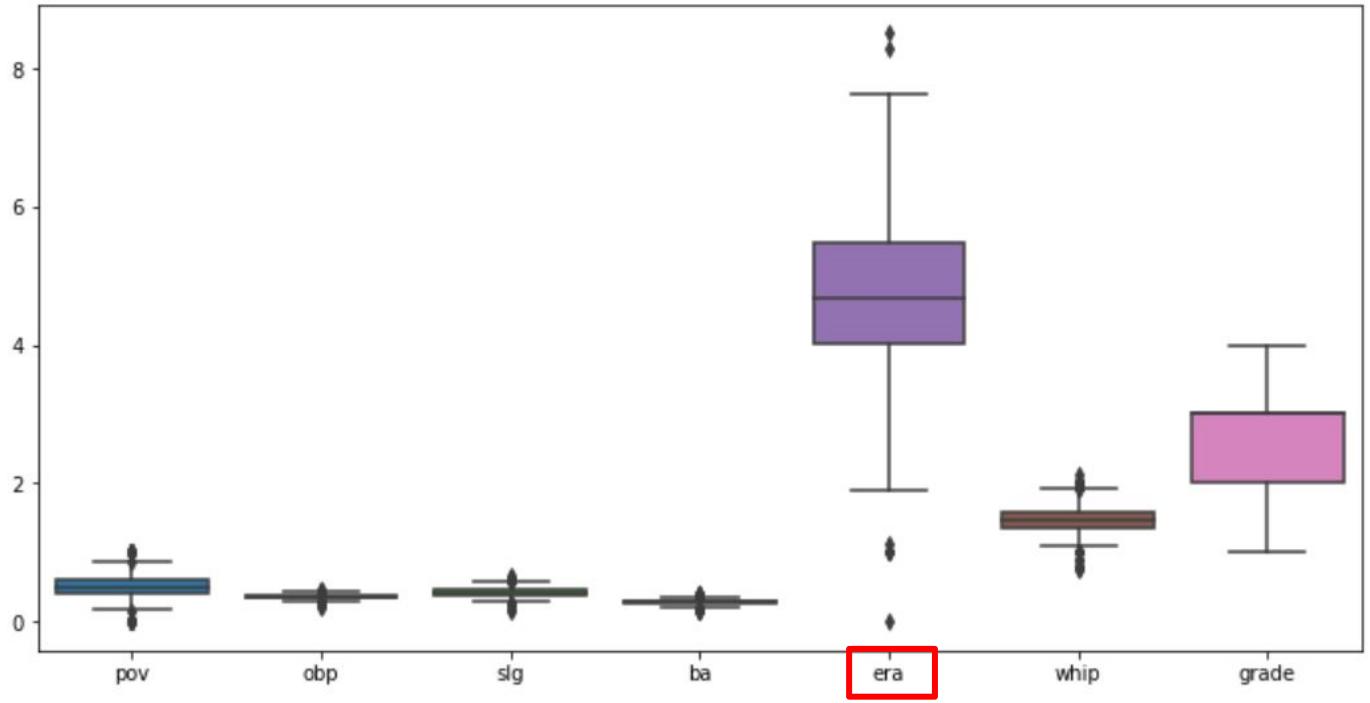
팀별 월 평균 승률 비교



- 각 팀의 승률은 월에 상관없이 박스권 내
- month는 승률에 영향을 미치지 않는 것으로 보임

Boxplot

```
plt.figure(figsize=(12,6))
sns.boxplot(data=bb.iloc[:, 3:]);
```

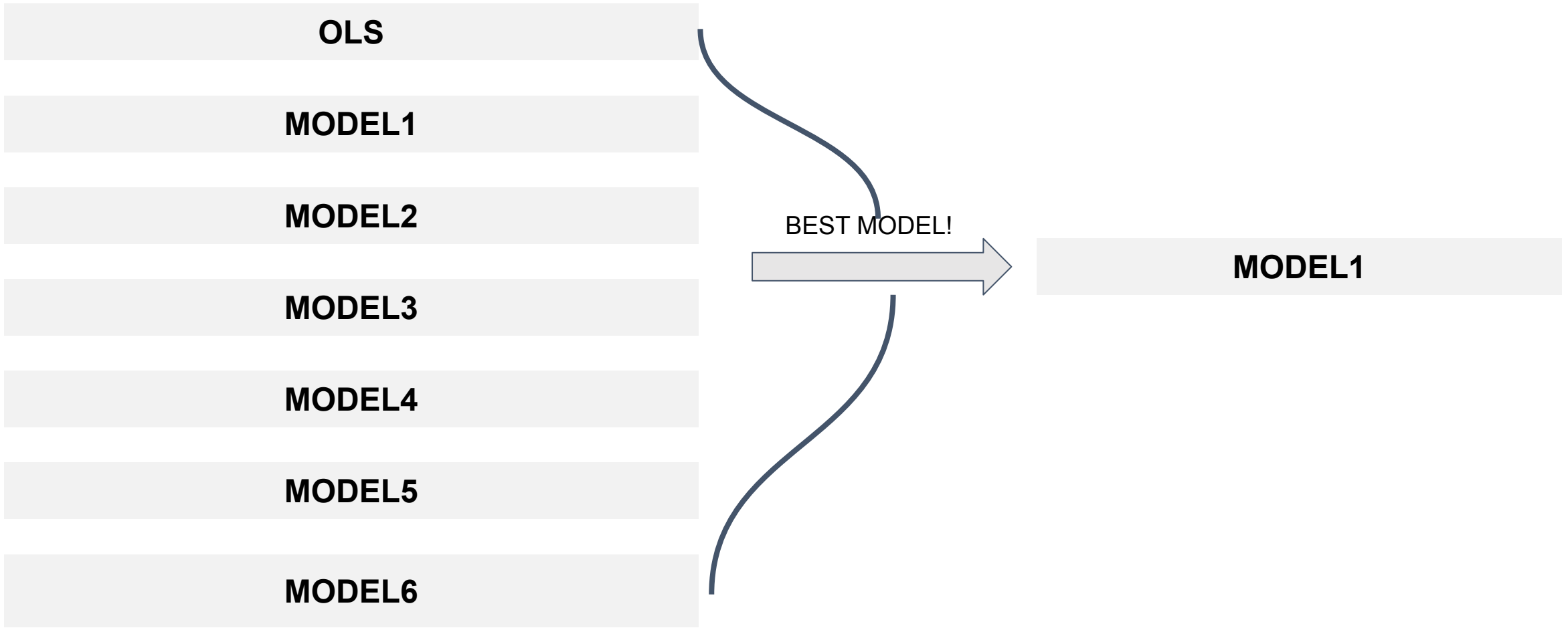


- era (평균자책점)의 경우 값의 범위가 다른 값에 비해 차이가 큰 것을 확인할 수 있음

4

모델링

MODEL 선정 과정



최종 MODEL 선정

MODEL 1	grade 1~4 기준 (Grade 4 = 0.60 이상 , Grade 3 = 0.50 이상 0.60 미만, Grade 2 = 0.40 이상 0.50 미만, Grade 1 = 0.40 미만)
MODEL 2	grade 0~1 기준 (Grade 1 = 0.51 이상, Grade 0 = 0.51 미만)
MODEL 3	grade 1~6 기준 (Grade 6 = 0.60 이상 , Grade 5 = 0.55 이상 0.60 미만, Grade 4 = 0.50 이상 0.55 미만, Grade 3= 0.45 이상 0.50 미만, Grade 2= 0.40 이상 0.45 미만, Grade 1 = 0.40 미만)
MODEL 4	grade 1~4 기준 (Grade 4 = 0.56이상, Grade 3 = 0.49이상 0.56미만, Grade 2 = 0.40이상 0.49미만 Grade 1 = 0.4미만)
MODEL 5	grade 1~4 기준 (Grade 4 = 0.65 이상 , Grade 3 = 0.55 이상 0.65 미만, Grade 2 = 0.45 이상 0.55 미만, Grade 1 = 0.45 미만)
MODEL 6	grade 1~3 기준 (Grade 3 = 0.56이상, Grade 3 = 0.49이상 0.56미만, Grade 1 = 0.49미만)

Y값(Target값 정의) - GRADE

1. 승률 - POV 에 대한 고민
2. GRADE 설정
3. 모델별 GRADE 조정 및 최적 구간값 설정

Decision Tree - Test Size 비교

Test Size 1. 0.2

```
X=bb[['obp', 'slg', 'ba', 'era','whip']]
y=bb['grade']

X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.2, random_state=13, stratify=y)
```

✓ 0.4s

```
cc = DecisionTreeClassifier(max_depth=7, random_state=13)
cc.fit(X_train, y_train)

y_pred_tr = cc.predict(X_train)
y_pred_test = cc.predict(X_test)


print('Train Acc : ', accuracy_score(y_train, y_pred_tr))
print('Test Acc : ', accuracy_score(y_test, y_pred_test))
```

✓ 0.6s

```
Train Acc :  0.7892271662763466
Test Acc :  0.4672897196261682
```

Test Size 2. 0.3

```
X=bb[['obp', 'slg', 'ba', 'era','whip']]
y=bb['grade']

X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.3, random_state=13, stratify=y)
```

✓ 0.6s

```
cc = DecisionTreeClassifier(max_depth=7, random_state=13)
cc.fit(X_train, y_train)

y_pred_tr = cc.predict(X_train)
y_pred_test = cc.predict(X_test)


print('Train Acc : ', accuracy_score(y_train, y_pred_tr))
print('Test Acc : ', accuracy_score(y_test, y_pred_test))
```

✓ 0.6s

```
Train Acc :  0.7989276139410187
Test Acc :  0.546583850931677
```

Test size 3. 비교

- 0.2 (max_depth=7)
  - Train Acc : 0.7892271662763466
  - Test Acc : 0.4672897196261682
- 0.3 (max_depth=7)
  - Train Acc : 0.7989276139410187
  - Test Acc : 0.546583850931677

Decision Tree - Best Max_depth

In [23]:
```
from sklearn.model_selection import GridSearchCV
import pprint

params={'max_depth' : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]}
baseball=DecisionTreeClassifier(random_state=13)

gridsearch=GridSearchCV(estimator=baseball, param_grid=params, cv=5)
gridsearch.fit(X, y)
```

Out[23]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=13),
             param_grid={'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                       13, 14, 15]})
```

In [24]:
```
pp=pprint.PrettyPrinter(indent=4)
pp.pprint(gridsearch.cv_results_)
```

In [25]:
```
gridsearch.best_estimator_
```

Out[25]:
```
DecisionTreeClassifier(max_depth=7, random_state=13)
```

Decision Tree - Confusion Matrix

In [29]:
```
from sklearn.metrics import confusion_matrix

def print_clf_eval_m(y_test, pred):
    confusion = confusion_matrix(y_test,pred)
    acc, pre, re, f1, auc = get_clf_eval_m(y_test, pred)

    print('=> confusion matrix')
    print(confusion)
    print('=========')

    print('Accuracy: {0:.4f}, Precision: {1:.4f}'.format(acc,pre))
    print('Recall: {0:.4f},  F1_1: {1:.4f}, AUC:{2:.4f}'.format(re, f1, auc))
```

In [30]:
```
from sklearn.metrics import confusion_matrix

def print_clf_eval_w(y_test, pred):
    confusion = confusion_matrix(y_test,pred)
    acc, pre, re, f1, auc = get_clf_eval_w(y_test, pred)

    print('=> confusion matrix')
    print(confusion)
    print('=========')

    print('Accuracy: {0:.4f}, Precision: {1:.4f}'.format(acc,pre))
    print('Recall: {0:.4f},  F1_1: {1:.4f}, AUC:{2:.4f}'.format(re, f1, auc))
```

In [35]:
```
print('average=micro:'), print_clf_eval_m(y_test, y_pred_test),
print('========================================'),
print('average=weighted:'), print_clf_eval_w(y_test, y_pred_test)
```

```
average=micro:
=> confusion matrix
[[19 11  3  2]
 [12 16 10  0]
 [ 5  6 36  3]
 [ 1  4 16 17]]
=========
Accuracy: 0.5466, Precision: 0.5466
Recall: 0.5466,  F1_1: 0.5466, AUC:0.7123
========================================
average=weighted:
=> confusion matrix
[[19 11  3  2]
 [12 16 10  0]
 [ 5  6 36  3]
 [ 1  4 16 17]]
=========
Accuracy: 0.5466, Precision: 0.5681
Recall: 0.5466,  F1_1: 0.5436, AUC:0.7123
```

	1	2	3	4
1	19	11	3	2
2	12	16	10	0
3	5	6	36	3
4	1	4	16	17
Precision	0.514	0.432	0.554	0.773
Recall	0.543	0.421	0.72	0.447
F1	0.528	0.427	0.626	0.567

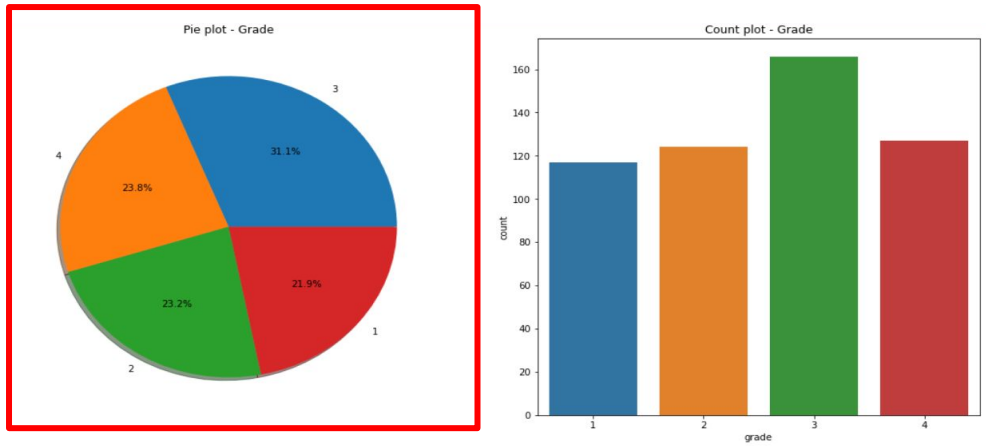
Decision Tree - Classification Report

In [36]:
```
# Grade로 얼마나 있는지 비율 확인

f, ax = plt.subplots(1, 2, figsize=(18, 8))

bb['grade'].value_counts().plot.pie(autopct='%1.1f%%', ax=ax[0], shadow=True)
ax[0].set_title('Pie plot - Grade')
ax[0].set_ylabel('')
sns.countplot('grade', data=bb, ax=ax[1])
ax[1].set_title('Count plot - Grade')

plt.show();
```



In [37]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_test))
```

	precision	recall	f1-score	support
1	0.51	0.54	0.53	35
2	0.43	0.42	0.43	38
3	0.55	0.72	0.63	50
4	0.77	0.45	0.57	38
accuracy			0.55	161
macro avg	0.57	0.53	0.54	161
weighted avg	0.57	0.55	0.54	161

비율 \ 등급	1	2	3	4
실제 data	21.9 %	23.2 %	31.1 %	23.8 %
test data	21.7 %	23.6 %	31.1 %	23.6 %

실제 값으로 Test 한 예측 값 및 예측 비율 (2022년 6월 13일 기준 data)

In [40]:
```
# grade 1~4 기준 (Grade 4 = 0.60 이상 , Grade 3 = 0.50 이상 0.60 미만, Grade 2 = 0.40 이상 0.50 미만, Grade 1 = 0.40 미만
ssg=np.array([[0.330,   0.370,  0.249,  3.56,   1.21]])
kium=np.array([[0.326,  0.365,  0.246,  3.49,   1.26]])
lg=np.array([[0.340,    0.392,  0.264,  3.78,   1.36]])
kia=np.array([[0.350,   0.409,  0.269,  4.03,   1.34]])
dusan=np.array([[0.331, 0.357,  0.256,  4.12,   1.42]])
samsung=np.array([[0.321,       0.365,  0.260,  3.86,   1.39]])
kt=np.array([[0.326,    0.352,  0.247,  3.58,   1.24]])
lotte=np.array([[0.313, 0.366,  0.254,  4.03,   1.36]])
nc=np.array([[0.320,    0.363,  0.248,  3.85,   1.36]])
hanhwa=np.array([[0.317,        0.355,  0.242,  5.30,   1.53]])
```

In [41]:
```
print('1. ssg : ', cc.predict(ssg)),
print('2. kium :', cc.predict(kium)),
print('3. lg : ', cc.predict(lg)),
print('4. kia : ', cc.predict(kia)),
print('5. dusan : ', cc.predict(dusan)),
print('6. samsung : ', cc.predict(samsung)),
print('7. kt :', cc.predict(kt)),
print('8. lotte: ', cc.predict(lotte)),
print('9. nc :', cc.predict(nc)),
print('10. hanhwa :', cc.predict(hanhwa))
```

```
1. ssg :  [3]
2. kium : [3]
3. lg :  [3]
4. kia :  [3]
5. dusan :  [3]
6. samsung :  [3]
7. kt : [3]
8. lotte:  [2]
9. nc : [3]
10. hanhwa : [1]
```

In [43]:
```
print('1. ssg : ', cc.predict_proba(ssg)),
print('2. kium :', cc.predict_proba(kium)),
print('3. lg : ', cc.predict_proba(lg)),
print('4. kia : ', cc.predict_proba(kia)),
print('5. dusan : ', cc.predict_proba(dusan)),
print('6. samsung : ', cc.predict_proba(samsung)),
print('7. kt :', cc.predict_proba(kt)),
print('8. lotte: ', cc.predict_proba(lotte)),
print('9. nc :', cc.predict_proba(nc)),
print('10. hanhwa :', cc.predict_proba(hanhwa))
```

```
1. ssg :  [[0.08 0.28 0.6  0.04]]
2. kium : [[0.08 0.28 0.6  0.04]]
3. lg :  [[0.0625 0.     0.875  0.0625]]
4. kia :  [[0.         0.         0.61111111 0.38888889]]
5. dusan :  [[0.08 0.28 0.6  0.04]]
6. samsung :  [[0.08 0.28 0.6  0.04]]
7. kt : [[0.08 0.28 0.6  0.04]]
8. lotte:  [[0.25 0.75 0.   0.  ]]
9. nc : [[0.08 0.28 0.6  0.04]]
10. hanhwa : [[0.78947368 0.15789474 0.05263158 0.        ]]
```

DT 외 다른 Model 확인

In [49]:

```
import time

models = [lr_clf, dt_clf, rf_clf, lgbm_clf]
model_names=['LogisticReg', 'DecisionTree', 'RandomForest', 'LightGBM']

start_time=time.time()
results=get_result_pd_m(models, model_names, X_train, y_train, X_test, y_test)

print('Fit time : ', time.time() - start_time), print(results)
```

```
Fit time :  28.359723329544067
```

	accuracy	precision	recall	f1	roc_auc
LogisticReg	0.453416	0.453416	0.453416	0.453416	0.712322
DecisionTree	0.546584	0.546584	0.546584	0.546584	0.712322
RandomForest	0.552795	0.552795	0.552795	0.552795	0.712322
LightGBM	0.478261	0.478261	0.478261	0.478261	0.712322

5

결론 및
한계 보완 방법

결론

1. 8개 구단이 포스트 시즌 진출 가능할 것으로 예측됨.

2. 롯데, 한화는 포스트 시즌 진출이 어려울 것으로 예측됨.

3. 아직 시즌 중이므로 정확한 1위 예측은 힘들 것으로 보임.

4. 추가적으로 예측 비율로 우승을 예측해봤을때 유의미한 값을 찾기엔 부족함.

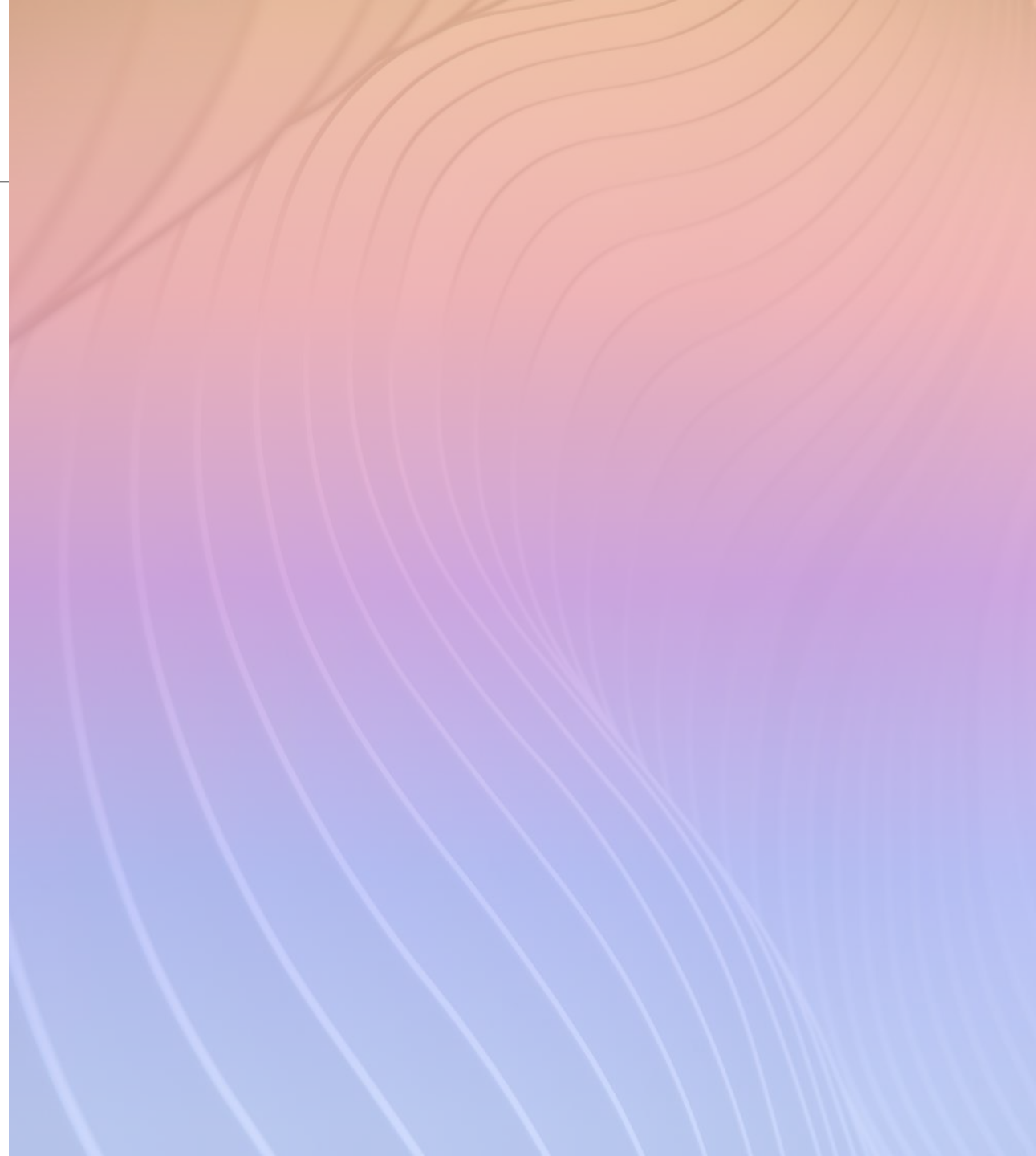
한계

1 데이터량 부족으로 프로젝트 주제인 우승팀 예측 목표에 도달하지 못함.

2 상대 승률, 포스트 시즌 대진표 등 중요 변수 고려하지않음.

3 실시간 데이터 누적을 통한 데이터 수집이 아닌 것이 예측을 어렵게 만듦.

4 이미 발생한 승차 고려하지 않음.



한계 보완 방안

1. 데이터가 부족하나 데이터가 쌓일 시 feature - 우승자로 분석 가능(연도 데이터 및, KBO 이외 리그 데이터)

2. 상대 승률 등은 우승 후보를 추린 뒤 적용 가능한 것으로 보임

3. 승차 등의 문제는 새로운 변수로 설정해서 해결할 수 있을듯 - 연구 필요

감사합니다!