BITamin 6기 복습 프로젝트 2조

FIFA MOBILE

길다영 서현재 이상연 정세영

Contents

I. 데이터 설명 및 선정배경

데이터 설명 및 목표 설정

1) 데이터 설명

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18207 entries, 0 to 18206
Data columns (total 88 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   ID                        18207 non-null  int64
 1   Name                      18207 non-null  object
 2   Age                       18207 non-null  int64
 3   Photo                     18207 non-null  object
 4   Nationality               18207 non-null  object
 5   Flag                      18207 non-null  object
 6   Overall                   18207 non-null  int64
 7   Potential                 18207 non-null  int64
 8   Club                      17966 non-null  object
 9   Club Logo                 18207 non-null  object
 10  Value                     18207 non-null  object
 11  Wage                      18207 non-null  object
 12  Special                   18207 non-null  int64
 13  Preferred Foot            18159 non-null  object
 14  International Reputation  18159 non-null  float64
 15  Weak Foot                 18159 non-null  float64
 16  Skill Moves               18159 non-null  float64
 17  Work Rate                 18159 non-null  object
 18  Body Type                 18159 non-null  object
 19  Real Face                 18159 non-null  object
 20  Position                  18147 non-null  object
 21  Jersey Number             18147 non-null  float64
 22  Joined                    16654 non-null  object
 23  Loaned From               1264 non-null   object
 24  Contract Valid Until      17918 non-null  object
 25  Height                    18159 non-null  object
 26  Weight                    18159 non-null  object
```

<데이터 프레임 구조>

18207 행, 88열로 이루어져 있음

<데이터 정보> - from Kaggle

- FIFA19의 선수 카드 데이터
- 선수들의 이름, 나이, 사진, 소속클럽, 국적과 같은 개인정보부터 각 포지션에 사용될 경우의 능력치, 선수 개개인의 기본 스탯에 대한 정보를 담고 있음

URL: https://www.kaggle.com/karangadiya/fifa19

데이터 설명 및 목표 설정

1) 데이터 설명

선수 기본 데이터

선수 능력치 데이터

포지션 능력치 데이터

```
data.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 18207 entries, 0 to 18206
Data columns (total 88 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   ID                        18207 non-null  int64
 1   Name                      18207 non-null  object
 2   Age                       18207 non-null  int64
 3   Photo                     18207 non-null  object
 4   Nationality               18207 non-null  object
 5   Flag                      18207 non-null  object
 6   Overall                   18207 non-null  int64
 7   Potential                 18207 non-null  int64
 8   Club                      17966 non-null  object
 9   Club Logo                 18207 non-null  object
 10  Value                     18207 non-null  object
 11  Wage                      18207 non-null  object
 12  Special                   18207 non-null  int64
 13  Preferred Foot            18159 non-null  object
 14  International Reputation  18159 non-null  float64
 15  Weak Foot                 18159 non-null  float64
 16  Skill Moves               18159 non-null  float64
 17  Work Rate                 18159 non-null  object
 18  Body Type                 18159 non-null  object
 19  Real Face                 18159 non-null  object
 20  Position                  18147 non-null  object
 21  Jersey Number             18147 non-null  float64
 22  Joined                    16654 non-null  object
 23  Loaned From               1264 non-null   object
 24  Contract Valid Until      17918 non-null  object
 25  Height                    18159 non-null  object
 26  Weight                    18159 non-null  object
 27  LS                        16122 non-null  object
 28  ST                        16122 non-null  object
 29  RS                        16122 non-null  object
 30  LW                        16122 non-null  object
 31  LF                        16122 non-null  object
 32  CF                        16122 non-null  object
 33  RF                        16122 non-null  object
 34  RW                        16122 non-null  object
 35  LAM                       16122 non-null  object
 36  CAM                       16122 non-null  object
 37  RAM                       16122 non-null  object
 38  LM                        16122 non-null  object
 39  LCM                       16122 non-null  object
 40  CM                        16122 non-null  object
 41  RCM                       16122 non-null  object
 42  RM                        16122 non-null  object
 43  LWB                       16122 non-null  object
 44  LDM                       16122 non-null  object
 45  CDM                       16122 non-null  object
 46  RDM                       16122 non-null  object
 47  RWB                       16122 non-null  object
 48  LB                        16122 non-null  object
 49  LCB                       16122 non-null  object
 50  CB                        16122 non-null  object
 51  RCB                       16122 non-null  object
 52  RB                        16122 non-null  object
 53  Crossing                  18159 non-null  float64
 54  Finishing                 18159 non-null  float64
 55  HeadingAccuracy           18159 non-null  float64
 56  ShortPassing              18159 non-null  float64
 57  Volleys                   18159 non-null  float64
 58  Dribbling                 18159 non-null  float64
 59  Curve                     18159 non-null  float64
 60  FKAccuracy                18159 non-null  float64
 61  LongPassing               18159 non-null  float64
 62  BallControl               18159 non-null  float64
 63  Acceleration              18159 non-null  float64
 64  SprintSpeed               18159 non-null  float64
 65  Agility                   18159 non-null  float64
 66  Reactions                 18159 non-null  float64
 67  Balance                   18159 non-null  float64
 68  ShotPower                 18159 non-null  float64
 69  Jumping                   18159 non-null  float64
 70  Stamina                   18159 non-null  float64
 71  Strength                  18159 non-null  float64
 72  LongShots                 18159 non-null  float64
 73  Aggression                18159 non-null  float64
 74  Interceptions             18159 non-null  float64
 75  Positioning               18159 non-null  float64
 76  Vision                    18159 non-null  float64
 77  Penalties                 18159 non-null  float64
 78  Composure                 18159 non-null  float64
 79  Marking                   18159 non-null  float64
 80  StandingTackle            18159 non-null  float64
 81  SlidingTackle             18159 non-null  float64
 82  GKDiving                  18159 non-null  float64
 83  GKHandling                18159 non-null  float64
 84  GKKicking                 18159 non-null  float64
 85  GKPositioning             18159 non-null  float64
 86  GKReflexes                18159 non-null  float64
 87  Release Clause            16643 non-null  object
dtypes: float64(38), int64(5), object(45)
memory usage: 12.4+ MB
```

데이터 설명 및 목표 설정

2) 목표 설정

어떤 선수가 전반적인 능력치가 가장 좋을까?	축구 선수의 평균적인 전성기는 언제일까? 축구 선수의 능력과 주급은 비례할까?
클럽별로 유의미한 특징은 무엇이 있을까?	클럽별로 선수 구성이 어떻게 되어 있을까? 클럽마다 특화된 포지션/능력치가 있을까?
포지션별로 유의미한 특징은 무엇이 있을까?	각 포지션에서 가장 중요한 능력치는 무엇일까? 포지션별로 주급에 유의미한 차이가 있을까?

II. 데이터 전처리

데이터 전처리

1) 열 삭제

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18207 entries, 0 to 18206
Data columns (total 88 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   ID                        18207 non-null  int64
 1   Name                      18207 non-null  object
 2   Age                       18207 non-null  int64
 3   Photo                     18207 non-null  object
 4   Nationality               18207 non-null  object
 5   Flag                      18207 non-null  object
 6   Overall                   18207 non-null  int64
 7   Potential                 18207 non-null  int64
 8   Club                      17966 non-null  object
 9   Club Logo                 18207 non-null  object
 10  Value                     18207 non-null  object
 11  Wage                      18207 non-null  object
 12  Special                   18207 non-null  int64
 13  Preferred Foot            18159 non-null  object
 14  International Reputation  18159 non-null  float64
 15  Weak Foot                 18159 non-null  float64
 16  Skill Moves               18159 non-null  float64
 17  Work Rate                 18159 non-null  object
 18  Body Type                 18159 non-null  object
 19  Real Face                 18159 non-null  object
 20  Position                  18147 non-null  object
 21  Jersey Number             18147 non-null  float64
 22  Joined                    16654 non-null  object
 23  Loaned From               1264 non-null   object
 24  Contract Valid Until      17918 non-null  object
 25  Height                    18159 non-null  object
 26  Weight                    18159 non-null  object
 27  LS                        16122 non-null  object
 28  ST                        16122 non-null  object
 29  RS                        16122 non-null  object
 30  LW                        16122 non-null  object
 31  LF                        16122 non-null  object
 32  CF                        16122 non-null  object
 33  RF                        16122 non-null  object
 34  RW                        16122 non-null  object
 35  LAM                       16122 non-null  object
 36  CAM                       16122 non-null  object
 37  RAM                       16122 non-null  object
 38  LM                        16122 non-null  object
 39  LCM                       16122 non-null  object
 40  CM                        16122 non-null  object
 41  RCM                       16122 non-null  object
 42  RM                        16122 non-null  object
 43  LWB                       16122 non-null  object
 44  LDM                       16122 non-null  object
 45  CDM                       16122 non-null  object
 46  RDM                       16122 non-null  object
 47  RWB                       16122 non-null  object
 48  LB                        16122 non-null  object
 49  LCB                       16122 non-null  object
 50  CB                        16122 non-null  object
 51  RCB                       16122 non-null  object
 52  RB                        16122 non-null  object
 53  Crossing                  18159 non-null  float64
 54  Finishing                 18159 non-null  float64
 55  HeadingAccuracy           18159 non-null  float64
 56  ShortPassing              18159 non-null  float64
 57  Volleys                   18159 non-null  float64
 58  Dribbling                 18159 non-null  float64
 59  Curve                     18159 non-null  float64
 60  FKAccuracy                18159 non-null  float64
 61  LongPassing               18159 non-null  float64
 62  BallControl               18159 non-null  float64
 63  Acceleration              18159 non-null  float64
 64  SprintSpeed               18159 non-null  float64
 65  Agility                   18159 non-null  float64
 66  Reactions                 18159 non-null  float64
 67  Balance                   18159 non-null  float64
 68  ShotPower                 18159 non-null  float64
 69  Jumping                   18159 non-null  float64
 70  Stamina                   18159 non-null  float64
 71  Strength                  18159 non-null  float64
 72  LongShots                 18159 non-null  float64
 73  Aggression                18159 non-null  float64
 74  Interceptions             18159 non-null  float64
 75  Positioning               18159 non-null  float64
 76  Vision                    18159 non-null  float64
 77  Penalties                 18159 non-null  float64
 78  Composure                 18159 non-null  float64
 79  Marking                   18159 non-null  float64
 80  StandingTackle            18159 non-null  float64
 81  SlidingTackle             18159 non-null  float64
 82  GKDiving                  18159 non-null  float64
 83  GKHandling                18159 non-null  float64
 84  GKKicking                 18159 non-null  float64
 85  GKPositioning             18159 non-null  float64
 86  GKReflexes                18159 non-null  float64
 87  Release Clause            16643 non-null  object
dtypes: float64(38), int64(5), object(45)
memory usage: 12.4+ MB
```

데이터 분석과 관련이 낮은 ‘
ID’, ‘Photo’, ‘Flag’, ‘Club Logo’, ‘Real Face’
열은 삭제해도 무방하기 때문에 drop할 예정

결측치가 너무 많은
‘Loaned From’ 열도 dorp할 예정

데이터 전처리

1) 열 삭제

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18207 entries, 0 to 18206
Data columns (total 88 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   ID                        18207 non-null  int64
 1   Name                      18207 non-null  object
 2   Age                       18207 non-null  int64
 3   Photo                     18207 non-null  object
 4   Nationality               18207 non-null  object
 5   Flag                      18207 non-null  object
 6   Overall                   18207 non-null  int64
 7   Potential                 18207 non-null  int64
 8   Club                      17966 non-null  object
 9   Club Logo                 18207 non-null  object
 10  Value                     18207 non-null  object
 11  Wage                      18207 non-null  object
 12  Special                   18207 non-null  int64
 13  Preferred Foot            18159 non-null  object
 14  International Reputation  18159 non-null  float64
 15  Weak Foot                 18159 non-null  float64
 16  Skill Moves               18159 non-null  float64
 17  Work Rate                 18159 non-null  object
 18  Body Type                 18159 non-null  object
 19  Real Face                 18159 non-null  object
 20  Position                  18147 non-null  object
 21  Jersey Number             18147 non-null  float64
 22  Joined                    16654 non-null  object
 23  Loaned From               1264 non-null   object
 24  Contract Valid Until      17918 non-null  object
 25  Height                    18159 non-null  object
 26  Weight                    18159 non-null  object
```

해당 열들을

모두 drop!

```
data.drop(['ID', 'Photo', 'Flag', 'Club Logo', 'Real Face'], axis=1, inplace=True)
data.drop(['Loaned From'], axis=1, inplace=True)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18207 entries, 0 to 18206
Data columns (total 82 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Name                      18207 non-null  object
 1   Age                       18207 non-null  int64
 2   Nationality               18207 non-null  object
 3   Overall                   18207 non-null  int64
 4   Potential                 18207 non-null  int64
 5   Club                      17966 non-null  object
 6   Value                     18207 non-null  object
 7   Wage                      18207 non-null  object
 8   Special                   18207 non-null  int64
 9   Preferred Foot            18159 non-null  object
 10  International Reputation  18159 non-null  float64
 11  Weak Foot                 18159 non-null  float64
 12  Skill Moves               18159 non-null  float64
 13  Work Rate                 18159 non-null  object
 14  Body Type                 18159 non-null  object
 15  Position                  18147 non-null  object
 16  Jersey Number             18147 non-null  float64
 17  Joined                    16654 non-null  object
 18  Contract Valid Until      17918 non-null  object
 19  Height                    18159 non-null  object
 20  Weight                    18159 non-null  object
```

데이터 전처리

2) 포지션 능력치 데이터 전처리

```
data.loc[:, 'LS':'RB']
```

	LS	ST	RS	LW	LF	CF	RF	RW	LAM	CAM	...	LWB	LDM	CDM	RDM	RWB	LB	LCB	CB	RCB	RB
0	88+2	88+2	88+2	92+2	93+2	93+2	93+2	92+2	93+2	93+2	...	64+2	61+2	61+2	61+2	64+2	59+2	47+2	47+2	47+2	59+2
1	91+3	91+3	91+3	89+3	90+3	90+3	90+3	89+3	88+3	88+3	...	65+3	61+3	61+3	61+3	65+3	61+3	53+3	53+3	53+3	61+3
2	84+3	84+3	84+3	89+3	89+3	89+3	89+3	89+3	89+3	89+3	...	65+3	60+3	60+3	60+3	65+3	60+3	47+3	47+3	47+3	60+3
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	82+3	82+3	82+3	87+3	87+3	87+3	87+3	87+3	88+3	88+3	...	77+3	77+3	77+3	77+3	77+3	73+3	66+3	66+3	66+3	73+3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
18202	42+2	42+2	42+2	44+2	44+2	44+2	44+2	44+2	45+2	45+2	...	44+2	45+2	45+2	45+2	44+2	45+2	45+2	45+2	45+2	45+2
18203	45+2	45+2	45+2	39+2	42+2	42+2	42+2	39+2	40+2	40+2	...	30+2	31+2	31+2	31+2	30+2	29+2	32+2	32+2	32+2	29+2
18204	45+2	45+2	45+2	45+2	46+2	46+2	46+2	45+2	44+2	44+2	...	34+2	30+2	30+2	30+2	34+2	33+2	28+2	28+2	28+2	33+2
18205	47+2	47+2	47+2	47+2	46+2	46+2	46+2	47+2	45+2	45+2	...	36+2	32+2	32+2	32+2	36+2	35+2	31+2	31+2	31+2	35+2
18206	43+2	43+2	43+2	45+2	44+2	44+2	44+2	45+2	45+2	45+2	...	46+2	46+2	46+2	46+2	46+2	46+2	47+2	47+2	47+2	46+2

18207 rows × 26 columns

'숫자+숫자'로 되어 있는 포지션별 능력치(['LS':'RB']) 열들을 '두 숫자의 합'으로 바꿔줘야 함

데이터 전처리

2) 포지션 능력치 데이터 전처리

```
data_stats=data.loc[:, 'LS':'RB']

stats=data_stats.columns

for i in range(len(data)):
    if data_stats.loc[i].isnull().sum() != 0:
        continue
    for stat in stats:
        data.loc[i, stat]=eval(data.loc[i, stat])
```

for문을 활용해 데이터를 순회하면서

Null 값을 가진 행을 제외하고

eval함수로 능력치의 합을 구함

*eval 함수 : 문자형 데이터를 숫자형처럼 연산을 가능케 함

```
data.loc[:, 'LS':'RB']
```

	LS	ST	RS	LW	LF	CF	RF	RW	LAM	CAM	...	LWB	LDM	CDM	RDM	RWB	LB	LCB	CB	RCB	RB
0	90	90	90	94	95	95	95	94	95	95	...	66	63	63	63	66	61	49	49	49	61
1	94	94	94	92	93	93	93	92	91	91	...	68	64	64	64	68	64	56	56	56	64
2	87	87	87	92	92	92	92	92	92	92	...	68	63	63	63	68	63	50	50	50	63
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	85	85	85	90	90	90	90	90	91	91	...	80	80	80	80	80	76	69	69	69	76
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
18202	44	44	44	46	46	46	46	46	47	47	...	46	47	47	47	46	47	47	47	47	47
18203	47	47	47	41	44	44	44	41	42	42	...	32	33	33	33	32	31	34	34	34	31
18204	47	47	47	47	48	48	48	47	46	46	...	36	32	32	32	36	35	30	30	30	35
18205	49	49	49	49	48	48	48	49	47	47	...	38	34	34	34	38	37	33	33	33	37
18206	45	45	45	47	46	46	46	47	47	47	...	48	48	48	48	48	48	49	49	49	48

18207 rows × 26 columns

데이터 전처리

3) 행 삭제

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18207 entries, 0 to 18206
Data columns (total 82 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Name                      18207 non-null  object
 1   Age                       18207 non-null  int64
 2   Nationality               18207 non-null  object
 3   Overall                   18207 non-null  int64
 4   Potential                 18207 non-null  int64
 5   Club                      17966 non-null  object
 6   Value                     18207 non-null  object
 7   Wage                      18207 non-null  object
 8   Special                   18207 non-null  int64
 9   Preferred Foot            18159 non-null  object
 10  International Reputation  18159 non-null  float64
 11  Weak Foot                 18159 non-null  float64
 12  Skill Moves               18159 non-null  float64
 13  Work Rate                 18159 non-null  object
 14  Body Type                 18159 non-null  object
```

… 중략 …

```
 70  Vision                    18159 non-null  float64
 71  Penalties                 18159 non-null  float64
 72  Composure                 18159 non-null  float64
 73  Marking                   18159 non-null  float64
 74  StandingTackle            18159 non-null  float64
 75  SlidingTackle             18159 non-null  float64
 76  GKDiving                  18159 non-null  float64
 77  GKHandling                18159 non-null  float64
 78  GKKicking                 18159 non-null  float64
 79  GKPositioning             18159 non-null  float64
 80  GKReflexes                18159 non-null  float64
 81  Release Clause            16643 non-null  object
```

공통적으로 결측 행이 48개 있음

```
data.iloc[13236:13283, 9:80]
```

	Preferred Foot	International Reputation	Weak Foot	Skill Moves	Work Rate	Body Type	Position	중	StandingTackle	SlidingTackle	GKDiving	GKHandling	GKKicking	GKPositioning
13236	NaN	NaN	NaN	NaN	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN
13237	NaN	NaN	NaN	NaN	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN
13238	NaN	NaN	NaN	NaN	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN
13239	NaN	NaN	NaN	NaN	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN
13240	NaN	NaN	NaN	NaN	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN
13241	NaN	NaN	NaN	NaN	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN
13242	NaN	NaN	NaN	NaN	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN
13271	NaN	NaN	NaN	NaN	NaN	NaN	NaN	략	NaN	NaN	NaN	NaN	NaN	NaN
13272	NaN	NaN	NaN	NaN	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN
13273	NaN	NaN	NaN	NaN	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN
13274	NaN	NaN	NaN	NaN	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN
13275	NaN	NaN	NaN	NaN	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN
13276	NaN	NaN	NaN	NaN	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN
13277	NaN	NaN	NaN	NaN	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN
13278	NaN	NaN	NaN	NaN	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN
13279	NaN	NaN	NaN	NaN	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN
13280	NaN	NaN	NaN	NaN	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN
13281	NaN	NaN	NaN	NaN	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN
13282	NaN	NaN	NaN	NaN	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN

```
data = data.dropna(subset=['Preferred Foot'],how='any', axis=0)
```

데이터 전처리

4) 단위 통일 – 키 & 몸무게

```
data['Height']
```

```
0         5'7
1         6'2
2         5'9
3         6'4
4        5'11
         ...
18202     5'9
18203     6'3
18204     5'8
18205    5'10
18206    5'10
Name: Height, Length: 18159, dtype: object
```

```
data['Height'] = data['Height'].str.replace('′', '.')
data['Height'].fillna(0, inplace=True)
data['Height'] = data['Height'].astype(float) * 30.84
```

① ['Height']를 작은 따옴표 (') 기준으로 분리

② NaN 값은 0으로 대체

③ feet ->cm이므로 * 30.84

```
data['Height'].round(2)
```

```
0        175.79
1        191.21
2        181.96
3        197.38
4        157.59
          ...
18202    181.96
18203    194.29
18204    178.87
18205    157.28
18206    157.28
Name: Height, Length: 18159, dtype: float64
```

```
data['Weight']
```

```
0        159lbs
1        183lbs
2        150lbs
3        168lbs
4        154lbs
          ...
18202    134lbs
18203    170lbs
18204    148lbs
18205    154lbs
18206    176lbs
Name: Weight, Length: 18159, dtype: object
```

```
data['Weight'] = data['Weight'].str.replace('lbs', '')
data['Weight'].fillna(0, inplace=True)
data['Weight'] = data['Weight'].astype(float) / 2.205
```

① ['Weight']의 lbs를 공백으로 대체

② NaN 값은 0으로 대체

③ lbs ->kg이므로 / 2.205

```
data['Weight'].round(2)
```

```
0        72.11
1        82.99
2        68.03
3        76.19
4        69.84
         ...
18202    60.77
18203    77.10
18204    67.12
18205    69.84
18206    79.82
Name: Weight, Length: 18159, dtype: float64
```

데이터 전처리

5) 단위 통일 – 화폐

```
data[['Value','Wage','Release Clause']]
```

	Value	Wage	Release Clause
0	€110.5M	€565K	€226.5M
1	€77M	€405K	€127.1M
2	€118.5M	€290K	€228.1M
3	€72M	€260K	€138.6M
4	€102M	€355K	€196.4M
...	...	...	...
18202	€60K	€1K	€143K
18203	€60K	€1K	€113K
18204	€60K	€1K	€165K
18205	€60K	€1K	€143K
18206	€60K	€1K	€165K

```
def money(col):
    col=col.str.replace('€', '')
    c=col.str.split(pat=(r'([A-Z])'), expand=True)
    c.iloc[:][1].replace('M', 1000000, inplace=True)
    c.iloc[:][1].replace('K', 1000, inplace=True)
    c.fillna(0, inplace=True)
    c[0] = c[0].astype(float)
    c[2] = c[0] * c[1]
    col=c[2]*1.21
    return col.astype(int)
```

① €를 공백으로 대체

② 숫자와 알파벳을 분리해 데이터프레임 생성 (expand=True)

③ M->1,000,000
K->1,000 으로 숫자 변환

④ NaN 값은 0으로 대체

⑤ 원래 금액 * M or K(숫자로 변환된)

⑥ € ->$ 이므로 *1.21

```
data['Value'] = money(data['Value'])
data['Wage'] = money(data['Wage'])
data['Release Clause'] = money(data['Release Clause'])
```

```
data[['Value','Wage','Release Clause']]
```

	Value	Wage	Release Clause
0	133705000	683650	274065000
1	93170000	490050	153791000
2	143385000	350900	276001000
3	87120000	314600	167706000
4	123420000	429550	237644000
...	...	...	...
18202	72600	1210	173030
18203	72600	1210	136730
18204	72600	1210	199650
18205	72600	1210	173030
18206	72600	1210	199650

6) 포지션 그룹핑

```
data['Position'].value_counts()
```

```
ST      2152
GK      2025
CB      1778
CM      1394
LB      1322
RB      1291
RM      1124
LM      1095
CAM      958
CDM      948
RCB      662
LCB      648
LCM      395
RCM      391
LW       381
RW       370
RDM      248
LDM      243
LS       207
RS       203
RWB       87
LWB       78
CF        74
RAM       21
LAM       21
RF        16
LF        15
Name: Position, dtype: int64
```

```
def classify_position(x):
    if x in (['ST','LW','RW','LS','RS','CF','LF','RF']):
        return 'FW'
    elif x in(['CM','RM','LM','CAM','CDM','RCM','LCM','RDM','LDM','RAM','LAM']):
        return 'MD'
    elif x in(['CB','LB','RB','RCB','LCB','RWB','LWB']):
        return 'DF'
    else:
        return 'GK'

data['Position simplified'] = data['Position'].apply(classify_position)
data['Position simplified'].value_counts(dropna=False)
```

```
MD    6838
DF    5866
FW    3418
GK    2037
Name: Position simplified, dtype: int64
```

포지션의 수가 너무 많아 공격수/미드필더/수비수/골키퍼로 새롭게 그룹핑 진행

'FW'	'ST','LW','RW','LS','RS','CF','LF','RF'
'MD'	'CM','RM','LM','CAM','CDM','RCM', 'LCM','RDM','LDM','RAM','LAM'
'DF'	'CB','LB','RB','RCB','LCB','RWB','LWB'
'GK'	'GK'

데이터 전처리

7) 시계열 데이터 전처리

```
data['Contract Valid Until'][20:40]
```

```
20          2023
21          2020
22          2021
23          2021
24          2020
25          2022
26          2023
27          2021
28     30-Jun-19
29          2022
30          2022
31          2020
32          2023
33          2021
34          2021
35          2022
36          2022
37          2022
38     30-Jun-19
39          2020
Name: Contract Valid Until, dtype: object
```

'Contract Valid Until' 열을 살펴보면 30-Jun-19와 같이 년도(year)형태가 아닌 데이터가 있음

```
data['Contract Valid Until']=pd.to_datetime(data['Contract Valid Until'])
data['Contract Valid Until']=data['Contract Valid Until'].dt.to_period(freq='A')
data['Contract Valid Until'][20:40]
```

```
20    2023
21    2020
22    2021
23    2021
24    2020
25    2022
26    2023
27    2021
28    2019
29    2022
30    2022
31    2020
32    2023
33    2021
34    2021
35    2022
36    2022
37    2022
38    2019
39    2020
Name: Contract Valid Until, dtype: period[A-DEC]
```

① pd.to_datetime을 활용해 날짜형 데이터로 변환

② dt.to_period를 이용해 연(year)을 기준으로 데이터를 반환 (freq='A')

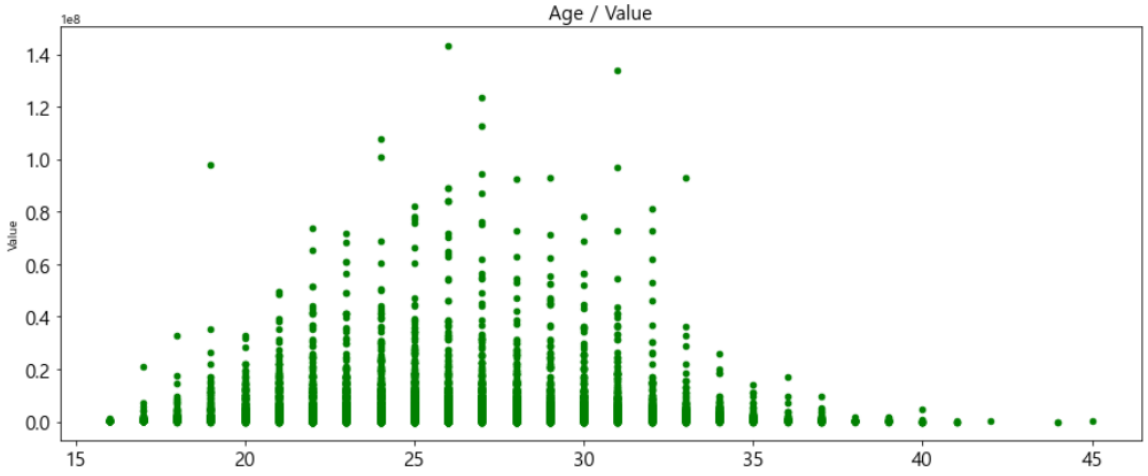
III. 데이터 시각화

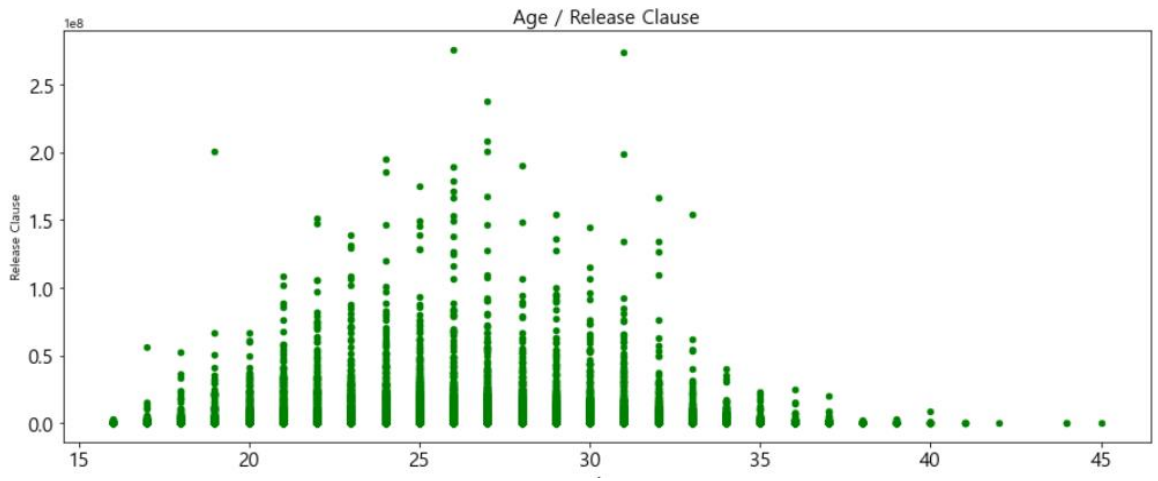
1-1) 선수 – 연령 & 주급/시장 가치/바이아웃

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
fig, axs = plt.subplots(figsize=(15, 20), nrows=3, ncols=1, squeeze=False)

data.plot(kind='scatter', x='Age', y='Wage', c='green', ax=axs[0][0], fontsize=15)
axs[0][0].set_title('Age / Wage', fontsize=15)
data.plot(kind='scatter', x='Age', y='Value', c='green', ax=axs[1][0], fontsize=15)
axs[1][0].set_title('Age / Value', fontsize=15)
data.plot(kind='scatter', x='Age', y='Release Clause', c='green', ax=axs[2][0], fontsize=15)
axs[2][0].set_title('Age / Release Clause', fontsize=15)
```

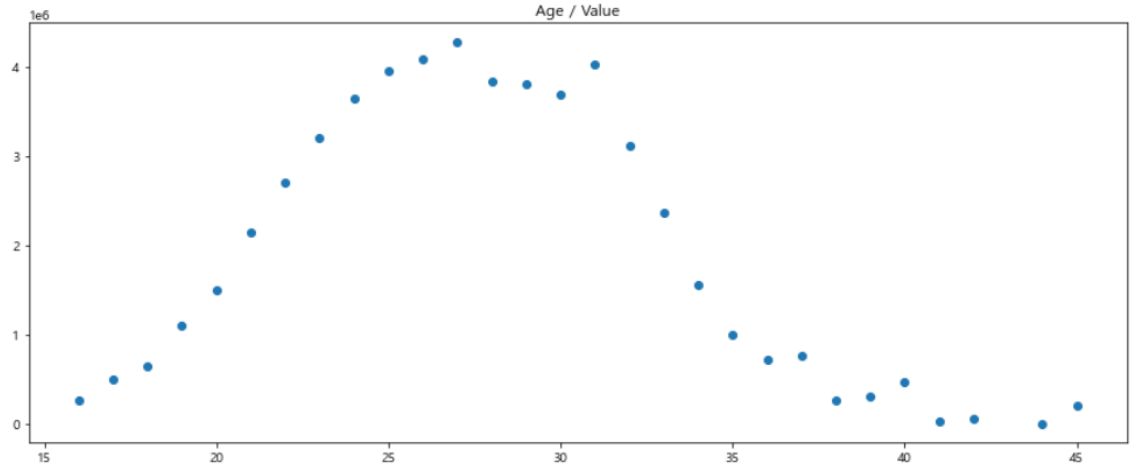
```
Text(0.5, 1.0, 'Age / Release Clause')
```

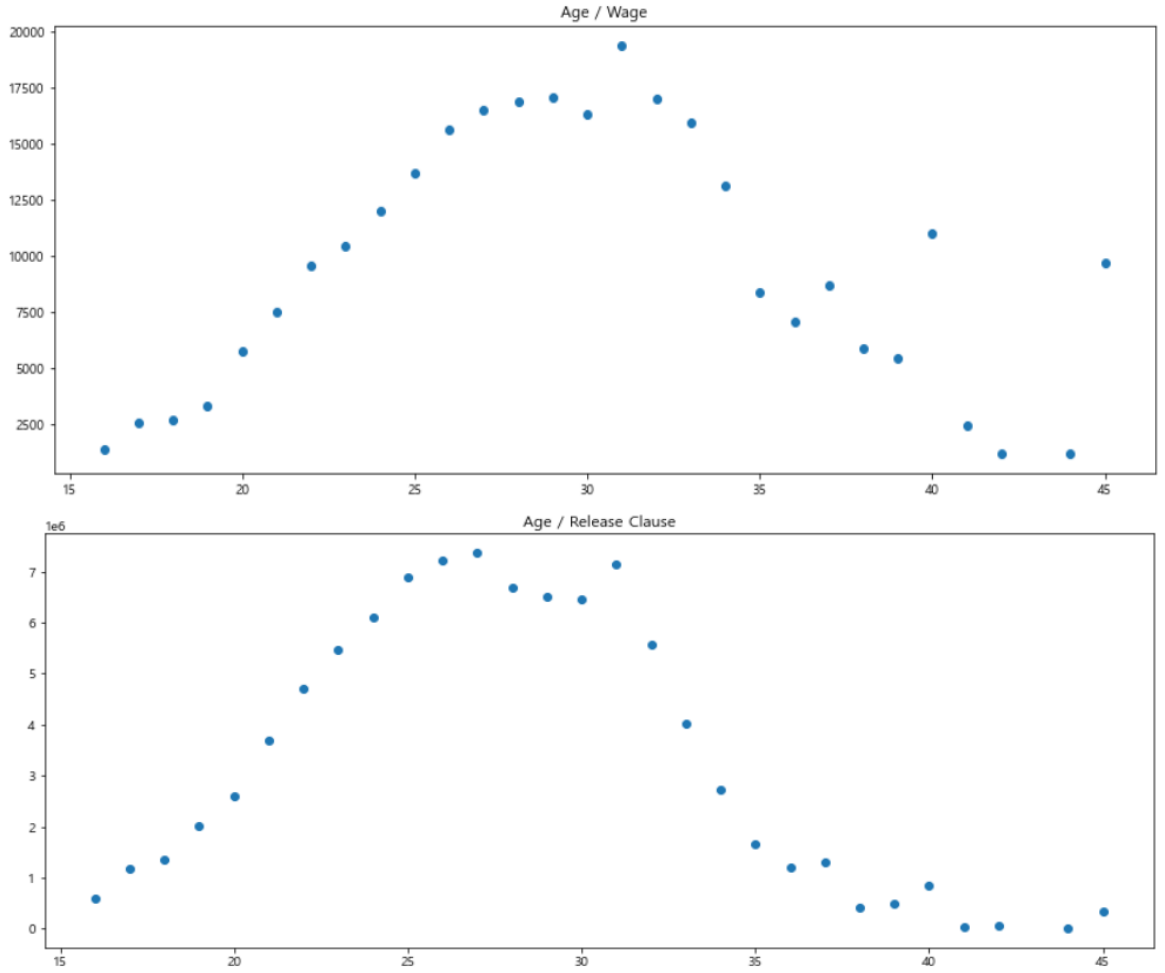




1-2) 선수 – 연령 & 주급/시장 가치/바이아웃 평균치

```
group_age_wage = data.groupby('Age')['Wage'].mean().reset_index()
group_age_value = data.groupby('Age')['Value'].mean().reset_index()
group_age_release_clause = data.groupby('Age')['Release Clause'].mean().reset_index()

fig, axs = plt.subplots(figsize=(15, 20), nrows=3, ncols=1, squeeze=False)

axs[0][0].scatter(data=group_age_wage, x='Age', y='Wage')
axs[1][0].scatter(data=group_age_value, x='Age', y='Value')
axs[2][0].scatter(data=group_age_release_clause, x='Age', y='Release Clause')

axs[0][0].title.set_text('Age / Wage')
axs[1][0].title.set_text('Age / Value')
axs[2][0].title.set_text('Age / Release Clause')
```

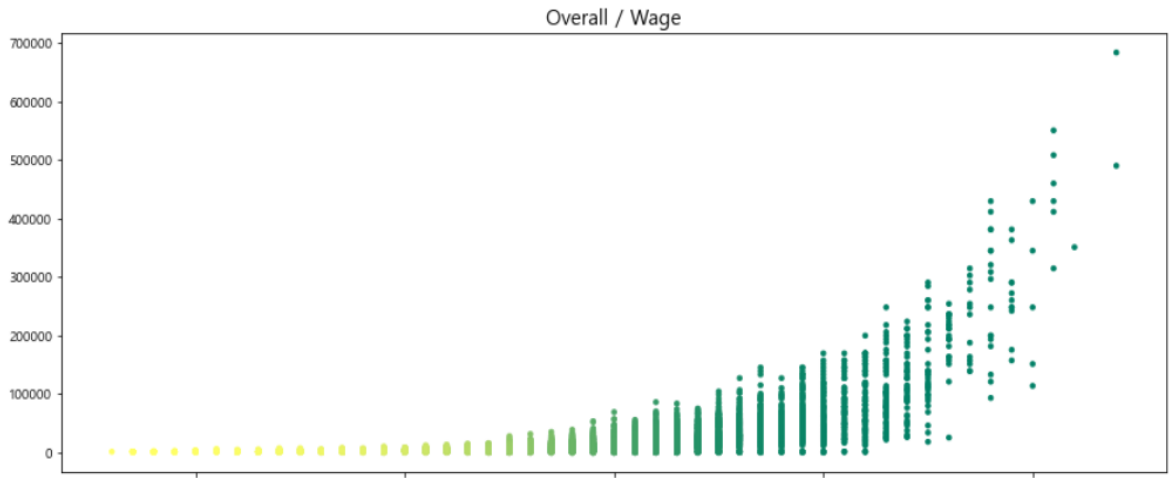


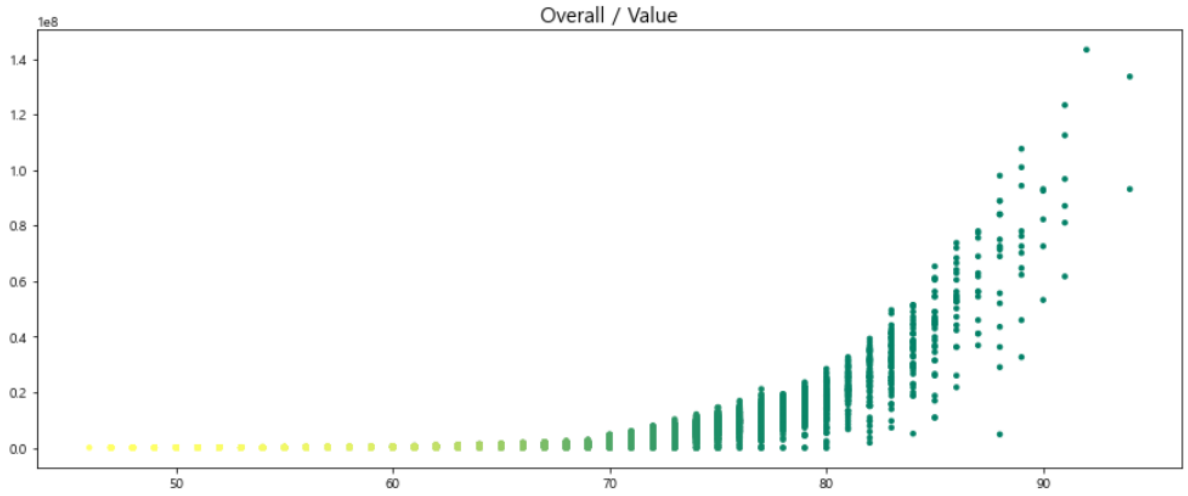


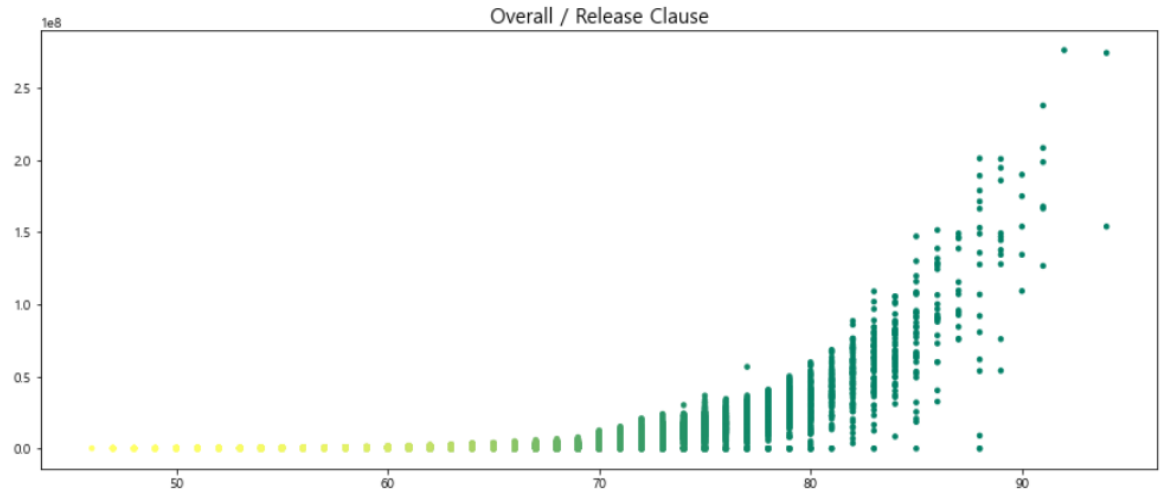
평균치로 분석, 1-1)과 유사한 결과를 얻음

1-3) 선수 전반 능력치 & 주급/시장 가치/바이아웃

```
fig = plt.figure(figsize=(15, 20))

colors = sns.color_palette("summer", len(data))

ax1 = fig.add_subplot(311)
ax1.scatter(data=data, x='Overall', y='Wage', s=12, color=colors)
ax1.set_title('Overall / Wage', fontsize=15)
ax2 = fig.add_subplot(312)
ax2.scatter(data=data, x='Overall', y='Value', s=12, color=colors)
ax2.set_title('Overall / Value', fontsize=15)
ax3 = fig.add_subplot(313)
ax3.scatter(data=data, x='Overall', y='Release Clause', s=12, color=colors)
ax3.set_title('Overall / Release Clause', fontsize=15)
```



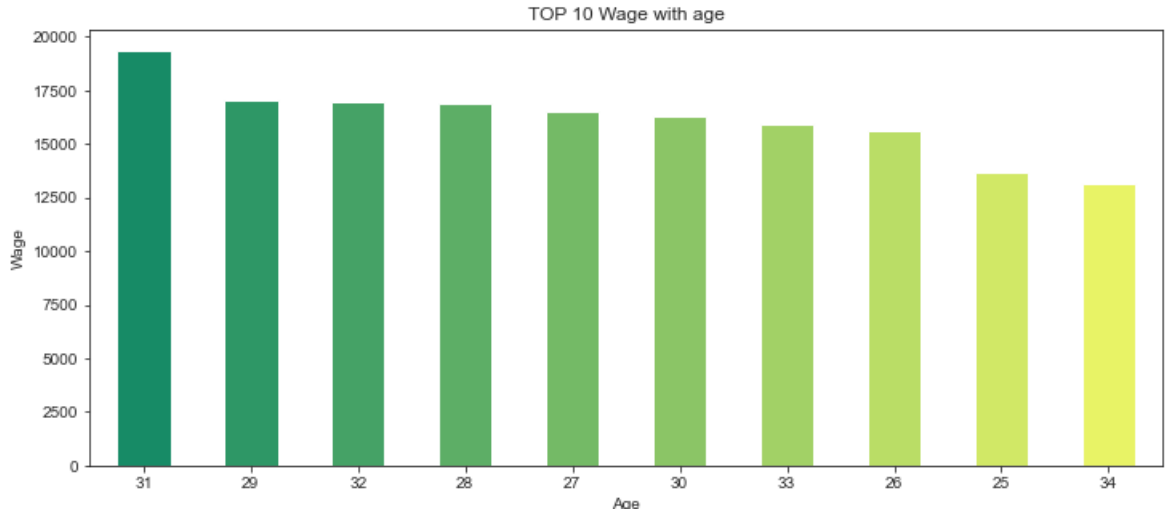




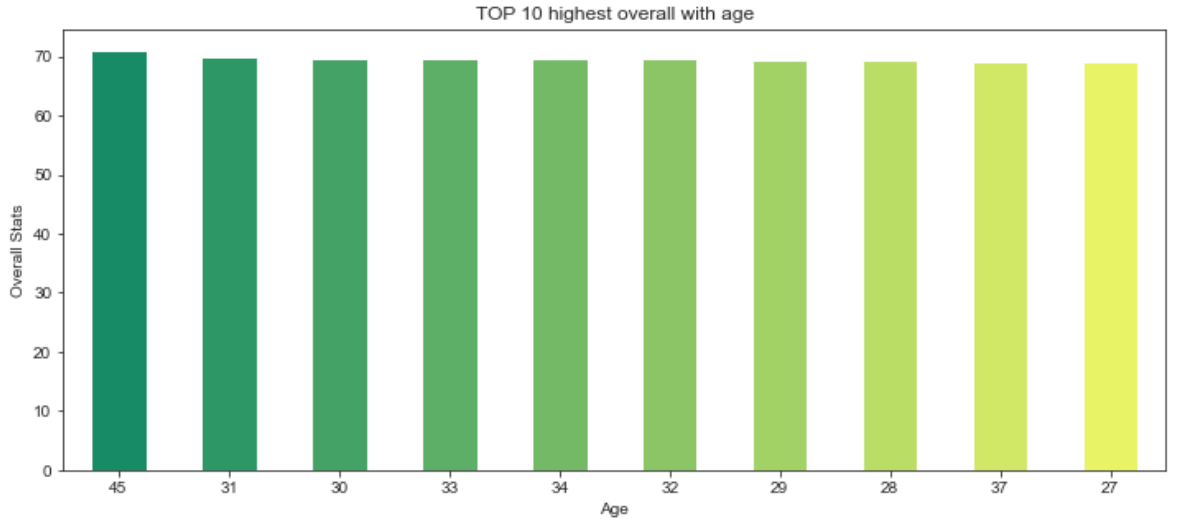
데이터 시각화

1-4) 선수 연령 & 주급/전반 능력치/시장 가치

```
grouped_wage_top = data.groupby('Age').Wage.mean().sort_values(ascending=False)
top10 = grouped_wage_top.index[0:10]

colors = sns.color_palette('summer', 10)
plt.figure(figsize=(12, 5))
grouped_wage_top[top10].plot(kind='bar', color=colors)
plt.xticks(rotation='horizontal')
plt.xlabel('Age')
plt.ylabel('Wage')
plt.title('TOP 10 Wage with age')
plt.show()
```



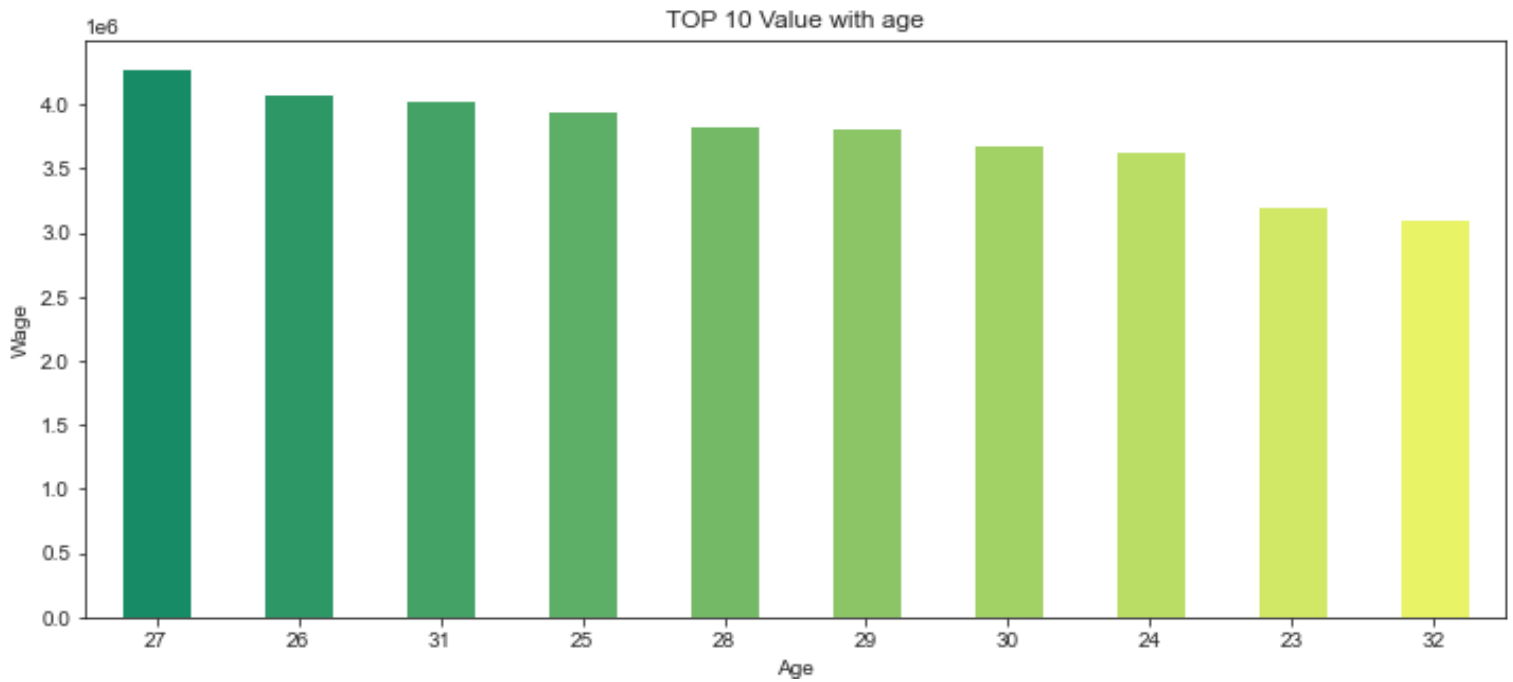

```
grouped_overall_top = data.groupby('Age').Overall.mean().sort_values(ascending=False)
top10 = grouped_overall_top.index[0:10]

colors = sns.color_palette('summer', 10)
plt.figure(figsize=(12, 5))
grouped_1[top10].plot(kind='bar', color= colors)
plt.xticks(rotation='horizontal')
plt.xlabel('Age')
plt.ylabel('Overall Stats')
plt.title('TOP 10 highest overall with age')
plt.show()
```



20대 후반~30대 초반에서 주급, 선수가치, 오버롤이 가장 높음

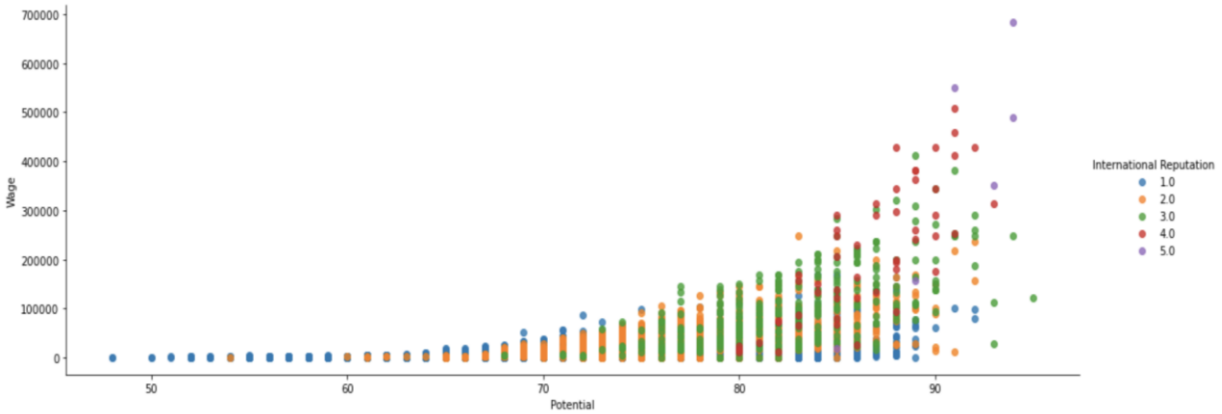
1-4) 선수 연령 & 주급/전반 능력치/시장 가치

```
grouped_value_top = data.groupby('Age').Value.mean().sort_values(ascending=False)
top10 = grouped_value_top.index[0:10]

colors = sns.color_palette('summer', 10)
plt.figure(figsize=(12, 5))
grouped_value_top[top10].plot(kind='bar', color=colors)
plt.xticks(rotation='horizontal')
plt.xlabel('Age')
plt.ylabel('Wage')
plt.title('TOP 10 Value with age')
plt.show()
```

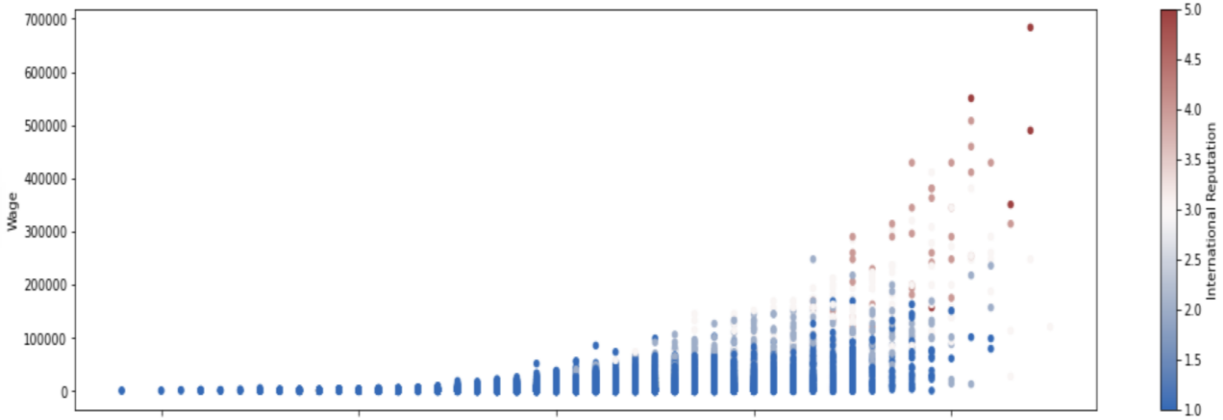


1-5) 선수 - Potential과 Wage의 산점도 (International Reputation으로 구분)

```
sns.lmplot(x='Potential', y='Wage', data=data,
           hue='International Reputation', aspect=3, fit_reg=False)
```



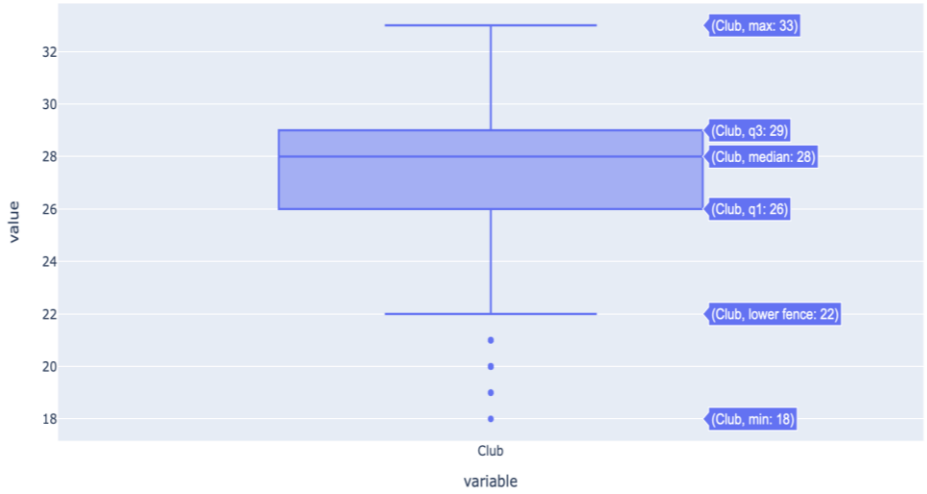

```
data.plot(kind="scatter", x="Potential", y="Wage",
          figsize=(20,5), c="International Reputation", colormap="vlag")
```



2) 클럽

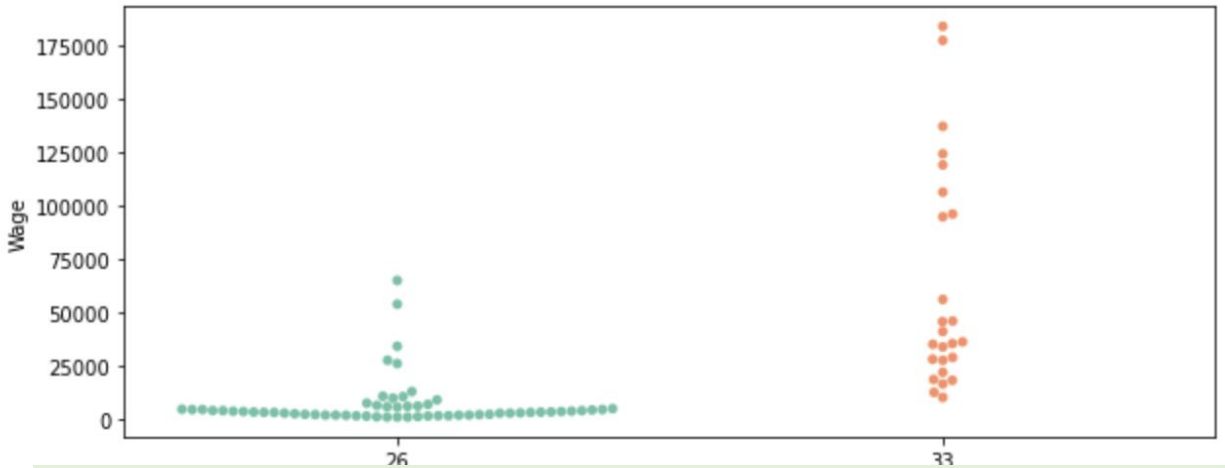
✓ 클럽 별 선수 인원 파악

```
import plotly.express as px
fig = px.box(data['Club'].value_counts())
fig.show()
```



클럽 인원이 26명, 33명인 클럽을 선택

```
a26 = pd.pivot_table(d26, index='Club', values=['Wage'], aggfunc='mean')
a26['num_player'] = 26
b33 = pd.pivot_table(d33, index='Club', values=['Wage'], aggfunc='mean')
b33['num_player'] = 33
ab = pd.concat([a26, b33])
sns.swarmplot('num_player', 'Wage', data = ab, palette="Set2")
```



- 26명이 있는 팀보다 33명이 있는 팀의 주급 평균이 높은 쪽에 분포해 있음
- 26명이 있는 팀은 주급 평균이 낮게 깔려있는 분포

데이터 시각화

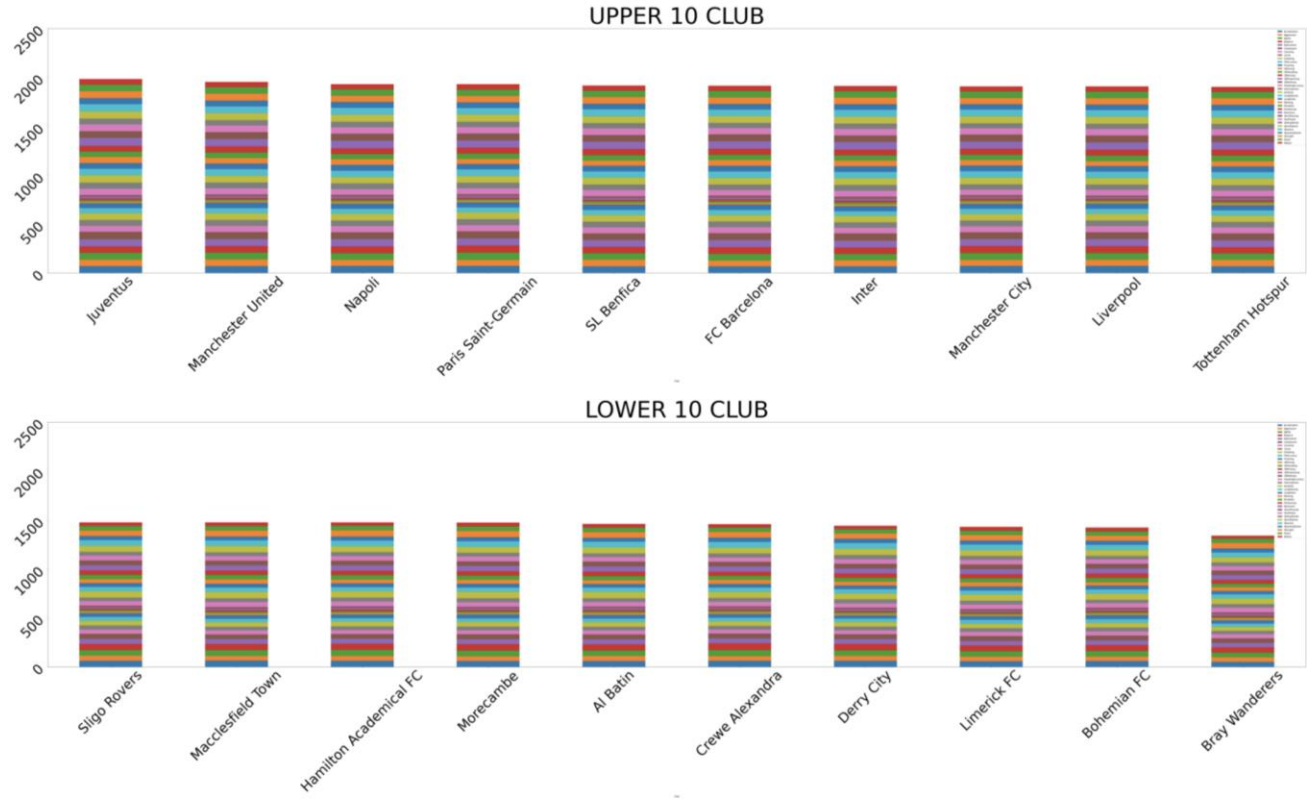
2) 클럽

✓ 클럽 별 능력치

```
ClubStat=pd.pivot_table(data, index='Club', values=data.iloc[:, 47:-2], aggfunc='mean')
ClubStat['Sum'] = ClubStat[:].sum(axis=1)
ClubStat.sort_values(by=['Sum'], axis=0, ascending=False, inplace=True)
ClubStat_Upper = ClubStat.iloc[:10, :].drop('Sum', axis=1)
ClubStat_Lower = ClubStat.iloc[-10:, :].drop('Sum', axis=1)
```

```
ClubStat_Upper.plot(kind='bar', stacked=True, legend=True, figsize=(100,20))
plt.ylim(0, 2500)
plt.title('UPPER 10 CLUB', fontsize=100)
plt.xticks(fontsize=60, rotation=45)
plt.yticks(fontsize=60, rotation=45)
ClubStat_Lower.plot(kind='bar', stacked=True, legend=True, figsize=(100,20))
plt.ylim(0, 2500)
plt.title('LOWER 10 CLUB', fontsize=100)
plt.xticks(fontsize=60, rotation=45)
plt.yticks(fontsize=60, rotation=45)
```

- 선수들의 능력치를 팀 별로 합산한 그래프
- 상위 10개 팀에 유벤투스, 맨유 등 유명팀 분포



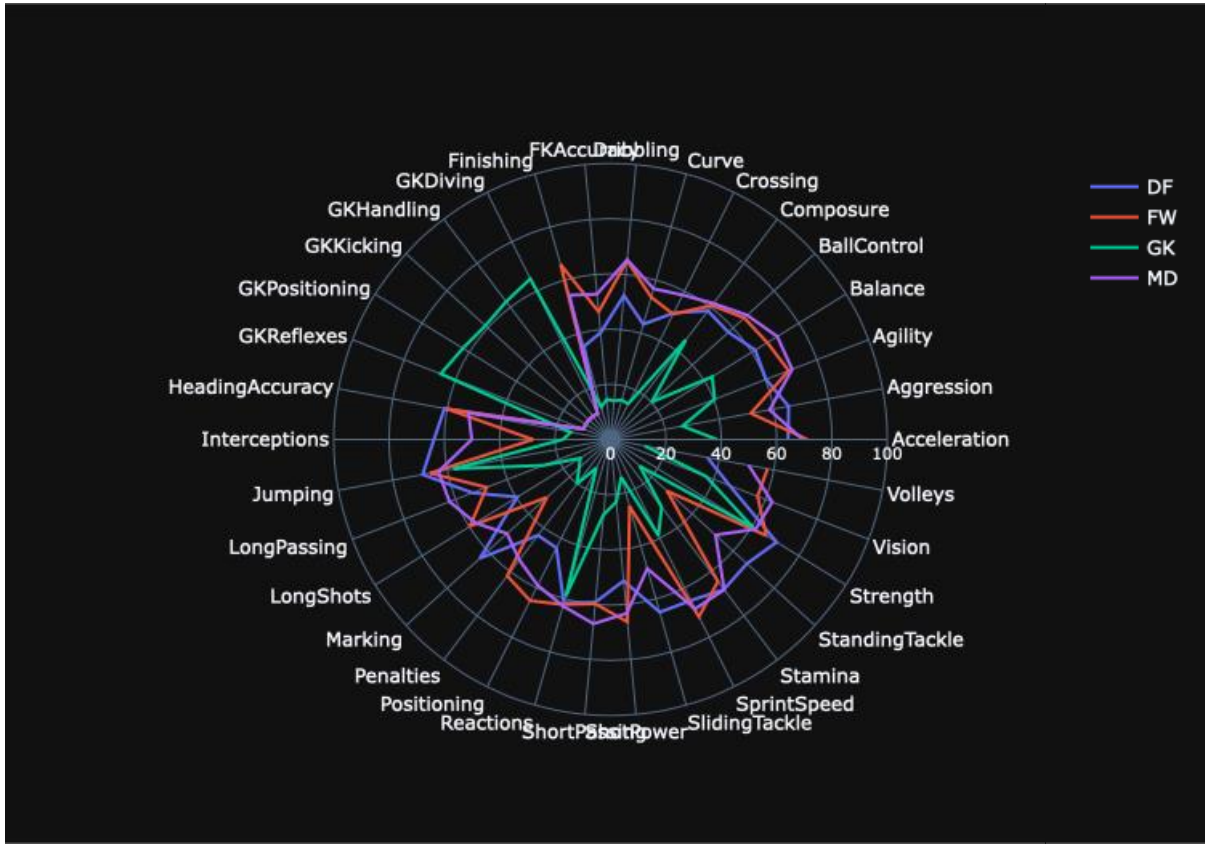
3) 포지션 - 능력치

✓ 포지션 별 능력치 평균 – GK포함

```
Position_stat=pd.pivot_table(data, index='Position simplified',
                             values=data.iloc[:, 47:-2].columns,
                             aggfunc='mean')
```

```
categories=Position_stat.columns
```

```
fig = go.Figure()

fig.add_trace(go.Scatterpolar(
      r=Position_stat.iloc[0],
      theta=categories,
      name='DF'
))
fig.add_trace(go.Scatterpolar(
      r=Position_stat.iloc[1],
      theta=categories,
      name='FW'
))
fig.add_trace(go.Scatterpolar(
      r=Position_stat.iloc[2],
      theta=categories,
      name='GK'
))
fig.add_trace(go.Scatterpolar(
      r=Position_stat.iloc[3],
      theta=categories,
      name='MD'
))
fig.update_layout(
  polar=dict(radialaxis=dict(visible=True, range=[0, 100], tickfont_size=10)),
  showlegend=True, template="plotly_dark"
)

fig.show()
```



- GK를 포함했기 때문에 플레이어들의 성격을 한 눈에 보기 어려움

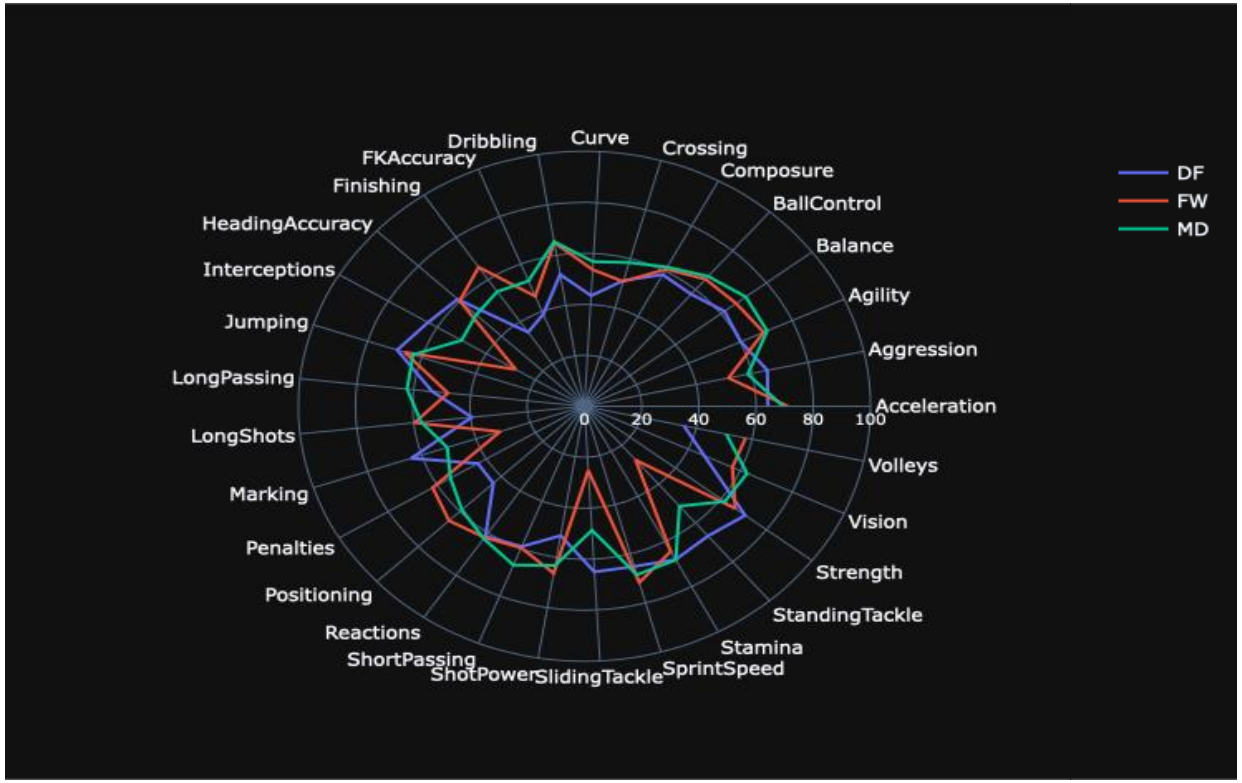
3) 포지션 - 능력치

✓ 포지션 별 능력치 평균 - GK제외

```
Position_stat_Field= pd.pivot_table(data, index='Position simplified',
                                    values=data.iloc[:, 47:-7].columns,
                                    aggfunc='mean')
```

```
categories=Position_stat_Field.columns
```

```
fig = go.Figure()

fig.add_trace(go.Scatterpolar(
      r=Position_stat_Field.iloc[0],
      theta=categories,
      name='DF'
))
fig.add_trace(go.Scatterpolar(
      r=Position_stat_Field.iloc[1],
      theta=categories,
      name='FW'
))
fig.add_trace(go.Scatterpolar(
      r=Position_stat_Field.iloc[3],
      theta=categories,
      name='MD'
))
fig.update_layout(
  polar=dict(radialaxis=dict(visible=True, range=[0, 100], tickfont_size=10)),
  showlegend=True, template="plotly_dark"
)
fig.show()
```



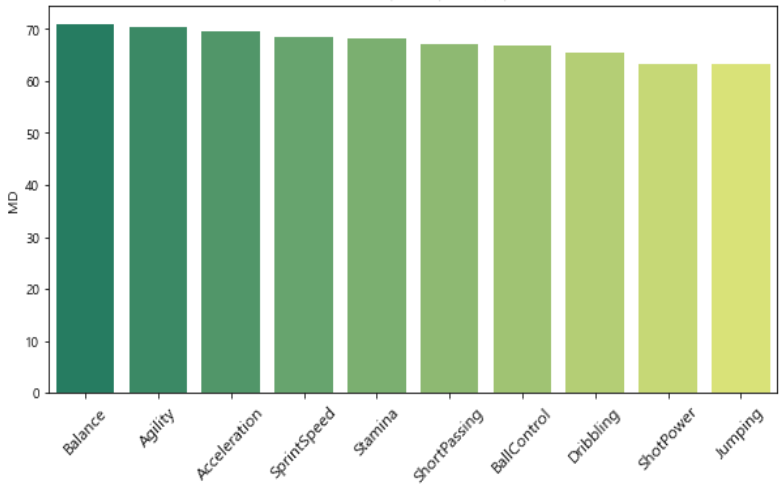
- 공격수는 다른 필드플레이어들에 비해 특정 능력치들이 특화되면 오히려 떨어지는 부분들도 있음
- 미드필더는 다른 필드 플레이어들이 비해 전반적으로 원만한 평균적인 능력치를 필요로함

3) 포지션 - 능력치

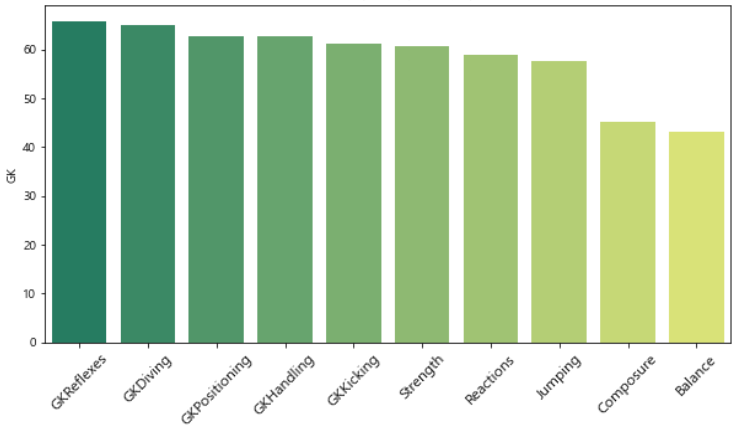
✓ 포지션 별 상위 10개 능력치

```
data_pos_simple = pd.concat([data['Position simplified'], data.loc[:, 'Crossing':'GKReflexes']], axis=1)

group_pos_simple = data_pos_simple.groupby('Position simplified').mean()

colors = sns.color_palette("summer", len(data))

top_ability_position = group_pos_simple.apply(lambda x: [x.nlargest(10).index.tolist(), x[x.nlargest(10).index.tolist(
)], x.name], axis=1)

for i in range(len(top_ability_position)):
    curr = top_ability_position[i]
    fig = plt.figure(figsize=(10, 5))
    x = curr[0]
    y = curr[1]
    pos = curr[2]
    plt.title(pos + ' 상위 15개 능력치', fontsize=15)
    sns.barplot(x, y, palette='summer')
    plt.xticks(rotation=45, fontsize=12)
```

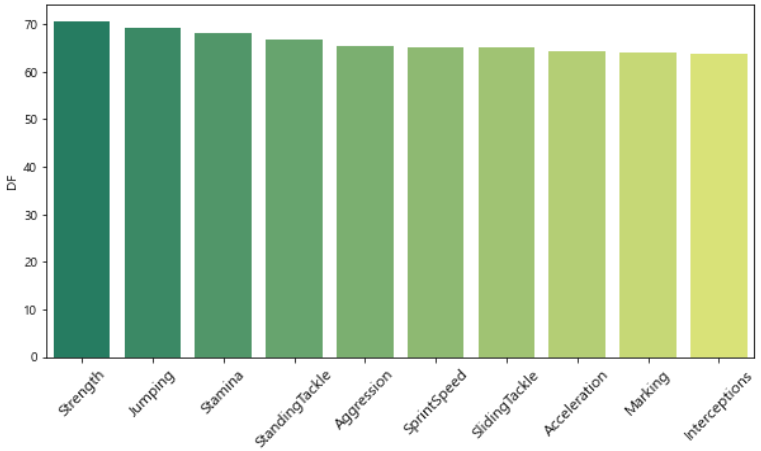
MD 상위 10개 능력치



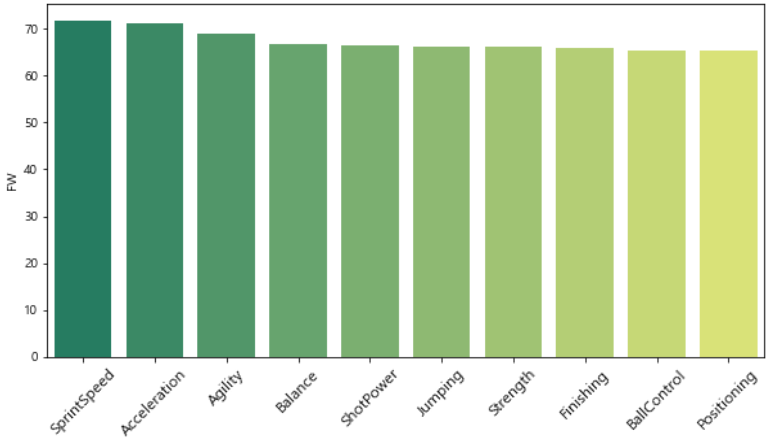
GK 상위 10개 능력치



DF 상위 10개 능력치



FW 상위 10개 능력치



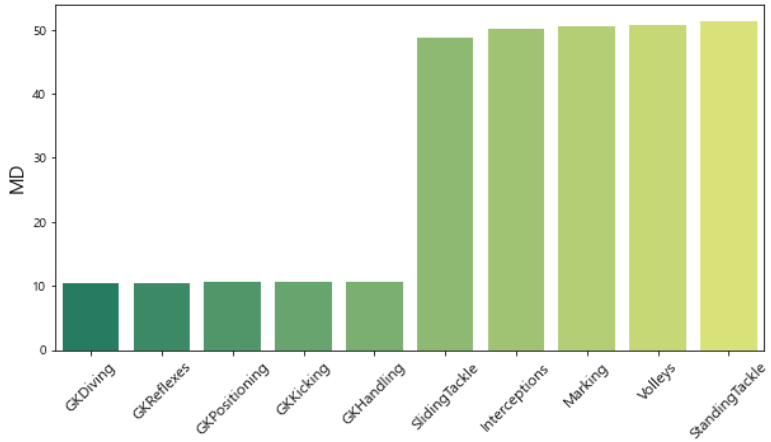
각 포지션에 어울리는 능력치가 가장 높음 (MD: Balance / GK: GKReflexes / DF: Strength / FW: Sprintspeed)

3) 포지션 - 능력치

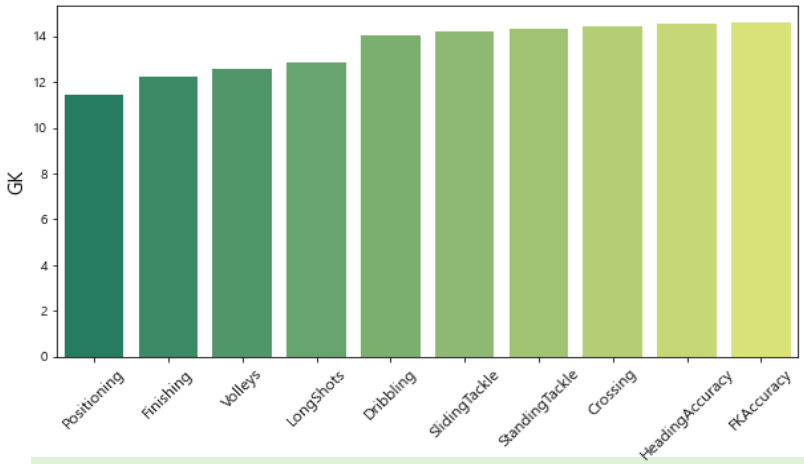
✓ 포지션 별 하위 10개 능력치

```
data_pos_simple = pd.concat([data['Position simplified'], data.loc[:, 'Crossing':'GKReflexes']], axis=1)

group_pos_simple = data_pos_simple.groupby('Position simplified').mean()

colors = sns.color_palette("summer", len(data))

top_ability_position = group_pos_simple.apply(lambda x: [x.nsmallest(10).index.tolist(), x[x.nsmallest(10).index.tolis
t()], x.name], axis=1)

for i in range(len(top_ability_position)):
    curr = top_ability_position[i]
    fig = plt.figure(figsize=(10, 5))
    x = curr[0]
    y = curr[1]
    pos = curr[2]
    plt.title(pos + ' 하위 15개 능력치', fontsize=15)
    plt.ylabel('Ability', fontsize=15)
    sns.barplot(x, y, palette='summer')
    plt.xticks(rotation=45, fontsize=12)
```

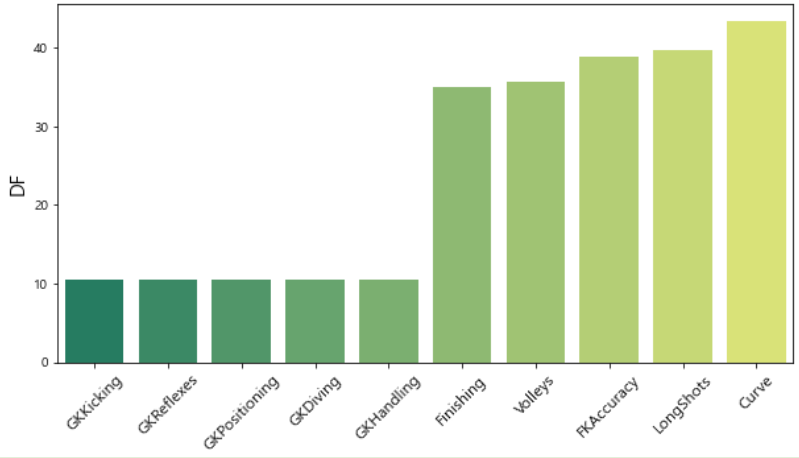
MD하위 10개능력치



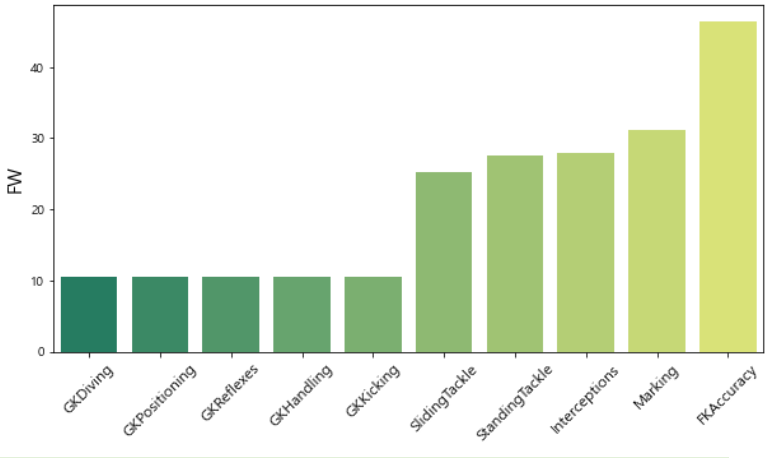
GK하위 10개능력치



DF하위 10개능력치



FW하위 10개능력치



각 포지션에 어울리지 않는 능력치가 가장 낮음 (MD: GKDriving / GK: Positioning / DF: GKKicking / FW:GKDriving)

데이터 시각화

3) 포지션 - 능력치

✓ 세분화된 포지션 별 상위 10개 능력치

```
def group_position_for_graph(x):
    if x in ['LS','ST','RS','LF','CF','RF']:
        return '공격수'
    elif x in ['LW','RW']:
        return '윙어'
    elif x in ['LAM','RAM','LM','RM']:
        return '측면 미드필더'
    elif x in ['CAM','CM','LCM','RCM']:
        return '중앙 미드필더'
    elif x in ['LDM','CDM','RDM']:
        return '수비형 미드필더'
    elif x in ['LWB','LB','RB','RWB']:
        return '풀백'
    elif x in ['LCB','CB','RCB']:
        return '센터백'
    else:
        return '골키퍼'
```

```
position_classify = data['Position'].apply(group_position_for_graph)
position_classify
```

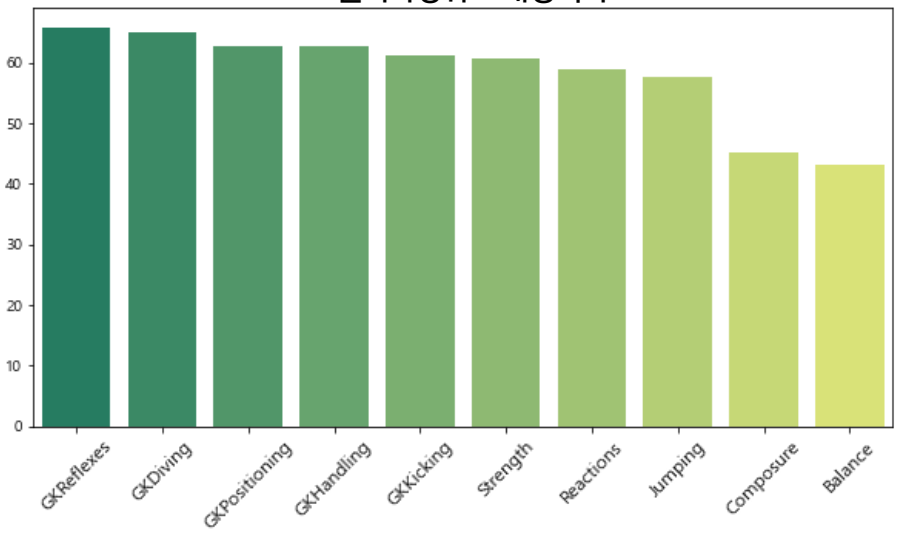
```
0              공격수
1              공격수
2               윙어
3              골키퍼
4          중앙 미드필더
           ...
18202      중앙 미드필더
18203          공격수
18204          공격수
18205           윙어
18206      중앙 미드필더
Name: Position, Length: 18159, dtype: object
```

```
data_pos_simple2 = pd.concat([position_classify, data.loc[:, 'Crossing':'GKReflexes']], axis=1)
group_pos_simple2 = data_pos_simple2.groupby('Position').mean()
top_ability_position = group_pos_simple2.apply(
    lambda x: [x.nlargest(10).index.tolist(), x[x.nlargest(10).index.tolist()], x.name], axis=1)

for i in range(len(top_ability_position)):
    curr = top_ability_position[i]
    fig = plt.figure(figsize=(10, 5))

    x = curr[0]
    y = curr[1]
    pos = curr[2]
    plt.title(pos + ' 상위 10개 능력치', fontsize=15)
    plt.xticks(rotation=45, fontsize=12)
    plt.ylabel('', fontsize=15)
    sns.barplot(x, y, palette='summer')
```

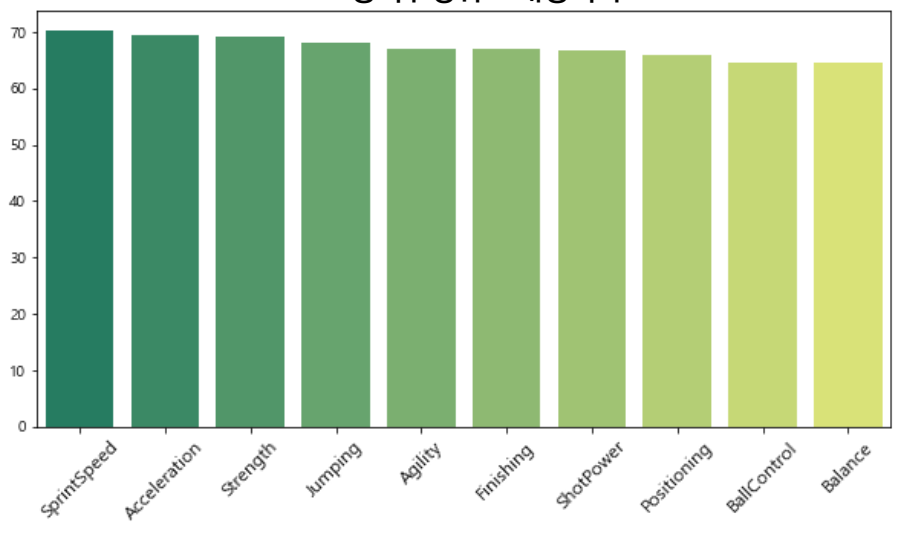
데이터 시각화

3) 포지션 - 능력치

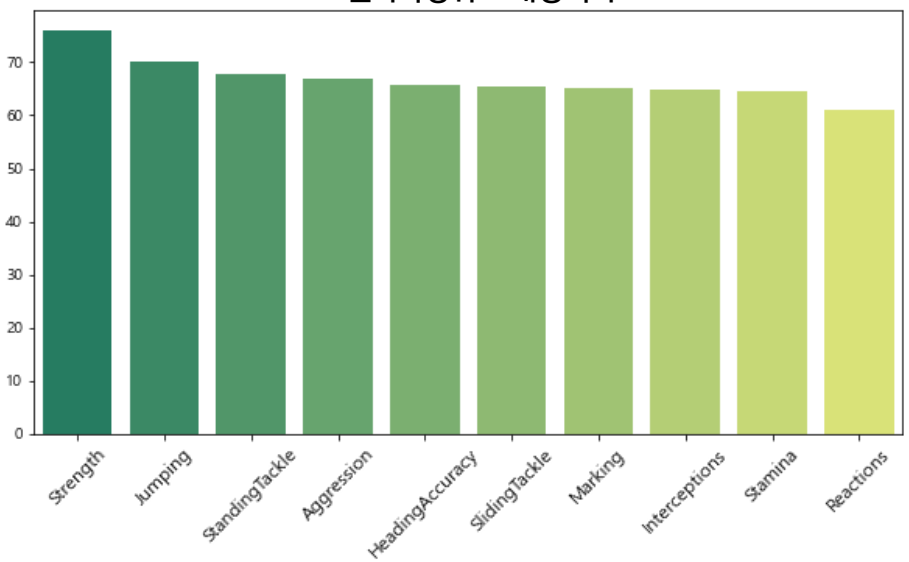
골키퍼상위 10개능력치



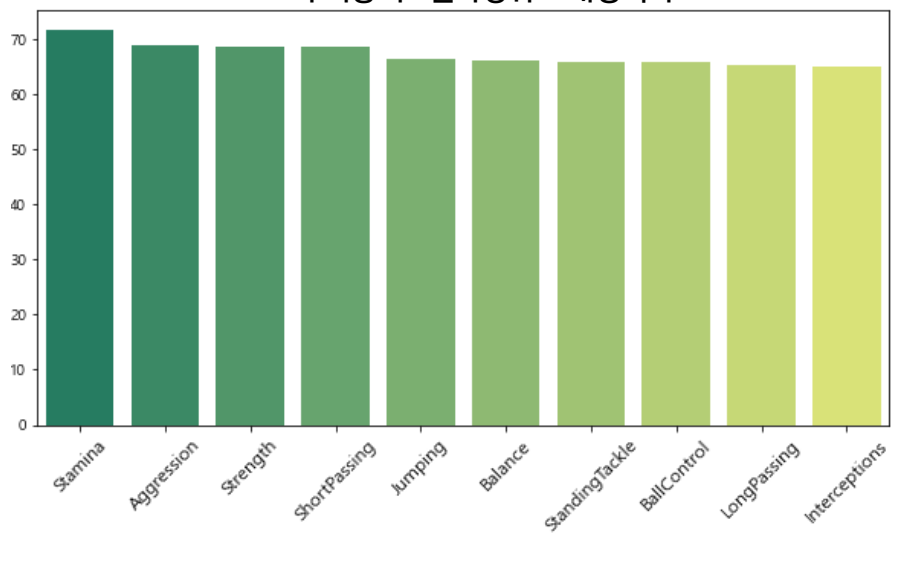
공격수상위 10개능력치



센터백상위 10개능력치



수비형미드필더상위 10개능력치

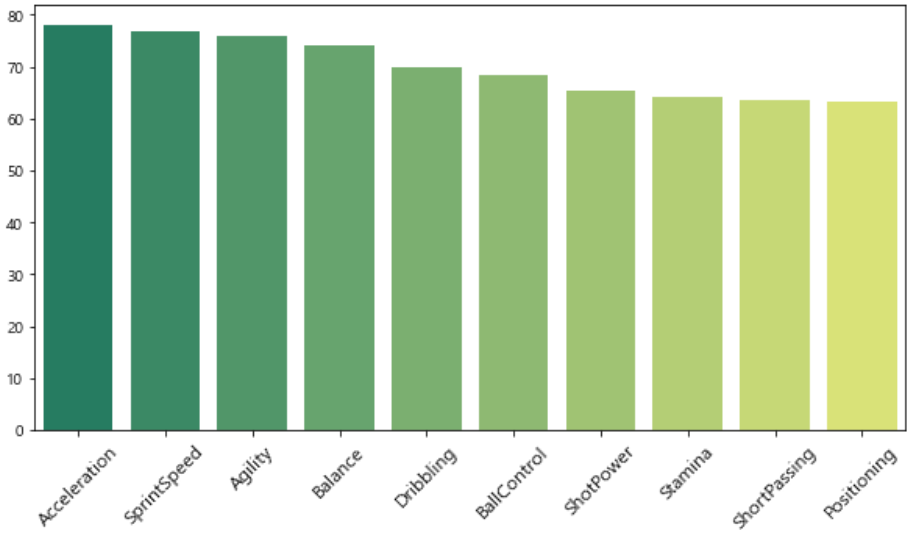


- 공격수는 속도, 힘, 민첩 관련 능력치가 높게 나옴
- 수비수는 힘, 태클 관련 능력치가 높게 나옴

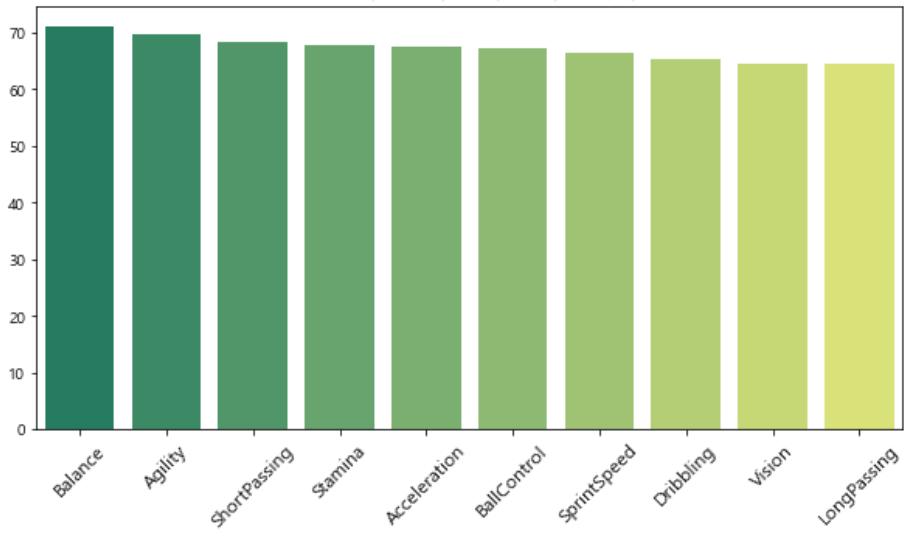
데이터 시각화

3) 포지션 - 능력치

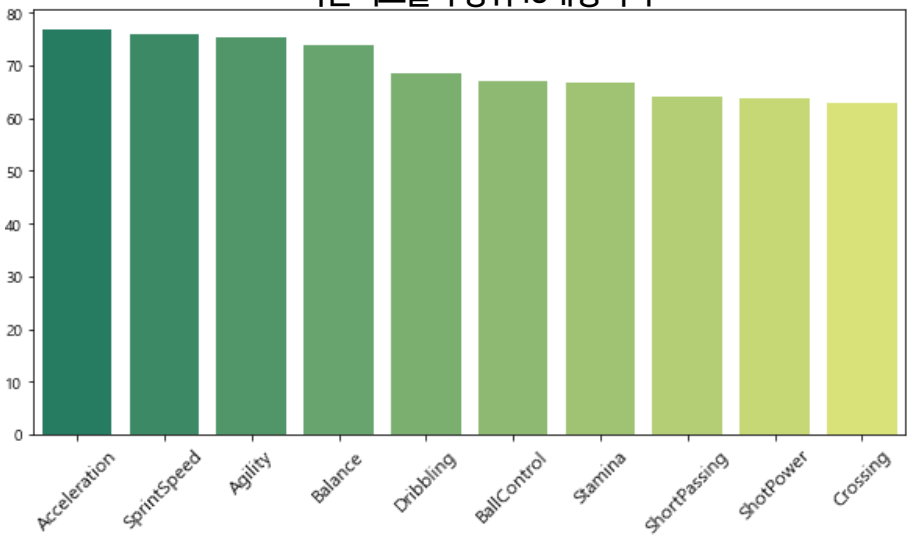
윙어상위 10개능력치



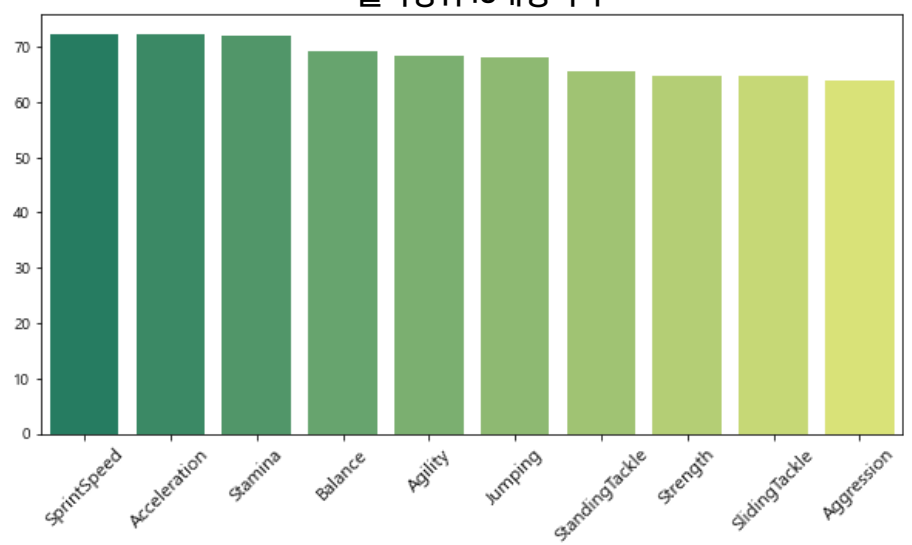
중앙미드필더 상위 10개능력치



측면미드필더 상위 10개능력치



풀백상위 10개능력치



- 중앙 포지션에서는 체력, 패스, 힘 관련 능력치가 높게 나옴
- 측면 포지션에서 속도, 민첩 관련 능력치가 높게 나옴

3) 포지션 - 능력치

✓ 포지션별 평균값 상위 8개 능력치 비교

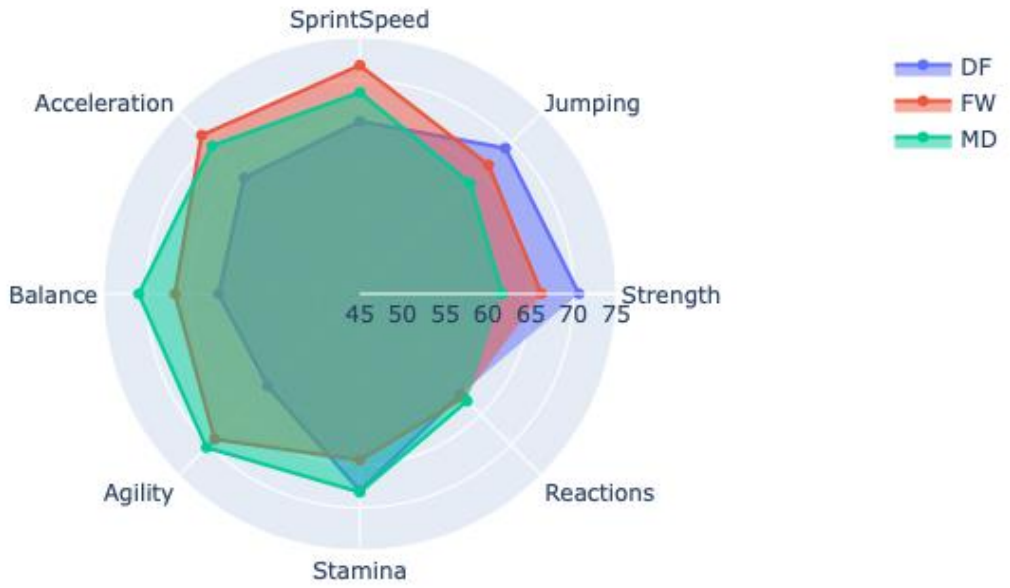
```
top_ability_index = data.loc[:, 'Crossing':'GKReflexes'].mean().sort_values(ascending=False)[:8].index

group_position = data.groupby('Position simplified')[top_ability_index].mean()
group_position = group_position.reindex(['DF','FW','MD','GK']).reset_index()

group_position
```

	Position simplified	Strength	Jumping	SprintSpeed	Acceleration	Balance	Agility	Stamina	Reactions
0	DF	70.643028	69.085578	65.173202	64.174736	61.536652	60.258609	68.038186	61.616434
1	FW	66.261849	66.301638	71.716501	71.214160	66.656232	69.050322	64.432417	62.168812
2	MD	61.676075	63.248464	68.552501	69.475285	70.880521	70.409330	68.185873	62.727991
3	GK	60.571429	57.727541	38.871870	38.486009	43.241532	40.359352	30.640157	58.920962

데이터 시각화

3) 포지션 - 능력치

```
import plotly.express as px
import plotly.graph_objects as go

fig = go.Figure()

fig.add_trace(go.Scatterpolar(r=group_position.iloc[0, 1:],
                              theta=top_ability_index,
                              fill='toself',
                              name='DF'))

fig.add_trace(go.Scatterpolar(r=group_position.iloc[1, 1:],
                              theta=top_ability_index,
                              fill='toself',
                              name='FW'))

fig.add_trace(go.Scatterpolar(r=group_position.iloc[2, 1:],
                              theta=top_ability_index,
                              fill='toself',
                              name='MD'))

fig.update_layout(
    title='포지션 별 능력치 비교',
    polar=dict(
    radialaxis=dict(
        visible=True,
        range=[45, 75]
    )),
)

fig.show()
```



데이터 시각화

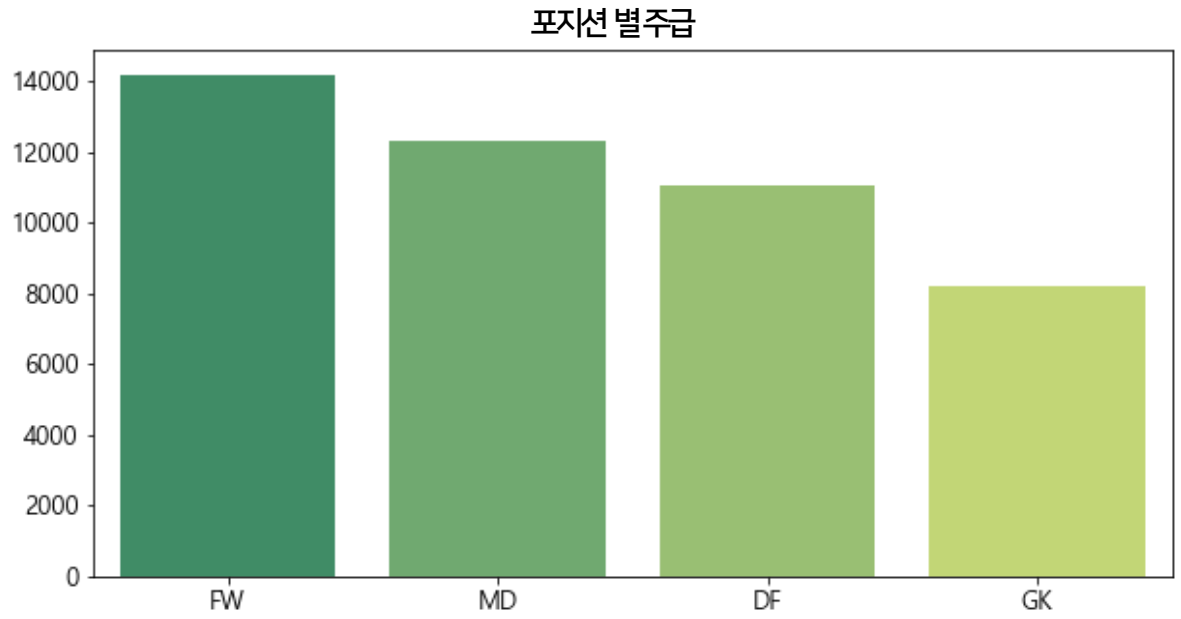
3) 포지션 - 주급

✓ 포지션 별 주급

```
group_position_wage = data.groupby('Position simplified')['Wage'].mean().reset_index()
group_position_wage.sort_values(by='Wage', ascending=False, inplace=True)
print(group_position_wage)
```

	Position simplified	Wage
1	FW	14179.090111
3	MD	12318.536122
0	DF	11066.776338
2	GK	8184.280805

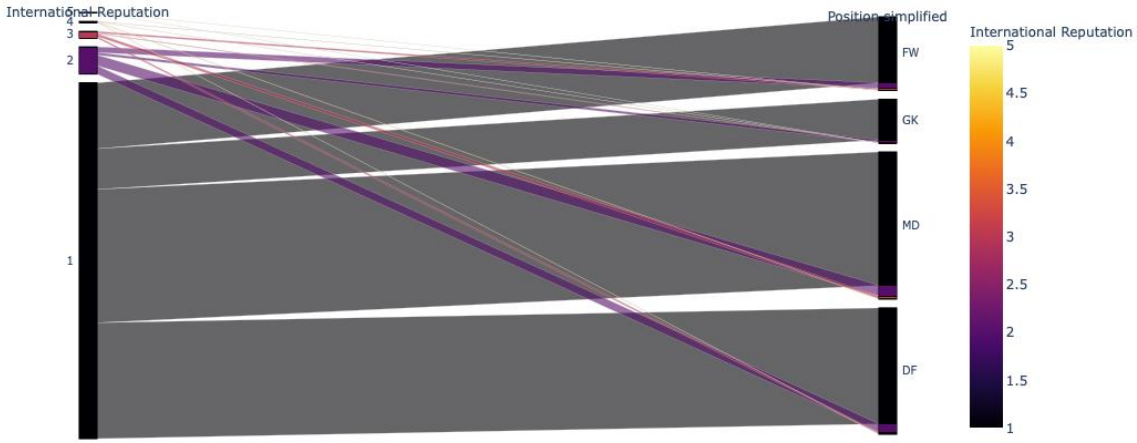
```
plt.figure(figsize=(10, 5))
plt.title('포지션 별 주급', fontsize=15)
plt.xticks(fontsize=13); plt.xlabel('Wage', fontsize=15)
plt.yticks(fontsize=13); plt.ylabel('Wage', fontsize=15)
sns.barplot(data=group_position_wage, x='Position simplified', y='Wage', palette='summer')
```



3) 포지션

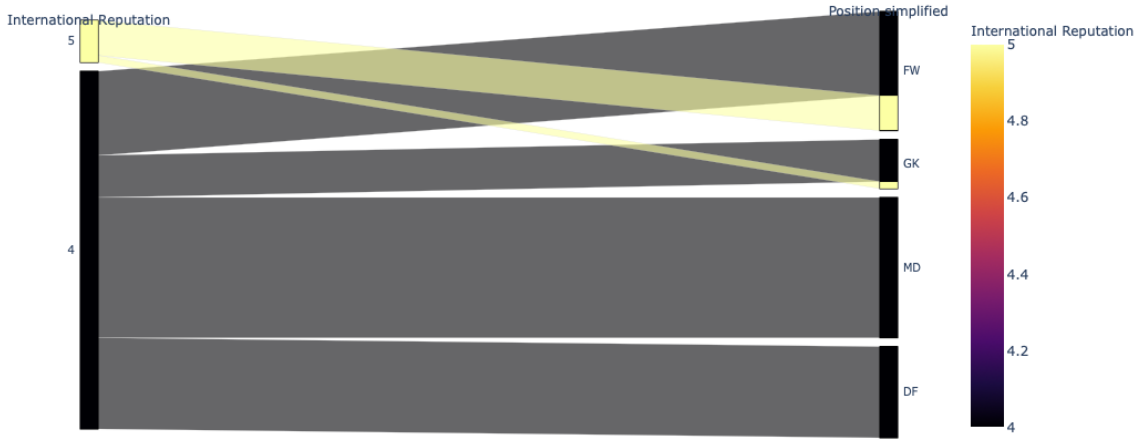
✓ 포지션 별 International Reputation ※ International Reputation은 1~5점으로 이뤄진 변수

```
fig=px.parallel_categories(data,
                           dimensions=['International Reputation', 'Position simplified'],
                           color='International Reputation',
                           color_continuous_scale=px.colors.sequential.Inferno)
fig.show()
```



International Reputation가 4, 5인 데이터가 매우 작아 확인하기 어려움

```
fig=px.parallel_categories(data[data['International Reputation']>=4],
                           dimensions=['International Reputation', 'Position simplified'],
                           color='International Reputation',
                           color_continuous_scale=px.colors.sequential.Inferno)
fig.show()
```



International Reputation가 4, 5인 데이터만 데이터만 추출해 포지션의 분포 확인

IV. 분류 모델 모델링

분류 모델 모델링

"선수별 능력치로
포지션 분류가 가능할까? "

"어떤 분류 모델이
가장 예측 성능이 좋을까?"

다양한 분류모델들 중 기준에 부합하는 모델들을 선정하여

직접 최적 하이퍼 파라미터를 찾아 튜닝한 후

가장 정확도가 높은 모델을 우리의 최적 모델로 채택!

분류 모델 모델링

✓ 모델 선정 시 고려사항

SVM

SVM은 학습데이터를 비선형 매핑(Mapping)을 통해 고차원으로 변환 시켜 새로운 차원에서 최적의 의사결정 영역을 구해주는 모델임

=> SVM은 복잡한 비선형 의사결정 영역을 모형화 할 수 있기 주로 고차원 데이터에 활용함

VS

KNN

KNN은 데이터가 주어지면 그 이웃의 데이터를 살펴본 뒤 더 많은 데이터가 포함되어 있는 범주로 분류하는 모델임

=> 저차원일수록 데이터간 거리가 가깝기 때문에 주로 저차원 데이터에 활용함

<성능 비교(상대치)>

LightGBM

XGBoost

Gradient Boost

그동안 학습한 분류 Boosting 모델들 중에서 속도가 가장 빠르고 성능이 좋은 LightGBM을 우선적으로 채택

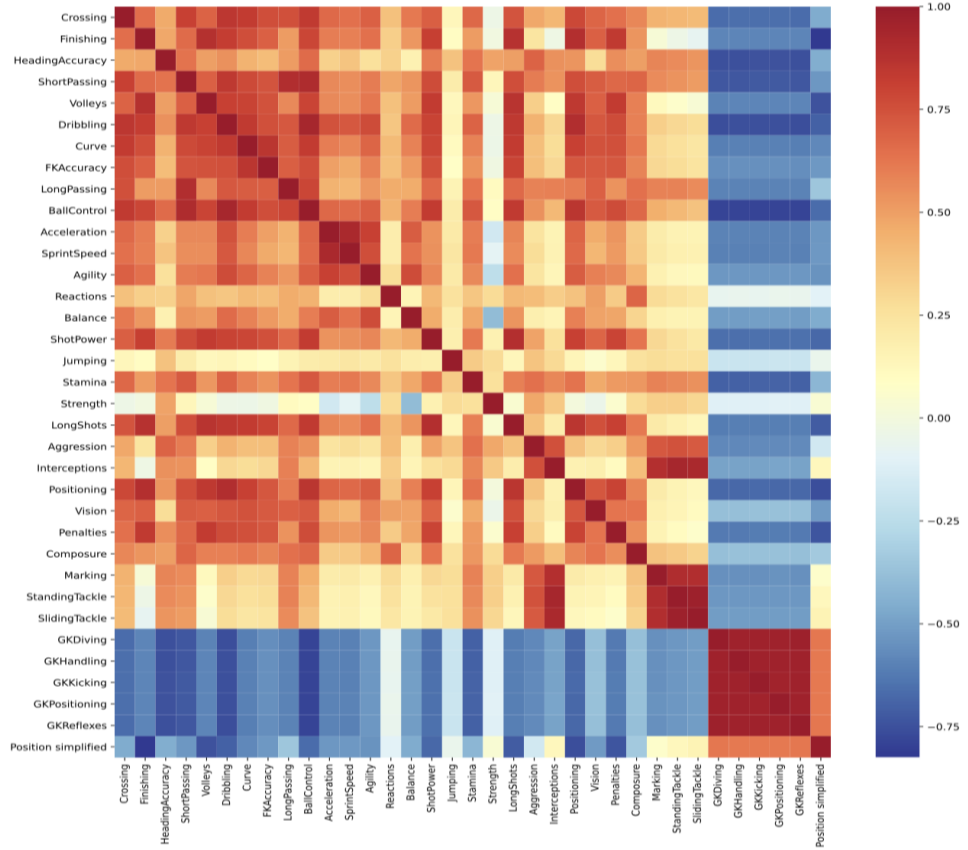
분류 모델 모델링

✓ 모델링을 위한 준비

1) Feature간의 상관관계 파악하기

```
X = data.loc[:, 'Crossing':'GKReflexes']
y = data.loc[:, 'Position simplified']
xy = pd.concat([X, y], axis=1)
```

```
import seaborn as sns
import matplotlib.pyplot as plt


plt.figure(figsize=(15, 15))
sns.heatmap(xy.corr(), cmap='RdYlBu_r')
```



- 피쳐 VS 피쳐 : Strength – Balance는 음의 상관관계
- 타켓 VS 피쳐 : GK 능력치를 제외하고 전반적으로 음의 상관관계

분류 모델 모델링

✓ 모델링을 위한 준비

2) Label Encoding

```
map_position = {'FW':0, 'MD':1, 'DF':2, 'GK':3}
col = ['Position simplified']
data[col] = data[col].applymap(map_position.get)
data['Position simplified']
```

```
0          0
1          0
2          0
3          3
4          1
          ..
18154      1
18155      0
18156      0
18157      0
18158      1
Name: Position simplified, Length: 18159, dtype: int64
```

3) 모델링을 위한 모듈 import 및 훈련/학습 데이터 분리

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
```

```
X=data.loc[:, 'Crossing':'GKReflexes']
y=data.loc[:, 'Position simplified']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0, stratify=y)
```

분류 모델 모델링

✓ 모델링을 위한 준비

4) 'get_model_train_eval' 함수 정의

```
from sklearn.metrics import classification_report, f1_score, confusion_matrix
```

```
def get_model_train_eval(model, X_train=None, X_test=None, y_train=None, y_test=None):
    model.fit(X_train, y_train)
    print('{} Test Accuracy: {}%'.format(model, round(model.score(X_test, y_test)*100, 2)))


    pred_model = model.predict(X_test)
    print('{} report:{}\n'.format(model.__class__.__name__, classification_report(y_test, pred_model)))
```

분류 모델 모델링

1. SVM

✓ Pipeline과 GridSearch 이용

```
import multiprocessing
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

pipe = make_pipeline(StandardScaler(), SVC(random_state=0))

param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]

param_grid = [
    {'svc__C': param_range, 'svc__gamma': param_range, 'svc__kernel': ['rbf']}
]

grid_model_svc = GridSearchCV(
    estimator=pipe,
    param_grid=param_grid,
    n_jobs=multiprocessing.cpu_count(),
    cv=3,
    verbose=True
)

grid_model_svc.fit(X_train, y_train)
```

```
grid_model_svc.best_estimator_
```

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('svc', SVC(C=10.0, gamma=0.01, random_state=0))])
```

최적의 파라미터로 예측하기

```
get_model_train_eval(grid_model_svc.best_estimator_, X_train, X_test, y_train, y_test)
```

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('svc', SVC(C=10.0, gamma=0.01, random_state=0))]) Test Accuracy: 88.49%
Pipeline report:               precision    recall  f1-score   support

           0       0.87      0.74      0.80       684
           1       0.83      0.89      0.86      1368
           2       0.92      0.93      0.93      1173
           3       1.00      0.99      0.99       407

    accuracy                           0.88      3632
   macro avg       0.91      0.89      0.89      3632
weighted avg       0.89      0.88      0.88      3632
```

Test Accuracy: 88.49%

분류 모델 모델링

2. LightGBM

```
from lightgbm import LGBMClassifier
lgbm_clf=LGBMClassifier()
```

✓ GridSearch 진행

```
parameters = {'n_estimators':[500], 'learning_rate':[0.01,0.05,0.1], 'objective':['multiclass']}
```

```
grid_lgbm=GridSearchCV(lgbm_clf, param_grid=parameters, cv=3, verbose=1)
grid_lgbm.fit(X_train, y_train)
print('GridSearchCV 최고 평균 정확도 수치: {:.6f}'.format(grid_lgbm.best_score_))
print('GridSearchCV 최적 하이퍼파라미터: ', grid_lgbm.best_params_)
```

```
Fitting 3 folds for each of 3 candidates, totalling 9 fits
GridSearchCV 최고 평균 정확도 수치: 0.881944
GridSearchCV 최적 하이퍼파라미터:  {'learning_rate': 0.01, 'n_estimators': 500, 'objective': 'multiclass'}
```

```
best_lgbm=grid_lgbm.best_estimator_
pred1=best_lgbm.predict(X_test)
accuracy=accuracy_score(y_test, pred1)
print('LGBM 예측 정확도: {0:.6f}'.format(accuracy))
```

LGBM 예측 정확도: 0.887115

분류 모델 모델링

3. Logistic Classifier

✓ Pipeline과 GridSearch 이용

```
pipe = make_pipeline(StandardScaler(), LogisticRegression(random_state=0))

param_grid = [
    {'logisticregression__C': [0.06, 0.08, 0.1],
    'logisticregression__intercept_scaling': [-90, -85, -80, -70],
    'logisticregression__max_iter': [64, 69, 74],
    'logisticregression__solver': ["lbfgs"],
    'logisticregression__multi_class' : ["multinomial"],
    'logisticregression__penalty' : ['l1', 'l2']
    }
]

grid_model_lr = GridSearchCV(
    estimator=pipe,
    param_grid=param_grid,
    n_jobs=multiprocessing.cpu_count(),
    cv=3,
    verbose=True
)

grid_model_lr.fit(X_train, y_train)
```

```
grid_model_lr.best_estimator_
```

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('logisticregression',
                 LogisticRegression(C=0.08, intercept_scaling=-90, max_iter=69,
                                    multi_class='multinomial',
                                    random_state=0))])
```

최적의파라미터로예측하기

```
get_model_train_eval(grid_model_lr, X_train, X_test, y_train, y_test)
```

```
Fitting 3 folds for each of 72 candidates, totalling 216 fits
[Parallel(n_jobs=8)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=8)]: Done  46 tasks      | elapsed:    7.1s
[Parallel(n_jobs=8)]: Done 200 tasks      | elapsed:   26.0s
[Parallel(n_jobs=8)]: Done 216 out of 216 | elapsed:   28.1s finished
GridSearchCV(cv=3,
             estimator=Pipeline(steps=[('standardscaler', StandardScaler()),
                                       ('logisticregression',
                                        LogisticRegression(random_state=0))]),
             n_jobs=8,
             param_grid=[{'logisticregression__C': [0.06, 0.08, 0.1],
                          'logisticregression__intercept_scaling': [-90, -85,
                                                                    -80, -70],
                          'logisticregression__max_iter': [64, 69, 74],
                          'logisticregression__multi_class': ['multinomial'],
                          'logisticregression__penalty': ['l1', 'l2'],
                          'logisticregression__solver': ['lbfgs']}],
             verbose=True) Test Accuracy: 88.13%
GridSearchCV report:               precision    recall  f1-score   support

           0       0.85      0.79      0.82       684
           1       0.84      0.86      0.85      1368
           2       0.91      0.92      0.91      1173
           3       1.00      0.99      0.99       407

    accuracy                           0.88      3632
   macro avg       0.90      0.89      0.89      3632
weighted avg       0.88      0.88      0.88      3632
```

Test Accuracy: 88.13%

분류 모델 모델링

4. RandomForest

```
from sklearn.ensemble import RandomForestClassifier
rt_clf = RandomForestClassifier()
```

✓ GridSearch 진행

```
parameters={'n_estimators':[500], 'max_depth':[8,10,12,14,16],
            'min_samples_split':[3,5,7,9,11], 'min_samples_leaf':[3,5,7,9,11]}
```

```
grid_rtclf=GridSearchCV(rt_clf, param_grid=parameters, scoring='accuracy', cv=3, n_jobs=-1)
grid_rtclf.fit(X_train, y_train)
print('GridSearchCV 최고 평균 정확도 수치: {:.6f}'.format(grid_rtclf.best_score_))
print('GridSearchCV 최적 하이퍼파라미터: ', grid_rtclf.best_params_)
```

```
GridSearchCV 최고 평균 정확도 수치: 0.878708
GridSearchCV 최적 하이퍼파라미터:  {'max_depth': 16, 'min_samples_leaf': 3, 'min_samples_split': 5, 'n_estimators': 500}
```

```
best_rt_clf=grid_rtclf.best_estimator_
pred_rt=best_rt_clf.predict(X_test)
accuracy=accuracy_score(y_test, pred_rt)
print('RandomForest 예측 정확도: {0:.6f}'.format(accuracy))
```

RandomForest 예측 정확도: 0.883811

5. GBM

```
from sklearn.ensemble import GradientBoostingClassifier
gb_clf=GradientBoostingClassifier()
```

✓ 학습 및 예측 진행

```
gb_clf.fit(X_train, y_train)
```

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

```
gb_pred=gb_clf.predict(X_test)
accuracy=accuracy_score(y_test, gb_pred)
```

```
print('GBM 정확도: {0:.6f}'.format(accuracy))
```

GBM 정확도: 0.886839

분류 모델 모델링

6. XGBoost

✓ Pipeline과 GridSearch 이용

```
import multiprocessing
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier

pipe = make_pipeline(StandardScaler(), XGBClassifier())

param_grid = [
    {'xgbclassifier__n_estimators': [1000],
    'xgbclassifier__learning_rate': [0.01, 0.05, 0.1, 0.15, 0.2],
    'xgbclassifier__max_depth': [3, 6, 9],
    'xgbclassifier__min_child_weight': [1, 3, 5],
    'xgbclassifier__objective': ['multi:softmax'],
    'xgbclassifier__gamma': [0, 0.1, 0.2, 0.3, 0.4, 0.5] }
]

grid_model_xgb = GridSearchCV(
    estimator=pipe,
    param_grid=param_grid,
    n_jobs=multiprocessing.cpu_count(),
    cv=3,
    verbose=True
)

grid_model_xgb.fit(X_train, y_train)
```

```
grid_model_xgb.best_estimator_
```

최적의 파라미터로 예측하기

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('xgbclassifier',
                 XGBClassifier(base_score=0.5, booster='gbtree',
                               colsample_bylevel=1, colsample_bynode=1,
                               colsample_bytree=1, gamma=0.3, gpu_id=-1,
                               importance_type='gain',
                               interaction_constraints='', learning_rate=0.05,
                               max_delta_step=0, max_depth=9,
                               min_child_weight=1, missing=nan,
                               monotone_constraints='()', n_estimators=1000,
                               n_jobs=8, num_parallel_tree=1,
                               objective='multi:softprob', random_state=0,
                               reg_alpha=0, reg_lambda=1, scale_pos_weight=None,
                               subsample=1, tree_method='exact',
                               validate_parameters=1, verbosity=None))])
```

```
get_model_train_eval(grid_model_xgb, X_train, X_test, y_train, y_test)
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits
                                    중    략
                verbose=True) Test Accuracy: 88.3%
GridSearchCV report:                precision    recall  f1-score   support

           0       0.85      0.76      0.80       684
           1       0.84      0.87      0.85      1368
           2       0.92      0.93      0.93      1173
           3       1.00      0.99      0.99       407

    accuracy                           0.88      3632
   macro avg       0.90      0.89      0.89      3632
weighted avg       0.88      0.88      0.88      3632
```

Test Accuracy: 88.3%

분류 모델 모델링

7. 스태킹 (Stacking) – CV 기반 스태킹 진행

✓ 스태킹에 사용될 머신러닝 알고리즘 클래스 생성

```
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
import numpy as np


rf = RandomForestClassifier(max_depth= 20, min_samples_leaf= 6, min_samples_split= 8, n_estimators= 500)
xgb = XGBClassifier(base_score=0.5,
                                                booster='gbtree',
                                                colsample_bylevel=1,
                                                colsample_bynode=1,
                                                colsample_bytree=1,
                                                eval_metric='mlogloss',
                                                gamma=0.3, gpu_id=-1,
                                                importance_type='gain',
                                                interaction_constraints='',
                                                learning_rate=0.05,
                                                max_delta_step=0,
                                                max_depth=9,
                                                min_child_weight=1,
                                                missing=np.nan,
                                                monotone_constraints='()',
                                                n_estimators=1000,
                                                n_jobs=8,
                                                num_parallel_tree=1,
                                                objective='multi:softprob',
                                                random_state=0,
                                                reg_alpha=0, reg_lambda=1,
                                                scale_pos_weight=None,
                                                subsample=1,
                                                tree_method='exact',
                                                validate_parameters=1,
                                                verbosity=None)

lgbm = LGBMClassifier(learning_rate= 0.04, n_estimators= 1000, min_child_weight=30, objective='multiclass', num_iterations=150)
gbm = GradientBoostingClassifier()
svm = SVC(kernel='rbf', C=10.0, gamma=0.01, random_state=0)
```

분류 모델 모델링

7. 스태킹 (Stacking)

✓ 'get_stacking_base_datasets' 함수 정의

```python
from sklearn.model_selection import KFold
from sklearn.metrics import mean_absolute_error

def get_stacking_base_datasets(model, X_train_n, y_train_n, X_test_n, n_folds):
    kf = KFold(n_splits=n_folds, shuffle=False, random_state=0)
    train_fold_pred = np.zeros((X_train.shape[0], 1))
    test_pred = np.zeros((X_test_n.shape[0], n_folds))
    print(model.__class__.__name__ , 'model 시작')

    for folder_counter, (train_index, valid_index) in enumerate(kf.split(X_train_n)):
        print('\t 폴드 세트: ', folder_counter, '시작')
        X_tr = X_train_n.iloc[train_index]
        y_tr = y_train_n.iloc[train_index]
        X_te = X_train_n.iloc[valid_index]

        model.fit(X_tr, y_tr)
        train_fold_pred[valid_index, :] = model.predict(X_te).reshape(-1, 1)
        test_pred[:, folder_counter] = model.predict(X_test_n)

    test_pred_mean = np.mean(test_pred, axis=1).reshape(-1, 1)

    return train_fold_pred, test_pred_mean
```

```python
rf_train, rf_test = get_stacking_base_datasets(rf, X_train, y_train, X_test, 3)
xgb_train, xgb_test = get_stacking_base_datasets(xgb, X_train, y_train, X_test, 3)
lgbm_train, lgbm_test = get_stacking_base_datasets(lgbm,  X_train, y_train, X_test, 3)
gbm_train, gbm_test = get_stacking_base_datasets(gbm, X_train, y_train, X_test, 3)
svm_train, svm_test = get_stacking_base_datasets(svm, X_train, y_train, X_test, 3)
```

```
RandomForestClassifier model 시작
         폴드 세트:  0 시작
         폴드 세트:  1 시작
         폴드 세트:  2 시작
XGBClassifier model 시작
         폴드 세트:  0 시작
         폴드 세트:  1 시작
         폴드 세트:  2 시작
LGBMClassifier model 시작
         폴드 세트:  0 시작
         폴드 세트:  1 시작
         폴드 세트:  2 시작
GradientBoostingClassifier model 시작
         폴드 세트:  0 시작
         폴드 세트:  1 시작
         폴드 세트:  2 시작
SVC model 시작
         폴드 세트:  0 시작
         폴드 세트:  1 시작
         폴드 세트:  2 시작
```

분류 모델 모델링

7. 스태킹 (Stacking)

✓ 각 모델별 학습 데이터와 테스트 데이터 결합 후 정확도 확인

```
Stack1_final_X_train = np.concatenate((xgb_train, lgbm_train, gbm_train, svm_train), axis=1)
Stack2_final_X_train = np.concatenate((rf_train, lgbm_train, gbm_train, svm_train),  axis=1)
Stack3_final_X_train = np.concatenate((rf_train, xgb_train, gbm_train, svm_train),  axis=1)
Stack4_final_X_train = np.concatenate((rf_train, xgb_train, lgbm_train, svm_train),  axis=1)
Stack5_final_X_train = np.concatenate((rf_train, xgb_train, lgbm_train, gbm_train),  axis=1)
```

```
Stack1_final_X_test = np.concatenate((xgb_test, lgbm_test, gbm_test, svm_test), axis=1)
Stack2_final_X_test = np.concatenate((rf_test, lgbm_test, gbm_test, svm_test), axis=1)
Stack3_final_X_test = np.concatenate((rf_test, xgb_test, gbm_test, svm_test), axis=1)
Stack4_final_X_test = np.concatenate((rf_test, xgb_test, lgbm_test, svm_test), axis=1)
Stack5_final_X_test = np.concatenate((rf_test, xgb_test, lgbm_test, gbm_test), axis=1)
```

```
from sklearn.metrics import accuracy_score

rf.fit(Stack1_final_X_train, y_train)
xgb.fit(Stack2_final_X_train, y_train)
lgbm.fit(Stack3_final_X_train, y_train)
gbm.fit(Stack4_final_X_train, y_train)
svm.fit(Stack5_final_X_train, y_train)

stack1_final = rf.predict(Stack1_final_X_test)
stack2_final = xgb.predict(Stack2_final_X_test)
stack3_final = lgbm.predict(Stack3_final_X_test)
stack4_final = gbm.predict(Stack4_final_X_test)
stack5_final = svm.predict(Stack5_final_X_test)

#랜덤포레스트
print('최종 메타 모델의 예측 정확도: {0:.4f}'.format(accuracy_score(y_test, stack1_final)))
#XGBoost
print('최종 메타 모델의 예측 정확도: {0:.4f}'.format(accuracy_score(y_test, stack2_final)))
#LGBM
print('최종 메타 모델의 예측 정확도: {0:.4f}'.format(accuracy_score(y_test, stack3_final)))
#GBM
print('최종 메타 모델의 예측 정확도: {0:.4f}'.format(accuracy_score(y_test, stack4_final)))
#SVM
print('최종 메타 모델의 예측 정확도: {0:.4f}'.format(accuracy_score(y_test, stack5_final)))
```

```
최종 메타 모델의 예측 정확도: 0.8835
최종 메타 모델의 예측 정확도: 0.8852
최종 메타 모델의 예측 정확도: 0.8791
최종 메타 모델의 예측 정확도: 0.8841
최종 메타 모델의 예측 정확도: 0.8844
```

RF
XGBoost
LGBM
GBM
SVM

✓ 최종 선택 모델은?

하이퍼 파라미터

learning_rate : 0.01
n_estimator : 500
objective : multiclass

정확도가 0.887115 인

LightGBM

분류 모델 모델링

8. AutoML

✓ 초기 세팅 (Initialize Setup)

```
from pycaret.classification import *
```

```
XX=data.loc[:, 'Crossing':'GKReflexes']
yy=data.loc[:, 'Position simplified']
```

```
data_auto=pd.concat([XX, yy], axis=1)
```

```
clf=setup(data=data_auto, target='Position simplified', train_size=0.8, session_id=0)
```

	Description	Value
0	session_id	0
1	Target	Position simplified
2	Target Type	Multiclass
3	Label Encoded	DF: 0, FW: 1, GK: 2, MD: 3
4	Original Data	(18159, 35)
5	Missing Values	False
6	Numeric Features	34
7	Categorical Features	0
8	Ordinal Features	False
9	High Cardinality Features	False
10	High Cardinality Method	None
	중 략	
47	Polynomial Degree	None
48	Trignometry Features	False
49	Polynomial Threshold	None
50	Group Features	False
51	Feature Selection	False
52	Features Selection Threshold	None
53	Feature Interaction	False
54	Feature Ratio	False
55	Interaction Threshold	None
56	Fix Imbalance	False
57	Fix Imbalance Method	SMOTE

분류 모델 모델링

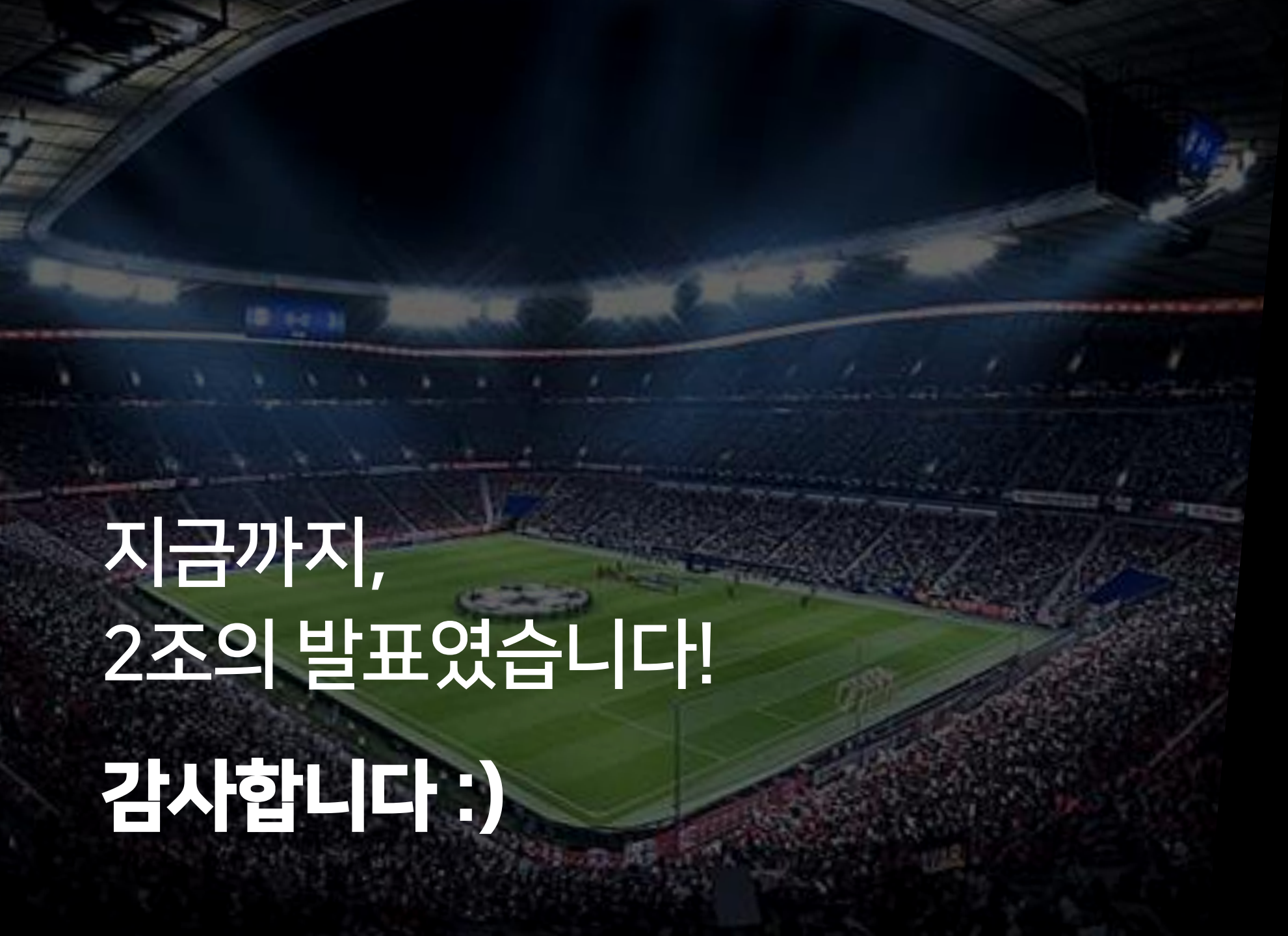
8. AutoML

✓ 모델 비교 (Compare Baseline)

```
best4models=compare_models(sort='Accuracy', n_select=4, fold=3)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
catboost	CatBoost Classifier	0.8849	0.973	0.8913	0.8849	0.8844	0.8364	0.8367	7.317
gbc	Gradient Boosting Classifier	0.8837	0.972	0.892	0.8835	0.8833	0.8348	0.835	4.49
lightgbm	Light Gradient Boosting Machine	0.8833	0.9723	0.8905	0.8831	0.8828	0.8343	0.8345	1.233
xgboost	Extreme Gradient Boosting	0.8826	0.9715	0.89	0.8823	0.8821	0.8332	0.8334	3.277
rf	Random Forest Classifier	0.8792	0.9704	0.8877	0.8791	0.8786	0.8283	0.8286	0.4833
et	Extra Trees Classifier	0.8789	0.9704	0.8857	0.8792	0.8783	0.8277	0.8282	0.36
lr	Logistic Regression	0.8785	0.9706	0.8889	0.8785	0.8783	0.8276	0.8277	2.083
lda	Linear Discriminant Analysis	0.8681	0.9655	0.883	0.8686	0.8682	0.8132	0.8133	0.07
ridge	Ridge Classifier	0.8665	0	0.8792	0.867	0.8665	0.8105	0.8107	0.0567
knn	K Neighbors Classifier	0.8623	0.9517	0.8739	0.863	0.8623	0.8044	0.8047	1.403
qda	Quadratic Discriminant Analysis	0.8555	0.9641	0.8777	0.8559	0.8549	0.7964	0.7971	0.07
svm	SVM - Linear Kernel	0.8528	0	0.8632	0.8666	0.8536	0.7899	0.7954	0.1933
dt	Decision Tree Classifier	0.8111	0.8613	0.8353	0.8113	0.8112	0.7328	0.7328	0.6533
nb	Naive Bayes	0.7859	0.9309	0.8323	0.7876	0.785	0.6998	0.701	0.0733
ada	Ada Boost Classifier	0.6733	0.8002	0.7547	0.6964	0.6512	0.5515	0.5724	0.35

GBM > LightGBM > XGBoost > RandomForest



지금까지,
2조의 발표였습니다!

감사합니다 :)