스포츠경기분석과 데이터사이언스

강지연

01 변수

값에 이름표를 붙인 것 = 변수

`home_goals = 4`

이름(home_goals)에 값(4)을 저장 · 저장된 값은 계산 가능 → home − away = **4 − 2 = 2**

값의 종류 = 자료형

INT	FLOAT	STR	BOOL
정수	**소수**	**글자**	**참 / 거짓**
4	7.5	"서울"	True

type()으로 자료형 확인 - 종류가 맞아야 계산·비교 가능 (숫자끼리 빼기, 글자끼리 잇기)

여러 값을 순서대로 담는 묶음 = 리스트

```
ratings = [6.8, 7.6, 7.4, 8.1, 7.0]  # 선수 5명의 평점
```

[0]	[1]	[2]	[3]	[4] = [-1]
6.8	7.6	7.4	8.1	7.0

인덱스는 **0부터** · 음수는 뒤에서부터

`len(ratings)`

개수 → **5**

`sum(ratings)`

합 → **36.9**

`max(ratings)`

최댓값 → **8.1**

개수·합·최댓값을 **함수 한 줄**로 - 손으로 셀 필요 없음

03 딕셔너리

이름표(key)로 값을 꺼내는 짝 구조 = 딕셔너리

```
roster = {7: "린가드", 9: "천성훈", 16: "최준"}
```

KEY · 등번호	KEY · 등번호	KEY · 등번호
7	9	16
린가드	**천성훈**	최준

`roster[9]` → **"천성훈"** · key로 즉시 조회

순서(번호) 대신 **이름으로 바로 조회** - 등번호→선수, 팀→점수 같은 '짝' 데이터에 적합

04 복합 자료형 정리

여러 값을 담는 4가지 그릇 - 기준은 순서 · 수정 · 중복

구분	기호	예시	특징	쓰임새
리스트	[ ]	`[6.8, 7.6, 8.1]`	**순서 있음 · 수정 가능**	평점 · 기록의 나열
튜플	( )	`(52.5, 34.0)`	순서 있음 · 수정 불가	좌표처럼 고정된 짝
딕셔너리	{ key: value }	`{7: "린가드"}`	**key로 조회**	등번호 → 선수 같은 '짝'
집합	{ }	`{"서울", "강원"}`	중복 없음 · 순서 없음	고유값 추리기

경기 데이터 분석은 **리스트와 딕셔너리**로 대부분 해결 - 튜플 · 집합은 만나면 알아볼 정도면 충분

05 조건문 · if

조건이 True일 때만 실행 = if

```
if xg >= 0.3:
    print("빅찬스")
```

1 조건식 검사

xg >= 0.3

결과는 항상 True 또는 False

→

2 TRUE

들여쓴 블록 실행

"빅찬스" 출력

2 FALSE

블록 건너뜀

아무 일도 일어나지 않음

==	!=	> · >=	< · <=
같다	다르다	크다 · 이상	작다 · 이하

문법은 **콜론(:)과 들여쓰기 4칸**이 전부 - 들여쓴 줄까지가 if에 속한 블록

여러 갈래로 나누기 = elif · else

```
if r >= 8.0:      "최상"
elif r >= 7.5:    "좋음"
else:             "보통"
```

r	r >= 8.0	r >= 7.5	결과
8.1	**True**	검사 안 함	**최상**
7.6	False	**True**	**좋음**
6.8	False	False	보통 (else)

위에서부터 차례로 검사 - **처음 True인 곳 하나만** 실행하고 종료

else = 위 조건이 모두 False인 나머지 경우 담당

조건의 **순서가 결과를 좌우** - 넓은 조건을 먼저 쓰면 아래 조건은 실행 기회 없음

묶음을 하나씩 꺼내 같은 처리 반복 = for

```
for r in ratings:
    print(r)
```

1회차	2회차	3회차	4회차	5회차
r = 6.8	r = 7.6	r = 7.4	r = 8.1	r = 7.0

반복 변수 **r** - 매회 다음 값으로 교체 · 들여쓴 블록이 매번 실행

range(5) → 0~4 숫자 반복 · 리스트가 없어도 횟수 반복 가능

5명이든 1,600건이든 **같은 규칙을 자동 반복** - 파이썬이 손보다 빠른 이유

반복 안에서 조건으로 판단 = for × if

```
for r in ratings:
    if r >= 7.5:  "좋음"
    else:         "보통"
```

6.8	7.6	7.4	8.1	7.0
보통	**좋음**	보통	**좋음**	보통

값마다 조건으로 **다른 결과** 산출 - 판정 · 분류의 기본 패턴

pandas의 **필터링 · 파생변수**가 내부에서 하는 일의 원형

경기 데이터 = 행 × 열의 표, DataFrame

	minute	team	event	xg
0	31	서울	슈팅	0.42
1	45	강원	패스	–
2	52	서울	태클	–
3	67	서울	슈팅	0.55

열(column) - 같은 종류의 값이 세로로

행(row) - 한 줄 = 한 이벤트

셀(cell) - 하나 = 한 값

인덱스(index) - 맨 왼쪽 번호 = 행의 주소

모든 pandas 작업 = **열을 고르고 · 행을 거르고 · 묶는 것** - 이 표 그림을 머리에 넣고 시작

새 표를 받으면 4가지부터 확인

`events.shape`

크기

(행, 열) → (1600, 12)

`events.columns`

열 이름 목록

['minute', 'team', ...]

`events.head()`

미리보기

첫 5행을 눈으로 확인

`events.describe()`

요약통계

숫자 열의 평균 · 최소 · 최대 · 표준편차

크기 · 열 이름 · 미리보기 · 요약 4종 세트로 데이터의 지도를 먼저 그린 뒤 분석 시작

조건 → 참/거짓 → True인 행만 남기기

```
shots = events[ events["event"] == "슈팅" ]
```

1 비교 → 참/거짓

패스	False
슈팅	**True**
슈팅	**True**
태클	False
슈팅	**True**

→

2 True인 행만 선택

23 패스	–
31 슈팅	0.42
45 슈팅	0.08
52 태클	–
67 슈팅	0.55

→

3 새 표 반환

min	event	xg
31	슈팅	0.42
45	슈팅	0.08
67	슈팅	0.55

대괄호 [ ] 안의 조건이 각 행을 True/False로 채점 → **True인 행만 새 표로 반환**

조건 두 개를 & (그리고) 로 겹치기

```
events[ (team == "서울") & (event == "슈팅") ]
```

행	team == "서울"	event == "슈팅"	A & B	결과
서울 · 슈팅	**True**	**True**	**True**	**선택**
강원 · 슈팅	False	True	False	제외
서울 · 패스	True	False	False	제외

& = 그리고(교집합) · **|** = 또는(합집합, 하나만 True여도 통과)

각 조건은 반드시 **괄호 ( )** 로 감싸기

원리는 필터링과 동일 - **참/거짓 마스크를 만들어 겹치는 것**

10 집계함수(groupby)

같은 것끼리 나누고 → 집계 → 합치기

```
events.groupby("team").size()  # split · apply · combine
```

1 팀 기준으로 구분

서울	슈팅
강원	패스
서울	태클
강원	슈팅
서울	패스

→

2 그룹별로 분리

서울 그룹

슈팅 · 태클 · 패스 (3건)

강원 그룹

패스 · 슈팅 (2건)

→

3 집계해 결합

team	count
서울	**3**
강원	**2**

size() 개수 · mean() 평균 · sum() 합

groupby = **나누고(split) · 함수 적용(apply) · 합치기(combine)** - 선수별·팀별 통계의 출발점

10 다중 집계(groupby/agg)

한 그룹에 여러 함수를 한 번에 = agg

```
passes.groupby("player")["pass_length"].agg(["count", "mean", "max"])
```

'린가드'의 패스 거리 (m)

12	30	8	25

→

count 개수	mean 평균	max 최댓값
4	18.8	30

세 결과가 **한 줄(한 선수)**로 정리

개수·합·평균·최대를 동시에 - **선수당 여러 지표를 한 표로**

기존 열을 계산해 새 열 만들기

```
p["forward"] = p["end_x"] - p["x"]  # 전진 거리
```

player	x (시작)	end_x (끝)
A	30	60
B	55	70
C	20	18

→ `end_x − x`

player	forward (새 열)
A	**+30**
B	**+15**
C	**−2**

양수 = 전진 패스 · **음수** = 뒤로 돌린 패스

원본에 없던 지표도 열끼리 계산으로 추가 - **copy()**로 원본 보존 후 작업

12 정렬

기준 열로 줄 세우기 = sort_values

```
p.sort_values("forward", ascending=False)  # 큰 값이 위로
```

정렬 전

player	forward
A	30
B	15
C	-2
D	42

→

정렬 후 · 내림차순

player	forward
D	**42**
A	30
B	15
C	-2

맨 위 = **가장 전진한 패스**

ascending=True → 오름차순

.head() 와 함께 쓰면 TOP N

값은 그대로, **행의 순서만 재배열** - '가장 ~한 선수/장면' 탐색에 필수

13 범주별 개수 세기

값 종류별 개수 세기 = value_counts()

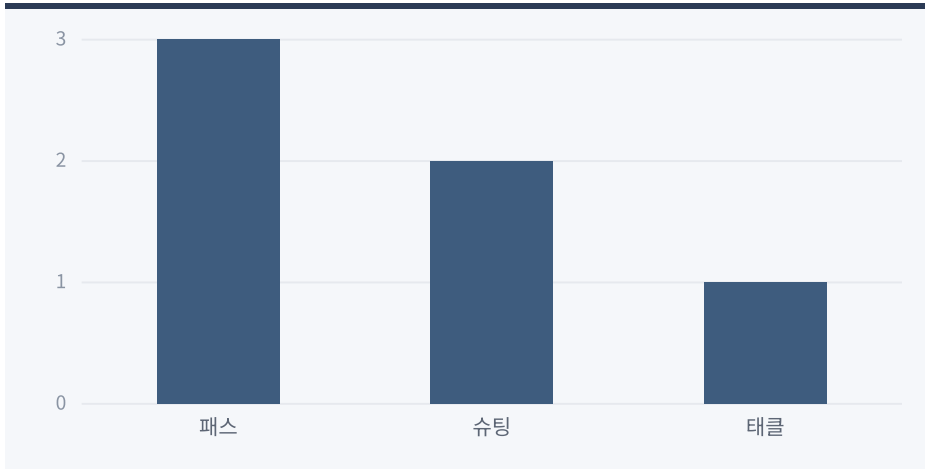
```
events["event"].value_counts()
```

1 한 열의 값들

패스
슈팅
패스
태클
패스
슈팅

→

2 종류별로 세어 정렬



많은 순으로 **자동 정렬** - '어떤 이벤트가 가장 많은가'에 한 줄로 답

14 시각화 기초(막대그래프)

집계 결과를 막대그래프로

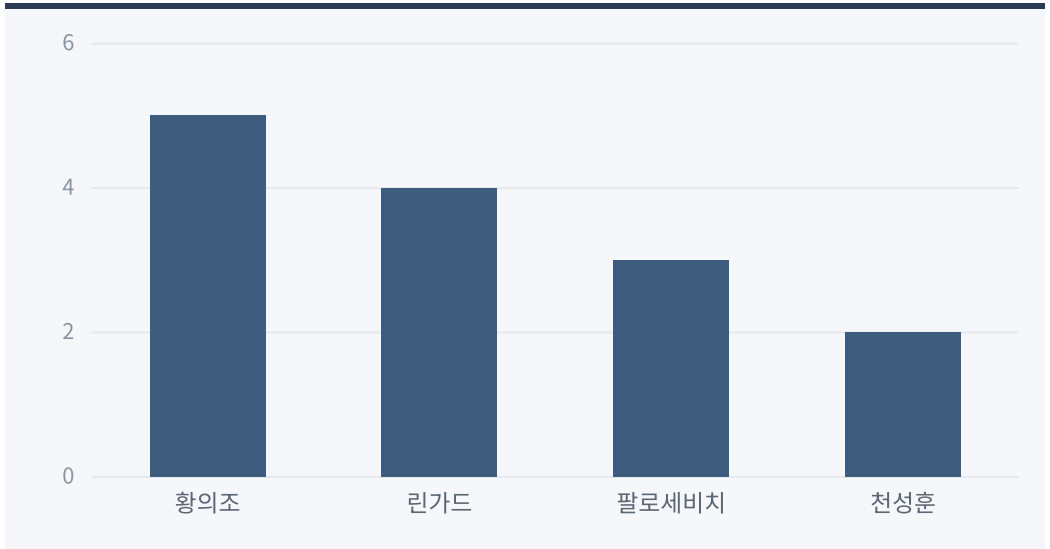
```
top = shots.groupby("player").size() ... → plot(kind="bar")
```

1 선수별 슛 수 표

player	shots
황의조	5
린가드	4
팔로세비치	3
천성훈	2

→

2 한눈에 보이는 순위



표의 숫자보다 **막대 길이가 즉시 비교됨** - 집계 → 그림이 마지막 단계

슛 위치 산점도 - 점 = 위치 · 크기 = xG · 색 = 결과

```
plt.scatter(x, y, s=xg*800, c=결과색)
```

위치 (x, y)

점의 자리

어디서 일어났는가

크기 (s)

기대득점 xG

클수록 좋은 찬스

색 (c)

슛 결과

골 · 유효슛 · 빗나감/막힘

숫자 세 개를 그림 하나에 - **어디서, 얼마나 좋은 슛**인지 즉시 파악