# Ch03. 프로 야구 데이터를 통해 데이터프레임 다루기

> KBO 데이터를 이용해 이대호 선수의 일별 누적 장타율 구하기

## 1. 웹크롤링 결과를 데이터프레임으로 불러오기 (read_html)

- KBO 사이트의 이대호 선수 일별기록

https://www.koreabaseball.com/Record/Player/HitterDetail/Daily.aspx?playerId=71564

**크롤링이란?**

- 웹페이지에서 데이터를 추출하는 행위
- 판다스에서는 read_html 함수로 가능하다

> pandas read_html

웹페이지의 테이블을 데이터프레임으로 불러오는 함수. 리스트에 데이터프레임을 담아준다.

---

**io**

웹페이지의 경로명

**match** (인수는 문자열 혹은 정규표현식 / 기본값은 '.+' )

테이블 중에서 특정 문자열을 포함한 테이블을 지정하는 인자. 기본값은 빈 문자열을 제외한 모든 문자열이다.(지정하지 않으면 문자열이 포함된 모든 테이블을 가져온다)

**header** (인수는 정수, 정수의 리스트 / 기본값은 None)

columns를 지정하는 인자. 리스트로 지정하면 멀티 인덱스인 columns가 된다.

**index_col** (인수는 정수, 정수의 리스트 / 기본값은 None)

index를 지정하는 인자. 지정하지 않으면 RangeIndex가 index로 부여된다. 리스트로 지정하면 멀티 인덱스인 index가 된다.

read_html 함수 설명 블로그

read_html 판다스 공식문서

```
# 강의 준비코드
import pandas as pd
pd.set_option('max_rows', 6)

# 해당 웹사이트의 모든 표를 데이터프레임으로 만들어 리스트에 담아준다
url = 'https://www.koreabaseball.com/Record/Player/HitterDetail/Daily.aspx?playerId=71564'
dfs = pd.read_html(url)
dfs
```

```
[      4월  상대   AVG1   PA  AB   R   H  2B  3B  HR  RBI  SB  CS  BB  HBP  SO  W
 0   04.02  키움  0.250    4   4   0   1   0   0   0    1   0   0   0    0   0
 1   04.03  키움  0.400    5   5   0   2   0   0   0    0   0   0   0    0   1
 2   04.05  NC  0.500    5   4   0   2   0   0   0    0   0   0   1    0   0
 ..    ...  ..    ...  ...  ..  ..  ..  ..  ..  ..  ...  ..  ..  ..  ...  ..
 22  04.29  LG  0.400    5   5   1   2   0   0   0    1   0   0   0    0   2
 23  04.30  LG  0.500    4   4   0   2   0   0   0    0   0   0   0    0   0
 24     합계  합계  0.356  100  90  13  32   2   0   2   10   0   0   7    1   9

     GDP   AVG2
 0     0  0.250
 1     0  0.333
 2     0  0.385
 ..  ...    ...
 22    0  0.349
 23    0  0.356
 24    4  0.356
```

```
[25 rows x 18 columns],
      5월  상대   AVG1  PA  AB  R   H  2B  3B  HR  RBI  SB  CS  BB  HBP  SO  W
0   05.03  KT  0.250   4   4  0   1   0   0   0    0   0   0   0    0   2
1   05.04  KT  0.000   4   4  0   0   0   0   0    0   0   0   0    0   0
2   05.05  KT  0.000   4   4  0   0   0   0   0    0   0   0   0    0   0
..    ...  ..    ...  ..  .. ..  ..  ..  ..  ..  ...  ..  ..  ..  ...  ..
22  05.29  키움  0.000   4   4  0   0   0   0   0    0   0   0   0    0   0
23  05.31  LG  0.500   4   4  1   2   1   0   0    0   0   0   0    0   0
24     합계  합계  0.355  98  93  8  33   4   0   4   13   0   0   5    0  11

    GDP   AVG2
0     0  0.351
1     1  0.337
2     0  0.324
..  ...    ...
22    0  0.352
23    0  0.355
24    2  0.355

[25 rows x 18 columns],
      6월  상대   AVG1  PA  AB  R   H  2B  3B  HR  RBI  SB  CS  BB  HBP  SO  W
0   06.01  LG  0.250   4   4  0   1   0   0   0    0   0   0   0    0   1
1   06.02  LG  0.250   6   4  0   1   0   0   0    0   0   0   2    0   0
2   06.03  NC  0.000   4   4  0   0   0   0   0    0   0   0   0    0   1
..    ...  ..    ...  ..  .. ..  ..  ..  ..  ..  ...  ..  ..  ..  ...  ..
21  06.28  두산  0.500   3   2  0   1   1   0   0    0   0   0   1    0   0
22  06.30  두산  0.250   4   4  0   1   1   0   0    1   0   0   0    0   1
23     합계  합계  0.341  96  88  6  30   7   0   3   18   0   0   6    0   9

    GDP   AVG2
0     0  0.353
1     0  0.351
2     0  0.344
..  ...    ...
21    0  0.352
22    0  0.351
23    4  0.351

[24 rows x 18 columns],
      7월  상대   AVG1  PA  AB  R   H  2B  3B  HR  RBI  SB  CS  BB  HBP  SO  W
```

# 데이터프레임의 리스트에서 첫번째 데이터프레임만 불러올 때
dfs[0]

| | 4월 | 상대 | AVG1 | PA | AB | R | H | 2B | 3B | HR | RBI | SB | CS | BB | HBP | SO | GDP | AVG2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 04.02 | 키움 | 0.250 | 4 | 4 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0.250 |
| **1** | 04.03 | 키움 | 0.400 | 5 | 5 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0.333 |
| **2** | 04.05 | NC | 0.500 | 5 | 4 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0.385 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **22** | 04.29 | LG | 0.400 | 5 | 5 | 1 | 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 2 | 0 | 0.349 |
| **23** | 04.30 | LG | 0.500 | 4 | 4 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.356 |
| **24** | 합계 | 합계 | 0.356 | 100 | 90 | 13 | 32 | 2 | 0 | 2 | 10 | 0 | 0 | 7 | 1 | 9 | 4 | 0.356 |

25 rows × 18 columns

# dfs의 모든 데이터프레임의 날짜 열의 이름이 달라 concat로는 outer join을 한다
pd.concat(dfs)

| | 4월 | 상대 | AVG1 | PA | AB | R | H | 2B | 3B | HR | ... | HBP | SO | GDP | AVG2 | 5월 | 6월 | 7월 | 8월 | 9월 | 10월 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 04.02 | 키움 | 0.25 | 4 | 4 | 0 | 1 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0.250 | NaN | NaN | NaN | NaN | NaN | NaN |
| **1** | 04.03 | 키움 | 0.4 | 5 | 5 | 0 | 2 | 0 | 0 | 0 | ... | 0 | 1 | 0 | 0.333 | NaN | NaN | NaN | NaN | NaN | NaN |
| **2** | 04.05 | NC | 0.5 | 5 | 4 | 0 | 2 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0.385 | NaN | NaN | NaN | NaN | NaN | NaN |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **2** | NaN | NC | 0.0 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 1 | 0 | 0.332 | NaN | NaN | NaN | NaN | NaN | 10.05 |
| **3** | NaN | LG | 0.25 | 4 | 4 | 0 | 1 | 1 | 0 | 0 | ... | 0 | 0 | 2 | 0.331 | NaN | NaN | NaN | NaN | NaN | 10.08 |
| **4** | NaN | 합계 | 0.143 | 16 | 14 | 1 | 2 | 1 | 0 | 1 | ... | 0 | 2 | 3 | 0.331 | NaN | NaN | NaN | NaN | NaN | 합계 |

149 rows × 24 columns

```
# 열의 이름이 다른 날짜에 해당 하는 열을 index로 만들어 dfs를 부르고 concat를 하자
dfs = pd.read_html(url, index_col=0)
pd.concat(dfs)
```

| | 상대 | AVG1 | PA | AB | R | H | 2B | 3B | HR | RBI | SB | CS | BB | HBP | SO | GDP | AVG2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **04.02** | 키움 | 0.25 | 4 | 4 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0.250 |
| **04.03** | 키움 | 0.4 | 5 | 5 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0.333 |
| **04.05** | NC | 0.5 | 5 | 4 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0.385 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **10.05** | NC | 0.0 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0.332 |
| **10.08** | LG | 0.25 | 4 | 4 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 0.331 |
| **합계** | 합계 | 0.143 | 16 | 14 | 1 | 2 | 1 | 0 | 1 | 4 | 0 | 0 | 2 | 0 | 2 | 3 | 0.331 |

149 rows × 17 columns

```
# 결과를 변수 df_ex1으로 지정하자 (메인 프로젝트 코드)
df_ex1 = pd.concat(dfs)
df_ex1
```

| | 상대 | AVG1 | PA | AB | R | H | 2B | 3B | HR | RBI | SB | CS | BB | HBP | SO | GDP | AVG2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **04.02** | 키움 | 0.25 | 4 | 4 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0.250 |
| **04.03** | 키움 | 0.4 | 5 | 5 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0.333 |
| **04.05** | NC | 0.5 | 5 | 4 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0.385 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **10.05** | NC | 0.0 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0.332 |
| **10.08** | LG | 0.25 | 4 | 4 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 0.331 |
| **합계** | 합계 | 0.143 | 16 | 14 | 1 | 2 | 1 | 0 | 1 | 4 | 0 | 0 | 2 | 0 | 2 | 3 | 0.331 |

149 rows × 17 columns

## 2. 데이터프레임의 정보 파악하기

**info : 데이터 프레임의 많은 정보를 알려주는 함수**
**shape : 데이터 프레임의 크기를 알려주는 속성**
**dtypes : 각 열의 dtype을 시리즈로 반환하는 속성**
**describe : 숫자로 된 열들의 간단한 통계들을 제공하는 함수**
**unique : 열의 고유값들을 반환하는 함수**

```
# 프로젝트 코드
import pandas as pd
pd.set_option('max_rows', 6)
pd.options.display.float_format = '{:.2f}'.format
url = 'https://www.koreabaseball.com/Record/Player/HitterDetail/Daily.aspx?playerId
dfs = pd.read_html(url, index_col=0)
df_ex1 = pd.concat(dfs)
df_ex1
```

| | 상대 | AVG1 | PA | AB | R | H | 2B | 3B | HR | RBI | SB | CS | BB | HBP | SO | GDP | AVG2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **04.02** | 키움 | 0.25 | 4 | 4 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0.25 |
| **04.03** | 키움 | 0.40 | 5 | 5 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0.33 |

```
# index 확인하기
df_ex1.index
```

```
Index(['04.02', '04.03', '04.05', '04.06', '04.07', '04.08', '04.09', '04.10',
       '04.12', '04.14',
       ...
       '09.23', '09.24', '09.29', '09.30', '합계', '10.02', '10.03', '10.05',
       '10.08', '합계'],
      dtype='object', length=149)
```

```
# columns 확인하기
df_ex1.columns
```

```
Index(['상대', 'AVG1', 'PA', 'AB', 'R', 'H', '2B', '3B', 'HR', 'RBI', 'SB', 'CS',
       'BB', 'HBP', 'SO', 'GDP', 'AVG2'],
      dtype='object')
```

```
df_ex1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 149 entries, 04.02 to 합계
Data columns (total 17 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   상대      149 non-null    object
 1   AVG1    149 non-null    object
 2   PA      149 non-null    int64
 3   AB      149 non-null    int64
 4   R       149 non-null    int64
 5   H       149 non-null    int64
 6   2B      149 non-null    int64
 7   3B      149 non-null    int64
 8   HR      149 non-null    int64
 9   RBI     149 non-null    int64
 10  SB      149 non-null    int64
 11  CS      149 non-null    int64
 12  BB      149 non-null    int64
 13  HBP     149 non-null    int64
 14  SO      149 non-null    int64
 15  GDP     149 non-null    int64
 16  AVG2    149 non-null    float64
dtypes: float64(1), int64(14), object(2)
memory usage: 21.0+ KB
```

```
# 데이터 프레임 크기 파악하기
df_ex1.shape
```

```
(149, 17)
```

```
# 데이터 프레임의 행의 수(주로 코드에 넣을 때)
df_ex1.shape[0]
```

```
149
```

```
# 데이터 프레임 각 열의 dtype
df_ex1.dtypes
```

```
상대       object
AVG1     object
PA        int64
         ...
SO        int64
GDP       int64
AVG2    float64
Length: 17, dtype: object
```

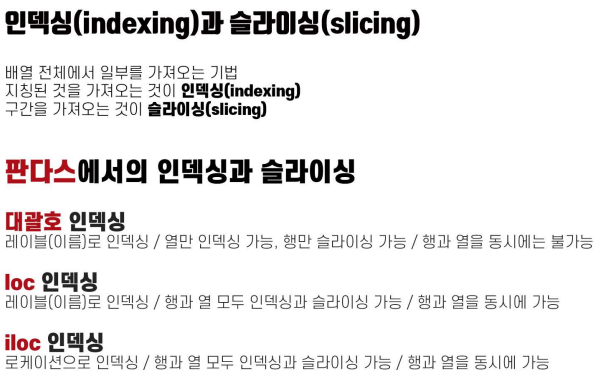
```
# 숫자 열들의 기본적인 통계자료
pd.set_option('max_rows', 8)
pd.options.display.float_format = '{:.2f}'.format
df_ex1.describe()
```

| | PA | AB | R | H | 2B | 3B | HR | RBI | SB | CS | BB | HBP | SO | GDP | AVG2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 149.00 | 149.00 | 149.00 | 149.00 | 149.00 | 149.00 | 149.00 | 149.00 | 149.00 | 149.00 | 149.00 | 149.00 | 149.00 | 149.00 | 149.00 |
| **mean** | 7.93 | 7.25 | 0.71 | 2.40 | 0.31 | 0.00 | 0.31 | 1.36 | 0.00 | 0.00 | 0.58 | 0.03 | 0.75 | 0.35 | 0.34 |
| **std** | 18.13 | 16.59 | 1.78 | 5.71 | 0.88 | 0.00 | 0.82 | 3.36 | 0.00 | 0.00 | 1.44 | 0.16 | 1.83 | 0.87 | 0.02 |
| **min** | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.24 |
| **25%** | 4.00 | 4.00 | 0.00 | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.33 |
| **50%** | 4.00 | 4.00 | 0.00 | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.34 |
| **75%** | 5.00 | 4.00 | 1.00 | 2.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 | 0.00 | 1.00 | 0.00 | 1.00 | 0.00 | 0.35 |
| **max** | 100.00 | 93.00 | 13.00 | 33.00 | 7.00 | 0.00 | 5.00 | 22.00 | 0.00 | 0.00 | 9.00 | 1.00 | 11.00 | 5.00 | 0.39 |

```
# 고유값
df_ex1['AB'].unique()
```

```
array([ 4,  5,  3,  1, 90, 93,  6,  0,  2, 88, 82, 85, 14])
```

## 3. 데이터프레임의 인덱싱과 슬라이싱 (loc와 iloc)




```
# 실습 준비코드
data = [[1, 3, 1, 0, 4, 1], [6, 5, 8, 9, 0, 3],[7, 6, 2, 3, 4, 8], [9, 0, 2, 0, 6, 5]]
df = pd.DataFrame(data, index=list('가나다라'), columns=list('ABCDEF'))
df
```

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| **가** | 1 | 3 | 1 | 0 | 4 | 1 |
| **나** | 6 | 5 | 8 | 9 | 0 | 3 |
| **다** | 7 | 6 | 2 | 3 | 4 | 8 |
| **라** | 9 | 0 | 2 | 0 | 6 | 5 |

```
# 대괄호 인덱싱 (각기 실행해보세요)
df['A'] # A열
df[['A', 'C']] # A열과 C열
df['가':'다'] # 가행부터 다행까지 슬라이싱
```

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 가 | 1 | 3 | 1 | 0 | 4 | 1 |
| 나 | 6 | 5 | 8 | 9 | 0 | 3 |
| 다 | 7 | 6 | 2 | 3 | 4 | 8 |

```
# loc 인덱싱 (각기 실행해보세요)
df.loc['가', 'A'] # 가 행의 A열
df.loc[['가', '나'], ['A', 'C']] # 가, 나 행과 A열과 C열
df.loc[['가', '나'], ['A', 'B', 'C']] # 가, 나 행과 A열과 B열과 C열 (인덱싱)
df.loc[['가', '나'], 'A':'C'] # 가, 나 행과 A열과 B열과 C열 (슬라이싱)
df.loc[['가', '나'], 'B':] # 가, 나 행과 B열에서 끝까지
df.loc['가':'다', :] # 가행부터 다행까지 열은 전부
df.loc['가':'다'] # 가행부터 다행까지 열은 전부
```

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 가 | 1 | 3 | 1 | 0 | 4 | 1 |
| 나 | 6 | 5 | 8 | 9 | 0 | 3 |
| 다 | 7 | 6 | 2 | 3 | 4 | 8 |

```
# iloc 인덱싱 (각기 실행해보세요)
df.iloc[0, :] # 첫번째 행
df.iloc[0] # 첫번째 행
df.iloc[:3, 2:] # 시작부터 다행까지 C열부터 끝까지
df.iloc[[0, 1], 2:] # 가행과 나행 C열부터 끝까지
```

| | C | D | E | F |
|---|---|---|---|---|
| 가 | 1 | 0 | 4 | 1 |
| 나 | 8 | 9 | 0 | 3 |

> pandas drop

데이터 프레임의 행이나 열을 삭제하는 함수

---

**labels** (인수는 레이블 혹은 리스트)

드롭할 행의 레이블(이름)이나 열의 레이블. 복수라면 리스트 로 묶어서 입력한다.

**axis** (인수는 0 또는 1/기본값은 0)

삭제할 부분이 행인지 열인지를 지정하는 인자. 기본값은 0이고 행을 삭제한다

**level** (멀티 인덱스의 레벨 / 기본값은 None)

멀티인덱스일 때 삭제할 레벨을 지정하는 인자

drop 함수 설명 블로그

drop 판다스 공식문서

```
# D열만 빼고 모두 가져오기
df[['A', 'B', 'C', 'E', 'F']] # 인덱싱으로는 번거롭다
df.drop('D', axis=1) # 삭제가 더 편리하다
```

| | A | B | C | E | F |
|---|---|---|---|---|---|
| 가 | 1 | 3 | 1 | 4 | 1 |
| 나 | 6 | 5 | 8 | 0 | 3 |
| 다 | 7 | 6 | 2 | 4 | 8 |
| 라 | 9 | 0 | 2 | 6 | 5 |

```
# 행은 나와 다 그리고 열은 C열만 빼고 다 가져오기
```

```
df.loc[['나', '다'], ['A', 'B', 'D', 'E', 'F']]
```

| | A | B | D | E | F |
|---|---|---|---|---|---|
| **나** | 6 | 5 | 9 | 0 | 3 |
| **다** | 7 | 6 | 3 | 4 | 8 |

```
# 인덱싱만 하는것보다 인덱싱과 drop 함수를 섞는 것이 편리할 때가 많다
df.loc[['나', '다']].drop('C', axis=1)
```

| | A | B | D | E | F |
|---|---|---|---|---|---|
| **나** | 6 | 5 | 9 | 0 | 3 |
| **다** | 7 | 6 | 3 | 4 | 8 |

인덱싱과 슬라이싱에 대해서 더 공부가 필요한 분들은 아래 강의를 참고하세요

- 엑셀투파이썬 채널 인덱싱과 슬라이싱 강의

```
# 프로젝트 코드
import pandas as pd
pd.set_option('max_rows', 6)
pd.options.display.float_format = '{:.3f}'.format
url = 'https://www.koreabaseball.com/Record/Player/HitterDetail/Daily.aspx?playerId=71564'
dfs = pd.read_html(url, index_col=0)
df_ex1 = pd.concat(dfs)
df_ex1
```

| | 상대 | AVG1 | PA | AB | R | H | 2B | 3B | HR | RBI | SB | CS | BB | HBP | SO | GDP | AVG2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **04.02** | 키움 | 0.250 | 4 | 4 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0.250 |
| **04.03** | 키움 | 0.400 | 5 | 5 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0.333 |
| **04.05** | NC | 0.500 | 5 | 4 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0.385 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **10.05** | NC | 0.000 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0.332 |
| **10.08** | LG | 0.250 | 4 | 4 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 0.331 |
| **합계** | 합계 | 0.143 | 16 | 14 | 1 | 2 | 1 | 0 | 1 | 4 | 0 | 0 | 2 | 0 | 2 | 3 | 0.331 |

149 rows × 17 columns

```
# 장타율을 구하는데 필요한 열만 인덱싱하고 합계행을 삭제한다
df_ex2 = df_ex1[['AB', 'H', '2B', '3B', 'HR']].drop('합계')
df_ex2
```

| | AB | H | 2B | 3B | HR |
|---|---|---|---|---|---|
| **04.02** | 4 | 1 | 0 | 0 | 0 |
| **04.03** | 5 | 2 | 0 | 0 | 0 |
| **04.05** | 4 | 2 | 0 | 0 | 0 |
| **...** | ... | ... | ... | ... | ... |
| **10.03** | 2 | 1 | 0 | 0 | 1 |
| **10.05** | 4 | 0 | 0 | 0 | 0 |
| **10.08** | 4 | 1 | 1 | 0 | 0 |

142 rows × 5 columns

## 4. 데이터프레임과 시리즈의 연산

# 판다스의 연산

데이터 프레임이나 시리즈와 **스칼라(상수)의 연산**
- 모든 요소에서 각각 연산한다

**시리즈 사이의 연산**
- 동일한 index 끼리 각각 연산한다

**데이터 프레임 사이의 연산**
- 동일한 index 와 columns 끼리 각각 연산한다

**데이터 프레임과 시리즈의 연산**
- 브로드 캐스팅(broad casting)

```
# 실습준비 코드
import pandas as pd
s1 = pd.Series([0, 1, 2, 3], index=list('ABCD'))
s2 = pd.Series([2, 3, 4, 5], index=list('ABCD'))
s3 = pd.Series([2, 3, 4], index=list('CAD'))
df1 = pd.DataFrame([[50, 40], [60, 70], [90, 70], [40, 20]],
                   index=list('ABCD'), columns=['국어', '영어'])
df2 = pd.DataFrame([[50, 60], [40, 70], [80, 70], [40, 90]],
                   index=list('BADC'), columns=['국어', '영어'])
```

```
# 시리즈와 스칼라(상수)의 연산 (각각 실행해보세요)
s1 + 1
s1 * 2
```

```
A    0
B    2
C    4
D    6
dtype: int64
```

```
# 시리즈와 스칼라(상수)의 비교연산
s1 > 2
```

```
A    False
B    False
C    False
D     True
dtype: bool
```

```
# 시리즈와 시리즈의 연산
s1 + s2
```

```
A    2
B    4
C    6
D    8
dtype: int64
```

```
# 시리즈와 시리즈의 연산 (인덱스의 구성이 서로 다를 때)
s1 + s3
```

```
A   3.000
B     NaN
C   4.000
D   7.000
dtype: float64
```

```
# 데이터 프레임과 스칼라(상수)의 연산 (각각 실행해 보세요)
df1 + 1
```

```
df1 == 70
```

| | 국어 | 영어 |
|---|---|---|
| **A** | False | False |
| **B** | False | True |
| **C** | False | True |
| **D** | False | False |

```
# 데이터 프레임과 데이터 프레임의 연산
df1 + df2
```

| | 국어 | 영어 |
|---|---|---|
| **A** | 90 | 110 |
| **B** | 110 | 130 |
| **C** | 130 | 160 |
| **D** | 120 | 90 |

```
# 데이터 프레임의 열간의 연산(시리즈의 연산)
df1['국어'] + df1['영어']
```

```
A     90
B    130
C    160
D     60
dtype: int64
```

연산에 대해서 더 공부가 필요한 분들은 아래 강의를 참고하세요

- 엑셀투파이썬 채널 연산 강의

## 5. 열 다루기

```
# 실습준비 코드
import pandas as pd
dict1 = {'A': 1, 'B': 2, 'C': 4}
df1 = pd.DataFrame([[50, 40], [60, 70], [90, 70], [40, 20]],
                   index=list('ABCD'), columns=['국어', '영어'])
df2 = pd.DataFrame([[50, 60], [40, 70], [80, 70], [40, 90]],
                   index=list('BADC'), columns=['국어', '영어'])
df1
```

| | 국어 | 영어 |
|---|---|---|
| **A** | 50 | 40 |
| **B** | 60 | 70 |
| **C** | 90 | 70 |
| **D** | 40 | 20 |

```
# 수학 열 만들기
df1['수학'] = 'pass'
df1
```

| | 국어 | 영어 | 수학 |
|---|---|---|---|

```
# 열 수정하기
df1['수학'] = '-'
df1
```

| | 국어 | 영어 | 수학 |
|---|---|---|---|
| **A** | 50 | 40 | - |
| **B** | 60 | 70 | - |
| **C** | 90 | 70 | - |
| **D** | 40 | 20 | - |

```
# 열 수정하기
df1['수학'] = [10, 20, 30, 40]
df1
```

| | 국어 | 영어 | 수학 |
|---|---|---|---|
| **A** | 50 | 40 | 10 |
| **B** | 60 | 70 | 20 |
| **C** | 90 | 70 | 30 |
| **D** | 40 | 20 | 40 |

```
# 시리즈를 배정하기
s1 = pd.Series([10, 20, 30, 40], index=list('ABCD'))
df1['수학'] = s1
df1
```

| | 국어 | 영어 | 수학 |
|---|---|---|---|
| **A** | 50 | 40 | 10 |
| **B** | 60 | 70 | 20 |
| **C** | 90 | 70 | 30 |
| **D** | 40 | 20 | 40 |

```
# 복수의 열 한번에 만들기
df1[['과학', '사회']] = 'pass'
df1
```

| | 국어 | 영어 | 수학 | 과학 | 사회 |
|---|---|---|---|---|---|
| **A** | 50 | 40 | 10 | pass | pass |
| **B** | 60 | 70 | 20 | pass | pass |
| **C** | 90 | 70 | 30 | pass | pass |
| **D** | 40 | 20 | 40 | pass | pass |

```
# 열간의 연산으로 열 만들기
df2['총점'] = df2['국어'] + df2['영어']
df2
```

열다루기에 대해서 더 공부가 필요한 분들은 아래 강의를 참고하세요

- 엑셀투파이썬 채널 열 강의

```
# 프로젝트 코드
import pandas as pd
pd.set_option('max_rows', 6) # 6행까지만 출력옵션
pd.options.display.float_format = '{:.3f}'.format # 소수점 세자리까지 출력옵션
url = 'https://www.koreabaseball.com/Record/Player/HitterDetail/Daily.aspx?playerId=71564'
dfs = pd.read_html(url, index_col=0)
df_ex1 = pd.concat(dfs)
df_ex2 = df_ex1[['AB', 'H', '2B', '3B', 'HR']].drop('합계')
df_ex2
```

| | AB | H | 2B | 3B | HR |
|---|---|---|---|---|---|
| **04.02** | 4 | 1 | 0 | 0 | 0 |
| **04.03** | 5 | 2 | 0 | 0 | 0 |
| **04.05** | 4 | 2 | 0 | 0 | 0 |
| **...** | ... | ... | ... | ... | ... |
| **10.03** | 2 | 1 | 0 | 0 | 1 |
| **10.05** | 4 | 0 | 0 | 0 | 0 |
| **10.08** | 4 | 1 | 1 | 0 | 0 |

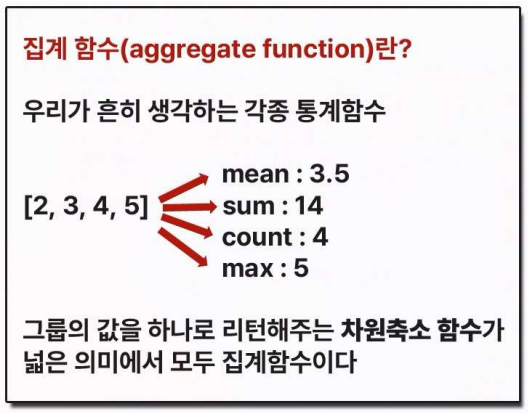
142 rows × 5 columns

```
# 루타 열을 만들자
df_ex2['루타'] = df_ex2['H'] + (df_ex2['2B'] * 2) + (df_ex2['3B'] * 3) + (df_ex2['H
df_ex2
```

| | AB | H | 2B | 3B | HR | 루타 |
|---|---|---|---|---|---|---|
| **04.02** | 4 | 1 | 0 | 0 | 0 | 1 |
| **04.03** | 5 | 2 | 0 | 0 | 0 | 2 |
| **04.05** | 4 | 2 | 0 | 0 | 0 | 2 |
| **...** | ... | ... | ... | ... | ... | ... |
| **10.03** | 2 | 1 | 0 | 0 | 1 | 5 |
| **10.05** | 4 | 0 | 0 | 0 | 0 | 0 |
| **10.08** | 4 | 1 | 1 | 0 | 0 | 3 |

142 rows × 6 columns

## 6. 데이터 프레임에 각종 통계함수 적용하기



```
# 프로젝트 코드
import pandas as pd
pd.set_option('max_rows', 6) # 6행까지만 출력옵션
pd.options.display.float_format = '{:.3f}'.format # 소수점 세자리까지 출력옵션
url = 'https://www.koreabaseball.com/Record/Player/HitterDetail/Daily.aspx?playerId=71564'
dfs = pd.read_html(url, index_col=0)
df_ex1 = pd.concat(dfs)
df_ex2 = df_ex1[['AB', 'H', '2B', '3B', 'HR']].drop('합계')
df_ex2['루타'] = df_ex2['H'] + (df_ex2['2B'] * 2) + (df_ex2['3B'] * 3) + (df_ex2['HR'] * 4)
df_ex2
```

| | AB | H | 2B | 3B | HR | 루타 |
|---|---|---|---|---|---|---|
| **04.02** | 4 | 1 | 0 | 0 | 0 | 1 |
| **04.03** | 5 | 2 | 0 | 0 | 0 | 2 |
| **04.05** | 4 | 2 | 0 | 0 | 0 | 2 |
| **...** | ... | ... | ... | ... | ... | ... |
| **10.03** | 2 | 1 | 0 | 0 | 1 | 5 |
| **10.05** | 4 | 0 | 0 | 0 | 0 | 0 |
| **10.08** | 4 | 1 | 1 | 0 | 0 | 3 |

142 rows × 6 columns

```
# df_ex2의 윗 다섯 줄을 따로 df로 지정하자
df = df_ex2.head(5).copy()
df
```

| | AB | H | 2B | 3B | HR | 루타 |
|---|---|---|---|---|---|---|
| **04.02** | 4 | 1 | 0 | 0 | 0 | 1 |
| **04.03** | 5 | 2 | 0 | 0 | 0 | 2 |
| **04.05** | 4 | 2 | 0 | 0 | 0 | 2 |
| **04.06** | 4 | 0 | 0 | 0 | 0 | 0 |
| **04.07** | 4 | 0 | 0 | 0 | 0 | 0 |

```
# 각 열의 합
df.sum()
```

```
AB    21
H      5
2B     0
3B     0
HR     0
루타     5
dtype: int64
```

```
# 각 열의 평균
df.mean()
```

```
AB   4.200
H    1.000
2B   0.000
3B   0.000
HR   0.000
루타   1.000
dtype: float64
```

```
# 각 행의 합
df.sum(axis=1)
```

```
04.02    6
04.03    9
04.05    8
04.06    4
04.07    4
dtype: int64
```

```
# 누적합
df.cumsum()
```

| | AB | H | 2B | 3B | HR | 루타 |
|---|---|---|---|---|---|---|
| **04.02** | 4 | 1 | 0 | 0 | 0 | 1 |
| **04.03** | 9 | 3 | 0 | 0 | 0 | 3 |
| **04.05** | 13 | 5 | 0 | 0 | 0 | 5 |
| **04.06** | 17 | 5 | 0 | 0 | 0 | 5 |
| **04.07** | 21 | 5 | 0 | 0 | 0 | 5 |

```
# 프로젝트 변수인 df_ex2에 장타율 열 생성
df_ex2['누적루타'] = df_ex2['루타'].cumsum()
df_ex2['누적타수'] = df_ex2['AB'].cumsum()
df_ex2['slg'] = df_ex2['누적루타'] / df_ex2['누적타수']
df_ex2
```

| | AB | H | 2B | 3B | HR | 루타 | 누적루타 | 누적타수 | slg |
|---|---|---|---|---|---|---|---|---|---|
| **04.02** | 4 | 1 | 0 | 0 | 0 | 1 | 1 | 4 | 0.250 |
| **04.03** | 5 | 2 | 0 | 0 | 0 | 2 | 3 | 9 | 0.333 |
| **04.05** | 4 | 2 | 0 | 0 | 0 | 2 | 5 | 13 | 0.385 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **10.03** | 2 | 1 | 0 | 0 | 1 | 5 | 314 | 532 | 0.590 |
| **10.05** | 4 | 0 | 0 | 0 | 0 | 0 | 314 | 536 | 0.586 |
| **10.08** | 4 | 1 | 1 | 0 | 0 | 3 | 317 | 540 | 0.587 |

142 rows × 9 columns

```
# 장타율 시각화 하기
df_ex2.plot(y='slg')
```

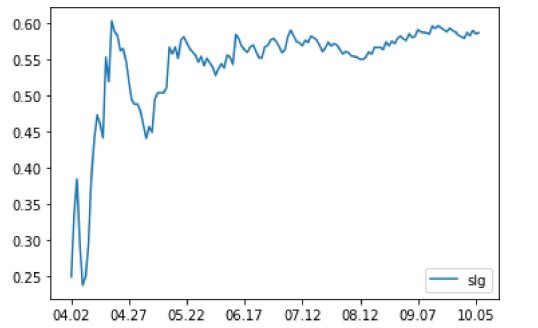
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f77ebd78130>
```


## 7. 요약강의

```
import pandas as pd
pd.set_option('max_rows', 6) # 6행까지만 출력옵션
pd.options.display.float_format = '{:.3f}'.format # 소수점 세자리까지 출력옵션

# 해당 웹사이트의 모든 표를 데이터프레임으로 만들어 리스트에 담아준다
```

```
url = 'https://www.koreabaseball.com/Record/Player/HitterDetail/Daily.aspx?playerId=71564'
dfs = pd.read_html(url, index_col=0)

# 리스트의 모든 데이터프레임 결합하기
df_ex1 = pd.concat(dfs)

# 일부 열만 인덱싱하고 합계 행 삭제하기
df_ex2 = df_ex1[['AB', 'H', '2B', '3B', 'HR']].drop('합계')

# 연산해서 열 만들기
df_ex2['루타'] = df_ex2['H'] + (df_ex2['2B'] * 2) + (df_ex2['3B'] * 3) + (df_ex2['HR'] * 4)
df_ex2['누적루타'] = df_ex2['루타'].cumsum()
df_ex2['누적타수'] = df_ex2['AB'].cumsum()
df_ex2['slg'] = df_ex2['누적루타'] / df_ex2['누적타수']

# 판다스의 시각화
df_ex2.plot(y='slg')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f77eb8284c0>
```




**참고 사이트**

판다스 공식 문서 https://pandas.pydata.org/docs/

강사 김판다의 블로그 https://kimpanda.tistory.com

강사 김판다의 유튜브 채널 엑셀투파이썬 https://www.youtube.com/channel/UCKsPvuqR7BucXi7umcUlmzg