

PANDAS

- 파이썬기초 -

PANDAS

테이터프레임(엑셀과 비슷) 과 시리즈라는 자료형
데이터 분석을 위한 다양한 기능을 제공하는 라이브러리
R의 데이터프레임에 영향
내부적으로 numpy 를 사용하므로 함께 import

<http://pandas.pydata.org>

기능요약

https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf

소스코드

99-1 Pandas.ipynb

Pandas 자료형

판다스	파이썬
object	string
int64	int
float64	float
datetime64	

소스코드

99-1 Pandas.ipynb

Series

numpy 의 1차원 array 이 비슷

```
import pandas as pd
import numpy as np
```

```
x = [1, 2, 3, 4, 5]
pd.Series(x) # Series 생성
```

```
# 넘파이 리스트도 사용가능
x = np.array([1, 2, 3, 4, 5])
pd.Series(x)
```

```
x = [1, 2, 3, 4, 5]
x = pd.Series(x)
```

```
x[0] # 1
x[1: 3] # 2, 3
x[::-1] # 5, 4, 3, 2, 1
x[x > 3] # 4, 5
```

```
x + 1 # 2, 3, 4, 5, 6
x * 10 # 10, 20, 30, 40, 50
x + x # 2, 4, 6, 8, 10
```

소스코드

99-1 Pandas.ipynb

Series Index

기본적으로 index 와 value 로 값이 구분되어 있다.

```
x = [1, 2, 3, 4, 5]
```

```
x = pd.Series(x)
```

```
print(x.index) # 인덱스
```

```
print(x.values) # 값
```

결과 :

```
RangeIndex(start=0, stop=5, step=1)
```

```
[1 2 3 4 5]
```

인덱스명을 변경할수있다.

```
x = [1, 2, 3, 4, 5]
```

```
x = pd.Series(x, index=['a','b','c','d','e'])
```

```
print(x['a']) # 명시적 인덱스접근
```

```
print(x[0]) # 묵시적 인덱스접근
```

```
print(x[['a','e']]) # 팬시색인, 한번에 여러값 접근
```

```
print(x.a) # 또다른 접근방법
```

결과 :

```
1
```

```
1
```

```
a 1
```

```
e 5
```

```
1
```

소스코드

99-1 Pandas.ipynb

가치를 높이는 금융 인공지능 실무교육

Insightcampus

딕셔너리 to Series

딕셔너리를 사용하여 Series 를 만들면 key 값을 index 로 사용한다.

```
x = {"수학":90, "영어":80, "과학":95, "미술":80}
```

```
x = pd.Series(x)
```

```
x
```

결과:

```
수학  90
영어  80
과학  95
미술  80
dtype: int64
```

```
x['수학']
```

결과:

```
90
```

슬라이싱 가능

```
x['영어:']
```

결과:

```
영어  80
과학  95
미술  80
```

#인덱스값을 지정하여 일부값만 Series 로 생성

```
x = pd.Series(x, index=["수학", "영어", "과학"])
```

결과:

```
수학  90
영어  80
과학  95
dtype: int64
```

소스코드

99-1 Pandas.ipynb

Multi Index

인덱스를 여러개 가질수있다.

```
student_1 = {"수학":90, "영어":80, "과학":95, "미술":80}
```

```
student_2 = {"수학":70, "영어":90, "과학":100, "미술":70}
```

```
#index_1 = ['홍길동','홍길동','홍길동','홍길동','이몽룡','이몽룡','이몽룡','이몽룡']
```

```
index_1 = ['홍길동' for i in range(len(student_1))] + ['이몽룡' for i in range(len(student_2))]
```

```
#index_2 = ['수학','영어','과학','미술','수학','영어','과학','미술']
```

```
index_2 = [i for i in student_1] + [i for i in student_2]
```

```
value_all = list(student_1.values()) + list(student_2.values())
```

```
students = pd.Series(value_all, index=[index_1, index_2])
```

소스코드

99-1 Pandas.ipynb

Multi Index

인덱스를 여러개 가질수있다.

```
student_1 = {"수학":90, "영어":80, "과학":95, "미술":80}
```

```
student_2 = {"수학":70, "영어":90, "과학":100, "미술":70}
```

```
#index_1 = ['홍길동','홍길동','홍길동','홍길동','이몽룡','이몽룡','이몽룡','이몽룡']
```

```
index_1 = ['홍길동' for i in range(len(student_1))] + ['이몽룡' for i in range(len(student_2))]
```

```
#index_2 = ['수학','영어','과학','미술','수학','영어','과학','미술']
```

```
index_2 = [i for i in student_1] + [i for i in student_2]
```

```
value_all = list(student_1.values()) + list(student_2.values())
```

```
students = pd.Series(value_all, index=[index_1, index_2])
```

소스코드

99-1 Pandas.ipynb

결측값 처리방법 (NaN, None)

비어있는 값들에 대한
처리

```
x = [1, None, 2, None, 3, 4, None]
x = pd.Series(x)
```

결측값을 무시하고 계산

```
print(x.sum()) # 10
print(x.max()) # 4
print(x.min()) # 1
```

#결측값 개수

```
print(x.isnull().sum()) # 3
print(x.notnull().sum()) # 4
```

#결측값 제거

```
x.dropna()
```

#결측값을 다른값으로 채우기

```
x.fillna(0)
```

소스코드

99-1 Pandas.ipynb

concat

데이터 연결, numpy concatenate 기능

```
x = pd.Series([1, 2, 3])  
y = pd.Series([4, 5, 6])  
z = pd.Series([7, 8, 9])
```

```
pd.concat([x, y, z])
```

```
# 같은인덱스번호가 있을경우 오류  
pd.concat([x, y, z], verify_integrity=True)
```

```
# 합치면서 인덱스 번호를 다시정리  
pd.concat([x, y, z], ignore_index=True)
```

```
# 축변경  
pd.concat([x, y, z], ignore_index=True, axis=1)
```

```
x = pd.Series([1, 2, 3, 4])  
y = pd.Series([4, 5, 6])  
z = pd.Series([7, 8, 9])
```

```
# 개수가 맞지 않을경우 교집합  
pd.concat([x, y, z], ignore_index=True, axis=1, join='inner')  
# 개수가 맞지 않을경우 합집합  
pd.concat([x, y, z], ignore_index=True, axis=1, join='outer')
```

연산과 집계함수

```
x = pd.Series([1, 2, 3, 4, 5])  
y = pd.Series([6, 7, 8, 9, 0])
```

더하기

```
print(x.add(10))  
print(x.add(y))
```

빼기

```
print(x.sub(y))
```

곱하기

```
print(x.mul(y))
```

나누기

```
print(x.floordiv(2))  
print(x.div(2))  
print(x.mod(2))
```

제곱

```
print(x.pow(2))
```

기초통계

```
print(x.count())  
print(x.min())  
print(x.max())  
print(x.mean())  
print(x.median()) # 중앙값  
print(x.sum())  
print(x.std()) # 표준편차  
print(x.var()) # 분산  
print(x.mad()) # 절대표준편차  
print(x.describe()) # 기초통계모두
```

```
print(x.head(2)) # 앞의 일부데이터  
확인
```

```
print(x.tail(2)) # 뒤의 일부데이터 확인 소스코드  
99-1 Pandas.ipynb
```

날짜 함수

날짜 자동 생성

```
pd.date_range(start='2019-01-01', end='2019-01-07')
```

#기간 사용

```
pd.date_range(start='2019-01-01', periods=7)
```

#증감

```
pd.date_range(start='2019-01-01', periods=7, freq='2D')
```

```
pd.date_range(start='2019-01-01', periods=7, freq='W')
```

#날짜형

```
x = pd.to_datetime(['2020-01-02 03:45', '2021-12-24 23:59'])
```

```
x.year
```

```
x.weekday
```

```
diff = x - pd.to_datetime('2020-02-12 15:39')
```

날짜 형식

- yyyy-mm-dd
- yyyy/mm/dd
- yyyy.mm.dd
- mm-dd-yyyy
- mm/dd/yyyy

소스코드

99-1 Pandas.ipynb

실습1

아래 코드를 실행하고 Series Y 에서

1. 결측값의 개수를 구하고
2. 결측값을 제거한 Series 를 만들고
3. 결측값에 y의 평균값을 넣은 Series 를 만들어주세요.

```
x = [np.nan, 1, 2, 3, 4, 5]  
y = pd.Series([x[np.random.randint(0, 6)] for i in range(20)])
```

소스코드

99-1 Pandas.ipynb

DataFrame

2차원 테이블 데이터 구조, 엑셀(스프레드시트) 와 비슷

```
sales_data = {  
    '연도':[2015, 2016, 2017, 2018, 2019, 2020],  
    '판매량':[103, 70, 130, 160, 190, 230],  
    '매출':[500000, 300000, 400000, 550000, 700000, 680000],  
    '순이익':[370000, 190000, 300000, 480000, 600000, 590000]}  
}
```

```
df = pd.DataFrame(sales_data)
```

	판매량	매출	순이익
2015	103	500000	370000
2016	70	300000	190000
2017	130	400000	300000
2018	160	550000	480000
2019	190	700000	600000
2020	230	680000	590000

소스코드

99-1 Pandas.ipynb

DataFrame 생성

Series로 부터 생성

```
a = pd.Series([103, 500000, 370000], ['판매량', '매출', '순이익'])  
b = pd.Series([70, 300000, 190000], ['판매량', '매출', '순이익'])  
c = pd.Series([130, 400000, 190000], ['판매량', '매출', '순이익'])
```

```
df = pd.DataFrame([a, b, c], index=[2015, 2016, 2017])
```

	판매량	매출	순이익
2015	103	500000	370000
2016	70	300000	190000
2017	130	400000	190000

소스코드

99-1 Pandas.ipynb

DataFrame

2차원 테이블 데이터 구조, 엑셀(스프레드시트) 와 비슷

열 선택

`df['판매량']` - 하나의 열(series)

`df[['판매량', '매출']]` - 복수의 열(dataframe)

행 선택

`df[2:3]`

`df.iloc[1]` - 하나의 행, `df.iloc[[1, 3]]` - 여러개 행

`df.loc[2015]` - 하나의 행, `df.loc[[2015, 2017]]`

#행과 열 동시 선택

`sales_data.loc[[2015, 2017], ['판매량', '매출']]` vs `sales_data.iloc[[1, 3], [0, 1]]`

columns - 원하는칼럼, index - 인덱스칼럼

`pd.DataFrame(sales_data, columns=['판매량', '매출', '순이익'], index=df['연도'])`

소스코드

99-1 Pandas.ipynb

CSV 파일읽기, 쓰기

```
%%writefile sales_data.csv
연도,판매량,매출,순이익
2015,103,500000,370000
2016,70,300000,190000
2017,130,400000,300000
2018,60,550000,480000
2019,190,700000,600000
2020,230,680000,590000
```

콤마로 데이터가 분리되어있는 텍스트파일
(CSV : Comma Separated Values)

파일읽기

```
sales_data = pd.read_csv('sales_data.csv', index_col='연도', header=0, sep=',')
```

index_col - 인덱스 칼럼

header - csv 에 헤더가 있을경우 0, 없으면 None

sep - 데이터를 분리하는 기호 (기본값 : ,)

usecols - 원하는 컬럼만 가져오기

파일쓰기

```
sales_data.to_csv('sales_data_save.csv', encoding='utf-8')
```

encoding - 인코딩

소스코드

99-1 Pandas.ipynb

데이터(Column) 추가

```
sales_data = {
    '연도':[2015, 2016, 2017, 2018, 2019, 2020],
    '판매량':[103, 70, 130, 160, 190, 230],
    '매출':[500000, 300000, 400000, 550000, 700000, 680000],
    '순이익':[370000, 190000, 300000, 480000, 600000, 590000]
}

sales_data = pd.DataFrame(sales_data)
sales_data['순이익율'] = (sales_data['순이익']/sales_data['매출']) * 100
```

```
sales_data['순이익율_비교'] = sales_data['순이익율'].apply(check)
sales_data
```

	연도	판매량	매출	순이익	순이익율	순이익율_비교
0	2015	103	500000	370000	74.000000	낮음
1	2016	70	300000	190000	63.333333	낮음
2	2017	130	400000	300000	75.000000	낮음
3	2018	160	550000	480000	87.272727	높음
4	2019	190	700000	600000	85.714286	높음
5	2020	230	680000	590000	86.764706	높음

```
def check(n):
    if n > 80:
        return '높음'
    else:
        return '낮음'
```

소스코드

99-1 Pandas.ipynb

데이터(Column) 추가2

인덱스에 조건가능

```
sales_data[sales_data['매출'] > 300000]
```

해당 조건에 일치하는것만 데이터 변경 **np.where(조건, True일경우, False일경우)**

```
sales_data['테스트1'] = np.where(sales_data['판매량'] > 200, 0, sales_data['판매량'])
```

해당 조건에 일치하는것만 바꾸고 나머지는 **NaN**

```
sales_data['테스트2'] = sales_data[sales_data['판매량'] < 100]['판매량'] + 50
```

행추가

```
sales_data.loc[6] = [2021, 720000, 650000, 360, 0, 0]
```

```
sales_data.loc[7] = sales_data.loc[5] + 100
```

	연도	판매량	매출	순이익	테스트1	테스트2
0	2015.0	103.0	500000.0	370000.0	103.0	NaN
1	2016.0	70.0	300000.0	190000.0	70.0	120.0
2	2017.0	130.0	400000.0	300000.0	130.0	NaN
3	2018.0	160.0	550000.0	480000.0	160.0	NaN
4	2019.0	190.0	700000.0	600000.0	190.0	NaN
5	2020.0	230.0	680000.0	590000.0	0.0	NaN
6	2021.0	720000.0	650000.0	360.0	0.0	0.0
7	2120.0	330.0	680100.0	590100.0	100.0	NaN

소스코드

99-1 Pandas.ipynb

데이터 병합

```
sales_data1 = {  
    '연도':[2015, 2016, 2017, 2018],  
    '판매량':[103, 70, 130, 160],  
    '매출':[500000, 300000, 400000, 550000],  
    '순이익':[370000, 190000, 300000, 480000]  
}
```

```
sales_df1 = pd.DataFrame(sales_data1)  
sales_df2 = pd.DataFrame(sales_data2)
```

append

```
sales_df1.append(sales_df2, ignore_index=True)
```

concat

```
pd.concat([sales_df1, sales_df2], ignore_index=True)
```

```
sales_data2 = {  
    '연도':[2019, 2020],  
    '판매량':[190, 230],  
    '매출':[700000, 680000],  
    '순이익':[ 600000, 590000]  
}
```

소스코드

99-1 Pandas.ipynb

데이터 병합 2

```
cost_data = {  
    '연도':[2015, 2016, 2017, 2018],  
    '이자':[1520, 1120, 1360, 1680],  
    '관리비':[650, 400, 750, 800],  
    '세금':[12000, 8900, 10900, 12800] }
```

```
cost_data = pd.DataFrame(cost_data)
```

merge

```
pd.merge(sales_df, cost_data, on='연도', how='left')
```

join

- 내부적으로 pandas.merge 함수 사용
- 기본적으로 index를 사용하여 left join
- 주로 merge 사용

how

- inner : 공통 항목만
- outer : 모두 선택
- left : 왼쪽 기준
- right : 오른쪽 기준

소스코드

99-1 Pandas.ipynb

데이터(Column) 삭제

```
sales_data = pd.DataFrame(sales_data)
sales_data['테스트1'] = 'test1'
sales_data['테스트2'] = 'test2'
sales_data['테스트3'] = 'test3'
```

del 키워드 사용

```
del sales_data['테스트1']
sales_data
```

drop 함수사용, inplace 를 True 로 하여 원본에 적용

```
sales_data.drop(['테스트2'], axis='columns', inplace=True)
```

팬시를 활용하여 한번에 여러개 지우기

```
sales_data.drop(sales_data.columns[[0, 2]], axis='columns', inplace=True)
```

행삭제

```
sales_data.drop(0, inplace=True)
sales_data.drop([3, 4, 5], inplace=True)
```

소스코드

99-1 Pandas.ipynb

Dataframe Multiindex

```
df = pd.DataFrame(np.random.randint(1, 100, size=(4, 4)), index=[['A','A','B','B'], ['a','b','a','b']],
```

```
columns=[['가가','가가','나나','나나'], ['가','나','가','나']])
```

df['가가']

		가	나
A	a	80	49
	b	84	94
B	a	4	52
	b	89	52

df.loc['A']

		가가		나나	
		가	나	가	나
a		15	19	93	41
		62	17	2	61

		가가		나나	
		가	나	가	나
A	a	80	49	61	81
	b	84	94	69	52
B	a	4	52	51	41
	b	89	52	18	30

인덱스확인

df.index

칼럼확인

df.columns

소스코드

99-1 Pandas.ipynb

GroupBy

```
df = pd.DataFrame(np.random.randint(1, 100, size=(8, 2)),
                  index=[['A참고','A참고','A참고','A참고','B참고','B참고','B참고','B참고'],
                        ['사과','배','바나나','사과','사과','배','바나나','배']],
                  columns=['판매','재고'])
df.index.names = ['참고명','상품명']
```

		판매 재고	
참고명	상품명		
A참고	사과	58	79
	배	27	46
	바나나	77	8
	사과	81	51
B참고	사과	86	84
	배	10	77
	바나나	37	43
	배	7	11

df.groupby('참고명').sum()

		판매 재고	
참고명			
A참고		243	184
B참고		140	215

df.groupby('상품명').sum()

		판매 재고	
	상품명		
바나나		114	51
	배	44	134
사과		225	214

df.groupby(['참고명','상품명']).sum()

		판매 재고	
참고명	상품명		
A참고	바나나	77	8
	배	27	46
	사과	139	130
B참고	바나나	37	43
	배	17	88
	사과	86	84

소스코드

99-1 Pandas.ipynb

sort_values

```
df = pd.DataFrame(np.random.randint(1, 100, size=(8, 2)),
                  index=[['A참고','A참고','A참고','A참고','B참고','B참고','B참고','B참고'],
                        ['사과','배','바나나','사과','사과','배','바나나','배']],
                  columns=['판매','재고'])
```

		판매 재고	
A참고	사과	47	98
	배	35	36
	바나나	58	77
B참고	사과	19	19
	배	17	27
	바나나	46	71
	배	54	33

```
df.sort_values(by="판매", ascending=True)
```

		판매 재고	
B참고	배	17	27
A참고	사과	19	19
B참고	사과	33	11
A참고	배	35	36
B참고	바나나	46	71
A참고	사과	47	98
B참고	배	54	33
A참고	바나나	58	77

```
df.sort_values(by=["판매", "재고"],
               ascending=[True, False])
```

		판매 재고	
B참고	배	17	27
A참고	사과	19	19
B참고	사과	33	11
A참고	배	35	36
B참고	바나나	46	71
A참고	사과	47	98
B참고	배	54	33
A참고	바나나	58	77

소스코드

99-1 Pandas.ipynb

실습2

아래와 같은 데이터프레임을
만들어주세요. (점수는 1~100까지 랜덤값)

		국어	영어	과학
1학년	1반	78	26	56
	2반	84	91	65
	3반	10	49	58
	4반	78	13	88
	5반	97	13	53
2학년	1반	43	12	31
	2반	7	11	4
	3반	68	49	62
	4반	26	74	14
	5반	42	56	80
3학년	1반	15	37	83
	2반	93	26	61
	3반	44	2	13
	4반	20	98	88
	5반	18	97	93

반별 합계와 평균을 추가해주세요

		국어	영어	과학	총점	평균
1학년	1반	78	26	56	160	122.666667
	2반	84	91	65	240	196.666667
	3반	10	49	58	117	78.333333
	4반	78	13	88	179	120.333333
	5반	97	13	53	163	127.666667
2학년	1반	43	12	31	86	65.333333
	2반	7	11	4	22	19.333333
	3반	68	49	62	179	137.666667
	4반	26	74	14	114	104.666667
	5반	42	56	80	178	124.666667
3학년	1반	15	37	83	135	79.666667
	2반	93	26	61	180	139.333333
	3반	44	2	13	59	50.333333
	4반	20	98	88	206	147.333333
	5반	18	97	93	208	146.000000

인덱스 명을 추가하고
학년별 합계를 구해주세요

	국어	영어	과학	총점	평균
학년					
1학년	347	192	320	859	645.666667
2학년	186	202	191	579	451.666667
3학년	190	260	338	788	562.666667

소스코드

99-1 Pandas.ipynb

실습3 (타이타닉 생존자체크) - 데이터설명

데이터설명

- * Passengerid: 탑승자 데이터 일련번호
- * survived: 생존 여부, 0 = 사망, 1 = 생존
- * Pclass: 티켓의 선실 등급, 1 = 일등석, 2 = 이등석, 3 = 삼등석
- * sex: 탑승자 성별
- * name: 탑승자 이름
- * Age: 탑승자 나이
- * sibsp: 같이 탑승한 형제자매 또는 배우자 인원수
- * parch: 같이 탑승한 부모님 또는 어린이 인원수
- * ticket: 티켓 번호
- * fare: 요금
- * cabin: 선실 번호
- * embarked: 중간 정착 항구 C = Cherbourg, Q = Queenstown, S = Southampton

소스코드

99-1 Pandas.ipynb

실습3 (타이타닉 생존자체크)

1. 결측지 처리를 해주세요. (age - 평균값적용, Cabin - N 으로, Embarked - N)

2. 성별(Sex) 생존자 합을 구해주세요

결과 :

Sex	
female	233
male	109

3. 클래스(Pclass) 별 생존자 합을 구해주세요

결과 :

Pclass	
1	136
2	87
3	119

4. 나이(Age) 를 활용 유아, 10대, 20대, 30대, 40대, 50대, 60대, 노인의 생존자 합을 구해주세요.

결과 :

Age_Range	
10대	41
20대	129
30대	73
40대	34
50대	20
60대	6
노인	1
유아	38

실습4 (타이타닉 생존자체크)

1. 30대 이면서 1등석에 탄사람의 수

2. 클래스(Pclass)별 나이 평균

	Pclass
1	37.048118
2	29.866958
3	26.403259

3. 클래스(Pclass) 와 성별에 따른 생존율

	Pclass	Sex	
1		female	0.968085
		male	0.368852
2		female	0.921053
		male	0.157407
3		female	0.500000
		male	0.135447

4. 나이(Age) 대 별로 생존율 구하기

0	0.612903
10	0.401961
20	0.324937
30	0.437126
40	0.382022
50	0.416667
60	0.315789
70	0.000000
80	1.000000