

Introduction to Data Mining Lecture

Week 6: Pandas (2)

Joon Young Kim

Assistant Professor, School of AI Convergence
Sungshin Women's University

Week 6 Outline

- Week 5 Review
- Pandas Data Processing
 - Data Selection
 - Data Setting and Search
 - Missing Data Processing
 - MultiIndex
- Pandas Data I/O and Type
 - Text File
 - Binary Data
 - SQL DB
 - Example
- Conclusion

Week 6 Outline

■ Week 5 Review

■ Pandas Data Processing

- Data Selection
- Data Setting and Search
- Missing Data Processing
- MultiIndex

■ Pandas Data I/O and Type

- Text File
- Binary Data
- SQL DB
- Example

■ Conclusion

Week 5 Review

■ 판다스 기초 부분 학습

→ Series & DataFrame부터 행렬 및 인덱스 객체까지 실습

```
In [90]: df3 = pd.DataFrame(np.arange(12).reshape(3, 4), columns=['A', 'B', 'C', 'D'])
```

```
In [91]: np.exp(df3)
```

```
In [92]: np.asarray(df3)
```

Week 5 Review

■ 판다스 주요 기능 학습

→ 바이너리 동작, 통계 및 함수 학습 및 실습

```
In [189]: df = pd.DataFrame(np.random.randn(4, 3), columns=['A', 'B', 'C'])
```

```
In [190]: df
```

```
In [191]: df.idxmin(axis=0)
```

```
In [192]: df.idxmin()
```

Week 6 Outline

■ Week 5 Review

■ Pandas Data Processing

- Data Selection
- Data Setting and Search
- Missing Data Processing
- MultiIndex

■ Pandas Data I/O and Type

- Text File
- Binary Data
- SQL DB
- Example

■ Conclusion

Pandas Data Processing

■ 데이터 선택

- 라벨에 의한 선택
- 위치에 의한 선택
- 호출에 의한 선택

```
In [274]: df1 = pd.DataFrame(np.random.randn(5, 4), columns=list('ABCD'),  
                             index=pd.date_range('20190701', periods=5))
```

```
In [275]: df1
```

```
In [276]: df1.loc['20190702':'20190703']
```

■ 라벨에 의한 선택

- .loc 속성의 사용

Data Selection

■ 인덱스에 의한 라벨 방법

```
In [277]: ser1 = pd.Series(np.random.randn(4), index=list('abcd'))
```

```
In [278]: ser1
```

```
In [279]: ser1.loc['c']
```

```
In [280]: ser1.loc['b']
```

■ .loc속성에 의한 동적할당

```
In [281]: ser1.loc['c'] = 0
```

```
In [282]: ser1
```


Data Selection

■ 라벨에 의한 선택 - DataFrame

```
In [283]: df1 = pd.DataFrame(np.random.randn(5, 4), index=list('abcde'), columns=list('ABCD'))
```

```
In [284]: df1
```

```
In [285]: df1.loc[['a', 'b', 'd'], :]
```

Data Selection

■ 슬라이싱 및 한 개의 라벨을 통한 데이터 접근

```
In [286]: df1.loc['c':, 'A':'C']
```

```
In [287]: df1.loc['a']
```

■ .loc속성 – 시작과 끝 라벨

```
In [288]: ser = pd.Series(list('abcde'), index=[0, 3, 2, 5, 4])
```

```
In [289]: ser
```

```
In [290]: ser.loc[3:5]
```

Data Selection

- ser객체에서 .loc[1:6] 속성 사용 – 1, 6은 라벨 명칭이고 선택에서 포함

```
In [291]: ser.sort_index()
```

```
In [292]: ser.sort_index().loc[1:6]
```

- 위치에 의한 선택
→ .iloc 속성

```
In [293]: ser1 = pd.Series(np.random.randn(5), index=list(range(0, 10, 2)))
```

```
In [294]: ser1
```

```
In [295]: ser1.iloc[:3]
```

```
In [296]: ser1.iloc[3]
```

Data Selection

■ 위치에 의한 선택 - DataFrame 객체 생성

```
In [297]: df1 = pd.DataFrame(np.random.randn(5, 4), index=list(range(0, 10, 2)), columns=list(range(0, 8, 2)))  
  
In [298]: df1
```

■ 위치에 의한 선택 - 정수 슬라이싱

```
In [299]: df1.iloc[:2]  
In [300]: df1.iloc[1:3, 0:3]
```

Data Selection

■ 위치에 의한 선택 - 정수리스트로 슬라이싱

```
In [301]: df1.iloc[[0, 2, 3], [1, 3]]
```

■ df1 객체를 횡단면으로 슬라이싱

```
In [302]: df1.iloc[1]
```

■ Callable에 의한 선택

→ df1 객체 생성

```
In [303]: df1 = pd.DataFrame(np.random.randn(5, 4), index=list('abcde'), columns=list('ABCD'))  
In [304]: df1
```

Data Selection

■ callable을 indexer로 적용

```
In [305]: df1.loc[lambda df: df.A>0, :]
```

■ callable을 적용하여 열을 선택

```
In [306]: df1.loc[:, lambda df: ['A', 'B']]
```

```
In [307]: df1.iloc[:, lambda df: [0, 1]]
```

Data Selection

■ Callable에 의한 선택 – columns 속성 사용

```
In [308]: df1[lambda df: df.columns[0]]
```

■ Callable에 의한 선택 – df1.A인 series

```
In [309]: df1.A.loc[lambda ser: ser > 0]
```

Data Setting and Search

■ 데이터 확장 설정 및 변경

→ 라벨 지정 후 동적할당과 데이터 확장

```
In [310]: ser = pd.Series(np.arange(3))
```

```
In [311]: ser
```

```
In [312]: ser[5] = 7
```

```
In [313]: ser
```

→ .loc로 축에 적용하여 확장

```
In [314]: df = pd.DataFrame(np.arange(9).reshape(3, 3), columns=['A',  
    'B', 'C'])
```

```
In [315]: df
```

```
In [316]: df.loc[:, 'D'] = df.loc[:, 'A']
```

```
In [317]: df
```


Data Setting and Search

■ 인덱스 라벨을 붙이고 새로운 값을 동적할당

```
In [318]: df.loc[3] = 7  
In [319]: df
```

■ at()와 iat() 메소드

```
In [320]: ser.iat[3]  
In [321]: ser.at[5]  
  
In [322]: df.at[3, 'E'] = 7  
In [323]: df.iat[3, 0] = 2  
  
In [324]: df
```

Data Setting and Search

■ 불리언 인덱싱 적용

→ 데이터를 필터링을 위한 불리언 벡터를 사용하는 연산

```
In [325]: ser = pd.Series(range(-3, 3))
```

```
In [326]: ser
```

```
In [327]: ser[ser > 0]
```

```
In [328]: ser[(ser < -1) | (ser > 1)]
```

```
In [329]: ser[~(ser < 2)]
```

→ 불리언 벡터에 조건 부여

```
In [330]: df[df['A'] < 3]
```

Data Setting and Search

■ Series에 isin() 메소드 적용

```
In [331]: ser[::-1].isin([-3, -1, 2])
```

```
In [332]: ser[ser[::-1].isin([-3, -1, 2])]
```

■ index객체에 isin() 메소드 적용

```
In [333]: ser.index.isin([2, 4, 6])
```

```
In [334]: ser[ser.index.isin([2, 4, 6])]
```

Data Setting and Search

■ DataFrame에 isin() 메소드 적용

```
In [335]: df = pd.DataFrame({'no': [1, 2, 3], 'ha': ['a', 'b', 'c'], 'hi': ['m', 'n', 'o']})
```

```
In [336]: val = ['a', 'n', 1, 3]
```

```
In [337]: df
```

```
In [338]: df.isin(val)
```

■ 특정 열에서 값을 매칭

```
In [339]: val = {'ha': ['a', 'c'], 'no': [1, 2]}
```

```
In [340]: df.isin(val)
```

Data Setting and Search

■ DataFrame에 isin() 메소드를 any() 또는 all() 메소드와 조합 사용

```
In [341]: val = {'ha': ['a', 'c'], 'hi': ['m', 'o'], 'no': [1, 2]}
```

```
In [342]: mask = df.isin(val).all(1)
```

```
In [343]: df[mask]
```

■ Take 메소드

→ DataFrame, Series, Index의 take() 메소드

```
In [344]: index = pd.Index(np.random.randint(0, 1000, 6))
```

```
In [345]: index
```

```
In [346]: positions = [0, 2, 5]
```

```
In [347]: index[positions]
```

```
In [348]: index.take(positions)
```

```
In [349]: ser = pd.Series(np.random.randn(10))
```

```
In [350]: ser.iloc[positions]
```

```
In [351]: ser.take(positions)
```

Data Setting and Search

■ DataFrame에 take()메소드 적용

```
In [352]: df = pd.DataFrame(np.random.randn(5, 3))
In [353]: df.take([1, 4, 3])
In [354]: df.take([0, 2], axis=1)

In [355]: d = {'one': [1.5, 2.2, -3.0], 'two': [1.0, -1.2, 5.0], 'three': [-1.1, 2.0, 4.0]}
In [356]: df = pd.DataFrame(d, index = ['a', 'c', 'f'])
In [357]: df['four'] = 'ha'
In [358]: df['five'] = df['one'] > 0
In [359]: df
In [360]: df1 = df.reindex(['a', 'b', 'c', 'd', 'e', 'f'])
In [361]: df1
```

Missing Data Processing

■ 손실값 탐지 – isna(), notna()

```
In [362]: df1['one']
```

```
In [363]: pd.isna(df1['one'])
```

```
In [364]: df1['four'].notna()
```

Missing Data Processing

■ 손실 데이터의 계산 → 손실값의 방송

```
In [367]: d1 = {'one': [1.0, 2.0, 3.0], 'two': [4.0, 5.0, 6.0]}
```

```
In [368]: df1 = pd.DataFrame(d1, index = ['a', 'b', 'c'])
```

```
In [369]: df2 = df1.copy()
```

```
In [370]: df2.loc['d'] = np.nan
```

```
In [371]: df2['three'] = 2.0
```

```
In [372]: df2.iloc[1:2, 1:2] = np.nan
```

```
In [373]: df1
```

```
In [374]: df2
```

```
In [375]: df1 + df2
```


Missing Data Processing

■ 비워있거나 모두 NA인 Series의 합은 0, 곱은 1

```
In [376]: pd.Series([np.nan]).sum()
```

```
In [377]: pd.Series([]).sum()
```

```
In [378]: pd.Series([np.nan]).prod()
```

```
In [379]: pd.Series([]).prod()
```

■ GroupBy – NA는 자동 제외

```
In [380]: df2
```

```
In [381]: df2.groupby('two').mean()
```

Missing Data Processing

■ 손실 데이터 채우기

→ fillna() – NA를 스칼라 값으로 대치

```
In [382]: df2.fillna(0)
```

```
In [383]: df2['one'].fillna('missing')
```

→ fillna() – method='pad' 적용

```
In [384]: df2
```

```
In [385]: df2.fillna(method='pad')
```

Missing Data Processing

■ fillna() – 팬더스 객체를 인수로 적용

```
In [386]: df2.loc['c', 'three'] = np.nan
```

```
In [387]: df2
```

```
In [388]: df2.mean()
```

```
In [389]: df2.fillna(df2.mean())
```

Missing Data Processing

■ fillna() – DataFrame 생성

```
df = pd.DataFrame([[np.nan, 2, 0, np.nan], [3, 4, np.nan, 1], [np.nan, 5, np.nan, 2], [np.nan, 1, 2, 3]], columns=list('ABCD'))  
df
```

■ NaN 요소들을 0으로 대체

■ non-null 값들의 전파

```
df.fillna(0)  
df  
df.fillna(method='ffill')
```

Missing Data Processing

■ NaN 요소들의 대치

```
val = {'A': 0, 'B': 1, 'C': 2, 'D': 3}  
df.fillna(value=val)
```

■ NaN 요소들의 대치 : limit=1

```
df.fillna(value=val, limit=1)  
df
```

■ 코드 [387]의 df2에 동적할당

```
In [391]: df2.iloc[2:3, 2:3] = 2.0
```

```
In [392]: df2
```

Missing Data Processing

■ 축에 따른 dropna 적용

```
In [393]: df2.dropna(axis=0)
```

```
In [394]: df2.dropna(axis=1)
```

```
In [395]: df2['two'].dropna()
```

■ 손실값 NaT를 갖는 df 생성

```
df = pd.DataFrame({'name': ['haena', 'suho', 'naeun'],  
...                'hobby': ['jogging', 'reading', np.nan],  
...                'born': [pd.NaT, pd.Timestamp('2001-01-01'), pd.NaT]})  
df
```

Missing Data Processing

■ 최소 1개의 요소가 손실된 행 또는 열을 제거

```
df.dropna()
```

```
df.dropna(axis='columns')
```

■ 모든 요소들이 손실된 행을 제거

```
df.dropna(how='all')
```

■ 최소 2개의 NA값들을 가진 행만을 제거

```
df.dropna(thresh=2)
```

Missing Data Processing

- 옵션에 해당하는 열에 손실값이 있는 행을 제거
- inplace - 저장하는 옵션

```
df.dropna(subset=['name', 'born'])    df.dropna(subset=['hobby'])  
df.dropna(inplace=True)  
df
```

- 손실값의 대체 – replace() 메소드
- ser 요소 치환

```
In [396]: ser = pd.Series([0, np.nan, 2, 3, 5])  
In [397]: ser  
In [398]: ser.replace(np.nan, 1.0)  
In [399]: ser.replace({np.nan: 1, 5: 4})
```


Missing Data Processing

■ 복수의 요소들을 하나의 값으로 치환

```
In [400]: ser.replace([0, 2], 1)
```

```
In [401]: ser.replace([np.nan, 5], [1, np.nan])
```

■ DataFrame에 replace() 메소드 적용

```
In [402]: df = pd.DataFrame({'A': [0, 1, np.nan], 'B': [3, 4, 5]})
```

```
In [403]: df.replace({'A': np.nan, 'B': 3}, 10)
```

Multindex

■ 멀티인덱스 객체 생성

→ 멀티인덱스 객체 생성 방법

→ `MultIndex.from_array()`

→ `MultIndex.from_tuples()`

→ `MultIndex.from_product()`

→ `MultIndex.from_frame()`

→ `from_tuples()` – list 생성

```
In [404]: li = [['ha', 'ha', 'hi', 'hi', 'ho', 'ho'], ['one', 'two', 'one', 'two', 'one', 'two']]
```

```
In [405]: li1 = list(zip(*li))
```

```
In [406]: li1
```

Multindex

■ Series 객체 생성 – 인수는 멀티인덱스

```
In [407]: ind = pd.MultiIndex.from_tuples(li1,names=['1st','2nd'])
```

```
In [408]: ind
```

```
In [409]: ser = pd.Series(np.random.randn(6), index=ind)
```

```
In [410]: ser
```

Multindex

■ 멀티인덱스 객체 생성 – from_product()

```
In [411]: iter = [['ha', 'hi', 'ho'], ['one', 'two']]
```

```
In [412]: pd.MultiIndex.from_product(iter, names=['1st', '2nd'])
```

■ 멀티인덱스 객체 생성 – from_frame()

```
In [413]: df = pd.DataFrame([['ha', 'one'], ['ha', 'two'], ['ho', 'one'], ['ho', 'two']], columns=['1st', '2nd'])
```

```
In [414]: pd.MultiIndex.from_frame(df)
```

Multindex

■ Multindex 자동 생성 – 배열리스트를 ser과 df에 넘김

```
In [415]: arr = [np.array(['ha', 'ha', 'hi', 'hi', 'ho', 'ho']), np.array(['one', 'two', 'one', 'two', 'one', 'two'])]
```

```
In [416]: ser = pd.Series(np.random.randn(6), index=arr)
```

```
In [417]: ser
```

```
In [418]: df = pd.DataFrame(np.random.randn(6, 3), index=arr)
```

```
In [419]: df
```

Multindex

■ Multindex 자동 생성 – 열 축의 지원

```
In [421]: df = pd.DataFrame(np.random.randn(3, 6), index=['A', 'B', 'C'], columns=ind)
```

```
In [422]: df
```

■ 계층적 인덱스의 인덱싱

→ 계층적 인덱싱 – 부분 선택

```
In [423]: df['ha']
```

```
In [424]: df['ha']['one']
```

Multindex

■ Series/DataFrame의 reindex() 메소드 적용

```
In [427]: ser.reindex(ind[:3])
```

```
In [428]: ser.reindex([(‘ho’, ‘one’), (‘ha’, ‘two’)])
```

■ 멀티인덱스를 .loc속성과 통합 인덱싱

```
In [429]: df = df.T
```

```
In [430]: df
```

```
In [431]: df.loc[(‘ha’, ‘two’)]
```

Multindex

■ loc로 특정 열 인덱싱

```
In [432]: df.loc[(‘ha’, ‘two’), ‘A’]
```

```
In [433]: df.loc[‘ha’]
```

■ df 객체의 부분 슬라이싱

```
In [434]: df.loc[‘ha’:’hi’]
```

■ 튜플의 형태의 범위로 슬라이싱

```
In [435]: df.loc[(‘hi’, ‘two’):(‘ho’, ‘one’)]
```

```
In [436]: df.loc[(‘hi’, ‘two’):’ho’]
```


Multindex

■ loc에 라벨 또는 튜플의 리스트 전달

```
In [437]: df.loc[[('ha', 'two'), ('ho', 'one')]]
```

■ 멀티인덱스를 순서정렬하기

→ `shuffle()` 함수 사용하여 요소들을 섞기

```
In [438]: li1
```

```
In [440]: li1
```

```
In [439]: np.random.shuffle(tup)
```

Multindex

■ 멀티인덱스의 순서 정렬

→ `sort_index()` 메소드

```
In [441]: ser = pd.Series(np.random.randn(6), index=pd.MultiIndex.from_tuples(l1))
```

```
In [442]: ser
```

```
In [443]: ser.sort_index()
```

```
In [444]: ser.sort_index(level=0)
```

```
In [445]: ser.sort_index(level=1)
```

Multindex

■ Multindex레벨에 이름 부여 – sort_index에 레벨 이름을 전달

```
In [446]: ser.index.set_names(['1st', '2nd'], inplace=True)
```

```
In [447]: ser.sort_index(level='1st')  
           = '2nd')
```

```
In [448]: ser.sort_index(level
```

Multindex

■ level에 의한 특정 축의 순서 정렬

```
In [449]: df  
In [450]: df.T.sort_index(level=1, axis=1)
```

Week 6 Outline

- Week 5 Review
- Pandas Data Processing
 - Data Selection
 - Data Setting and Search
 - Missing Data Processing
 - MultiIndex
- Pandas Data I/O and Type
 - Text File
 - Binary Data
 - SQL DB
 - Example
- Conclusion

Pandas Data I/O and Type

포맷 타입	데이터 유형	읽기 함수	쓰기 함수
텍스트	CSV	read_csv	to_csv
	JSON	read_json	to_json
	HTML	read_html	to_html
	local clipboard	read_clipboard	to_clipboard
이진 데이터	MS Excel	read_excel	to_excel
	HDF5 Format	read_hdf	to_hdf
	Feather Format	read_feather	to_feather
	Parquet Format	read_parquet	to_parquet
	Msgpack	read_msgpack	to_msgpack
	Stata	read_stata	to_stata
	SAS	read_sas	-
	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
	Google Big Query	read_gbq	to_gbq

Text File

■ CSV(comma-separated values) 파일

→ df 생성

```
In [451]: data = {'name': ['haena', 'naeun', 'una', 'bum', 'suho'],  
                 'age': [30, 27, 28, 23, 18],  
                 'address': ['dogok', 'suwon', 'mapo', 'ilsan', 'yeoyi'],  
                 'grade': ['A', 'B', 'C', 'B', 'A'],  
                 'score': [100, 88, 73, 83, 95]}
```

```
In [452]: df = pd.DataFrame(data, columns=['name', 'age', 'address', 'score',  
                                           'garde'])
```

```
In [453]: df
```

Text File

■ 원하는 디렉토리에 csv 파일 생성 – to_csv() 메소드 사용

```
In [454]: df.to_csv('C:/Users/chae/data/student_grade.csv')
```

■ read_csv()의 적용

```
In [455]: !type C:\Users\jchae\data\student_grade.csv
```

```
In [456]: df1=pd.read_csv('C:/Users/jchae/data/student_grade.csv')
```

```
In [457]: df1
```


Text File

■ 불필요한 열 Unnamed: 0을 삭제

```
In [458]: df1 = df1.iloc[0:5, 1:6]
```

```
In [459]: df1
```

■ read_csv() – header, nrows 적용

```
In [460]: df.to_csv(student_grade.csv)
```

```
In [461]: df2 = pd.read_csv('student_grade.csv',  
                             header=None,  
                             nrows=3)
```

```
In [462]: df2
```

Text File

■ 인수 index_col

```
In [463]: df2 = pd.read_csv('student_grade.csv', index_col=0)

In [464]: df2

In [465]: df2 = pd.read_csv('student_grade.csv', index_col=['name'])

In [466]: df2
```

Text File

■ 인수 names

```
In [467]: df2 = pd.read_csv('student_grade.csv', names=['  
            No', 'name',  
            'age', 'address', 'score', '  
            grade'], nrows=3)
```

```
In [468]: df2
```

■ 열의 특정 위치를 NaN으로 지정

```
In [469]: to_na = {'address': ['mapo', 'NA'], 'score': [83  
    ]}
```

```
In [470]: df2 = pd.read_csv('student_grade.csv', na_valu  
    es=to_na)
```

```
In [471]: df2
```

Text File

■ 인수 skiprows

```
In [472]: df2 = pd.read_csv('student_grade.csv', skiprows=3)
```

```
In [473]: df2
```

■ 인수 sep

```
In [483]: ! type student_grade1.csv
```

```
In [484]: pd.read_csv('student_grade1.csv', sep='|')
```

Text File

■ JSON(JavaScript Object Notation)

→ DataFrame을 JSON 스트링으로 변환 : orient='split'

```
In [485]: dfj = pd.DataFrame([[ 'a', 'b'], [ 'c', 'd']], index=[ 'row1', 'row2'], columns=[ 'col1', 'col2'])
```

```
In [486]: dfj.to_json()
```

```
In [487]: dfj.to_json(orient='split')
```

→ DataFrame을 JSON 스트링으로 변환 : orient='records', 'index'

```
In [488]: dfj.to_json(orient='records')
```

```
In [489]: dfj.to_json(orient='index')
```

Text File

- DataFrame을 JSON 스트링으로 변환 : orient='columns', 'values'

```
In [490]: dfj.to_json(orient='columns')
```

```
In [491]: dfj.to_json(orient='values')
```

- DataFrame을 JSON 스트링으로 변환 : orient='table'

```
In [492]: dfj.to_json(orient='table')
```

Text File

■ json 파일 생성 및 저장

```
In [493]: df = pd.DataFrame(data, columns=['name', 'age', 'address', 'score', 'grade'])
```

```
In [494]: df.to_json('C:/Users/jchae/data/student_grade.json')
```

■ json 스트링 읽기

```
In [495]: pd.read_json('C:/Users/jchae/data/student_grade.json')
```

Text File

■ DataFrame 생성

```
In [496]: df = pd.DataFrame({'ha': [1, 2, 3, 4],  
                             'hi': ['a', 'b', 'c', 'd'],  
                             'ho': pd.date_range('2019-09-01', freq='d', periods=4),  
                             'hu': pd.Categorical(['a', 'b', 'c', 'd'])}, index=pd.Index(range(4)  
                             , name='ind'))  
  
In [497]: df  
  
In [498]: df.dtypes
```


Text File

■ json 파일을 저장하고 저장파일 읽기

```
In [499]: df.to_json('hello.json', orient='table')
```

```
In [500]: dfj = pd.read_json('hello.json', orient='table')
```

```
In [501]: dfj
```

```
In [502]: dfj.dtypes
```

Text File

■ HTML(Hyper Text Markup Language)

→ read_html()

```
In [503]: url = 'http://www.fdic.gov/bank/individual/failed/banklist.html'
```

```
In [504]: dfh = pd.read_html(url)
```

```
In [505]: dfh
```

Text File

■ to_html() – df 생성

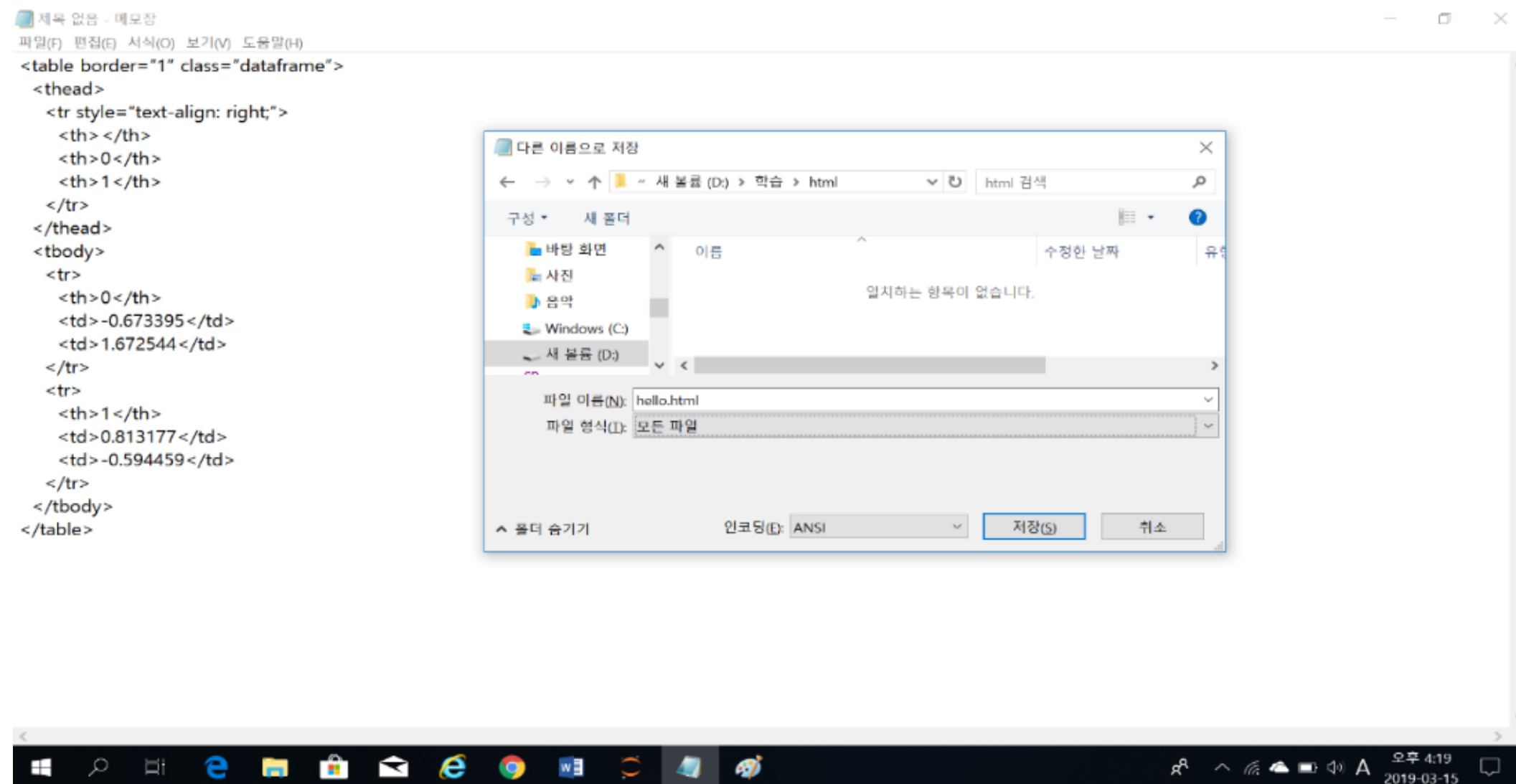
```
In [506]: df = pd.DataFrame(np.random.rand  
n(2,2))  
  
In [507]: df
```

■ df에 to_html() 적용

```
In [508]: print(df.to_html())
```

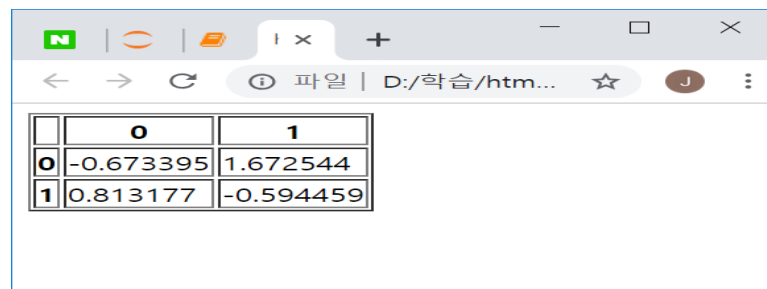
Text File

- hello.html – 메모장에서 앞의 html 코드를 저장



Text File

- 저장된 hello.html을 더블 클릭하여 웹브라우저로 읽기



A screenshot of a web browser window. The address bar shows the file path 'D:/학습/htm...'. The main content area displays a table with two columns and two rows of numerical data.

	0	1
0	-0.673395	1.672544
1	0.813177	-0.594459

- 클립보드(Clipboard) - 클립보드로 복사하고 읽기

```
In [509]: df = pd.DataFrame([[1, 2, 3], [4, 5, 6]], columns=['A', 'B', 'C'])
```

```
In [510]: df.to_clipboard(sep=',', index=False)
```

```
In [511]: pd.read_clipboard()
```

Out[511]:

```
      A,B,C
0  1,2,3
1  4,5,6
```

Binary Data

■ 엑셀(Excel) 파일

→ read_excel()로 엑셀파일 읽기

```
In [512]: df = pd.read_excel('shoppingcenter.xlsx')
```

```
In [513]: df
```

Binary Data

■ read_excel()로 엑셀파일의 특정 sheet 읽기

```
In [514]: df1 = pd.read_excel('shoppingcenter.xlsx', sheet_name='2018년 하반기 입  
력현황')
```

```
In [515]: df1
```

■ to_excel()로 d를 엑셀파일의 특정 sheet로 쓰기

```
In [ ]: df.to_excel('file_with_path.xlsx', sheet_name='Sheet1')
```

Binary Data

■ HDF5(Hierarchical Data Format 5)

→ HDF5 기술의 특징

- 복잡한 데이터 객체와 많은 종류의 메타데이터를 나타내는 다양한 데이터 모델
- PC부터 대용량 병렬시스템까지 다양한 연산플랫폼에서 실행되는 S/W 라이브러리
- C, C++, Fortran90 및 Java 인터페이스의 고차원 API를 실행하는 S/W 라이브러리
- 접속시간과 저장공간 최적화를 허용하는 풍부한 셋의 집종화된 성능
- 데이터를 관리, 조작, 시각화 및 분석하기 위함 어플리케이션

1 클래스 `pandas.io.pytables.HDFStore` – HDF5파일 생성 및 저장

```
In [516]: hfs = pd.HDFStore('D:/cjs/data/store.h5')
```

```
In [517]: hfs
```


Binary Data

2 df 객체

```
In [518]: ind = pd.date_range('1/1/2021', per  
iods=8)
```

```
In [519]: ser = pd.Series(np.random.randn(5)  
,  
                           index=['a', 'b',  
                                'c', 'd', 'e'])
```

```
In [520]: df = pd.DataFrame(np.random.rand  
n(8, 3),  
                             index=ind, columns=  
                             ['A', 'B', 'C'])
```

```
In [521]: country = ['KOR', 'US', 'ITALY']  
          mind = pd.MultiIndex  
               .from_product([count  
ry, ind])  
          col = ["item_%d" % i for i in rang  
e(1, 4)]  
          data = np.random.randn(24, 3)
```

3 df1 객체 생성

```
In [522]: df1 = pd.DataFrame(data, index=mind, col  
umns=col)
```

```
In [523]: df1
```

Binary Data

4 HDFStore 객체인 hfs에 동적 할당

```
In [524]: hfs['ser'] = ser  
hfs['df'] = df
```

```
In [525]: hfs['df1'] = df1
```

```
In [526]: hfs
```

```
In [524]:
```

5 hfs에서 df확인

```
In [527]: hfs['df']
```

Binary Data

5 df1 객체 삭제와 파일 닫기

```
In [528]: del hfs['df1']
```

```
In [529]: hfs
```

```
In [530]: hfs.close()
```

```
In [531]: hfs
```

■ 읽기 read_hdf와 쓰기 to_hdf

```
In [532]: df_s = pd.DataFrame({'A': list(range(5)), 'B': list(range(5))})
```

```
In [533]: df_s.to_hdf('D:/cjs/data/hfs_s.h5', 'table', append=True)
```

```
In [534]: pd.read_hdf('D:/cjs/data/hfs_s.h5', 'table', where=['index>2'])
```

Out[534]:

	A	B
3	3	3
4	4	4

Binary Data

■ HDFStore – 인수 dropna 적용

```
In [535]: df_na = pd.DataFrame({'A': [0, np.nan, 2], 'B': [1, np.nan, np.nan]})  
  
In [536]: df_na  
  
In [537]: df_na.to_hdf('D:/cjs/data/hfs_na.h5', 'with_na', format='table', mode='w')  
  
In [538]: pd.read_hdf('D:/cjs/data/hfs_na.h5', 'with_na')  
  
In [539]: df_na.to_hdf('D:/cjs/data/hfs_na.h5', 'with_na', format='table', mode='w', dropna=True)  
  
In [540]: pd.read_hdf('D:/cjs/data/hfs_na.h5', 'with_na')
```

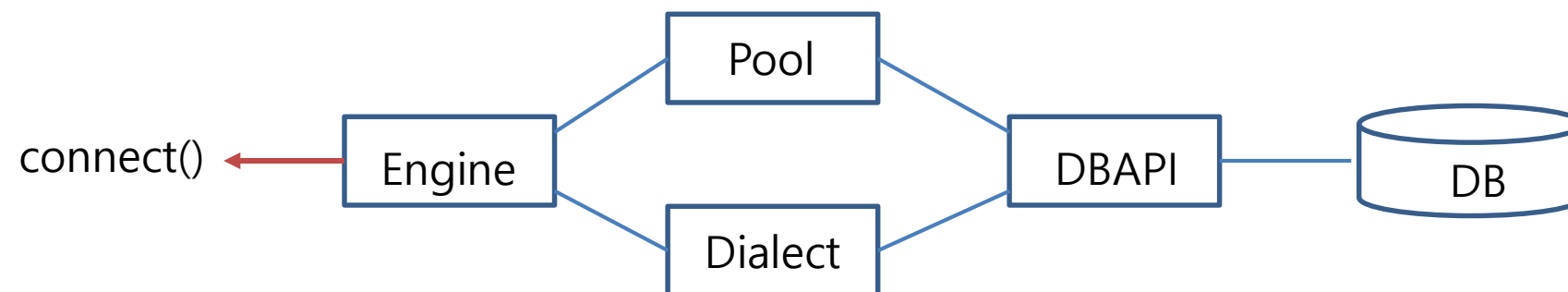
SQL DB

■ SQLAlchemy의 Engine 구성

■ SQLAlchemy

- 어플리케이션을 개발하기 위한 SQL 툴킷 및 객체 관계형 mapper(ORM)
- 호환성이 없는 시스템 사이에서 데이터를 변환토록하는 프로그래밍 기술
- MySQL, Oracle, PostgreSQL, MS SQL Server 등의 dialects들이 소통되도록 함
- SQL 데이터 베이스를 관리하고 연산 A

■ SQLAlchemy 구성도



SQL DB

■ engine 생성 – create_engine() 호출

- Dialect와 Pool은 데이터베이스 행태와 DBAPI의 모듈 기능을 해석
- Engine은 Dialect와 Pool을 참조
- user는 scott, 비밀번호는 tiger, host는 localhost, port는 5432, dbname은 mydb

```
In [ ]: from sqlalchemy import create_engine
```

```
In [ ]: engine = create_engine('postgresql://scott:tiger@localhost:5432/mydb')
```

■ 판다스 SQL관련 함수 적용

- 판다스 제공 주요 함수

종 류	기 능
read_sql_table	SQL 데이터베이스 테이블을 DataFrame으로 읽는다.
read_sql_query	SQL query를 DataFrame으로 읽는다.
read_sql	SQL query 또는 데이터베이스 테이블을 DataFrame으로 읽는다.
DataFrame.to_sql	DataFrame에 저장된 records를 SQL데이터베이스로 쓴다.

SQL DB

■ SQLite SQL db engine – 데이터가 memory에 저장된 임시의 SQLite db 사용

```
In [541]: from sqlalchemy import create_engine  
  
In [542]: engine = create_engine('sqlite:///memory:')
```

■ to_sql() – df 데이터를 데이터베이스에 쓰기

```
In [543]: df2.to_sql('data_db', engine)  
  
In [544]: df2.to_sql('data_db1', engine, chunksize=1000  
,
```

■ read_sql_table() – SQL 데이터베이스 테이블을 df로 읽기

```
In [545]: pd.read_sql_table('data_db', engine)
```

SQL DB

■ read_sql_table() – 인수 index_col 적용

```
In [546]: pd.read_sql_table('data_db', engine, index_col='name')
```

Out[546]:

	index	age	address	s core	grade
name					
haena	0	30	dogok	100	A
naeun	1	27	suwon	88	B
una	2	28	mapo	93	A
bum	3	23	ilsan	73	C
suho	4	18	yeoyi	85	B

```
In [547]: pd.read_sql_table('data_db', engine, columns=['name', 'grade'])
```

Out[547]:

	name	grade
0	haena	A
1	naeun	B
2	una	A
3	bum	C
4	suho	B

SQL DB

■ A

→ A

● read_sql_query() – SQL

```
In [548]: pd.read_sql_query('SELECT * FROM data_db', engine)
```

Out[548]:

	index	name	age	address	score	grade
A	0	0	haena	30	dogok	100
B	1	1	naeun	27	suwon	88
A	2	2	una	28	mapo	93
	3	3	bum	23	ilsan	73

● read_sql_query()

```
In [549]: pd.read_sql_query("SELECT name, address FROM data_db WHERE grade='A'", engine)
```

Out[549]:

	name	address
0	haena	dogok
1	una	mapo

SQL DB

■ to_sql()

```
In [550]: dfc = pd.DataFrame(np.random.randn(9, 3), columns=list('abc'))
```

```
In [550]: dfc.to_sql('data_ck', engine, index=False)
```

■ read_sql_query() – 인주 chunksize 식풍

```
In [551]: for chunk in pd.read_sql_query("SELECT * FROM data_ck", engine,
chunksize=3):
    print(chunk)
```

SQL DB

■ sqlite3

```
In []: import sqlite3

In []: con = sqlite3.connect(':memory:')

In []: data.to_sql('data', con)

In []: pd.read_sql_query("SELECT * FROM data", con)
```

Example

1 기상청 데이터 셋 - df 생성

```
In [552]: df = pd.read_csv('air_test.csv')
```

```
In [553]: df
```

2 chunksize=10000을 전달

```
In [554]: df_chunk = pd.read_csv('air_test.csv', chunksize=10000)
```

```
In [555]: df_chunk
```

■ chunksize=10000의 의미

- 기상청 데이터 292000개
- 10000행의 29개 chunk, 1600행의 chunk 1개
- 메모리를 효율적으로 사용

Example

3 TextFileReader 객체 df_chunk – 반복 처리(iteration)

```
In [556]: for chunk in df_chunk:
          print(chunk)
```

	site_code	test_no	average
0	104	1	0.008
1	102	8	101.000
...
9999	125	5	0.900

[10000 rows x 3 columns]

	site_code	test_no	average
10000	125	1	0.005
10001	125	3	0.050
...
19999	110	9	38.000

[10000 rows x 3 columns]

	site_code	test_no	average
290000	109	5	0.600
290001	109	6	0.017
...
291599	116	6	0.025

[1600 rows x 3 columns]

4 df_chunk의 연접

```
In [557]: type(chunk)
```

```
Out[557]: pandas.core.frame.DataFrame
```

```
In [558]: pd.concat(df_chunk, ignore_index=True)
```

```
Out[558]:
```

	site_code	test_no	average
0	104	1	0.888
1	102	8	101.00
2	102	9	64.00
...
291598	110	9	11.0
291599	116	6	0.025

291600 rows x 3 columns

Example

5 test_no별 측정데이터 구하기

```
In [559]: df_chunk = pd.read_csv('air_test.csv', chunksize=10,000)
```

```
In [560]: ser = pd.Series([])
           for chunk in df_chunk:
               ser = ser.add(chunk['test_no'].value_counts, fill_value=0)
           ser = ser.sort_values()
```

```
In [561]: ser
```

```
Out[561]: 1    48600.0
           3    48600.0
           5    48600.0
           6    48600.0
           8    48600.0
           9    48600.0
           dtype: float64
```

```
In [562]: ser[:5]
```

```
Out[562]: 1    48600.0
           3    48600.0
           5    48600.0
```

Week 6 Outline

- Week 5 Review
- Pandas Data Processing
 - Data Selection
 - Data Setting and Search
 - Missing Data Processing
 - MultiIndex
- Pandas Data I/O and Type
 - Text File
 - Binary Data
 - SQL DB
 - Example
- **scikit-learn example**
- Conclusion

scikit-learn과 pandas 관계

- scikit-learn과 pandas는 기본적으로 다른 존재
 - 직접적인 연동이나 데이터 입력은 어려운 상태
 - numpy array 형태나 혹은 list 형태등으로 변경 필요
 - 생각보다 어렵지는 않음

Why does Scikit-learn not directly work with, for example, **pandas**.DataFrame?

The homogeneous NumPy and SciPy data objects currently expected are most efficient to process for most operations. Extensive work would also be needed to support **Pandas** categorical types. Restricting input to homogeneous types therefore reduces maintenance cost and encourages usage of efficient data structures.

scikit-learn과 pandas 강점

■ scikit-learn

→ 데이터 훈련 및 예측

■ pandas

→ 데이터 import 및 정렬

scikit-learn과 pandas 예제

■ 예제 1

→ 타이타닉 데이터 예제

```
from sklearn.linear_model import LogisticRegression
import pandas as pd
```

```
url = 'http://bit.ly/kaggletrain'
train = pd.read_csv(url)
```

```
feature_cols = ['Pclass', 'Parch']
X = train.loc[:, feature_cols]
y = train.Survived
```

```
logreg = LogisticRegression()
logreg.fit(X, y)
```

```
url_test = 'http://bit.ly/kaggletest'
test = pd.read_csv(url_test)
X_new = test.loc[:, feature_cols]
```

```
new_pred_class = logreg.predict(X_new)
```

```
kaggle_data = pd.DataFrame({'PassengerId':test.PassengerId, 'Survived':new_pred_class}).set_index('PassengerId')
kaggle_data.to_csv('sub.csv')
```

```
train.to_pickle('train.pkl')
pd.read_pickle('train.pkl')
```

scikit-learn과 pandas 예제

■ 예제 2

→ 데이터 훈련 및 예측

```
from sklearn.neighbors import KNeighborsRegressor
import pandas as pd
```

```
url = 'http://bit.ly/kaggletrain'
train = pd.read_csv(url)
```

```
feature_cols = ['Pclass', 'Parch']
X = train.loc[:, feature_cols]
y = train.Survived
```

```
knr = KNeighborsRegressor()
knr.fit(X, y)
url_test = 'http://bit.ly/kaggletest'
test = pd.read_csv(url_test)
X_new = test.loc[:, feature_cols]
new_pred_class = knr.predict(X_new)
```

```
kaggle_data = pd.DataFrame({'PassengerId':test.PassengerId, 'Survived':new_pred_class}).set_index('PassengerId')
kaggle_data.to_csv('sub.csv')
```

```
train.to_pickle('train.pkl')
pd.read_pickle('train.pkl')
```

Remember this?

■ 생각보다 간단합니다.

→ 아래의 과정을 따르시면 됩니다.

→ 단 scikit-learn에만 적용

```
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
```

```
X, y = make_classification(n_samples=100, random_state=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
                                                    random_state=1)
```

① 데이터 로드

```
clf = MLPClassifier(random_state=1, max_iter=300)
```

② 모델 로드

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

③ 모델 훈련

```
clf.predict_proba(X_test[:1])
```

④ 데이터 샘플 테스트

```
clf.predict(X_test[:5, :])
```

⑤ 데이터 테스트

```
clf.score(X_test, y_test)
```

⑥ 모델 정확률 계산

기존 코드와의 관계

```
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
```

```
X, y = make_classification(n_samples=100, random_state=1)
① X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
                                                    random_state=1)
```

```
② clf = MLPClassifier(random_state=1, max_iter=300)
```

```
③ clf.fit(X_train, y_train)
   clf.predict_proba(X_test[:1])
```

```
⑤ clf.predict(X_test[:5, :])
   clf.score(X_test, y_test)
```

```
from sklearn.linear_model import LogisticRegression
import pandas as pd
```

```
url = 'http://bit.ly/kaggletrain'
train = pd.read_csv(url)
url_test = 'http://bit.ly/kaggletest'
test = pd.read_csv(url_test)
feature_cols = ['Pclass', 'Parch']
X = train.loc[:, feature_cols]
y = train.Survived
```

```
logreg = LogisticRegression()
```

```
logreg.fit(X, y)
```

```
X_new = test.loc[:, feature_cols]
new_pred_class = logreg.predict(X_new)
```

Week 6 Outline

- Week 5 Review
- Pandas Data Processing
 - Data Selection
 - Data Setting and Search
 - Missing Data Processing
 - MultiIndex
- Pandas Data I/O and Type
 - Text File
 - Binary Data
 - SQL DB
 - Example
- Conclusion

Week 6 Key Takeaway

■ 판다스 데이터 처리 학습

→ 데이터 선택/세팅 및 미싱 데이터 처리등 실습

■ 판다스 데이터 I/O 및 종류

→ 텍스트 파일 부터 SQL DB까지 종합적인 학습 진행

Assignment 2

- 5개 columns에 숫자 200개씩 담은 csv가 존재한다. 해당 파일을 기반으로 numpy, pandas 각각 활용하여서 통계적인 분석 값들을 도출하여 출력하시오.
 - 주요 분석 값: 평균, 표준편차, 최고, 최저, 5분위
(15%,35%,50%,65%,85%)
 - csv파일을 읽는 방식 제한 없음
 - 샘플개수 지정방식 제한 없음
 - 다만 판다스의 describe()를 포함한 기본적인 종합 통계값 계산 함수/메소드 사용 금지.
(다만 자체적으로 만든 함수 및 메소드는 100% 사용 가능)

다음 장에서는

- 판다스 고급
- 중간고사 상세 내용

lms