|  |  |
| --- | --- |
| 교육제목 | 데이터 기반 인공지능 시스템 엔지니어 양성 과정 |
| 교육일시 | 210929 |
| 교육장소 | YGL 학과장 및 자택(디스코드 사용 온라인 학습) |
| **교육내용** | |
| 오전 | 1. 행 인덱스 재배열   reindex() 메소드를 사용하면, 데이터프레임의 행 인덱스를 새로운 배열로 재지정 가능.  기존 객체를 변경하지 않고, 새로운 데이터 프레임 객체를 반환함.  새로운 배열로 행 인덱스를 재지정 :  DataFrame 객체.reindex(새로운 인덱스 배열)   * reindex : index를 재배치   import pandas as pd  #딕셔너리를 정의  dict\_data = {'c0':[1,2,3], 'c1':[4,5,6], 'c2':[7,8,9], 'c3':[10,11,12], 'c4':[13,14,15]}  # 딕셔서리를 데이터프레임으로 변환. 인덱스를 [r0, r1, r2]로 지정  df = pd.DataFrame(dict\_data, index=['r0', 'r1', 'r2'])  print(df)  print('\n')  #index 초기화 : reset\_index  df.reset\_index()  df = pd.DataFrame(dict\_data, index=['r0','r1','r2'])  print('[df]\n',df)  # sort : 인덱스로 정렬,특정 칼럼의 값으로 정렬  print(df.sort\_index(ascending=False)) # index로 정렬 : ascending( = False : 역으로정렬)  df.sort\_values(by=['c0','c1'], ascending=False)   1. 산술 연산   판다스 객체의 산술연산은 내부적으로 3단계 프로세스를 거침.  행/열 인덱스를 기준으로 모든 원소를 정렬한다.  동일한 위치에 있는 원소끼리 일대일로 대응시킨다.  일대일 대응이 되는 원소끼리 연산을 처리한다. 이때, 대응되는 원소가 없으면 NaN으로 처리한다  # 시리즈 연산  import pandas as pd  #딕셔너리 데이터로 판다스 시리즈 만들기  student1 = pd.Series({'국어':100, '영어':80,'수학':90, '역사':70})  student2 = pd.Series({'국어':90, '영어':85,'수학':95, '과학':100})  print(student1)  print('\n')  print('student1/200','\n',student1/200,'\n')  print('student1 + student2','\n',student1 + student2,'\n') # series ( +, -, \*, / ) series 또는 숫자  print('student1 - student2','\n',student1 - student2,'\n')  print('student1 \* student2','\n',student1 \* student2,'\n')  print('student1 / student2','\n',student1 / student2,'\n')  # 내부적으로 인덱스 정렬 후 각각의 인덱스로 계산 함  print(student1.add(student2, fill\_value=0)) # fill+value = n : 없는곳엔 n을 채워라  # add, sub, mul, div = 연산자와 다르게 자료가 없으면 원하는 값으로 대체할수있음  # 연산식에 NaN이 포함되면 결과는 Nan값으로 나타남 => 옵션 fill\_value=원하는 값  print(student1.sub(student2, fill\_value=0),'\n')  print(student1.mul(student2, fill\_value=0),'\n')  print(student1.div(student2, fill\_value=0),'\n')  # inf : 무한대,0으로 나눈 값   * dataframe 연산, 기존의 모듈에서 dataset을 불러들임   import seaborn as sns  titanic = sns.load\_dataset('titanic') # load\_dataset : 모듈에서 제공되는 dataset을 가져오는 메서드  type(titanic)  df = titanic.loc[ 100 : 201 ,['age','fare']] # loc[ : ] : 모든 행, 칼럼은 age,fare만  # 100~200인덱스 => df = titanic.iloc[100:201]  print(df.head(5)) # head(n) : 상위 n개만  #df.tail(10) # tail(n) : 하위 n개만  print(df + 10)  df.add(10, fill\_value=0) # 연산으로 NaN나오는거 0으로 바꿔서 계산하기  df1 = df.head(10)  df2 = df.tail(10)  print('df1.add(df2, fill\_value=0)=\n',df1.add( df2, fill\_value=0))  df = titanic.loc[ 100 : 201 ,['age','fare']]  # iloc[100 : 200] 100~200 / loc[100:200] 100~199  print(df.age.sum() / df.age.count()) # 평균나이출력  age\_avg = int(df.age.sum() / df.age.count())  #df\_age = df.age  # 평균 나이보다 적은 연령의 자료만 출력  for idx, age in enumerate(df\_age): # enumarate() : index와 data를 return  if age < age\_avg:  print("{} : {}".format(df.iloc[idx,0], df.iloc[idx,1]))  # 원소 검색 : .iloc 또는 .loc  print(titanic.dtypes)  print(titanic.columns)  df\_copy = []  for idx, item in enumerate(df.age):  if item < age\_avg:  df\_copy.append(df.iloc[idx])  print("{} : {}".format(df.iloc[idx,0], df.iloc[idx,1])) |
| 오후 | 1. 데이터 입출력   읽을때 메서드 : read\_  쓸때 메서드 : to\_  -csv파일 : 데이터 값을 쉼표로 구분하는 텍스트 파일.  데이터프레임: pandas.read\_csv("파일경로(이름)")  import pandas as pd  file\_path = "./test.csv"  df = pd.read\_csv(file\_path)  print(df)  df1 = pd.read\_csv(file\_path,header=None,index\_col=None)  print("df1 ===")  print(df1)  df2= pd.read\_csv(file\_path, index\_col='c0') # index컬럼을 지정  print('df2 ===')  print(df2)\   1. csv   read\_csv(옵션...) : 읽을 때  to\_csv(옵션..) : 쓸 때  path = 파일의 위치 포함한 파일명  sep : 필드를 구분하는 구분자 ','  header : 헤더가 정의되어 있는지, None : 헤더가 없음  index\_col : 인덱스로 사용될 컬럼명 , None : 인덱스 없음  문)  #tatanic 데이터를 load, age가 평균 age보다 적은 데이터를 가져와서  # 평균 나이보다 적은 연령의 자료만 출력  # save\_titanic.csv 파일에 저장(인덱스 제외)  # titanic.dtypes, titanic.columns  import seaborn as sns  titanic1 = sns.load\_dataset('titanic')  a\_list = [[1,23,5],[4,5,6]]  print(type(pd.DataFrame(a\_list)))  df\_copy =[] # 리스트  avg\_age = titanic.age.sum()/titanic.age.count()  for idx, age in enumerate(titanic.age):  if age < avg\_age:  df\_copy.append(titanic.iloc[idx]) # 2차원 배열의 리스트    pd.DataFrame(df\_copy).to\_csv("./titanic\_save\_file1.csv", index=None)  print(type(df\_copy) ," : ", type(pd.DataFrame(df\_copy)))#데이터타입 : list, DataFrame  data\_types = titanic.dtypes  column\_names = titanic.columns  columns = [ ] # 컬럼명을 리스트로  for idx, dtype in enumerate(data\_types):  if dtype in ['float64', 'int64']:  columns.append(column\_names[idx]) # 데이터 타입이 float/int 64  titanic\_select = titanic.loc[ : , columns] #df.loc[행 또는 배열,컬럼 또는 배열]  titanic\_select.to\_csv("./titanic\_save\_file.csv",index=None)   1. json   : 파일 형식  df = pd.read\_json("./read\_json\_sample.json") # 상대경로  # C:\Users/Lim/python\_src/read\_json\_sample.json > 절대경로  import os  print(os.getcwd()) # 작업하고 있는 경로 확인  print(df,'\n')  print(df.index)  df   1. html   url = "./sample.html"  tables = pd.read\_html(url)  tables  print(tables)  len(tables)  for i in range(len(tables)):  print("tables {}".format(i))  print(tables[i])  print('\n')  df = tables[0]  df.set\_index(['name'], inplace=True) # 'name'칼럼을 인덱스로 setting  df  df1 = tables[0]  df1.set\_index(['c0'],inplace=True)  df1   * 위키피디아 미국 etf 웹페이지에서 필요한 정보를 스크래핑하여 출력하기   url = "./sample.html"  tables = pd.read\_html(url)  tables  print(tables)  len(tables)  for i in range(len(tables)):  print("tables {}".format(i))  print(tables[i])  print('\n')  df = tables[0]  df.set\_index(['name'], inplace=True) # 'name'칼럼을 인덱스로 setting  df  df1 = tables[0]  df1.set\_index(['c0'],inplace=True)  df1  =정리=  정리  파일처리하는 방법  읽어오기  - read\_csv, read\_html, real\_excel, read\_json  저장하기  - df.to\_csv(파일명), df.to\_html, df.to\_excel, df.to\_json  json 파일은 구조가 DataFrame과 동일  csv 파일은 2차원 배열로 -> DataFrame으로 변경해서 사용  excel파일도 뭐..  df = pandas.DataFrame ..  행 df.loc[인덱스명], df.iloc[인덱스첨자] 여러개 start:end, [ , , ]  열 df.열명 df[열명]  원소 df.loc[[인덱스명],[컬럼명]]  df.iloc[[인덱스 첨자],[컬럼의 순서 정수]] |