|  |  |
| --- | --- |
| 교육제목 | 데이터 기반 인공지능 시스템 엔지니어 양성 과정 |
| 교육일시 | 211005 |
| 교육장소 | YGL 학과장 |
| **교육내용** | |
| 오전 | 1. 시계열 데이터   : 주식, 환율 등 금융 데이터를 다루기 위해 개발된 판다스는 시계열 데이터를 다루는 여러가지 유용한 기능을 제공함. 특히 시계열 데이터를 데이터프레임의 행 인덱스로 사용하면 시간으로 기록된 데이터를 분석하는 것이 매우 편리함    시계열 데이터 분석 순서  1. 데이터 확인 df.info(), df.head()  2. 날짜형으로 형변환 ( 컬럼을 추가 함 )  3. 시게열 데이터를 인덱스로 지정  4. 기존의 자료 삭제  5. 자료형 및 데이터 확인 df.info(), df.head()  예)  import pandas as pd  df = pd.read\_csv('./stock-data.csv')  print(df.head(),'\n')  print(df.info())  df['New\_date'] = pd.to\_datetime(df['Date']) #데이터 형변환  # 새로운 컬럼 New\_Date를 index로 설정  df.set\_index('New\_date', inplace=True)  # 기존의 컬럼 Date를 삭제  df.drop('Date',axis=1, inplace=True)  df.reset\_index(inplace=True) # index reset : 인덱스가 제거됨.  # df 속성을 이용하여 new\_date 열의 년월일 정보를 년, 월, 일...일일일ㅇ잉ㄹ잉로 구분  df['Year'] = df['New\_date'].dt.year  df['Month'] = df['New\_date'].dt.month  df['Day'] = df['New\_date'].dt.day  # 그래프 만들기  import matplotlib.pyplot as plt  x = df['New\_date']  y = df['High']  plt.plot(x, y)   1. 데이터프레임의 다양한 응용   < 함수 매핑 >  시리즈에 함수를 적용 : .apply(함수명)  DataFrame에 함수 적용 : .applymap(함수명)  예)  import seaborn as sns  titanic = sns.load\_dataset('titanic')  df = titanic.loc[:,['age','fare']] # age, fare 2개 열을 선택하여 데이터프레임 만들기, 칼럼명이니까 loc.  # 사용자 함수 정의, 10을 더하는 함수.  def add\_10(n):  return n+10  # 두 객체의 합을 구하는 함수  def add\_two\_obj(a,b):  return a + b  # 시리즈 객체에 10을 더하는 함수를 적용. -> 보통은 함수명(값, ...)  # 시리즈.apply(함수명) : 시리즈 각 원소에 함수를 적용.  df['New\_age'] = df['age'].apply(add\_10) # apply 함수를 쓰면 반복문처럼 하나하나에 함수 적용.  # add\_10(값 : df['age']의 각 원소의 값)  print(df,'\n')  # 람다함수로 똑같이 적용해보기  df['age'].apply(lambda x: add\_10(x))  df['age'].apply(lambda x: x +10)  df['age\_obj']=df['age'].apply(add\_two\_obj, b=20)  # DataFrame에 함수 적용하는 applymap() 사용하기  df\_map = df.applymap(add\_10)  print(df\_map.head())  #데이터프레임의 각 열에 대해서 함수 매핑  result = df.apply(add\_10, axis = 0) # 데이터프레임에 apply(함수명, axis=0), 각 열에 매핑함수  print(result.head())  # 최대값 - 최소값  df.apply(lambda x : x.max() - x.min())  result\_row = df.apply(add\_10, axis = 1) # 각 행에 매핑함수, 세로축시리즈 axis=1 // 가로축시리즈 -> axis=0  print(result\_row.head())  df = titanic.loc[ : ,['age','fare']]  df\_map = df.applymap(add\_10).head(3)  result = df.apply(add\_10, axis = 0).head(3)  result\_row = df.apply(add\_10, axis = 1).head(3)  def min\_max(x):  return x.max() - x.min()  df.apply(min\_max, axis=0)  df.apply(min\_max, axis=1)  # df['add'] = df.apply(lambda x : add\_two\_obj(x['age'] ,x['fare'], axis=1)  df['add'] = df.apply(lambda x : x['age'] + x['fare'], axis=1)   1. 데이터프레임 객체에 함수 매핑   : 데이터프레임 객체.pipe(매핑함수)  <pipe() 함수 알아보기>  df = titanic.loc[: , ['age','fare']]  # 각 열의 NaN 찾기 - 데이터프레임 전달하면 데이터프레임을 반환  def missing\_value(x):  return x.isnull()  # 각 열의 NaN 개수 반환 - 데이터프레임을 전달하면 시리즈 반환  def missing\_count(x):  return missing\_value(x).sum()  # 데이터 프레임의 총 NaN 개수 - 데이터프레임 전달하면 값을 반환  def total\_number\_missing(x):  return missing\_count(x).sum()  # 데이터 프레임에 함수 매핑  result\_df = df.pipe(missing\_value)  result\_ser = df.pipe(missing\_count)  print('┌--result\_df---┐ \n',result\_df.head(),'\n')  print('┌-result\_ser--┐ \n',result\_ser.head(),'\n')  print(type(result\_df))  result\_value = df.pipe(total\_number\_missing)   1. 열 재구성(열 순서 변경)   : 열 순서 변경 : DataFrame객체[재구성한 열 이름 리스트]  예)  titanic.columns.values # array로 나옴 = list()함수 사용하면 리스트형태로 바꿔줄 수 있음.  df = titanic.loc[0:4, 'survived':'age']  # 열 이름의 리스트 만들기  columns = list(df.columns.values)  # 열 이름을 알파벳 순으로 정렬  new\_columns = sorted(columns)  new\_columns  # 열 이름을 정렬한 데이터프레임 생성.  df\_sorted = df[new\_columns]  print(df\_sorted, '\n')  # 열 이름을 역으로 정렬하기  df\_reversed = df[list(reversed(columns))]  print(df\_reversed)   1. 열 분리  * 시리즈의 문자열 리스트 인덱싱 : Series 객체.str.get(인덱스)   예)  import pandas as pd  df = pd.read\_excel('./주가데이터.xlsx', engine='openpyxl')  # 1. 연월일을 문자열로 변경  df['연월일'] = df['연월일'].astype('str')  # 2. 연월일을 '-'기준으로 split  dates = df['연월일'].str.split('-')  dates  # 3. 데이터프레임에 '연','월','일' 칼럼 추가.  # 시리즈에서 각 리스트의 동일 위치의 원소를 추출하고자 할 때,  # 리스트.str.get(위치 인덱스)  df['연'] = dates.str.get(0) # dates 변수의 원소 리스트의 0번째 인덱스 값을 가져옴.  # df['연'] = dates[0][0] 0인덱스의 0번째 -> 0인덱스의 값 1개만 나오니까 str.get()을 쓰는게 맞음  df['월'] = dates.str.get(1) # dates 변수의 원소 리스트의 1번째 인덱스 값을 가져옴.  # df['월'] = dates[0][1] 0인덱스의 1번째  df['일'] = dates.str.get(2) # dates 변수의 원소 리스트의 2번째 인덱스 값을 가져옴.  df\_a = df.drop('연월일',axis=1)  print(df.info)  print(df.dtypes)  s\_list = pd.Series([[1, 2, 3],['abc','a','c'],['10','20','30']]) # 리스트를 시리즈로 바꿔준다.  s\_list.str.get(1) # 서로 다른 리스트 요소 중 동일 위치에서 값을 찾을 때.   1. 필터링   불린 인덱싱  : **조건식**을 적용, 각 원소가 조건을 만족하는지 True, False값으로 표시하여 불린 시리즈를 만들 수 있음.  이 불린 시리즈를 데이터프레임에 대입하면 조건을 만족하는 행들만 선택할 수 있음.   * 데이터프레임의 불린 인덱싱 : DataFrame 객체[불린 시리즈]   예)  # 나이가 10대(10~19세)인 승객만 따로 선택  mask\_age = (titanic.age >= 10) & (titanic.age < 20)  print(mask\_age)  print(type(mask\_age)) # 불린 시리즈  mask\_age.values  df\_teenage = titanic.loc[mask\_age, : ]  #df\_teenage.describe()  #df\_teenage.info()  Titanic  # 나이가 10세 미만, 60세 이상인 승객의 age, fare, sex, pclass, alive 컬럼 정보만 출력.  mask\_age2 = (titanic.age < 10) | (titanic.age >= 60)  #print(mask\_age2) # 불린 시리즈 만들었음.  df\_1060 = titanic.loc[mask\_age2, ['age','fare','sex','pclass','alive']]  df\_1060  # 나이가 10세 미만(0~9세)이고 여성인 승객만 따로 선택  df\_femi = titanic.loc[(titanic.age < 10) & (titanic.sex == 'female'), :]  df\_femi  # 함께 탑승한 형제 또는 배우자의 수(sibsp\*)가 3, 4, 5인 승객만 따로 추출 - 불린 인덱싱  mask4 = (titanic.sibsp > 2) & (titanic.sibsp <= 5)  df\_sib = titanic[mask4]  df\_sib   * 여기에서 isin() 메서드를 활용하면 더 간편하게 추출 가능   mask5 = titanic['sibsp'].isin([3, 4, 5]) <- sibsp에 3, 4, 5가 들어있나??  df\_isin = titanic.loc[mask5,['age','fare','sex','sibsp']]  df\_isin.head()  <정리>  원하는 데이터만 추출 : boolean mask 이용, isin() 메서드 이용   1. 데이터프레임 연결   : pandas.**concat**(데이터 프레임의 리스트)   * 연결 함수, 메소드는 대표적으로 concat(), merge(), join() 이 있음.   예)  # 데이터프레임 합치기 : pandas.concat(데이터프레임 리스트)  df1 = pd.DataFrame({'a': ['a0', 'a1', 'a2', 'a3'],  'b': ['b0', 'b1', 'b2', 'b3'],  'c': ['c0', 'c1', 'c2', 'c3']},  index=[0, 1, 2, 3])    df2 = pd.DataFrame({'a': ['a2', 'a3', 'a4', 'a5'],  'b': ['b2', 'b3', 'b4', 'b5'],  'c': ['c2', 'c3', 'c4', 'c5'],  'd': ['d2', 'd3', 'd4', 'd5']},  index=[2, 3, 4, 5])  df3 = pd.concat([df1,df2]) # concat() : 위에서 아래로 행을 붙이듯이 이어준다.  print('<df3> \n',df3,'\n')  # 기존의 인덱스 무시하고 새로운 인덱스 부여  df4 = pd.concat([df1,df2],ignore\_index=True)  print('<df4> \n',df4,'\n')  # 열로 붙이기  df5 = pd.concat([df1,df2], axis=1)  print('<df5> \n',df5,'\n')  # join 키워드  df6 = pd.concat([df1,df2], axis=1, join='inner') # join=outer(Default) : 합집합 // join=inner : 교집합만 출력  print('<df6> \n',df6,'\n')  # 데이터프레임과 시리즈 붙이기 : concat() -> 여러개의 데이터프레임을 연결.  # 시리즈 생성  sr1 = pd.Series(['e0', 'e1', 'e2', 'e3'], name='e')  sr2 = pd.Series(['f0', 'f1', 'f2'], name='f', index=[3, 4, 5])  sr3 = pd.Series(['g0', 'g1', 'g2', 'g3'], name='g')  # df1과 sr1을 좌우 열 방향으로 연결하기  df\_s1 = pd.concat([df1, sr1], axis=1)  df\_s1  # df1과 sr1을 좌우 열 방향으로 연결하기 ( 시리즈에 인덱스 존재함 )  df\_s2 = pd.concat([df2,sr2], axis=1)  df\_s2  # sr1과 sr3을 좌우 열 방향으로 연결하기( 결과 : 데이터프레임)  ssr1 = pd.concat([sr1, sr3], axis=1, sort=True)  print(ssr1,'\n','type :',type(ssr1),'\n')  # sr1과 sr3을 위아래 행방향으로 연결하기 ( 결과 : 시리즈 )  ssr2 = pd.concat([sr1,sr3], axis=0)  print(ssr2,'\n','type :',type(ssr2)) |
| 오후 | 1. 데이터 프레임 병합 : merge()   : pandas.merge(df\_left, df\_right, how='inner(교집합)/outer(합집합)', on=None)  How옵션으로 교집합만 출력할지, 합집합을 출력할지 결정 가능. On=None 옵션으로 두 데이터프레임에 공통으로 속하는 모든 열을 기준(키)으로 병합할 수 있음.  예)  import pandas as pd  df1 = pd.read\_excel('./stock\_price.xlsx')  df2 = pd.read\_excel('./stock\_valuation.xlsx')  print("df1 요소 갯수 {}개 , df2 요소 갯수 {}개".format(len(df1),len(df2)))  # 데이터프레임 합치기 - 교집합(inner)  merge\_inner = pd.merge(df1,df2)  print('★merge\_inner───────────────────────────────────────── \n', merge\_inner,'\n')  # 데이터프레임 합치기 - 합집합(outer)  merge\_outer = pd.merge(df1,df2, how='outer', on='id')  print('★merge\_outer───────────────────────────────────────── \n', merge\_outer,'\n')  # 데이터프레임 합치기 - 왼쪽프레임 기준, 키 값 분리, how='left', left\_on=컬럼명  merge\_left = pd.merge(df1,df2, how='left', left\_on='id', right\_on='id')  print("★merge\_left───────────────────────────────────────── \n", merge\_left,'\n')  merge\_right = pd.merge(df1,df2, how='right', left\_on='id', right\_on='id')  print('★merge\_right───────────────────────────────────────── \n', merge\_right,'\n')  # 불린 인덱싱과 결합하여 원하는 데이터 찾기  price = df1[df1['price'] < 50000] # 불린 시리즈 생성  value = pd.merge(price, df2)  df1.set\_index('id', inplace=True)  df2.set\_index('id', inplace=True)  # 데이터프레임결합 (join) - 교집합, 인덱스로 join.  df3 = df1.join(df2, how='inner') # how : left,right,outer,inner  df3   1. 그룹 연산   : 그룹 연산(분할) : Daraframe 객체.groupby(기준열)  예)  titanic['class'].unique() # unique() : 유일한 값 출력  df = titanic.loc[ :, ['age','sex','class','fare','survived']]  print(len(df))  # class 열을 기준으로 분할  grouped = df.groupby(['class'])  print(grouped.head())  grouped.get\_group('Third')  # 위 아래 똑같음.  df.loc[df['class'] =='Third', :]  df = titanic.loc[ :, ['age','sex','class','fare','survived']]  # class 열을 기준으로 분할  grouped = df.groupby(['class'])  # 각 그룹에 대한 모든 열의 표준편차를 집계하여 데이터프레임으로 반환  std\_all = grouped.std()  std\_all  # 각 그룹에 대한 fara열의 표준편차를 집계하여 시리즈로 반환  grouped.fare.std()  grouped\_two = df.groupby(['class','sex'])  # groupde\_two 그룹 객체에 연산 메소드 적용  average\_two = grouped\_two.mean()  average\_two  # group\_two 그룹 객체에서 개별 그룹 선택하기  grouped\_two.get\_group(('Third','female'))  # 여러 함수를 각 열에 동일하게 적용하여 집계  agg\_all = grouped.agg(['min','max'])  print('agg\_all = \n',agg\_all,'\n')  # 각 열마다 다른 함수를 적용하여 집계  agg\_sep = grouped.agg({'fare':['min','max'], 'age':'mean'})  print('agg\_sep = \n',agg\_sep)   1. sql 문장   select : 보여주는 리스트  from : 테이블 또는 뷰(데이터프레임)  wehre : 조인, 머지 조건)  group by : 원하는 컬럼으로 그룹, 컬럼의 조합도 가능  having : 그룹에 함수 적용... 조건  order by : 정렬  df.groupby(컬럼 또는 컬럼리스트  grouped.함수() (min,max,mean,std...)  grouped.get\_group(원하는 그룹의 컬럼 리스트)  grouped.agg([함수리스트]) grouped.agg( { 컬럼명: [함수리스트], 컬럼명:함수 / [함수리스트] } )  grouped.filter(lambda x: len(x) >= 200)  grouped.filter(lambda x: x.age.mean() < 30)  grouped.apply(lambda x: x.age.mean() < 30) -> 위랑 결과값은 같음.  예)  # 데이터 개수가 200개 이상인 그룹만을 필터링하여 데이터프레임으로 반환  grouped.filter(lambda x: len(x) >= 200)  # age 열의 평균이 30보다 작은 그룹만을 필터링하여 데이터프레임으로 반환  grouped.filter(lambda x: x.age.mean() < 30)  grouped.apply(lambda x: x.age.mean() < 30)   1. xs 인덱서   : df.loc[('First','female')] = df.xs('male',level='sex')  grouped = df.groupby(['class','sex'])  gdf = grouped.mean()  gdf  # class 값이 First인 행을 선택.  gdf.loc['First']  # class 값이 First, sex 값이 male인 행을 선택.  gdf.loc[('First','male')]  # sex값이 male인 행을 선택 : 그룹함수결과.xs(그룹컬럼의 값, level=그룹컬럼명)  gdf.xs('male',level='sex')  # 행, 열, 값, 집계에 사용할 열을 1개씩 지정 - 평균 집계  pdf1 = pd.pivot\_table(df, # 피벗할 데이터 프레임  index='class', # 행 위치에 들어갈 열  columns='sex', # 열 위치에 들어갈 열  values='age', # 데이터로 사용할 열  aggfunc='mean') # 데이터 집계 함수  pdf1  # 행, 열, 값, 집계에 사용할 열을 1개씩 지정 - 평균 집계  pdf2 = pd.pivot\_table(df, # 피벗할 데이터 프레임  index='class', # 행 위치에 들어갈 열 ['class', 'sex']  columns='sex', # 열 위치에 들어갈 열  values='age', # 데이터로 사용할 열 ['age', 'fare']  aggfunc=['mean','sum','count']) # 데이터 집계 함수  pdf2  pdf2.xs('First') # ('First', 'female'), ('male', level='sex')  # ('Second', 'male'), level=[0, 'sex'])  # ('mean', axis=1), ('mean','age',axis=1)  # ('max','fare',0),level=[0, 1, 2], axis=1)  # 행, 열, 값, 집계에 사용할 열을 2개 이상 지정 가능 - 평균 나이, 최대 요금 집계  pdf3 = pd.pivot\_table(df, # 피벗할 데이터 프레임  index=['class','sex'], # 행 위치에 들어갈 열  columns='survived', # 열 위치에 들어갈 열  values=['age','fare'], # 데이터로 사용할 열  aggfunc=['mean','max']) # 데이터 집계 함수  pdf3.xs('First') |