|  |  |
| --- | --- |
| 교육 제목 | python 10 |
| 교육 일시 | 2021. 10. 05 |
| 교육 장소 | 영우글로벌러닝 |
| **교육 내용** | |
| 오전 | import pandas as pd  # read.csv() 함수로 csv 파일을 가져와서 df로 변환  df=pd.read\_csv('./stock-data.csv')  df['New\_Date']=pd.to\_datetime(df['Date']) # 데이터 형변환을 위해  df.info()  # 새로운 컬럼 New\_Data를 index로 설정  df.set\_index('New\_Date', inplace=True)  df.head()  #기존의 컬럼 Data를 삭제  df.drop('Date', axis=1, inplace=True)  df.head()  # 1. 데이터 확인 df.info(), df.head()  # 2. 날짜형으로 형변환(컬럼을 추가 함)  # 3. 시계열 테이터를 인덱스로 지정  # 4. 기존의 자료 삭제  # 5. 자료형 및 데이터 확인 df.info(), df.head()  df.reset\_index(inplace=True) # index reset : 인덱스 제거  df.head()  # df 속성을 이용하여 New\_Date 열의 년월일 정보를 년, 월, 일로 구분  df['Year']=df['New\_Date'].dt.year  df['Month']=df['New\_Date'].dt.month  df['Day']=df['New\_Date'].dt.day  df.head()  ------------------------------------------------------------------------  import malplotlib.pyplot as plt  x=df['New\_Date']  y=df['High']  plt.plot(x,y)  -------------------------------------------------------------------------  import seaborn as sns  # titanic 데이터셋에서 age, fare 2개 열을 선택하여 데이터프레임 만들기  titanic=sns.load\_dataset('titanic')  # titanic info()  df=titanic.loc[ : ,['age','fare']]  df.head(2)  # 사용자 함수 정의 : 10을 더하는 함수  def add\_10(a):  return a+10  # 두 객체의 합을 더하는 함수  def add\_two\_obj(a,b):  return a+b  # 시리즈 객체에 10을 더하는 함수를 적용  # 시리즈.apply(함수명) : 시리즈 각 원소에 함수를 적용  df['new\_age']=df['age'].apply(add\_10) # add\_10(값 : df['age'의 원소의 값])  df['add\_obj']=df['age'].apply(add\_two\_obj, b=20)  df['age'].apply(lambda x: x+10)  # 데이터프레임.applymap(함수명) : 데이터프레임에 함수 적용  df\_map=df.applymap(add\_10)  # 데이터프레임의 각 열에 대해서 함수 매핑  rusult=df.apply(add\_10, axis=0) # 데이터프레임에 apply(함수명, axis=0)  df\_map.head()  # 최대값-최소값  df.apply(lambda x: x.max()-x.min())  result\_row=df.apply(add\_10, axis=1)  print(result\_row.head(2))  print(df.head(2))  df=titanic.loc[ : ,['age','fare']]  df.head()  df.applymap(add\_10).head()  df.apply(add\_10, axis=1).head()  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)  df.head()  # pipe 함수 알아보기  df=titanic.loc[ : , ['age','fare']]  # 각 열의 NaN 찾기 - 데이터프레임 전달하면 데이터프레임을 변환  def missing\_value(x):  return x.isnull()  # 각 열의 NaN 개수 반환 - 데이터프레임 전달하면 시리즈 변환  def missing\_count(x):  return missing\_value(x).sum()  # 데이터프레임의 총 NaN 개수 - 데이터프레임 전달하면 값을 반환  def total\_number\_missing(x):  return missing\_count(x).sum()  result\_df=df.pipe(missing\_count)  result\_df  result\_value=df.pipe(total\_number\_missing)  result\_value  titanic.columns.values  df=titanic.loc[0:4, 'survived': 'age']  # 열 이름의 리스트 만들기  columns=list(df.columns.values)  columns  # 열 이름을 알파벳 순으로 정렬  new\_columns=sorted(columns)  new\_columns  # 열 이름을 정렬한 데이터프레임 생성  df\_sorted=df[new\_columns]  # 열 이름을 역으로 정렬  df\_reversed=df[list(reversed(columns))]  new\_columns  -----------------------------------------------------------------------  # 열 분리  df=pd.read\_excel('./주가데이터.xlsx')  # 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' 컬럼을 추가  # 리스트.str.get(위치인덱스)  df['연']=dates.str.get(0) # dates 변수의 원소 리스트 0번재 인덱스 값  # df['연']=dates[0][0]........dates[1][0]  df['월']=dates.str.get(1) # dates 변수의 원소 리스트 1번재 인덱스 값  df['일']=dates.str.get(2) # dates 변수의 원소 리스트 2번재 인덱스 값  df  s\_list=[[1,2,3],['abc'],['10','20,','30']]  s\_list.str.get(1)  df.dtypes  # 나이가 10대(10~19세)인 승객만 따로 선택  mask\_age=(titanic.age >=10) & (titanic.age<20)  mask\_age.values  df\_teenage=titanic.loc[mask\_age, : ]  # df\_teenage.describe()  # df\_teenage.info()  # 나이가 10세 미만 60세 이상인 승객의 age, fare, sex, class, alive 컬럼 정보만 출력  mask1=(titanic.age<10) | (titanic.age>=60)  df\_alive=titanic.loc[mask1 , ['age','fare','sex','class','alive']]  df\_alive  # 나이가 10세 미만이고 여성인 승객만 따로 선택  df\_female=titanic.loc[(titanic.age<10) & (titanic.sex=='female')]  df\_female  # 함께 탑승한 형제 또는 배우자의 수가 3, 4, 5인 승객만 따로 추출  df\_sibsp=titanic.loc[(titanic.sibsp>=3)&(titanic.sibsp<=5)]  df\_sibsp  # isin() 매소드를 활용하여 간편하게 추출  mask3=titanic['sibsp'].isin([3,4,5])  # df\_isin=titanic[mask3]  df\_isin=titanic.loc[mask3, ['age','fare','sex','sibsp']]  df\_isin.head()  # 원하는 데이터만 추출 : boolean mask 이용, isin 매소드 이용 |
| 오후 | # 데이터프레임 합치기 : pandas .concat(데이터프레임리스트)  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]) # 위에서 아래로 행을 붙이듯이 이어줌  df3  # 기존의 인덱스 무시하고 새로운 인덱스 부여  df4=pd.concat([df1, df2], ignore\_index=True)  df4  # 열로 붙이기  df5=pd.concat([df1, df2], axis=1)  df5  # join 키워드 'outer', 'inner' : 교집합  df6=pd.concat([df1, df2], axis=1, join='inner')  df6  # 데이터프레임과 시리즈 연결 : 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')  # dr1과 sr1을 좌우 열 방향으로 연결하기 (index 없음)  df\_s1=pd.concat([df1, sr1], axis=1)  df\_s1  # df2와 sr2를 좌우 열 방향으로 연결하기(시리즈에 인덱스 존재 함)  df\_s2=pd.concat([df2, sr2], axis=1)  df\_s2  # sr1과 sr3를 좌우 열 방향으로 연결하기 (결과 : 데이터프레임)  st3=pd.concat([sr1, sr3], axis=1)  st3  # sr1과 sr3을 위 아래 행방향으로 연결하기 (결과 : 시리즈)  st4=pd.concat([sr1, sr3], axis=0)  st4  --------------------------------------------------------------------------  # 주식 데이터를 가져와서 데이터프레임 만들기  df1=pd.read\_excel('./stock\_price.xlsx') # 상대경로로 파일명 작성  df2=pd.read\_excel('./stock\_valuation.xlsx')  # 데이터프레임 합치기 - 교집합  merge\_inner=pd.merge(df1, df2)  merge\_inner  # 데이터프레임 합치기 - 합집합  merge\_outer=pd.merge(df1, df2, how='outer', on='id')  merge\_outer  # 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')  merge\_left  # 데이터프레임 합치기 - 왼쪽프레임 기준, 키 값 분리, how='right', right\_on= 컬럼명  merge\_right=pd.merge(df1, df2, how='right', left\_on='id', right\_on='id')  merge\_right  # 불린 인덱시오가 결합하여 원하는 데이터 찾기  price=df1[df1['price']<50000]  value=pd.merge(price, df2)  value  df1.set\_index('id', inplace=True)  df2.set\_index('id', inplace=True)  # 데이터프레임 결합 join - 교집합, 인덱스로 결합  df3=df1.join(df2, how='left') # how : left, right, outer, inner  df3  titanic['class'].unique()  df=titanic.loc[ :, ['age','sex','class','fare','survived']]  len(df)  # class 열을 기준으로 분할  grouped=df.groupby(['class'])  # 각 그룹에 대한 모든 열의 표준편차를 집계하여 데이터프레이믕로 반환  std\_all=grouped.std()  std\_all  # 각 그룹데 대한 fare 열의 표준편차를 집계하여 시리즈로 반환  grouped.fare.std()  #grouped.head()  grouped.get\_group('Third')  grouped\_two=df.groupby(['class','sex'])  # grouped\_two 그룹 객체에 연산 메소드 적용  average\_two=grouped\_two.mean()  average\_two  # grouped\_two 그룹 객체에서 개별 그룹 선택하기  grouped\_two.get\_group(('Third','female'))  # 여러 함수를 각 열에 동일하게 적용하여 집계하기  agg\_all=grouped.agg(['min','max'])  agg\_all  # 각 열마다 다른 함수를 적용하여 집계  agg\_sep=grouped.agg({'fare':['min','max'], 'age':'mean'})  agg\_sep  ------------------------------------------------------------------------  sql 문장  select 보여주는 리스트  form 테이블 또는 뷰(데이터프레임)  where (join, merge 조건)  group by (월하는 칼럼 또는 칼럼의 조합으로 그룹  having 그룹에 함수 적용을 해서 조건  order by 정렬  grouped=df.groupby(칼럼 또는 칼럼리스트)  grouped.함수() (min,max,mean,std...)  grouped.get\_group(원하는 그룹의 칼럼리스트)  grouped.agg([함수리스트])  grouped.agg({칼럼명:[함수리스트], 칼럼명:함수})  ------------------------------------------------------------------------  grouped=df.groupby(['class','sex'])  gdf=grouped.mean()  gdf  # class값이 First인 행을 선택  gdf.loc['First']  # class값이 First이고 sex 값이 male인 행을 선택  gdf.loc[('First','male')]  # sex값이 male인 행을 선택 : 그룹함수결과.xs(그룹칼럼의 값, level=그룹칼람명)  gdf.xs('male',level='sex')  -----------------------------------------------------------------------  # 행, 열, 값, 집계에 사용할 열을 1개씩 지정 - 평균 집계  pdf1=pd.pivot\_table(df, # 피벗할 데이터프레임  index='class', # 행 위치에 들어갈 열  columns='sex', # 열 위치에 들어갈 열  values='age', # 데이터로 사용할 열  aggfunc='mean') # 데이터 집계 함수  pdf1  pdf2=pd.pivot\_table(df, # 피벗할 데이터프레임  index='class', # 행 위치에 들어갈 열  columns='sex', # 열 위치에 들어갈 열  values='age', # 데이터로 사용할 열  aggfunc=['mean','sum']) # 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)  -------------------------------------------------------------------------------------  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  import seaborn as sns  '''  [Step 1 ~ 4] 데이터 준비  '''  # CSV 파일을 데이터프레임으로 변환  df = pd.read\_csv('./auto-mpg.csv', header=None)  # 열 이름 지정  df.columns = ['mpg','cylinders','displacement','horsepower','weight',  'acceleration','model year','origin','name']  # horsepower 열의 자료형 변경 (문자열 ->숫자)  df['horsepower'].replace('?', np.nan, inplace=True) # '?'을 np.nan으로 변경  df.dropna(subset=['horsepower'], axis=0, inplace=True) # 누락데이터 행을 삭제  df['horsepower'] = df['horsepower'].astype('float') # 문자열을 실수형으로 변환  # 분석에 활용할 열(속성)을 선택 (연비, 실린더, 출력, 중량)  ndf = df[['mpg', 'cylinders', 'horsepower', 'weight']]  # ndf 데이터를 train data 와 test data로 구분(7:3 비율)  X=ndf[['weight']] #독립 변수 X  y=ndf['mpg'] #종속 변수 Y  # train data 와 test data로 구분(7:3 비율)  from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=10)  print('훈련 데이터: ', X\_train.shape)  print('검증 데이터: ', X\_test.shape)  print('\n')  '''  Step 5: 비선형회귀분석 모형 - sklearn 사용  '''  # sklearn 라이브러리에서 필요한 모듈 가져오기  from sklearn.linear\_model import LinearRegression #선형회귀분석  from sklearn.preprocessing import PolynomialFeatures #다항식 변환  # 다항식 변환  poly = PolynomialFeatures(degree=2) #2차항 적용  X\_train\_poly=poly.fit\_transform(X\_train) #X\_train 데이터를 2차항으로 변형  print('원 데이터: ', X\_train.shape)  print('2차항 변환 데이터: ', X\_train\_poly.shape)  print('\n')  # train data를 가지고 모형 학습  pr = LinearRegression()  pr.fit(X\_train\_poly, y\_train)  # 학습을 마친 모형에 test data를 적용하여 결정계수(R-제곱) 계산  X\_test\_poly = poly.fit\_transform(X\_test) #X\_test 데이터를 2차항으로 변형  r\_square = pr.score(X\_test\_poly,y\_test)  print(r\_square)  print('\n')  # train data의 산점도와 test data로 예측한 회귀선을 그래프로 출력  y\_hat\_test = pr.predict(X\_test\_poly)  fig = plt.figure(figsize=(10, 5))  ax = fig.add\_subplot(1, 1, 1)  ax.plot(X\_train, y\_train, 'o', label='Train Data') # 데이터 분포  ax.plot(X\_test, y\_hat\_test, 'r+', label='Predicted Value') # 모형이 학습한 회귀선  ax.legend(loc='best')  plt.xlabel('weight')  plt.ylabel('mpg')  plt.show()  plt.close()  # 모형에 전체 X 데이터를 입력하여 예측한 값 y\_hat을 실제 값 y와 비교  X\_ploy = poly.fit\_transform(X)  y\_hat = pr.predict(X\_ploy)  plt.figure(figsize=(10, 5))  ax1 = sns.kdeplot(y, label="y")  ax2 = sns.kdeplot(y\_hat, label="y\_hat", ax=ax1)  plt.legend()  plt.show() |