**교육일지**

**교육 제목 : 데이터프레임 다양한 응용**

**교육 장소 : YGL C6 강의실**

**교육 일시 : 2021/10/05**

**df['New\_Data'] = pd.to\_datetime(df['Data']) #데이터 형변환을 위해 컬럼 추가**

**df.info()**

**df.set\_index('New\_Data', inplace=True) #새로운 컴럼 New Data를 indexfh tjfwjd**

**df.drop('Data', axis=1, inplace=True) #기존의 컬럼 Data를 삭제**

**df.reset\_index(inplace=True) # index reset : 인덱스 제거**

**#dt 속성을 이용하여 new\_Data 열의 년월일 정보를 년, 월,일 로 구분**

**df['Year'] = df['New\_Date'].dt.year**

**df['Month'] = df['New\_Date'].dt.month**

**df['Day'] = df['New\_Date'].dt.day**

**# 사용자 함주 정의 : 10을 더하는 함수 # 두 객체의 합을 구하는 함수**

**def add\_10(a): def add\_two\_obj(a,b):**

**return a+10 return a+b**

**#시리즈 객체에 10을 더하는 함수를 적용-> 함수명(값, ..)**

**#시리즈.apply(함수명) : 시리즈의 각 원소에 함수를 적용**

**df['new\_age'] = df['age'].apply(add\_10) # add\_10(값 : df['age'의 원소의 값])**

**#시리즈 .apply(함수명) : 시리즈 각 원소에 함수 적용**

**#데이터프레임.applymap(함수명) : 데이터프레임에 함수 적용**

**df\_map = df.applymap(add\_10)**

**#데이터프레임의 각 열에 대해서 함수 매핑**

**result = df.apply(add\_10, axis=0)# 데이터프레임에 apply(함수명,axis=0)**

**#최대값 - 최소값**

**df.apply(lambda x: x.max() - x.min())**

**result\_row = df.apply(add\_10, axis=1 #세로축 시리즈**

**#pipe 함수 알아보기**

**df = titanic.loc[ :, ['age','fare']]**

**#각 열의 NaN 찾기 - 데이터 프레임 전달하면 데이터 프레임 반환**

**def missing\_value(x):**

**return x.isnull()**

**#각 열의 NaN 개수 반환 - 데이터 프레임 전달하면 시리즈 반환**

**def missing\_count(x):**

**return missing\_value(x).sum()**

**#데이터프레임의 총 NaN 개수 - 데이터 프레임 전달하면 값을 반환**

**def total\_number\_missing(x):**

**return missing\_count(x).sum()**

**#데이터 프레임 합치기 - 교집합**

**merge\_inner = pd.merge(df1,df2)**

**merge\_inner**

**#데이터 프레임 합치기 - 합집합**

**merge\_outer = pd.merge(df1,df2, how='outer', on='id')**

**merge\_outer**

**#데이터프레임 합치기 - 왼쪽 프레임 기준, 키 값 분리, how='left'**

**merge\_left = pd.merge(df1,df2, how='left', left\_on='id', right\_on='id')**

**merge\_right = pd.merge(df1,df2, how='right', left\_on='id', right\_on='id')**

**#불린 인덱싱과 결합하여 원하는 데이터 찾기**

**price = df1[df1['price']<50000]**

**value = pd.merge(price,df2)**

**#데이터 프레임 결합(join) - 교집합, 인덱스로 병합**

**df3 = df1.join(df2, how='left') #how : lefr, right, outer, inner**

**sql 문장**

**selct 보여주는 리스트 from 테이블 또는 뷰(데이터 프레임) where (조인, 머지 조건) group by (원하는 컬럼 또는 컬럼의 조합으로 그룹 having 그룹에 함수 적용을 해서 조건 order by 정렬grouped = df.groupby(컬럼 또는 컬럼 리스트) grouped.함수() (min,max,mean,std ...) grouped.get\_group(원하는 그룹의 컬럼 리스트) grouped.agg([함수리스트]) grouped.agg({컬럼명:[함수리스트], 컬럼명:함수})데이터 개수가 200개이상인 그룹만을 필터링하여 데이터 프레임으로 반환 grouped.filter(lambda x : x.age.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**

**머신러닝**

**[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()**