Week2

<chapter 5>

[데이터 파악 시 사용하는 명령어]

**(데이터프레임).head( )** : 데이터의 앞에서부터 5번째 행까지 출력

: 괄호에 숫자 입력 시, 입력한 행까지 데이터 출력

ex) exam.head( ) #exam데이터프레임의 5행까지 출력

ex) exam.head(10) #exam데이터프레임의 10행까지 출력

**(데이터프레임).tail(** **)** : 데이터의 뒤에서부터 5번째 행까지 출력

: 괄호에 숫자 입력 시, 입력한 행까지 데이터 출력

ex) exam.tail( ) #exam데이터프레임의 뒤에서부터 5번째 행까지 출력

ex) exam.tail(10) #exam데이터프레임의 뒤에서부터 10번째 행까지 출력

**(데이터프레임).shape** : 데이터프레임의 행, 열 개수를 알려줌.

: (=데이터프레임의 크기 알려줌)

: 함수 아님. 데이터프레임이 가지고 있는 attribute(속성)임.

: 따라서 명령어 뒤에 괄호 입력 X

ex) exam.shape #exam데이터프레임의 행, 열 개수 출력

**(데이터프레임).info( )** : 데이터프레임에 들어있는 모든 변수의 속성을 한눈에 출력.

ex) exam.info( ) #exam데이터프레임의 변수 속성을 보여줌

**(데이터프레임).describe( )** : 데이터프레임의 (숫자)변수 값을 요약한 요약 통계량을 출력.

(나타나는 요약 통계량) 빈도(값의 개수), 평균, 표준편차, 최소값, 1사분위수, 중앙값, 3사분위수, 최대값

: 문자 변수의 요약 통계량도 출력하려면

**(df명).describe(include=’all’)** 으로 입력.

Cf) NaN은 결측치(누락 값)의미.

숫자 변수의 요약통계량 나타낼 때, 문자변수의 경우엔 NaN 출력되고,

문자 변수의 요약통계량 나타낼 때, 숫자변수의 경우엔 NaN 출력됨

ex) exam.describe( ) #exam데이터프레임에 있는 변수의 각종 통계량 보여줌

ex) exam.describe(include=’all’) #exam데이터프레임에 있는 문자 변수의 통계량까지 출력함.

[함수 종류에 따른 표기법 차이]

**내장함수**

: 함수명( )

ex) sum(var), max(var)

**패키지함수**

: 패키지명.함수명( )

: 패키지함수 사용하기 전에 패키지를 로드하는 과정 있어야 함.

* import 패키지명 (as 약어)

ex) import pandas as pd

pd.read\_csv(‘exam.csv’)

pd.DataFrame({‘x’ : [1,2,3]})

**매서드함수**

: 변수명.함수명( )

: 변수가 지니고 있는 함수.

: 변수의 자료구조에 따라 사용할 수 있는 매서드 함수가 다름.

* head( ) : 데이터프레임에 들어있는 매서드함수. 따라서 리스트에서는 사용할 수 없음.
* 변수의 자료구조(데이터프레임/리스트)는 type( ) 이용하면 알 수 있음.

ex) df.head( ), df.info( )

+) **어트리뷰트(attribute)**

: 변수명.어트리뷰트명

: 변수가 지니고 있는 값

: 함수 X. 따라서 괄호 입력 X

: 변수의 자료구조에 따라 사용할 수 있는 어트리뷰트 다름

Cf) 매서드와 어트리뷰트의 차이

: 매서드 = 기술 / 어트리뷰트 = 능력치

: 매서드는 명령하면 실행하는 기술이라 생각하면 됨. 변수를 이용해서 매서드를 실행

: 어트리뷰트는 능력치라 생각. 변수의 특징을 알고자 할 때, 어트리뷰트라는 값을 출력해서 봄.

[변수명 바꾸기]

: **데이터프레임 = 데이터프레임.rename(columns = { ‘기존변수명’ : ‘바꿀변수명’}}**

: 기존 데이터 변형 시, 꼭 복사본 만들어서 복사본으로 작업하기

+) **데이터프레임명.copy( )** : 복사본 만들기

Ex) df\_new = df\_raw.copy( ) #df\_raw의 복사본을 df\_new라고 함

p.116 <파생변수>

: 기존의 변수를 변형해 만든 변수

**데이터프레임.plot.hist( )**

: 히스토그램을 만드는 함수

Ex) mpg[‘total’].plot.hist( )

**데이터프레임.plot.bar( )**

: 막대그래프를 만드는 함수

Ex) count\_test.plot.bar( )

Ex) count\_test.plot.bar(rot = 0) #rot은 축 이름을 원하는 각도만큼 회전해 주는 파라미터

**np.where( )**

: 지정한 조건에 맞을 때와 맞지 않을 때 서로 다른 값을 반환하는 함수. 즉, 조건문 함수

: 중첩 조건문으로도 사용됨

: numpy패키지 내의 함수

Ex) mpg[‘test’] = np.where(mpg[‘total’] >= 20, ‘pass’, ‘fail’)

* Total 값이 20이상이면 pass를, 20 미만이면 fail을 부여

Ex) mpg[‘grade’] = np.where(mpg[‘total’]>=30, ‘A’,

np.where(mpg[total’]>=20, ‘B’, ‘C’))

* Total 값이 30이상이면 A를, 20이상이면 B를, 20미만이면 C를 부여
* 중첩 조건문

Ex) mpg[‘size’] = np.where( (mpg[‘category’] == ‘compact’) |

(mpg['category’] == ‘subcompact’) |

(mpg['category’] == ‘2seater’),

‘small’ , ‘large’ )

* Mpg의 category값이 compact / subcompact / 2seater이면 small을, 아니면 large 값을 부여
* 여러 조건 입력할 때 or을 뜻하는 | (shift + \) 이용

+) **데이터프레임명.isin( ) 함수**

Ex) mpg[‘size’] = np.where( mpg[‘category’].isin([‘compact’ , ‘subcompact’, ‘2seater’]),

‘small’ , ‘large’ )

* 바로 위의 ex와 같은 코드임
* 줄줄이 조건을 입력하지 않고, isin( ) 함수와 [ ]를 이용해서 간단하게 표현 가능

**데이터프레임.value\_counts( )**

: 변수의 값이 종류별로 몇 개씩 있는지, 값의 개수를 나타내는 빈도표를 만드는 함수

: 알파벳 순으로 내림차순 정렬해서 나타내고 싶을 땐 .sort\_index( )함수를 뒤에 붙여주면 됨.

Ex) count\_grade = mpg[‘grade’].value\_counts( ).sort\_index( )

* 이렇게 점(.)을 이용해서 메서드를 계속 이어서 작성하는 걸 ‘메서드 체이닝’이라고 함.

Ex) mpg[‘test’].value\_counts( )

<chapter 6>

[데이터 전처리 과정 시 사용하는 명령어]

**(데이터프레임).query( 조건 )** : 조건에 해당하는 데이터만 추출. 일종의 조건문

Ex) exam.query( ‘nclass == 1’ ) #exam에서 nclass가 1인 경우만 추출

Ex) exam.query(‘nclass == 1 | nclass == 3 | nclass == 5’) #nclass가 1/3/5인 경우 추출

+) **in과 [ ] 이용해서 더 간단한 코드로 나타내기**

Ex) exam.query( ‘nclass in [1,3,5]’ ) #nclass가 1/3/5인 경우 추출

**+) 외부 변수를 이용해서 행 추출**

: 변수명 앞에 @를 붙여서 조건 입력

Ex) var = 3

exam.query( ‘nclass == @var’ ) #nclass가 3인 경우 추출

cf) 데이터프레임 출력 제한 설정 풀기

: 아래의 코드 실행하면 됨

* pd.set\_option( ‘display.max\_rows’ , None) #모든 행 출력하도록 설정
* pd.set\_option( ‘display.max\_columns’ , None) #모든 열 출력하도록 설정

cf) 다시 출력 제한 설정하기

: 주피터랩 새로 실행 / 커널 새로 실행 / 아래 코드 실행

* pd.reset\_option( ‘display.max\_rows’ ) #행 출력 제한 되돌리기
* pd.reset\_option( ‘display.max\_columns’ ) #열 출력 제한 되돌리기
* pd.reset\_option( ‘all’ ) #모든 설정 되돌리기

**데이터프레임명[ ‘추출할 변수명’ ]** : 추출할 변수만 출력하기

Ex) exam[ ‘math’ ] #exam에서 math만 추출

Cf) 변수 1개만 출력할 경우, 자료구조가 ‘시리즈(Series)’로 바뀜. 데이터프레임 자료구조 형태로 유지하려면 여러 변수 출력하듯 변수명을 [ ] 로 한번 더 감싸야 함.

Ex) exam[ [ ‘math’] ] #exam에서 math만 추출. 자료구조는 데이터프레임 형태

+) **데이터프레임명[ [ ‘추출 변수1’, ‘추출 변수2’ ] ]** : 여러 변수 출력

Ex) exam[ [‘nclass’ , ‘math’ , ‘english’ ] ] #exam에서 nclass, math, English 출력

**데이터프레임명.drop( columns = ‘추출제외할변수’** ) : 특정 변수만 제외하고 나머지 모든 변수 추출

Ex) exam.drop(columns = ‘math’ ) #exam에서 math 변수 제외한 나머지 모든 변수 출력

+) **데이터프레임명.drop( columns = [‘추출제외변수’, ‘추출제외변수’ ] )**

: 출력 제외할 변수가 여러 개인 경우

Ex) exam.drop(columns = [‘math’ , ‘english’ ]) #math와 English 제외한 나머지 모든 변수 출력

**데이터프레임명.sort\_values( ‘정렬기준인 변수’ )** : 지정한 변수를 기준으로 오름차순 정렬

Ex) exam.sort\_values(‘math’) #math 오름차순 정렬

**데이터프레임명.sort\_values(‘정렬기준변수’ , ascending = False)** : 지정변수 기준 내림차순 정렬

Ex) exam.sort\_values(‘math’, ascending = False ) #math 내림차순 정렬

+) **정렬 기준 여러 개인 경우**

Ex) exam.sort\_values( [ ‘nclass’, ‘math’ ] ) #nclass 오름차순 정렬 후, math 오름차순 정렬

+) **정렬 기준 여러 개인데 정렬순서 다른 경우**

Ex) exam.sort\_values( [‘nclass’ , ‘math’ ] , ascending = [True, False] )

#nclass에 대해 오름차순 정렬 후, math에 대해 내림차순 정렬

**데이터프레임명.assign( 변수명 = 변수 만드는 공식 )** : 파생변수 추가하기

Ex) exam.assign( total = exam[‘math’] + exam[‘english’] + exam[‘science’] )

#total 파생변수 추가 / total은 수학+영어+과학점수

+) **여러 파생변수 한번에 추가하기**

Ex) exam.assign( total = exam[‘math’] + exam[‘english’] + exam[‘science’] ,

mean = (exam[‘math’] + exam[‘english’] + exam[‘science’]) / 3

)

# total 파생변수 추가 / total은 수학+영어+과학점수

# mean 파생변수 추가 / mean은 total의 평균점수

+) **조건문 이용해서 파생변수 추가하기**

Ex) exam.assign( test = np.where( exam[‘science’] >= 60, ‘pass’, ‘fail’ ) )

# 과학점수가 60점 이상이면 pass, 60점 미만이면 fail을 부여하여 test 파생변수 추가

+) **데이터프레임명.assign( 파생변수 = lambda 약어 : 변수 만드는 공식 내용 )**

: lambda 이용해서 데이터프레임명 대신 약어 사용하기

Ex) long\_name = pd.read\_csv(‘exam.csv’)

long\_name.assign(new = lambda x : x[‘math’] + x[‘english’] + x[‘science’] )

# 원래대로라면 long\_name.assign(new = long\_name[‘math’] + long\_name[‘english’] + long\_name[‘science’]) 가 입력되는 코드지만, lambda 이용해서 long\_name이라는 데이터프레임명을 x라는 약어 사용해서 간단하게 나타냄

+) **파생변수를 동시에 2개 이상 만들 때, 먼저 만든 파생변수를 이용해서 다음 파생변수 만들 경우,**

**이때는 반드시 lambda 이용해서 데이터프레임명을 약어로 입력해야 함. 이렇게 안하면 에러남**

Ex) exam.assign( total = exam[‘math’] + exam[‘english’] + exam[‘science’] , \

mean = lambda x : x[‘total’] / 3)

# total 파생변수 추가 / total은 수학+영어+과학점수

# mean 파생변수 추가 / mean은 total의 평균점수

+) **위의 코드를 좀 더 간단하게 나타내보자**

Ex) exam.assign( total = lambda x : x[‘math’] + x[‘english’] + x[‘science’] , \

mean = lambda x : x[‘total’] / 3)

**데이터프레임명.agg(요약값 할당할 변수명 = (‘요약할변수명’ , ‘사용 함수’)** : 전체 요약 통계량 구하기

Ex) exam.agg( mean\_math = (‘math’ , ‘mean’) #math 평균 구하기

Cf) 요약 통계량 함수 종류

mean( ) : 평균

std( ) : 표준편차

sum( ) : 합계

median( ) : 중앙값

min( ) : 최소값

max( ) : 최대값

count( ) : 빈도(개수)

**데이터프레임명.groupby(‘변수명’)** : 변수의 범주 별로 데이터 분리

Ex) exam.groupby( ‘nclass’ ) #nclass별로 분리

Cf) groupby( ) 함수는 기본적으로 변수를 인덱스로 바꾸도록 설정되어 있음. 변수를 인덱스로 바꾸지 않고 그대로 유지시켜려면, 변수명 입력 후, as\_index = False 라고 작성해주면 됨

Ex) exam.groupby( ‘nclass’ , as\_index = False)

+) **집단별 요약 통계량 구하기** : df.agg( ) + df.groupby( )

Ex) exam.groupby( ‘nclass’ ) . agg(mean\_math = (‘math’ , ‘mean’) #반별 수학 평균 구하기

+) **집단별 빈도(개수) 구하기**

Ex) mpg.groupby(‘drv’).agg(n = (‘drv’, ‘count’) #drv별 빈도 수 구하기

Cf) **위의 코드를 value\_counts( ) 하나만 이용해서 구할 수도 있음**

: 애초에 value\_counts( ) 함수가 변수 종류 별로 빈도 수를 구해주는 함수임

: 단, value\_counts( )함수는 자료구조가 시리즈(Series)라서 query( ) 적용 불가

: value\_counts( )함수에 query( ) 적용하려면 자료구조를 데이터프레임으로 바꿔준 후

적용

Ex) mpg[‘drv’].value\_counts( ) #drv별 빈도 수 구하기

**pd.merge(df1 , df2, how = ‘left’ , on = ‘기준변수명’**) : df1에 데이터프레임 가로로 합치기

: 데이터프레임을 가로로 합친다 = 변수(열)을 추가한다

: 입력한 기준변수명을 기준으로 두 df를 합침

**pd.concat( [df1, df2]** **)** : 데이터프레임 세로로 합치기

: 데이터프레임을 세로로 합친다 = 행을 추가한다

: 단, 세로로 데이터 합칠 때는 변수명이 같아야 함

: 만약 변수명이 다르면, pd.rename( )을 이용해서 변수명 같게 맞춘 후, 합치면 됨

<chapter 7>

**pd.isna(데이터프레임명)** : 데이터프레임에 결측치 존재 유무 확인

: True/False로 결측치 유무 알려줌

+) **pd.isna(데이터프레임명).sum( )** : 결측치의 빈도(개수) 알려줌

**(데이터프레임명).dropna(subset = ‘변수명’)** : 데이터프레임의 결측치가 있는 행 제거

: 제거할 결측치가 있는 열이 2개 이상일 경우, subset = [‘변수명’, ‘변수명’] 이런 식으로 [ ] 이용.

: 아무 변수도 지정하지 않으면 모든 변수에 결측치를 제거하고, 결측치가 없는 행만 출력해줌

Ex) df.dropna(subset = ‘score’) #score의 결측치 제거

Ex) df.dropna(subset = [‘score’ , ‘sex’] #score와 sex의 결측치 제거

Ex) df.dropna( ) #df에 존재하는 모든 결측치 제거한 행들만 출력

**(데이터프레임명).fillna(대체값)** : 결측치를 다른 데이터값으로 변경

Ex) exam[‘math’].fillna(55) #math의 결측치를 55로 대체

**(데이터프레임명).value\_counts(** **)** : 데이터에 이상치가 들어있는지 확인

: 뒤에 .sort\_index( )를 적용하면 빈도 기준 내림차순이 아니라, 변수 값 순서대로 정렬해서 출력

+) **이상치가 있을 경우, 이상치를 결측치로 변경한 후, 변경한 결측치를 제거해서 분석하는 과정으로 진행**

Ex) df[‘sex’].values\_counts( ).sort\_index( ) #df 데이터프레임의 sex변수에 이상치가 들어있는지 확인

**극단치 처리법) 1사분위수/3사분위수 구하기 > IQR 구하기 > 하한/상한 구하기 (여기까지 극단치 기준값 구하는 과정) > 극단치를 결측 처리 > 결측치 제거 및 분석**

Cf) IQR : 1사분위수와 3사분위수의 거리

Cf) 하한 : 1사분위수보다 IQR의 1.5배만큼 더 작은 값

상한 : 3사분위수보다 IQR의 1.5배만큼 더 큰 값

**(데이터프레임명).quantile( )** : 분위수 구하기

Ex) mpg[‘hwy’].quantile(.25) #하위 25%에 해당하는 1사분위수 구하기

Ex) mpg[‘hwy’].quantile(.75) #하위 75%에 해당하는 3사분위수 구하기

<chapter 8>

**sns.scatterplot(data=사용할 데이터, x = ‘x축으로 설정할 변수명’, y = ‘y축으로 설정할 변수명’)**

: 산점도 만들기

Ex) sns.scatterplot(data=mpg, x=’displ’, y=’hwy’) #x축은 displ, y축은 hwy로 하는 산점도 만들기

+) **(데이터프레임명).set(xlim = [나타낼 범위], ylim = [나타낼 범위])** : 나타낼 축 범위 설정하기

Ex) sns.scatterplot(data=mpg, x=’displ’, y=’hwy’).set(xlim = [3,6], ylim=[10,30])

#산점도에 나타낼 x축 범위는 3~6까지, y축 범위는 10~30까지로 설정

+) **산점도 표식의 색을 종류별로 다르게 표현 시, hue를 이용**

Ex) sns.scatterplot(data=mpg, x = ‘displ’, y = ‘hwy’, hue = ‘drv’)

# 산점도의 색을 drv별로 다르게 표현

**sns.barplot(data=사용할 데이터, x = ‘x축으로 설정할 변수명’, y=’y축으로 설정할 변수명’)**

: 막대 그래프 만들기

Ex) sns.barplot(data=df\_mpg, x = ‘drv’, y = ‘mean\_hwy’)

#x축은 drv, y축은 mean\_hwy로 하는 막대그래프 만들기

+) **막대그래프를 내림차순 정렬하기 위해서는 먼저 데이터프레임을 내림차순 정렬한 후,**

**막대그래프를 출력하기**

Ex) df\_mpg = df\_mpg.sort\_values(‘mean\_hwy’, ascending = False)

# df\_mpg를 mean\_hwy를 기준으로 내림차순 정렬

// cf)ascending=False 안쓰면 기본 오름차순 정렬

sns.barplot(data=df\_mpg, x = ‘drv’, y = ‘mean\_hwy’)

# 이미 df\_mpg를 내림차순으로 정렬했기 때문에 출력되는 막대그래프도

내림차순 정렬된 순으로 출력됨

**sns.countplot(data=사용할 데이터, x = ‘x축으로 사용할 변수명’)**

: 집단별빈도표 만들지 않고 바로 빈도막대그래프 만들기

: cf) sns.barplot( )을 이용하여 빈도막대그래프 만들려면

1)df.groupby( )와 df.agg( )이용해서 집단별빈도표를 만든 후,

2)sns.barplot( )으로 막대그래프 만들어야 함

+) **변수에 존재하는 데이터 값의 종류를 알고 싶을 때 unique( ) 이용**

Ex) mpg[‘drv’].unique( ) #mpg의 drv에 존재하는 값의 종류 출력

Ex) df\_mpg[‘drv’].unique( ) #df\_mpg의 drv에 존재하는 값의 종류 출력

+) **sns.countplot( )이용해서 만든 빈도막대그래프에서 화면상 나타나는 막대 순서를**

**정렬하기 위해 order = [ ] 이용**

Ex) sns.countplot(data=mpg, x = ‘drv’, order = [‘4’, ’f’, ’r’])

# 4, f, r 순으로 막대 정렬하여 그래프 출력

+) **df.value\_counts( ).index** : 빈도가 높은 순으로 변수 값 출력

Ex) mpg[‘drv’].value\_counts( ).index #drv의 값을 빈도가 높은 순으로 출력

+) **countplot( )에 value\_counts( ).index를 넣어서 자동으로 빈도 높은 순으로**

**막대 정렬해서 그래프 출력 가능**

Ex) sns.countplot(data=mpg, x = ‘drv’,

order = mpg[‘drv’].value\_counts( ).index)

#drv 빈도 높은 순으로 막대 정렬해서 빈도막대그래프 출력

**sns.lineplot(data=사용할 데이터, x = ‘x축으로 사용할 변수명’, y = ‘y축으로 사용할 변수명’)**

: 선 그래프 만들기

Ex) sns.lineplot(data=economics, x = ‘date’, y = ‘unemploy’)

#x축은 시간, y축은 실업자 수를 나타내는 선그래프

+) **errorbar = None** : 선 그래프에 표시되는 신뢰구간 없애기 위해 추가로 입력하는 코드

Cf) 신뢰구간 : 선그래프의 선의 위아래에 표시된 면적

Ex) sns.lineplot(data=economics, x = ‘year’, y = ‘unemploy’, errorbar = None)

#신뢰구간 제거

**pd.to\_datetime( )** : 변수의 타입을 날짜 시간 타입으로 변경

ex) economics[‘date2’] = pd.to\_datetime(economics[‘date’])

#date2 변수는 economics의 date변수를 날짜 시간 타입으로 저장한 것

+) **df.dt 이용해서 연/월/일 추출 가능**

Ex) economics[‘date2’].dt.year #연 추출

Ex) economics[‘date2’].dt.month #월 추출

Ex) economics[‘date2’].dt.day #일 추출

**sns.boxplot(data=사용할 데이터, x=’x축으로 사용할 변수명’, y = ‘y축으로 사용할 변수명’)**

: 상자그림 만들기

Ex) sns.boxplot(data=mpg, x = ‘drv’, y = ‘hwy’) #x축을 drv, y축을 hwy로 하는 상자그림 출력

[패키지 종류]

* pandas패키지 : 데이터를 가공할 때 사용하는 패키지. 데이터 전처리 작업 위해 가장

많이 사용됨.

* numPy패키지 : 배열 연산, 통계치 계산 등 수치연산을 할 때 사용하는 패키지
* seaborn패키지 : 그래프를 만들 때 사용하는 패키지
* matplotlib.pyplot 패키지 : 그래프의 설정을 바꾸는 패키지