빅데이터 분석기사 실기 정리 문서

**Chap1. Pandas Practice**

▶ Check DataFrame Information

1. df.info()

- 각 Column의 Data type ( dtypes ) 정보.

- Null 값의 개수

- DataFrame Memory usage.

|  |
| --- |
| <class 'pandas.core.frame.DataFrame'>  Index: 3500 entries, 0 to 3499  Data columns (total 9 columns):  # Column Non-Null Count Dtype  --- ------ -------------- -----  0 총구매액 3500 non-null int64  1 최대구매액 3500 non-null int64  2 환불금액 1205 non-null float64  3 주구매상품 3500 non-null object  4 주구매지점 3500 non-null object  5 내점일수 3500 non-null int64  6 내점당구매건수 3500 non-null float64  7 주말방문비율 3500 non-null float64  8 구매주기 3500 non-null int64  dtypes: float64(3), int64(4), object(2)  memory usage: 273.4+ KB |

Table 1. df.info() 결과

2. df.describe()

- 수치형 데이터에 대한 요약 통계 제공 ( mean, std, min, quantile, max )

- 옵션(include=’object’)을 통해 문자열(object) 자료도 열람 가능.

|  |
| --- |
| 총구매액 최대구매액 ... 주말방문비율 구매주기  count 3.500000e+03 3.500000e+03 ... 3500.000000 3500.000000  mean 9.191925e+07 1.966424e+07 ... 0.307246 20.958286  std 1.635065e+08 3.199235e+07 ... 0.289752 24.748682  min -5.242152e+07 -2.992000e+06 ... 0.000000 0.000000  25% 4.747050e+06 2.875000e+06 ... 0.027291 4.000000  50% 2.822270e+07 9.837000e+06 ... 0.256410 13.000000  75% 1.065079e+08 2.296250e+07 ... 0.448980 28.000000  max 2.323180e+09 7.066290e+08 ... 1.000000 166.000000  [8 rows x 7 columns] |

Table 2. df.describe() 결과

|  |
| --- |
| 주구매상품 주구매지점  count 3500 3500 # 각 Column별 Row 개수  unique 42 24 # count 개수의 Row 중 고유한 값의 도메인 개수  top 기타 본 점 # 전체에서 가장 많이 존재하는 값  freq 595 1077 # top 값의 빈도(출현) 수 |

Table 3. df.describe( include=’object’ ) 결과

3. df.shape

- DataFrame의 행과 열을 튜플 형태로 확인. (ex. (20, 5) 🡪 20행 5열. (13, 10) 🡪 13행 10열. )

4. df.head(N)

- 첫 Row를 포함한 N개의 Row만 출력 (N 입력이 없으면 기본값 5 )

|  |
| --- |
| 총구매액 최대구매액 환불금액 주구매상품 ... 내점일수 내점당구매건수 주말방문비율 구매주기  cust\_id ...  0 68282840 11264000 6860000.0 기타 ... 19 3.894737 0.527027 17  1 2136000 2136000 300000.0 스포츠 ... 2 1.500000 0.000000 1  2 3197000 1639000 NaN 남성 캐주얼 ... 2 2.000000 0.000000 1  3 16077620 4935000 NaN 기타 ... 18 2.444444 0.318182 16  4 29050000 24000000 NaN 보석 ... 2 1.500000 0.000000 85  [5 rows x 9 columns] |

Table 4. df.head() 또는 df.head(5)의 결과

5. df.tail(N)

- 마지막 Row를 포함한 하위 5개의 Row만 출력 (N 입력이 없으면 기본값 5)

6. df.columns

- DataFrame의 Column 이름을 출력한다.

|  |
| --- |
| Index(['총구매액', '최대구매액', '환불금액', '주구매상품', '주구매지점', '내점일수', '내점당구매건수', '주말방문비율',  '구매주기'],  dtype='object') |

Table 5. df.columns 결과 (Column명 리스트를 볼 수 있다).

7. df.index

- DataFrame의 Index를 출력한다.

- pd.read\_csv() 함수에서 ‘index\_col=’ 옵션을 통해 인덱스를 설정할 수 있다.

ㆍ1개의 Column으로만 인덱스 설정 시 index\_col=0 또는 index\_col=’열이름1’ 처럼 적용 가능하다.

ㆍ2개 이상의 Column으로 구성하려면 리스트 형태로 입력한다 (index\_col = [0, 1] 또는 index\_col = [‘열1’, ‘열2’]

|  |
| --- |
| Index([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,  ...  3490, 3491, 3492, 3493, 3494, 3495, 3496, 3497, 3498, 3499],  dtype='int64', name='cust\_id', length=3500) |

Table 6. ‘cust\_id’ Column을 Index로 구성한 DataFrame의 df.index 결과

8. df.dtypes

- DataFrame의 각 Column별 데이터 타입을 출력한다 (df.info() 결과에 포함된 값 중 하나).

|  |
| --- |
| 총구매액 int64  최대구매액 int64  환불금액 float64  주구매상품 object  주구매지점 object  내점일수 int64  내점당구매건수 float64  주말방문비율 float64  구매주기 int64  dtype: object |

Table 7. df.dtypes 결과 (‘object’는 문자열)

9. df.isnull()과 df.isnull().sum()

- df.isnull() 함수는 데이터프레임을 구성하는 각 열에서 Null(None or NaN)값이 있는 경우를 확인해

[‘True’, ‘False’]의 Table을 출력하는 함수.

|  |
| --- |
| 총구매액 최대구매액 환불금액 주구매상품 주구매지점 내점일수 …  cust\_id  0 False False False False False False …  1 False False False False False False ...  2 False False **True** False False False ...  3 False False **True** False False False ...  4 False False **True** False False False ...  ... ... ... ... ... ... ... ... ... |

Table 8. df.isnull() 결과 Table

- 특정 Column의 Null 여부만 확인하려면 df[‘열이름’].isnull() 방식으로 작성한다.

- 출력 결과에 .sum() 함수를 추가하여 Null 값이 존재하는 전체 Row 수를 Count할 수 있다.

|  |  |
| --- | --- |
| # df[‘환불금액’].isnull() 명령 결과  cust\_id  0 False  1 False  2 True  3 True  4 True  ...  3495 True  3496 False  3497 True  3498 True  3499 False  Name: 환불금액, Length: 3500, dtype: bool | # df.isnull().sum() 결과  총구매액 0  최대구매액 0  환불금액 2295  주구매상품 0  주구매지점 0  내점일수 0  내점당구매건수 0  주말방문비율 0  구매주기 0  dtype: int64 |

Table 9. 특정 Column의 isnull() 결과, 결측값(Null)이 존재하는 Row 수 계산 결과

10. df.nunique()

- 각 Column의 고유값의 수(Number of Unique value)를 출력한다.

|  |
| --- |
| 총구매액 3396  최대구매액 2576  환불금액 1114  주구매상품 42  주구매지점 24  내점일수 147  내점당구매건수 1107  주말방문비율 1142  구매주기 135  dtype: int64 |

Table 10. df.nunique() 결과

11. df.corr()

- 각 Column끼리의 상관관계를 계산하여 출력한다.

- 중요) ‘수치형’ 자료만 계산이 가능하므로 ‘범주형’ 자료의 경우 계산에서 제외시켜야 오류가 발생하지 않는다.

|  |
| --- |
| # 전처리 과정  # float, int형의 데이터타입 Column만 가져와 새 df 생성.  **df\_new = df.select\_dtypes( include= [‘float’, ‘int’] )**  **print(df\_new.corr())** |
| 총구매액 최대구매액 환불금액 내점일수 내점당구매건수 주말방문비율 구매주기  총구매액 1.000000 **0.700080** 0.419734 **0.659084** 0.090022 0.014396 -0.212944  최대구매액 0.700080 1.000000 0.410562 0.374147 0.018980 0.022277 -0.115837  환불금액 0.419734 0.410562 1.000000 0.270290 -0.063114 -0.062397 -0.211125  내점일수 0.659084 0.374147 0.270290 1.000000 0.225264 -0.010325 -0.293200  내점당구매건수 0.090022 0.018980 -0.063114 0.225264 1.000000 0.007659 -0.091151  주말방문비율 0.014396 0.022277 -0.062397 -0.010325 0.007659 1.000000 0.003372  구매주기 -0.212944 -0.115837 -0.211125 -0.293200 -0.091151 0.003372 1.000000 |

Table 11. Column 간의 상관계수 결과 테이블

- 상관계수의 값은 -1 ~ 1 사이의 값을 가지며 다음과 같은 성격이 있다.

ㆍ -1 : 완벽한 음의 상관 ( 한 변수의 값이 증가할 때 다른 변수의 값은 감소).

ㆍ 0 : 상관관계가 없음 ( 두 변수의 선형적 관계가 없다 ).

ㆍ 1 : 완벽한 양의 상관 ( 한 변수의 값이 증가할 때 다른 변수의 값도 증가).

12. df[‘특정 열’].value\_counts()

- 특정 Column의 값의 빈도를 계산해서 출력한다.

- 항목 분류 등의 ‘범주형’ 변수의 Column에서 분석하기 용이하다.

|  |
| --- |
| # **df[‘주구매상품’].value\_counts()** |
| 주구매상품  기타 595  가공식품 546 … 일부 생략  골프 82  스포츠 69  일용잡화 64  모피/피혁 57  육류 57  남성 캐주얼 55  소형가전 2  악기 2  Name: count, dtype: int64 |

Table 12. ‘주구매상품’ Column의 데이터 빈도수 확인

▶ 사분위 수 (Quantile, Quartile) 정리.

- 사분위 수 계산은 DataFrame에서 quantile() 함수를 사용해 적용한다.

ㆍ Parameter로 분위수 값 ( 1사분: 0.25, 3사분: 0.75 )을 지정해 출력한다.

ㆍ 2분위 수의 경우 ‘중앙값’이므로 median() 함수를 적용해도 된다.

ㆍ 4분위 수의 경우 ‘최대값’이므로 max() 함수를 적용해도 된다.

- 하위 25% : 전체 데이터의 25%는 1사분위 수 이하의 값을 갖는다.

- 상위 25% (하위 75%) : 전체 데이터의 25%는 3사분위 수 이상의 값을 갖는다.

또는 전체 데이터의 75%는 3사분위 수 이하의 값을 갖는다.

- IQR (InterQuartileRange) : 사분위 수 범위 🡪 3사분 - 1사분 (Q3-Q1)

- 이상치(Outlier) 상/하한선 : Q3+1.5\*IQR (상한선). Q3-1.5\*IQR (하한선)

ㆍ상/하한선을 넘는 값의 경우 이상치로 간주할 수 있다.

|  |
| --- |
| # 임시 데이터 프레임 만들기 data = [1,1,2,3,4,5,6,6,6,7,8,9,9] dtmp = pd.DataFrame(data, columns=['value'])  print(f"사분위 수 0% : {dtmp['value'].**quantile(0)**}") print(f"min() 결과: {dtmp['value'].**min()**}") print(f"사분위 수 하위 25% : {dtmp['value'].**quantile(0.25)**}") print(f"median() 결과: {dtmp['value'].**median()**}") print(f"사분위 수 하위 50% : {dtmp['value'].**quantile(0.5)**}") print(f"사분위 수 하위 75% (상위 25%) : {dtmp['value'].**quantile(0.75)**}") print(f"max() 결과: {dtmp['value'].**max()**}") print(f"사분위 수 하위 100% : {dtmp['value'].**quantile(1.0)**}") |
| 사분위 수 0% : 1.0  min() 결과: 1  사분위 수 하위 25% : 3.0  median() 결과: 6.0  사분위 수 하위 50% : 6.0  사분위 수 하위 75% (상위 25%) : 7.0  max() 결과: 9  사분위 수 하위 100% : 9.0 |

Table 13. 사분위 수 테스트 코드와 실행 결과

▶ 데이터 정렬과 인덱싱

- sort\_values( [정렬할 기준 Columns], ascending= [True, False], inplace= [True, False] )

ㆍ 특정 Column을 기준으로 DataFrame 정렬하는 함수

- DataFrame 인덱스 활용

ㆍloc (Location) : 특정 열에 대한 ‘이름’, 특정 행에 대한 ‘조건’을 적용해 데이터에 접근할 수 있다.

ㆍiloc (Integer Location) : 특정 행과 열을 인덱스 번호인 ‘정수’ 값으로만 접근할 수 있다.

|  |
| --- |
| # 0~4번 인덱스의 [0, 1, 3, 4]번 째 Column에 대한 데이터프레임 행 가져오기.  # df.iloc[ 행시작:행끝+1, “Column인덱스번호”]로 구성되며 ‘ : ‘, ‘ [] ‘ 등의 구분 기호를 통해 인덱스를 다룬다.  # 정수 값의 인덱스 번호로만 파라미터를 구성할 수 있으므로 ‘조건’ ‘이름’ 등은 적용할 수 없다.   * **df.iloc[0:5, [0,1,3,4]]** |
| 총구매액 최대구매액 주구매상품 주구매지점  cust\_id  0 68282840 11264000 기타 강남점  1 2136000 2136000 스포츠 잠실점  2 3197000 1639000 남성 캐주얼 관악점  3 16077620 4935000 기타 광주점  4 29050000 24000000 보석 본 점 |

Table 14. iloc 데이터 접근법을 통해 인덱스 번호 기준의 데이터프레임 출력

|  |
| --- |
| # loc에서는 인덱스 슬라이싱 시 행 번호 ‘그대로’를 적용하므로 ‘행끝+1’이 아닌 ‘행끝’ 값 그대로를 입력한다.   * df.loc[0:4, [‘총구매액’, ‘최대구매액’, ‘주구매상품’, ‘주구매지점’]]   🡪 위의 출력 결과와 동일한 결과를 얻는다. |
| # 특정 행만 추출하기 위한 ‘조건식’을 추가해서 loc에 반영할 수 있다.  # ex. 30000000 이상의 ‘총구매액’을 갖는 Row만 추출하고, ‘총구매액’과 ‘주구매상품’, ‘주구매지점’ Column 확인.   * printdf = df.loc[df['총구매액'] >= 30000000, ['총구매액', '주구매상품', '주구매지점']] print(printdf[0:5])   총구매액 주구매상품 주구매지점  cust\_id  0 68282840 기타 강남점  7 514570080 명품 본 점  8 688243360 기타 본 점  10 245110760 디자이너 부산본점  13 103217000 시티웨어 분당점 |

Table 15. loc 데이터 접근법을 통해 ‘조건식’ 및 Column명 기준의 데이터프레임 출력

|  |
| --- |
| # ‘주구매지점’ 기준의 오름차순과 내림차순 결과의 데이터프레임 0~4번 인덱스 행 출력  ㆍ ascending= True 🡪 오름차순 정렬. <-> False 🡪 내림차순  ㆍ inplace= False 🡪 원본 데이터프레임은 변경 안 함.  True 🡪 원본 데이터프레임도 같이 변경.   * tmpdf = df.sort\_values(['주구매지점'], ascending=True, inplace=False) * print(tmpdf.iloc[0:4, [3, 4, 5, 6]]) # 특정 Column의 값들만 보기   주구매상품 주구매지점 내점일수 내점당구매건수  cust\_id  0 기타 강남점 19 3.894737  169 건강식품 강남점 3 1.666667  2697 시티웨어 강남점 1 2.000000  1750 피혁잡화 강남점 3 2.333333 |
| # 주의) loc를 이용해 출력하려고 행 파라미터에 0:3을 입력하면 큰 문제가 발생한다.  ㆍ loc는 인덱스 슬라이싱이 ‘라벨’ 기준이기 때문에 0:3은 첫 행에서 두 번째 행까지가 아니라  “0번 인덱스의 행부터 3번 인덱스의 행까지’로 인식한다.   * print(tmpdf.loc[0:4, ['주구매상품', '주구매지점', '내점일수', '내점당구매건수']])   주구매상품 주구매지점 내점일수 내점당구매건수  cust\_id  **0** 기타 강남점 19 3.894737  169 건강식품 강남점 3 1.666667  2697 시티웨어 강남점 1 2.000000  1750 피혁잡화 강남점 3 2.333333  451 농산물 강남점 3 1.333333  ... ... ... ... ...  3441 수산품 본 점 19 1.947368  3495 골프 본 점 1 2.000000  3493 농산물 본 점 68 3.852941  3489 기타 본 점 6 1.833333  **4**  보석 본 점 2 1.500000 |

Table 16. 데이터프레임 정렬과 인덱스 활용 주의 예시

▶ 결측값에 대한 처리

1. dropna()

- 결측값이 있는 데이터 위치를 기준으로 ‘행’, ‘열’ 기준의 제거.

- **axis=0** (행 단위 제거). **axis=1** (열 단위 제거)

ㆍ **axis=’index’** 🡪 행 단위 제거. **axis=’column’** 🡪 열 단위 제거.

- **how=’any’** 🡪 기본값. 1개라도 결측값이 있으면 기준(행/열) 단위로 제거.

**how=’all’** 🡪 기준(행/열) 단위로 전부 결측값이면 제거.

- **subset=[‘열1’, ‘열2’]** 🡪 특정 열에만 결측값이 있는지 검사.

- inplace= [True 또는 False] 🡪 처리 결과를 원본 데이터프레임에 반영할 것인지 여부.

|  |
| --- |
| # 특정 Column의 결측값이 있는 데이터 추출   * nadf = df[df['환불금액'].isnull() == True] 🡪 2295개의 Row가 나왔다고 한다.   # 결측값이 있는 Row를 ‘행’ 단위로 제거 후의 데이터프레임 확인   * nodf = df.dropna(axis=0, subset=['환불금액'], inplace=False) * nodf.info()   - 원본 df는 3,500개의 Row가 있었으므로 ‘환불금액’ Column에 결측값이 존재했던 2,295개의 Row가 삭제되어  총 1,205개의 Row가 출력된 것을 확인할 수 있다.  <class 'pandas.core.frame.DataFrame'>  Index: **1205** entries, 0 to 3499  Data columns (total 9 columns):  # Column Non-Null Count Dtype  --- ------ -------------- -----  0 총구매액 1205 non-null int64  1 최대구매액 1205 non-null int64  2 환불금액 1205 non-null float64  3 주구매상품 1205 non-null object  4 주구매지점 1205 non-null object  5 내점일수 1205 non-null int64  6 내점당구매건수 1205 non-null float64  7 주말방문비율 1205 non-null float64  8 구매주기 1205 non-null int64  dtypes: float64(3), int64(4), object(2)  memory usage: 94.1+ KB |

Table 17. 결측값에 대한 dropna() 함수 처리 결과

▶ DataFrame 특정 Row 및 Column 삭제 처리

1. df.drop()

- 특정 행과 열을 삭제할 때 용이.

- **labels=** : 삭제할 행 또는 열의 라벨. Index나 Column명을 추가한다.

- **axis=** : 기준(행,열)을 정한다. 0이면 행 기준, 1이면 열 기준.

- **inplace=** : 변경 내용을 원본 DataFrame에 반영할 것인지 설정한다. True면 반영, False면 반영하지 않는다.

- **errors=** : 삭제할 행/열이 없을 때 오류가 발생하는데 그 때의 동작을 명시한다.

ㆍ’raise’ : 오류 발생 표시 (기본값)

ㆍ ‘ignore’ : 오류를 무시하고 계속 진행.

- 주로 DataFrame에서 특정 Column을 ‘종속변수’로 사용하기 위해 분리(삭제)할 목적으로 사용한다.

|  |
| --- |
| * **newdf = df.drop(['내점일수', '내점당구매건수'], axis=1, inplace=False )**   # 삭제 전 (df.columns)  Index(['총구매액', '최대구매액', '환불금액', '주구매상품', '주구매지점', '내점일수', '내점당구매건수', '주말방문비율',  '구매주기'], dtype='object')  # 삭제 후 (newdf.columns)  Index(['총구매액', '최대구매액', '환불금액', '주구매상품', '주구매지점', '주말방문비율', '구매주기'], dtype='object') |

Table 18. Column 기준의 drop() 후 결과

▶ Row Index 또는 Column명 변경

1. df.rename()

- 특정 Index(행) 또는 Column의 명칭을 변경할 때 적용한다.

- 전체 Column명을 변경하려면 df.columns= [‘열1’, ‘열2’, ‘열3’ …] 코드로 변경한다.

- 변경할 값을 입력하는 파라미터는 ‘Dictionary’ 또는 ‘함수’로 적용한다.

Ex) **df.rename(columns= {‘내점일수’ : ‘방문일수’}, inplace=True)**

ㆍ ‘내점일수’ Column명을 ‘방문일수’로 변경하고, 원본 데이터프레임(df)에 변경 내용을 반영한다.

Ex) **df.rename(index= {0 : ‘Changed\_Index’}, inplace=True)**

ㆍ 0번 행 (0 인덱스)의 인덱스 값을 ‘Changed\_Index’로 변경하고, 원본 데이터프레임(df)에 변경 내용을 반영한다.

|  |
| --- |
| 총구매액 최대구매액 환불금액 ... 내점당구매건수 주말방문비율 구매주기  cust\_id ...  **Changed\_Index** 68282840 11264000 6860000.0 ... 3.894737 0.527027 17  1 2136000 2136000 300000.0 ... 1.500000 0.000000 1  2 3197000 1639000 NaN ... 2.000000 0.000000 1 … |

Table 19. 행 인덱스 값을 변경한 결과 (0 인덱스를 ‘Changed\_Index’로 변경)

▶ 날짜 데이터 다루기

- datetime Type의 날짜 데이터를 다룰 수 있다.

- 날짜 형식의 문자열(object) 데이터를 datetime으로 변환해 적용할 수 있다.

- apply()

ㆍDataFrame 또는 Series 자료에서 사용하며 각 행 또는 열에 대해 ‘주어진 함수를 반복 실행’하는 역할.

ㆍ내장 함수 또는 lambda 형식의 사용자 지정 함수를 자유롭게 적용 가능.

ㆍ axis= 파라미터를 사용해 적용할 방향(행/열)을 지정할 수 있으며 기본값은 axis=0.

|  |
| --- |
| # 다음과 같은 자료로 구성된 DataFrame이 있다 (df)  dataTime stationName so2Value coValue o3Value no2Value pm10Value  1 2022-08-03 중구 0.003 0.2 0.035 0.017 30  2 2022-07-01 한강대로 0.004 0.5 0.015 0.036 48  3 2022-06-03 종로구 0.002 0.4 0.042 0.014 28  4 2022-06-13 청계천로 0.004 0.5 0.034 0.024 36  5 2022-06-22 종로 0.003 0.2 0.027 0.016 29 |
| # dataTime Column의 값을 datetime 자료로 변환한 결과를 ‘측정연월일’ Column을 새로 만들어 저장한다.  **df['측정연월일'] = df['dataTime'].apply(lambda row: pd.to\_datetime(str(row), format='%Y-%m-%d'))**  # datetime 라이브러리를 이용해 datetime 자료를 다룰 수 있다.  # import datetime as dt  **df['연도'] = df['측정연월일'].dt.year df['월'] = df['측정연월일'].dt.month df['일'] = df['측정연월일'].dt.day**  **tmpdf = df[['dataTime', '측정연월일', '연도', '월', '일']] print(tmpdf.head()) print(tmpdf.dtypes)**  # tmpdf.head() 결과  dataTime 측정연월일 연도 월 일  1 2022-08-03 2022-08-03 2022 8 3  2 2022-07-01 2022-07-01 2022 7 1  3 2022-06-03 2022-06-03 2022 6 3  4 2022-06-13 2022-06-13 2022 6 13  5 2022-06-22 2022-06-22 2022 6 22  # tmpdf.dtypes 결과  dataTime object  측정연월일 datetime64[ns]  연도 int32  월 int32  일 int32  dtype: object |

▶ 데이터 변환

1. 범주형 변수 변환

- Column 중 ‘object’ Type이지만 ‘명목척도’로 사용할 수 있는 경우가 있다.

ㆍ 명목척도 = 데이터를 ‘분류’, ‘범주화’하는 데 사용할 수 있는 척도.

ㆍ 명목척도는 ‘서열’, ‘크기’에 대한 정보를 제공하지 않으며 단순히 ‘그룹’을 나누는 목적으로 사용된다.

- DataFrame에서 범주형 데이터로 사용하려면 ‘category’ Type으로 변환해야 한다.

ㆍ astype( ‘ Type ’) 함수로 데이터 타입 변환이 가능하다.

|  |
| --- |
| # 성별 데이터를 관리하는 ‘성별’ Column의 범주화 작업.   * **df['성별'].unique()** 🡪 ['남자' '여자']   # ‘gender’라는 새로운 column에 범주화 데이터를 담기.   * **df['gender'] = df['성별'].astype('category')**   # print(df[‘gender’])  (고객번호  190105 남자  … …  190191 여자  190192 여자  190193 남자  190194 남자  Name: gender, Length: 90, dtype: **category**  **Categories (2, object): ['남자', '여자']** |

1. 변수 데이터 값 매핑

- map() 함수를 적용해 데이터 값을 특정 기준으로 변환 분류 할 수 있다.

ㆍex) ‘남자’를 1, ‘여자’를 0으로 값 변환이 가능하다.

ㆍ 딕셔너리 구조를 파라미터로 넣어 변환한다.

|  |
| --- |
| * **df['gender'] = df['gender'].map({'남자':1, '여자':0})**   190105 1  … …  190191 0  190192 0  190193 1  190194 1  Name: gender, Length: 90, dtype: category  Categories (2, int64): [1, 0] |

▶ 데이터 결합과 인덱스 재구성

- concat(), merge() 함수를 적용해 2개의 DataFrame을 행/열 결합할 수 있다.

ㆍmerge()의 경우 SQL JOIN과 유사한 구조로 접근할 때 용이.

ㆍ일반적으로 concat()을 사용해 행/열 기준의 결합을 수행한다.

1) concat()

- 옵션

ㆍaxis: 결합 축 설정 (0: 행 기준 🡪 Row 증가, 1: 열 기준 🡪 Column 증가)

ㆍjoin: 공통 index/column의 결합 방법 정의 ( ‘outer’-기본값 : 일치하지 않는 부분은 NaN 처리.

‘inner’ : 두 index 또는 column이 완전히 일치하는 부분만 결합).

ㆍignore\_index: 결합 후의 index 재구성 여부 (True=인덱스 새로 부여)

|  |  |
| --- | --- |
| # 행 기준 데이터프레임 결합  df1 = pd.DataFrame({'A': [1, 2, 3], 'B': [3, 4, 6], 'C': ['대전', '대구', '광주']}) df2 = pd.DataFrame({'A': [4, 5, 6], 'B': [2, 9, 5], 'C': ['인천', '울산', '부산']}) | |
| result = pd.concat([df1, df2], axis=0)  A B C  **0** 1 3 대전  **1** 2 4 대구  **2** 3 6 광주  **0** 4 2 인천  **1** 5 9 울산  **2** 6 5 부산 | result = pd.concat([df1, df2], axis=0, ignore\_index=True)  또는  result = pd.concat([df1, df2], axis=0) result = result.reset\_index(drop=True)  A B C  **0** 1 3 대전  **1** 2 4 대구  **2** 3 6 광주  **3** 4 2 인천  **4** 5 9 울산  **5** 6 5 부산 |
| # 열 기준 데이터프레임 결합  df3 = pd.DataFrame({'A': [1, 2, 3], 'D': [3, 4, 6], 'C': ['대전', '대구', '광주']}) df4 = pd.DataFrame({'B': [2, 5, 7], 'E': ['남자', '남자', '여자']})  rst = pd.concat([df3, df4], axis=1)  A D C B E  0 1 3 대전 2 남자  1 2 4 대구 5 남자  2 3 6 광주 7 여자 | |

- 수동 인덱스 재구성: **df.reset\_index(drop=True)**

ㆍdrop 옵션: True 🡪 이전의 인덱스를 삭제 후 재구성, False 🡪 이전의 인덱스를 Column으로 남겨둠.

2) merge()

- 특정 Column을 기준으로 A, B DataFrame을 병합할 때 적용한다.

- SQL JOIN과 처리 방식이 유사하므로 관련 옵션을 활용해 INNER, OUTER JOIN Table을 구성할 수 있다.

- 함수 구조: merge(left, right, how=, on=, left\_on=, right\_on=, left\_index=, right\_index=, sort=)

ㆍleft : 좌측 데이터프레임 테이블

ㆍright : 우측 데이터프레임 테이블

ㆍhow (기본값=’inner’) : JOIN 방식 선택 (‘left’, ‘right’, ‘inner’, ‘outer’)

* ‘left’, ‘right’는 각각 LEFT/RIGHT OUTER JOIN이다.
* ‘outer’는 FULL OUTER JOIN이다.

ㆍon : 기준이 되는 Column이며 두 DataFrame의 Column명이 동일할 때 적용할 수 있다.

ㆍleft\_on / right\_on : 두 DataFrame의 기준 Column명이 서로 다를 때 각각의 Column명을 명시하는 옵션이다.

ㆍleft\_index / right\_index : 두 DataFrame의 기준 Column을 명시할 수 없을 때, 인덱스 기준으로 명시하는 옵션.

ㆍsort (기본값 False) : 병합 후, 기준 Column을 기준으로 정렬할지 명시한다.

|  |
| --- |
| # 테스트용 데이터 프레임 (다른 Column 명을 가진 두 데이터 프레임 결합)  df1 = pd.DataFrame({'key\_left': ['A', 'B', 'C'], 'value1': [1, 2, 3]}) df2 = pd.DataFrame({'key\_right': ['A', 'B', 'D'], 'value2': [4, 5, 6]}) |
| **INNER JOIN 결과**   * **rst = pd.merge(df1, df2, left\_on='key\_left', right\_on='key\_right', how='inner')**   key\_left value1 key\_right value2  0 A 1 A 4  1 B 2 B 5 |
| **OUTER JOIN 결과**   * **rst = pd.merge(df1, df2, left\_on='key\_left', right\_on='key\_right', how='outer')**   key\_left value1 key\_right value2  0 A 1.0 A 4.0  1 B 2.0 B 5.0  2 C 3.0 NaN NaN  3 NaN NaN D 6.0 |

Table 20. Column명이 서로 다른 두 DataFrame의 merge()

|  |
| --- |
| # 테스트용 데이터 프레임 ( JOIN Column의 명칭이 서로 같은 두 데이터프레임 결합)  df3 = pd.DataFrame({'key': ['Dog', 'Cat', 'Fish'], 'value1': [20, 15, 5]}) df4 = pd.DataFrame({'key': ['Cat', 'Dog', 'Deer'], 'value2': ['Open', 'Close', 'Close']}) |
| INNER JOIN 결과   * **rst = pd.merge(df3, df4, on='key', how='inner', sort=True)**   key value1 value2  0 Cat 15 Open  1 Dog 20 Close |
| OUTER JOIN 결과   * **rst = pd.merge(df3, df4, on='key', how='outer', sort=False)**   key value1 value2  0 Cat 15.0 Open  1 Deer NaN Close  2 Dog 20.0 Close  3 Fish 5.0 NaN |

Table 21. 서로 같은 Column을 가진 두 DataFrame의 merge()

▶ 기술통계량

- **df.shape[0]** 또는 **len(df)** : 표본 수, **min()**: 최소값, **max()**: 최대값, **mean()**: 평균, **std()**: 표준편차

- CV (Coefficient of Variation – 변동계수) : **df.std() / df.mean()**

ㆍ평균에 대한 표준편차의 비율이며 데이터의 상대적인 변동성을 파악할 수 있는 지표.

- 왜도(Skewness) : 자료 분포의 대칭 정도.

ㆍ 0> 왼쪽 치우침, 0< 오른쪽 치우침, 0= 대칭

ㆍ **from scipy.stats import skew** 후, **skew(df.price)** 형식으로 통계량 수집 가능.

- 첨도(Kurtosis) : 자료 분포의 뾰족한 정도.

ㆍ 양수: 정규분포보다 높아서 뾰족함.

ㆍ 음수: 정규분포보다 낮아서 퍼져 있는 상태.

ㆍ 0 : 정규분포와 같음.

ㆍ **from scipy.stats import kurtosis** 후, **kurtosis(df.price)** 형식으로 통계량 수집 가능.

- Pandas 객체에서도 왜도/첨도 구하는 함수가 있으며 **df.price.skew()**, **df.price.kurt()** 형식으로 사용한다.

ㆍscipy.stats의 왜도/첨도는 ‘**모표준편차 공식**’을 사용한다.

ㆍPandas 객체의 왜도/첨도는 ‘**표본 표준편차 공식**’을 사용한다.

**Chap2. Data Model Creation**

▶ Correlation Analysis (상관관계 분석)

- 데이터 내의 두 변수가 서로 어떠한 연관성이 있는지 파악하기 위한 목적으로 사용 (선형적 관계).

- 함수적인 관계를 파악하기 위해서는 ‘회귀분석’ 모델을 적용한다.

- ( X, Y가 함께 변하는 정도 ) / ( X, Y가 각각 변하는 정도 )

ㆍ Cov(X, Y) / (X표준편차 x Y표준편차)

폰트, 텍스트, 화이트, 친필이(가) 표시된 사진

자동 생성된 설명

Figure 1. 공분산(Covariance) 공식

폰트, 텍스트, 화이트, 친필이(가) 표시된 사진

자동 생성된 설명

Figure 2. 상관계수(Correlation Coefficient) 공식

- **공분산**: 두 변수의 선형 관계를 나타내는 값.

ㆍ양수 : 한 변수가 증가하면 다른 변수도 증가하는 경향을 보임.

ㆍ음수 : 한 변수가 증가하면 다른 변수는 감소하는 경향을 보임.

ㆍ 0 : 두 변수는 서로 유의한 관계가 없음 (비선형 관계).

- 공분산은 측정하는 변수의 단위에 따라 결과값이 영향을 받으므로 값의 범위가 유동적이다.

(키와 몸무게의 관계를 비교할 때, 키가 cm, m 단위인지 몸무게가 g, kg 단위인지에 따라 값이 다르다.)

‘크고 작음’을 고정적으로 판단하기 어렵기 때문에 ‘**두 변수의 방향성 해석은 좋지만 크기를 해석하긴 어렵다**’.

- **상관계수**는 공분산의 크기 해석 어려움을 개선할 수 있도록 ‘**비교 가능한 표준화된 값**’으로 변환된 결과이다.

그래서 변수의 단위에 상관없이 -1 ~ 1 사이의 값을 가지며 각각은 다음과 같다.

ㆍ +1에 가까움 : 두 변수는 강한 양의 상관관계를 가짐.

ㆍ -1에 가까움 : 두 변수는 강한 음의 상관관계를 가짐.

ㆍ 0에 가까움 : 두 변수는 서로 상관관계가 거의 없음 (비선형 관계).

1) **Pearson** 상관계수

- 두 변수 간의 ‘선형 관계’를 측정하는 방법 (직선적 관계의 강도와 방향을 파악).

- 비선형 관계에서는 적절하지 않으며 ‘정규분포’를 따르는 데이터일 경우 유용하다.

- 이상치(Outlier)에 민감하다 (이상치에 의해 왜곡될 가능성이 크다).

2) **Spearman** 상관계수

- 두 변수 간의 ‘서열(순위) 관계’를 측정하는 방법 (선형이지 않더라도 순서 관계를 통해 상관성을 평가).

- 비선형 관계, 단조(Monotonic) 관계를 측정하는 것에 적합.

ㆍ비선형 : 두 변수 간의 관계가 ‘직선’의 형태로 나타나지 않는 경우.

ㆍ단조 : 한 변수가 증가할 때 다른 변수도 계속해서 증가하거나 계속해서 감소하는 경향.

-- 증가/감소 방향이 변하지 않지만, 선형적으로 비례한 증가와 감소가 아닌 경우를 의미한다.

-- 그래서 그래프로 표현하면 직선보다 휘어진 곡선의 경향을 보인다.

3) **Kendall** Tau 상관계수

- 두 변수 간의 ‘순서 일치도’를 측정하는 방법 (순서쌍을 비교).

- 순서(순위)쌍 간의 일치/불일치를 기반으로 계산하여 Spearman보다 순위에 대한 세밀한 평가가 가능하다.

- 작은 표본의 데이터 또는 이상치가 존재하는 데이터의 경우에도 민감하지 않은 분석이 가능하다.

ㆍ이상치가 있더라도 ‘순위’에 크게 변화를 주지 않기 때문.