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.

|  |
| --- |
| # Pearson 상관계수  # Pandas DataFrame 객체에서 corr() 함수로 활용 가능.  import pandas as pd  “””  Data columns (total 2 columns):  # Column Non-Null Count Dtype  --- ------ -------------- -----  0 height 15 non-null int64  1 weight 15 non-null int64  “””  **print(df.corr(method=’pearson’))**  “””  height weight height 1.000000 0.995495 weight 0.995495 1.000000  “””  # 스피어만은 df.corr(method=’spearman’)  # 켄달은 df.corr(method=’kendall’) |

Table 22. Pandas DataFrame의 corr() 함수로 상관계수 출력

|  |
| --- |
| # scipy 라이브러리의 stats 모듈에서 pearson 상관계수 적용 (함수명은 **pearsonr**임에 주의).  from scipy import stats  # df의 ‘weight’와 ‘height’ column의 상관계수를 알고 싶다고 가정.  **rst = stats.pearsonr(df[‘height’], df[‘weight’])**  # PearsonRResult(statistic=0.9954947677842163, pvalue=1.0909729585995878e-14)  - statistic : T-검정 통계량, pvalue : 유의 확률 p-값  - rst[0] = statistic값 반환, rst[1] = pvalue값 반환 |

Table 23. scipy 라이브러리의 stats 모듈로 pearson 상관계수 출력

▶ Regression Analysis (회귀분석)

- 영향을 주는 독립변수 X와 영향을 받는 종속변수 Y의 함수적 관계를 규명.

ㆍ독립변수와 종속변수 사이의 회귀식으로 종속변수에 대한 독립변수들의 영향력이 유의한지 분석한다.

- 독립변수와 종속변수 모두 ‘등간척도’ 또는 ‘비율척도’같은 ‘연속형 변수’ 자료여야 한다.

ㆍ등간척도(Interval Scale): 측정 값들 사이의 차이(간격)가 일정한 척도.

절대적인 0점이 없는 것이 중요한 특징 (ex. 온도는 0도라고 해서 열이 아예 없는 상태가 아니다).

크거나 작다는 것을 비교할 수는 있으나 ‘배수’로 비교할 수는 없다.

(ex. 온도에서 10도가 20도보다 작지만, 1/2배 따뜻하다고 볼 수는 없다)

[ 온도, 날짜, IQ 등 ]

ㆍ비율척도(Ratio Scale): 등간척도의 성질을 가지면서 절대적인 0점이 존재하는 척도.

크거나 작다를 포함해 ‘배수(비율)’로도 비교할 수 있다 🡪 모든 산술 연산이 가능한 척도.

(ex. 몸무게가 40kg인 사람은 80kg인 사람보다 작으며, 2배 가볍다. 또한, 0kg은 무게가 없음을 의미한다).

[ 키, 몸무게, 시간, 길이, 나이 등 ]

|  |
| --- |
| **명목척도**는 대상을 구분하는 데 사용되는 척도이다. 명목척도는 대상을 구분하는 데만 사용될 수 있고, 크기나 순서를 나타낼 수 없다. 성별(남자,여자), 출신 지역(서울,경기,인천), 혈액형(A형, B형, O형, AB형) 등이 명목척도이다.  **서열척도**는 대상을 크기나 순서에 따라 나열하는 데 사용되는 척도이다. 서열척도는 대상의 크기나 순서를 나타낼 수 있지만, 크기나 순서 간의 차이를 나타낼 수는 없다. 예를 들어, 운동선수의 순위, 음식의 맛 순위 등이 서열척도이다.  **등간척도**는 대상의 크기나 순서 간의 차이를 나타낼 수 있으며 크기/순서의 차이가 동일하다는 것을 의미한다. 절대적인 0이 존재하지 않으며 단순히 측정상의 값으로 적용한다. 온도, 설문조사 결과 등에 해당한다.  **비율척도**는 등간척도에서 크기의 절대적 크기를 나타낼 수 있는 척도이다. 비율척도는 0을 기준으로 하고(절대0점), 0을 기준으로 한 크기의 차이를 나타낼 수 있다. 예를 들어, 키, 몸무게, 시간, 거리 등에 해당한다.  일반적으로 **명목**형 데이터와 **서열** 데이터는 **범주형** 데이터로 구분한다. 절대값이 없으며 평균, 분산, 표준편차 등 연산을 적용할 수 없는 변수다.  반면 일반적으로 **등간**형 데이터와 **비율**형 데이터는 **수치형** 데이터로 취급한다. 경우에 따라 등간형 데이터를 범주형 데이터로 구분하기도 하는데, 리커트 척도와 같이 수치형으로 취급되어 연산이 적용되기도 한다. |

◈ 빅데이터 분석 기사 시험에서는 모든 모델을 다룰 필요가 없다. 따라서 회귀 모델로 사용할 수 있는 대표적인

‘선형 회귀(Linear regression)’, ‘랜덤포레스트(RandomForest)’, ‘LGBM(LightGBM)’ 3가지를 집중 공략한다.

1) 선형 회귀

- from sklearn.linear\_model import LinearRegression

- from statsmodels.formula.api import ols

- import statsmodels.api import sm ( sm.OLS() 활용 가능)

ㆍ 연속형/범주형 변수를 독립변수로 사용 가능. 연속형 변수의 경우, 값의 크기가 크고 범위가 넓은 경우 표준화(Scaling) 작업으로 학습에 적용하는 것이 좋다 (표준화 종류는 별도 설명). 범주형 변수의 경우 대체로 문자열(Object) 타입을 가지고 있으므로 이를 카테고리(Category) 타입으로 변환해야 하며 이항 분류의 경우 One-Hot Encoding 방식, 외에는 LabelEncoding이나 더미 변수(Dummy Variable)를 적용하여 변환 후 학습에 적용한다.

|  |
| --- |
| # LinearRegression 모듈 사용 예시 (from sklearn.linear\_model import LinearRegression 작성 가정)  # 데이터프레임 df에서 trainx, testx, trainy, testy를 8:2로 Row 분할했다고 가정.  model = LinearRegression()  model.fit(trainx, trainy)  # 모델 학습 후 예측  pred = model.predict(testx)  # 모델 성능 확인 (예측값과 실제값의 비교를 통한 비율 산정)  from sklearn.metrics import r2\_score # R2 결정 계수 – Coefficient of Determination  from sklearn.metrics import mean\_squared\_error # 평균 제곱 오차 – 값이 클수록 예측 오차가 크다.  from sklearn.metrics import mean\_absolute\_error # 평균 절대 오차 – 값이 작을수록 예측이 실제 값에 가깝다.  np.mean(np.abs((y\_true – y\_pred) / y\_true)) \* 100 # MAPE – 예측과 실제값의 차이를 백분율로 나타낸 지표.  R2 = r2\_score(testy, pred)  mse = mean\_squared\_error(testy, pred)  mae = mean\_absolute\_error(testy, pred) |

|  |
| --- |
| # OLS (Ordinary Least Squares – 최소제곱법) 회귀 모델 적용.  # ( from statsmodels.formula.api import ols 작성 가정)  # import statsmodels.api import sm 모듈에서 OLS 사용 가능.  # 데이터프레임 df에서 독립변수가 ‘A’, ‘B’ Column이고, 종속변수가 ‘C’ Column이라고 가정.  y = df[‘C] # 종속변수 부분만 가져와 1차원 y 데이터프레임 생성.  x = df[[‘A’, ‘B’]] # 독립변수 부분만 가져와 2차원 x 데이터프레임 생성.  model = ols(‘y ~ x’ , data = df).fit() # ols 모델로 학습.  # 또는 model = sm.OLS(y, x).fit()  # 예측 결과 확인  ypred = model.fittedvalues # 예측 결과값에 대한 Series 객체 생성.  summary\_note = model.summary() # 모델에 대한 전체적인 성능 통계 보고서 확인 가능.  “””  OLS Regression Results  ==============================================================================  Dep. Variable: y R-squared: 0.791  Model: OLS Adj. R-squared: 0.787  Method: Least Squares F-statistic: 177.3  Date: Sun, 13 Oct 2024 Prob (F-statistic): 1.78e-62  Time: 16:25:17 Log-Likelihood: -742.43  No. Observations: 192 AIC: 1495.  Df Residuals: 187 BIC: 1511.  Df Model: 4  Covariance Type: nonrobust  ===============================================================================  coef std err t P>|t| [0.025 0.975]  -----------------------------------------------------------------------------------------------------  Intercept 122.8021 0.846 145.216 0.000 121.134 124.470  scaled\_x[0] 23.4401 1.467 15.978 0.000 20.546 26.334  scaled\_x[1] -0.4562 1.532 -0.298 0.766 -3.479 2.567  scaled\_x[2] -0.7177 0.980 -0.732 0.465 -2.651 1.215  scaled\_x[3] 0.4882 1.006 0.485 0.628 -1.496 2.472  ==============================================================================  Omnibus: 1.692 Durbin-Watson: 1.851  Prob(Omnibus): 0.429 Jarque-Bera (JB): 1.770  Skew: 0.197 Prob(JB): 0.413  Kurtosis: 2.742 Cond. No. 3.72  ==============================================================================  Notes:  [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.  “”” |

2) 랜덤포레스트 (RandomForest)

- from sklearn.ensemble import RandomForestRegressor

ㆍ학습에 사용할 독립변수의 데이터는 실수 형태(정수, 부동소수점)로 표현해야 하므로 연속형의 경우는 표준화 작업, 범주형의 경우는 인코딩 작업을 수행해야 한다.

|  |
| --- |
| # RandomForest 회귀 모델 적용 ( from sklearn.ensemble import RandomForestRegressor 작성 가정)  # 데이터프레임 df에서 trainx, testx, trainy, testy를 8:2로 Row 분할했다고 가정.  model = RandomForestRegressor()  model.fit(trainx, trainy)  pred = model.predict(testx)  # 모델 성능 평가 ( r2\_score, mean\_absolute\_error 등) |

- RandomForestRegressor 적용 가능한 파라미터

ㆍ n\_estimators : (Default: 100) 트리 개수 지정 (많을수록 성능은 좋지만 시간이 오래걸림).

ㆍ criterion : (Default: ‘squared\_error’) 트리 분할 시 사용할 기준 (‘absolute\_error’).

ㆍ min\_samples\_split : (2) 노드 분할을 위한 최소 샘플 수. 값이 클수록 트리가 덜 복잡, 과적합 방지.

ㆍ min\_samples\_leaf : (1) 리프 노드에 있어야 하는 최소 샘플 수. 값이 클수록 트리 복잡성이 준다.

ㆍ max\_depth : (None) 트리 최대 깊이 지정. 기본값일 경우 트리는 리프 노드에 도달할 때까지 계속 분할된다.

ㆍ max\_leaf\_nodes : (None) 리프 노드의 최대 개수를 지정해 트리의 복잡도를 제한.

1. Lightgbm

- import lightgbm as lgbm

ㆍ회귀 모델 생성은 ‘lgbm.LGBMRegressor()’이며 학습 및 검증 방법은 동일하다.

- 적용 가능한 파라미터

ㆍboosting\_type : (Default: ‘gbdt’) 부스팅 알고리즘의 종류 지정.

> ‘gbdt’ : 그레디언트 부스팅 트리 (Gradient Boosting Tree)

> ‘dart’ : DropOut 방식을 적용한 부스팅.

> ‘goss’ : Gradient-based Oen-Side Sampling 방식.

> ‘rf’ : 랜덤 포레스트 방식.

ㆍ learning\_rate : (Default: 0.1) 낮은 값을 지정하면 학습이 천천히 이루어지며 정확한 모델을 만들지만 속도 느림.

ㆍ n\_estimators : (Default: 100) 생성할 부스팅 트리 개수 지정. 많을수록 성능 향상되지만 학습 시간이 증가.

ㆍ objective : (‘regression’) 모델의 손실 함수 지정.

> ‘regression’ : 일반적인 회귀 문제 적용.

> ‘poisson’ : 포아송 분포 기반 회귀.

> ‘huber’ : 허버 손실 기반 회귀. 이상치에 덜 민감하다.

> ‘quantile’ : 분위 회귀.

▶ Classification Analysis (분류분석)

◈ 빅데이터분석기사 ‘분류 (Classifier)’ 모델

- 회귀(Regressor)에서 사용한 ‘랜덤포레스트’와 ‘LGBM’을 동일하게 사용한다 (둘 다 사용 가능한 대표적인 모델).

- 단, 회귀에서는 ‘RandomForestRegressor()’, ‘LGBMRegressor()’였지만,

분류에서는 ‘RandomForestClassifier()’, ‘LGBMClassifier()’를 사용함에 주의한다.

1) 랜덤포레스트 분류 모형 (RandomForestClassifier)

- from sklearn.ensemble import RandomForestClassifier

ㆍ학습에 사용할 독립변수의 데이터는 실수 형태(정수, 부동소수점)로 표현해야 하므로 연속형의 경우는 표준화 작업, 범주형의 경우는 인코딩 작업을 수행해야 한다.

ㆍ사용할 수 있는 파라미터는 랜덤포레스트 회귀와 동일하므로 위의 파라미터 설명을 참고한다.

|  |
| --- |
| # 랜덤 포레스트 분류 적용 예시 (전체적인 절차는 동일함)  from sklearn.ensemble import RandomForestClassifier # 모델 객체 생성  from sklearn.preprocessing import StandardScaler # 데이터 전처리 (표준화)  from sklearn.model\_selection import train\_test\_split # 학습/검증 데이터 분할  from sklearn.metrics import classification\_report # 분류 검증 결과  # 독립변수는 x, 종속변수는 y라고 가정.  scaler = StandardScaler()  scaled\_x = scaler.fit\_transform(x) # 독립변수 x 세트를 표준화 (Z-Score 정규화)  trainx, testx, trainy, testy = train\_test\_split(scaled\_x, y, test\_size=0.2, random\_state=55) # 데이터 분리 8:2  model = RandomForestClassifier(n\_estimators=100, random\_state=55) # 모델 객체 생성  model.fit(trainx, trainy) # 훈련 데이터셋으로 학습 수행  predictions = model.predict(testx) # 검증 데이터로 예상 분류 계산  conf = classification\_report(testy, predictions) # 검증 정답과 분류 계산을 비교하며 성능 평가  “””  precision recall f1-score support  0 1.00 1.00 1.00 10  1 0.90 1.00 0.95 9  2 1.00 0.91 0.95 11  accuracy 0.97 30  macro avg 0.97 0.97 0.97 30  weighted avg 0.97 0.97 0.97 30  “””  # metrics에서 accuracy\_score, f1\_score, precision\_score, recall\_score, r2\_score를 개별 import해서 평가 가능. |

2) LGBM (LGBMClassifier)

- from lightgbm import LGBMClassifier

ㆍ사용 방법은 랜덤포레스트와 동일하다.

ㆍ레이블 값이 불균형한 데이터셋인 경우 ‘boost\_from\_average’ 파라미터를 False로 설정한다 (기본값 True로 진행

하면 ROC-AUC curve의 수치가 낮게 평가되는 결과가 나올 수 있다).

- 방식이 똑같은 관계로 교차 검증을 포함한 과정으로 정리한다.

- 교차 검증 (Cross-Validation) : 주어진 데이터를 여러 개의 집합으로 분할 후 모델을 반복 학습하고 평가하는 방법.

ㆍK-fold의 경우 K개의 동일한 크기의 폴드(fold)로 나누어 교차 검증을 수행하는 것이다.

ㆍ데이터를 다양한 학습-검증 세트로 평가할 수 있어서 모델의 일반화 성능을 정확하게 평가할 수 있다.

|  |
| --- |
| # LGBMClassifier 분류 모델로 교차 검증 수행하는 예시  from lightgbm import LGBMClassifier  from sklearn.model\_selection import cross\_validate # scikit-learn 패키지에 있는 교차 검증 사용  # 독립변수는 x, 종속변수는 y로 가정.  model = LGBMClassifier(n\_estimators=100, num\_leaves=64, n\_jobs=-1)  cross\_val = cross\_validate( model, x, y, cv=6, scoring=’accuracy’, return\_train\_score=True)  print(f"훈련세트에 대한 정확도 평균 = {cross\_val['train\_score'].mean()}") print(f"검증세트에 대한 정확도 평균 = {cross\_val['test\_score'].mean()}") |

- cross\_validate()에 사용할 수 있는 파라미터

ㆍestimator : model을 명시한다 (예시에서는 LGBMClassifier() 객체인 model을 넣었다).

ㆍscoring : 평가 지표이며 ‘accuracy’, ‘f1’, ‘roc\_auc’ 적용 가능하다.

ㆍreturn\_train\_score : True 시 훈련세트의 점수(train\_score)를 확인할 수 있다.

ㆍcv : 교차 검증 폴드 수를 지정한다 (기본값 5).

3) 기타 적용 모델 예시 ( 회귀, 분류 전부 )

from sklearn.svm import SVC 🡪 model = SVC( kernel=’rbf', C=10, gamma=0.1, random\_state=55)

from sklearn.tree import DecisionTreeClassifier 🡪 model = DecisionTreeClassifier(criterion=’entropy’)

from sklearn.linear\_model import LogisticRegression 🡪 model = LogisticRegression()

**Chap3. Data Model Evaluation (모델 성능 평가)**

▶ 분류 모델 성능 평가

- 분류는 간단히 말해서 각 조건들(독립변수)을 분석(비교)했을 때 ‘1’인지 ‘0’인지 구분(종속변수)하는 과정이다.

따라서 성능 지표는 ‘실제 분류와 예측 분류가 얼마나 일치하는지’ 본다.

**- Confusion Matrix** (혼동 행렬, 오차 행렬, 정오 행렬)

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

ㆍ실제 값에 대한(N 또는 P) 예측 값의 분포(T 또는 F)를 표현한 2X2 표이다. 주대각선은 예측이 실제와 정확히

일치하는 지표로 볼 수 있으며 주대각선 외의 값은 일치하지 않는 지표로 볼 수 있다.

- 오차 행렬을 통해 평가할 수 있는 지표들을 설명한다.

① 정확도 (Accuracy) : **주대각선 합계 / 전체 합계.**

‘전체’에서 올바르게 실제 범주를 추정(예측)한 비율. (1-오차 비율)로도 계산한다.

② 오차 비율(오류율-Error Rate) : **(FP+FT) / 전체 합계.**

‘전체’에서 주대각선이 아닌 성분의 합계의 비율. 실제 범주를 올바르게 예측하지 못한 수치이며 (1-정확도).

③ 정밀도(Precision) : **TP / (TP+FP)**

“Positive로 예측한 그룹(범주)에서 실제 Positive로 맞은 비율” 🡪 그래서 P 집단으로만 계산한다.

④ 재현율(민감도-Sensitivity, Recall, Hit Ratio) : **TP / (TP+FN)**

“실제 Positive인 그룹(범주)에서 Positive로 예측한 비율” 🡪 정밀도와의 그룹 기준이 다르다는 것을 명심한다.

참 긍정률 ( TP Rate )

⑤ 거짓 긍정률 ( FP Rate ) : **FP / (TN+FP)**

“실제 Negative인 그룹(범주)에서 Positive로 예측한 비율” 🡪 (1-특이도)

⑥ 특이도(Specificity) : **TN / (TN+FP)**

“실제 Negative인 그룹(범주)에서 Negative로 맞은 비율” 🡪 (1-FP Rate)

⑦ F1-Score : (2 x (Precision x Recall) ) / (Precision + Recall)

정밀도와 재현율(민감도)의 “조화 평균”

- ROC ( Receiver Operating Characteristic )

ㆍ참 긍정률(재현율- TP Rate)과 거짓 긍정률(FP Rate) 사이의 관계를 그래프로 표현한 성능 지표.

텍스트, 라인, 폰트, 도표이(가) 표시된 사진

자동 생성된 설명

ㆍTP Rate 값이 클수록, FP Rate 값이 작을수록 성능이 우수한 모델임을 나타낸다 (좌측 상단으로 커브가 치우친다).

ㆍ곡선으로 만들어지는 하단의 공간을 AUC ( Area Under the ROC Curve)라고 하며 공간이 커질수록 우수한 모델.

> 0.5 ~ 0.6 : 판별 능력 없음 > 0.7 ~ 0.8 : 모델 성능 양호 > 0.9 ~ 1.0 : 모델 성능 뛰어남

> 0.6 ~ 0.7 : 모델 성능 불량 > 0.8 ~ 0.9 : 모델 성능 우수

▶ 회귀 모델 성능 평가

- 회귀 모델은 ‘독립변수가 종속변수와 어떤 관계성으로 영향을 주는지’를 분석하고 ‘종속변수의 실측값을 판단(예측)’

하는 역할이다. 따라서 성능 평가 지표는 실측값과 예측값의 비교를 통해 ‘잔차가 평균으로 얼마나 회귀하는지’를

확인한다.

1. R2 score ( 결정계수 – R-squared )

- 회귀 모델에서 독립변수가 종속변수를 얼마나 잘 설명해주는지 평가하는 지표. (상관계수를 제곱한 형태)

- 상관계수는 ‘독립-독립’ 또는 ‘독립-종속’ 간의 상관성을 확인하지만, 결정계수는 ‘(회귀 모델+독립변수)-종속’ 간의

상관성을 설명하는 것으로 이해하면 된다.

- 결정계수는 0~1 사이의 값을 가지며 1에 가까울수록 회귀 모델이 높은 연관성을 가지고 있다고 해석할 수 있다.

- 결정계수 계산은 다음과 같다.

텍스트, 폰트, 화이트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

|  |  |
| --- | --- |
|  | SST ( Total Sum of Squares – 총 제곱합 )  - 실측값과 평균의 차이를 제곱한 합계  SSR ( Regression Sum of Squares – 회귀 제곱합 )  - 예측값과 평균의 차이를 제곱한 합계  SSE ( Error Sum of Squares – 잔차 제곱합 )  - 실측값과 예측값의 차이를 제곱한 합계 |

텍스트, 라인, 도표, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Figure 3. R2 Score에 필요한 데이터 예시

- 결정계수는 독립변수 구성이 늘어날수록 함께 증가하는 성질이 있기 때문에 ‘조정된 결정계수-Adjusted R2)’를 사용

한다.

텍스트, 폰트, 스크린샷, 화이트이(가) 표시된 사진

자동 생성된 설명

- python에서 scikit-learn Lib로 편리하게 결정계수를 계산할 수 있다.

ㆍEx) from sklearn.metrics import r2\_score

r2 = r2\_score( y, y\_pred ) # y는 종속변수, y\_pred는 모델의 predict() 결과로 얻은 예측값에 해당한다.

adj\_r2 = 1- (1- r2) \* ( (len(y) -1) / (len(y) – train\_x.shape[1] – 1) )

**Chap4. 피처(특성-Feature) 표준화(Scaling) / 정규화(Normalization)**

- Feature는 종류별로 값이 다양하게 분포하는 특징이 있다 (체중, 키, 빈도수, 거리 등). 따라서 분석 과정에서 값들을

비교할 때 값의 크기에 의해 특정 Feature에 편향(Bias)된 학습이 될 수 있다. 이를 방지하기 위해 값의 범위를 0~1

사이에 맞춰 동일한 비율로 분석에 참여할 수 있도록 한다.

▶ Scaler

- 대표적으로 많이 쓰이는 z-score Scaler와 Min-Max Scaler를 적용한다. Scikit-learn Lib에 있으며 연속형 변수에 적

용한다. 범주형 변수는 Encoder로 변환하며 이후에 기술한다.

1) StandardScaler ( from sklearn.preprocessing import StandardScaler )

- Z-score 정규화를 적용하는 방식. 평균이 0이고 표준편차가 1인 분포로 변환하며 정규분포를 따르는 데이터에 효

과적이다. 이상치가 있는 경우 Scaling 결과가 왜곡될 수 있으며 정규분포를 따르지 않는 데이터에 부적절하다.

도표, 라인, 그래프, 평행이(가) 표시된 사진

자동 생성된 설명

Figure 4. 정규분포를 따를 때, 왜도와 첨도는 0에 가까운 값을 보인다

- StandardScaler 적용 예시

|  |
| --- |
| from sklearn.preprocessing import StandardScaler  # x는 연속형 데이터를 가진 독립변수임을 가정  scaler = StandardScaler()  scaled\_x = scaler.fit\_transform(x) # scaler.fit(x), scaler.transform(x) 나누어 적용도 가능 |

ㆍ직접 계산식을 적용하려면 “ ( x – (X의 평균)) / (X의 표준편차) “ 로 계산한다.

Ex) df[‘col’] = df[‘col’].apply(lambda row: (row – df[‘col’].mean()) / df[‘col’].std()

|  |  |
| --- | --- |
| [[5.1 3.5 1.4 0.2]  [4.9 3. 1.4 0.2]  [4.7 3.2 1.3 0.2]  [4.6 3.1 1.5 0.2]] | [[-9.00681170e-01 1.01908435e+00 -1.34022653e+00 -1.31544430e+00]  [-1.14301091e+00 -1.31979479e-01 -1.34022653e+00 -1.31544430e+00]  [-1.38534205e+00 3.28414053e-01 -1.39706395e+00 -1.31544430e+00]  [-1.50652052e+00 9.82172869e-02 -1.28338910e+00 -1.31544430e+00]] |

Figure 5. Scaling 전과 후

2) Min-Max Scaler ( from sklearn.preprocessing import MinMaxScaler )

- 데이터를 선형 변환해 0~1 사이의 값으로 투영되도록 축소/확대하는 방법. 이미지 처리에서 픽셀의 정규화에 사용.

이상치에 매우 민감하기 때문에 정규분포의 형태가 변형될 수 있으며 신규 데이터에 대해 범위를 벗어날 수 있다.

- MinMaxScaler 적용 예시

|  |
| --- |
| from sklearn.preprocessing import MinMaxScaler  # x는 연속형 데이터를 가진 독립변수로 가정  scaler = MinMaxScaler()  scaled\_x = scaler.fit\_transform(x) # scaler.fit(x), scaler.transform(x) 나누어 적용도 가능 |

ㆍ직접 계산식을 적용하려면 “ (x- (X의 최솟값) ) / ((X의 최댓값) – (X의 최솟값))“ 식으로 계산한다.

|  |
| --- |
| # Min-Max Scaling  X\_min = np.min(df[‘col’]) # 또는 df.col.min()  X\_max = np.max(df.col) # 또는 df[‘col’].max()  df[‘col’] = df[‘col’].apply(lambda row: (row – X\_min) / (X\_max – X\_min)) |

▶ Encoder

- 범주형 데이터(Category)의 경우 대부분 문자열(Object)로 특정 상태를 설명하기 때문에 학습에 무리가 있다.

이를 적절하게 수치형으로 구분하고 학습에 적용할 수 있도록 데이터를 변환하는 방법이 Encoding이다.

1) LabelEncoder ( from sklearn.preprocessing import LabelEncoder )

- 각 Category에 순차적인 숫자를 할당한다. 가장 간단하고 메모리 효율적이라 보편적으로 사용한다.

- 순서에 의미가 없는 데이터지만 순서가 생기는 값의 구조로 인해 해석에 오류가 발생할 수 있다.

|  |
| --- |
| from sklearn.preprocessing import LabelEncoder  encoder = LabelEncoder()  encoded = encoder.fit\_transform(['red', 'blue', 'green'])  # 결과 : ['red', 'blue', 'green'] → [2 0 1] |

2) OneHotEncoder ( from sklearn.preprocessing import OneHotEncoder )

- 각 카테고리를 이진(0, 1) 열로 변환하여 구분한다. 순서에 의미가 없는 데이터에 적합한 변환 방식.

- 차원이 증가하기 때문에 메모리 사용량이 증가한다 (카테고리 항목별로 Column이 증가하기 때문).

- Series 객체로는 수행할 수 없는 구조이므로 넘파이 행렬(2차원) 또는 DataFrame(2차원)으로 작업해야 한다.

🡪 만약 벡터 같은 1차원 구조라면 reshape(-1, 1)로 변환해 Matrix 형태로 만들어야 한다.

- OneHotEncoder로 변환된 데이터를 이전 독립변수에 결합해 학습에 사용해야 하므로 전체적인 작업에 도움이 되는

파이프라인(Pipeline)을 적용할 수 있다. 이를 위해 ColumnTransformer 사용법도 같이 기술한다.

|  |  |  |
| --- | --- | --- |
| from sklearn.preprocessing import OneHotEncoder, StandardScaler  from sklearn.compose import ColumnTransformer  from sklearn.linear\_model import LogisticRegression  from sklearn.model\_selection import train\_test\_split  from sklearn.pipeline import Pipeline  import pandas as pd  encoder = OneHotEncoder(drop=’first’, sparse\_output=False)  # sparse\_output : 결과 반환 시 ndarray. 기본값은 희소행렬  # drop=’first’ : 인코딩 결과에서 첫 Column을 제거함 (다중공선성 방지 – 추가 학습 필요)  # Test DataFrame  sample = pd.DataFrame({  'age': [25, 30, 35, 40, 45, 50, 20, 32],  'salary': [30000, 45000, 55000, 60000, 70000, 80000, 28000, 45000],  'color': ['red', 'blue', 'green', 'red', 'blue', 'green', 'red', 'blue'],  'target': [0, 1, 1, 0, 1, 1, 0, 1] })S  X = data[[‘color’]] # DataFrame(2차원)구조. data[‘color’]는 Series(1차원) 구조이므로 오류 발생한다.  X\_encoded = encoder.fit\_transform(X) # One-Hot Encoding 결과를 확인할 수 있다.  # 학습용 데이터를 위해 ColumnTransformer로 전체 전처리 작업 진행  process = ColumnTransformer(  transformers=[  ( ‘num’, StandardScaler(), [‘age’, ‘salary’] ), # StandardScaler()를 적용할 수치형 데이터 세팅  ( ‘cat’, encoder, [‘color’] ) # OneHotEncoder()를 적용할 범주형 데이터 세팅  ]  )  # process로 변환된 데이터 확인하기.  checkData = process.fit\_transform(sample)  print(checkData)   |  |  | | --- | --- | | age salary color target  0 25 30000 red 0  1 30 45000 blue 1  2 35 55000 green 1  3 40 60000 red 0  4 45 70000 blue 1  5 50 80000 green 1  6 20 28000 red 0  7 32 45000 blue 1 | [[-1.02321629 -1.26499983 0. 1. ]  [-0.49167536 -0.3875433 0. 0. ]  [ 0.03986557 0.19742772 1. 0. ]  [ 0.5714065 0.48991323 0. 1. ]  [ 1.10294743 1.07488425 0. 0. ]  [ 1.63448835 1.65985526 1. 0. ]  [-1.55475722 -1.38199403 0. 1. ]  [-0.27905899 -0.3875433 0. 0. ]] |   Figure 6. 원본데이터(좌측)와 변환데이터(우측) 비교  # process 데이터를 바탕으로 분류 학습 진행  pipeline = Pipeline([  ( ‘preprocessor’, process), # 전처리 ‘preprocessor’ 모델로 process 활용  ( ‘classifier’, LogisticRegression()) # 분류 학습 모델로 LogisticRegression 적용.  ])  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)  pipline.fit(X\_train, y\_train) # Pipeline을 모델로 학습과 평가하는 과정은 동일하다. pred = pipline.predict(X\_test) |

- One-Hot Encoding 방법으로 Pandas에서 get\_dummies() 함수를 사용할 수 있다.

Ex) ret = sample[‘color’].get\_dummies()

ㆍ옵션 prefix=[‘col1’, ‘col2’] 🡪 Column명이 ‘col1’ 또는 ‘col2’로 시작하는 데이터만 반영.

ㆍdrop\_first = True/False 🡪 다중공선성 방지를 위한 첫 Column 제외 여부.

ㆍdummy\_na = True/False 🡪 인코딩 값이 전부 False로 나온 행에 대해 True/False 값을 보여주는 Column 추가.