

강원대학교
AI 소프트웨어학과

데이터 전처리

- 데이터프레임 -

데이터 프레임(Dataframe 이란?)

- 데이터 프레임은 프로그래밍 및 데이터 분석에 일반적으로 사용되는 표 형식의 데이터 구조
- 행과 열로 구성된 다양한 형태를 가지고 있는 리스트의 집합
- 데이터 프레임에서 각 열은 변수 또는 특정 속성을 나타냄
- 각 행은 개별 관찰 또는 데이터 포인트를 나타냄
- 데이터 프레임은 다목적이며 숫자, 범주 및 텍스트 데이터를 포함하여 다양한 유형의 데이터를 처리할 수 있음

데이터 프레임이란?

- **데이터 조작 용이성:** 데이터 프레임을 사용하면 행과 열의 추가, 삭제, 수정을 포함하여 데이터를 쉽게 조작할 수 있음
- **조건에 따른 행 필터링, 누락된 데이터 처리, 데이터 세트 병합과 같은 작업은 데이터 프레임을 사용하면 더 간단함**
- **CSV, Excel, SQL 데이터베이스 및 JSON과 같은 다양한 형식 및 소스에서 데이터 가져오기 및 내보내기에 대한 광범위한 지원을 제공함**

데이터 프레임이란?

```
pip install pandas  
import pandas as pd
```

```
data = np.array([  
    [1, 'Alice', 30],  
    [2, 'Bob', 25],  
    [3, 'Charlie', 35]  
])
```



길이가 같아야 생성가능

```
df = pd.DataFrame(data)
```


```
df = pd.DataFrame(data, columns=['ID', 'Name', 'Age'])
```

```
print(df)
```

데이터 프레임이란?

```
pip install pandas  
import pandas as pd
```

```
data = {  
    'Name': ['Alice', 'Bob', 'Charlie'],  
    'Age': [25, 30, 35],  
    'City': ['New York', 'Paris', 'London']  
}
```




길이가 같아야 생성가능

```
df = pd.DataFrame(data)
```

```
print(df)
```

TEXT

- 텍스트 파일은 데이터를 저장하고 표현하기 위해 간단하고 널리 사용되는 형식
- 텍스트 파일의 데이터는 일반적으로 각 데이터 포인트가 구분 기호(예: 쉼표 또는 탭)로 구분된 일반 텍스트로 저장됨
- 텍스트 파일의 주요 이점은 단순성과 다양한 프로그래밍 언어 및 소프트웨어 응용 프로그램과의 호환성이 좋음
- 복잡한 데이터 구조에 대한 지원 부족
- 데이터 조작 및 분석 기능이 제한됨
- 고급 데이터 작업을 위해 수동 처리가 필요함

 a - Windows 메모장

파일(F)	편집(E)	서식(O)	보기(V)	도움말(H)	
					키
					나이
					수학
					영어
					성적
					178
					17
					81
					92
					A
					188
					17
					71
					75
					B
					160
					17
					52
					36
					C
					170
					17
					55
					62
					C
					175
					17
					65
					47
					C
					181
					17
					71
					92
					B
					167
					17
					67
					78
					B
					158
					17
					84
					68
					B
					180
					17
					97
					91
					A
					172
					17
					100
					81
					A

Excel

- 특히 .xlsx 형식의 Excel 파일은 Microsoft Excel에서 만들고 사용하는 스프레드시트 파일
- Excel 파일은 데이터 구성, 조작, 시각화 및 분석을 위한 포괄적인 기능 세트를 제공함
- 복잡한 수식, 조건부 서식, 그래픽 표현 및 다양한 데이터 유형을 지원함
- Excel 파일은 사용자 친화적인 인터페이스와 광범위한 기능을 제공하여 기본 및 고급 데이터 관리 작업에 모두 적합함
- 그러나 Excel 파일은 크기가 상대적으로 큼
- 타사 소프트웨어와의 호환성 문제

	A	B	C	D	E
1	키	나이	수학	영어	성적
2	178	17	81	92	A
3	188	17	71	75	B
4	160	17	52	36	C
5	170	17	55	62	C
6	175	17	65	47	C
7	181	17	71	92	B
8	167	17	67	78	B
9	158	17	84	68	B
10	180	17	97	91	A
11	172	17	100	81	A

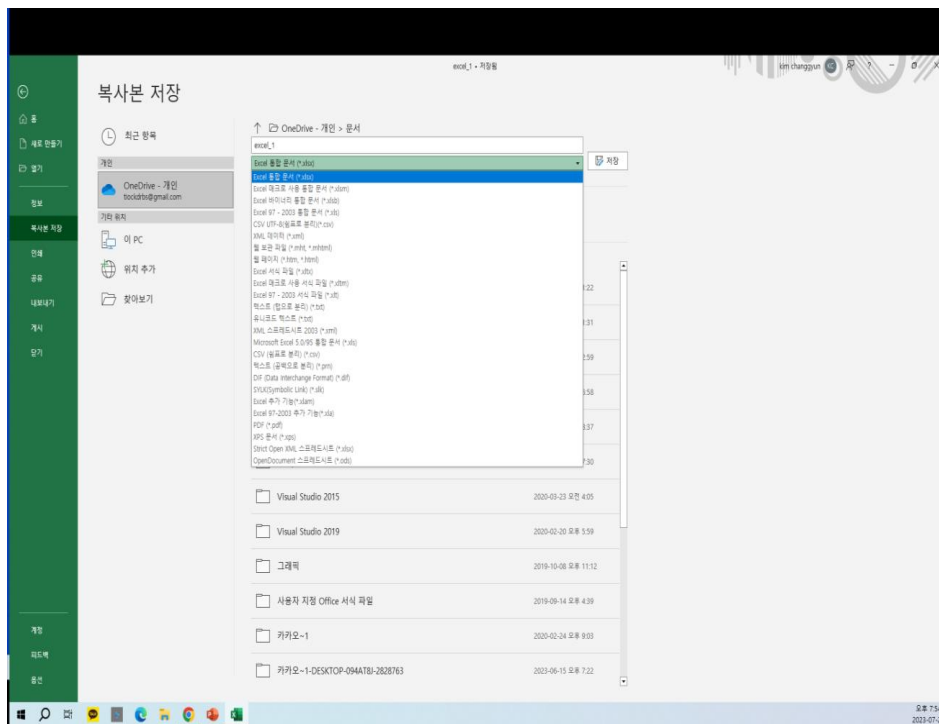
CSV(Comma-Separated Values)

- 파일은 테이블 형식 데이터 저장 및 교환에 일반적으로 사용되는 특정 유형의 텍스트 파일 형식
- CSV 파일에서 각 행은 데이터 레코드를 나타내며 행 내의 각 필드는 쉼표 또는 기타 지정된 구분 기호로 구분됨
- CSV 파일은 스프레드시트 소프트웨어 및 데이터베이스 응용 프로그램에서 광범위하게 지원되므로 데이터 공유 및 상호 운용성을 위해 많이 사용 가능함
- 데이터를 행과 열로 구성하기 위한 기본 구조를 제공하지만 복잡한 수식이나 서식 옵션은 지원하지 않음
- 복잡한 데이터 구조 또는 수식에 대한 제한된 지원
- 고급 서식 옵션이 부족함

	A	B	C	D	E
1	키	나이	수학	영어	성적
2	178	17	81	92	A
3	188	17	71	75	B
4	160	17	52	36	C
5	170	17	55	62	C
6	175	17	65	47	C
7	181	17	71	92	B
8	167	17	67	78	B
9	158	17	84	68	B
10	180	17	97	91	A
11	172	17	100	81	A

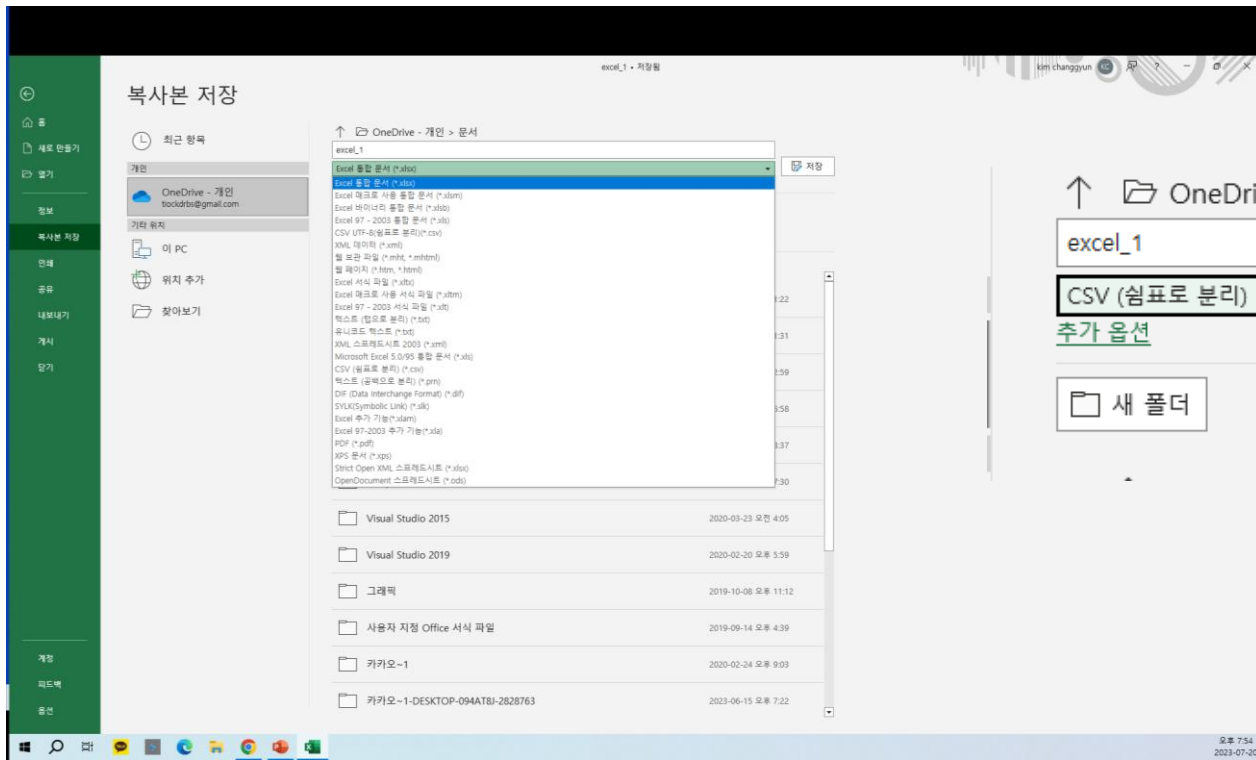
Excel to CSV(Comma-Separated Values)

- 다른 이름으로 저장 or 복사본 저장



Excel to CSV(Comma-Separated Values)

- 저장 탭에서 csv (쉼표로 분리) 선택



↑ OneDrive - 개인 > 문서

excel_1

CSV (쉼표로 분리) (*.csv)

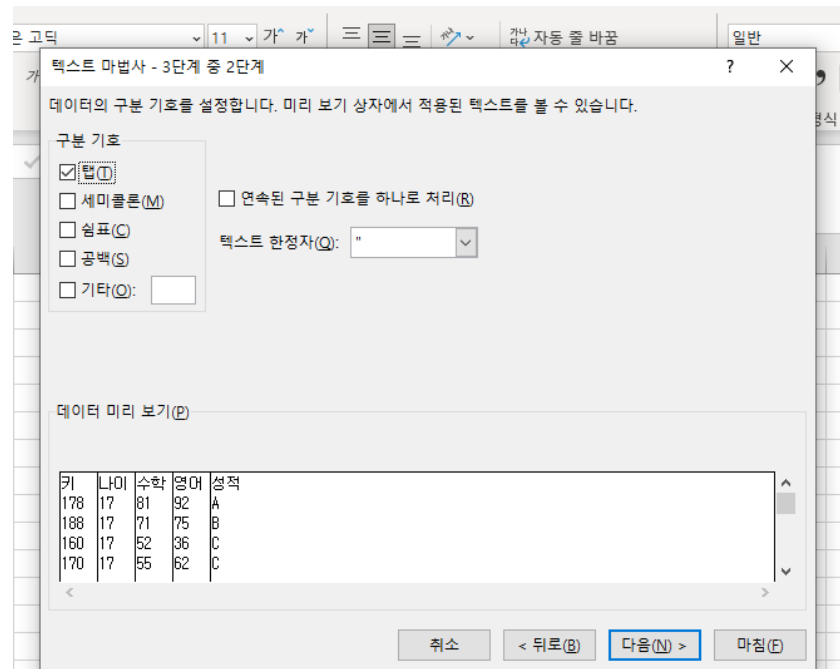
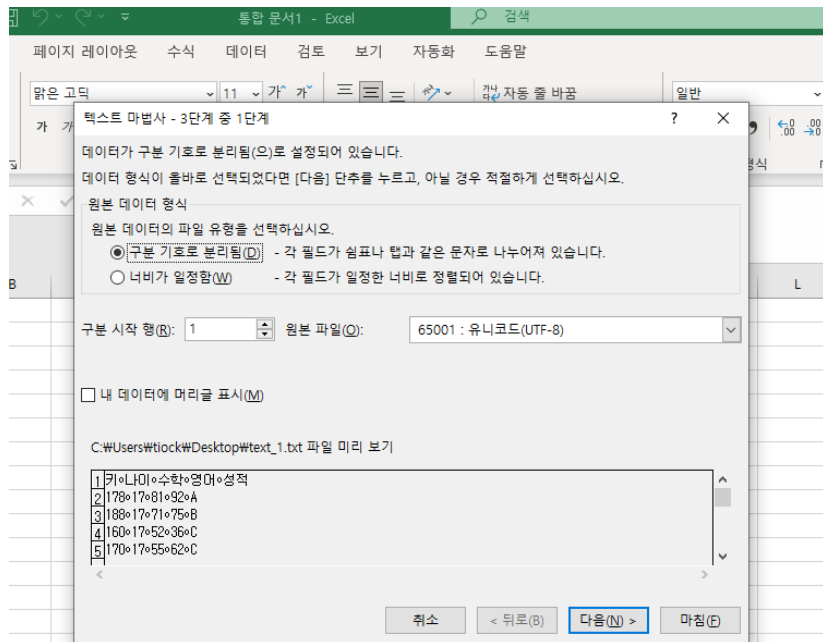
추가 옵션

새 폴더

01

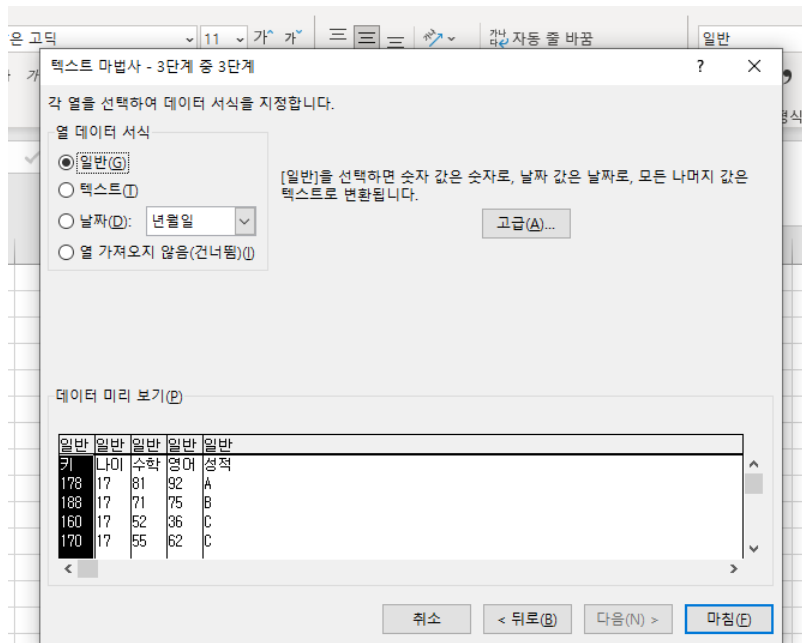
TEXT to CSV(Comma-Separated Values)

- Excel → 열기 → Text파일 선택



TEXT to CSV(Comma-Separated Values)

- Excel → 열기 → Text파일 선택



데이터 프레임 불러오기

```
df_h=pd.read_csv('data.csv', header=0)
```

```
df_nh= pd.read_csv('data.csv', header=None)
```

```
df_utf8 = pd.read_csv('data.csv', encoding='utf-8')
```

```
df_cp949 = pd.read_csv('data.csv', encoding='cp949')
```

```
df= pd.read_csv('data.csv', header=0, encoding='cp949')
```

- header = 데이터 프레임의 첫 행에 변수명이 있는지 없는지에 따라 0과 None으로 구분됨
- encoding = 영어 이외의 단어들이 들어 갔을 때, 오류가 발생할 수 있으므로 영어 이외의 단어를 인식할 수 있도록 인코딩 하는 방식(주로 한국어는 cp949, euc-kr, utf-8)세 가지로 저장됨

데이터 프레임 불러오기

```
df=pd.read_csv("data/국영수.csv", encoding='cp949')
```

df.shape → 데이터 프레임의 차원확인

df.columns → 데이터 프레임의 변수확인

df.dtypes → 데이터 프레임의 변수 타입확인

df.describe() → 수치형 변수들에 대한 요약

df.head() → 위에서 다섯개의 값 도출

df.tail() → 아래에서 다섯개의 값 도출

df.isnull().sum() → 결측값이 몇개인지

df['column'].value_counts() → 카테고리 변수의 어떻게 분포되어 있는지

df['column'].unique() → 카테고리 변수의 값들 중 독립적인 값 보기

df['column'].nunique() → 카테고리 변수의 값들 중 독립적인 값 갯수

```
#데이터 프레임 불러오기
```

```
df=pd.read_csv("data/국영수.csv",encoding='cp949')
```

```
#해당 변수 추출
```

```
df["변수명"]
```

```
#새로운 변수 생성
```

```
df["새로운 변수명"]=df["변수명"]+df["변수명"]
```

```
print(df)
```

방법 1

#하나의 변수 추출

```
df["변수명"]
```

#두개 이상의 변수 추출

```
columns = ['column_a', 'column_b', 'column_c']
```

```
selected_columns = df[columns]
```

방법 2

#하나의 변수 추출

```
df.loc[:, "변수명"] → loc : 변수명을 입력
```

```
df.iloc[:,0] → iloc : 변수위치를 입력
```

#두개 이상의 변수 추출

```
columns = ['column_a', 'column_b', 'column_c']
```

```
selected_columns = df.loc[:, columns]
```

```
selected_columns = df.iloc[:, [0,1,2]]
```


#특정 변수의 지정한 위치 값 추출
`df["변수명"][2]`

#특정 변수의 모든 값을 추출
`df["변수명"][:]`

#4번째 전까지의 값을 추출
`df["변수명"][:4]`

#하나의 변수 삭제

```
df = df.drop(['삭제 변수명'], axis=1) #axis = 0 → 행을 삭제, 1 → 열을 삭제
```

#두개 이상의 변수 삭제

```
df = df.drop(['column_name1', 'column_name2'], axis=1)
```

```
data = {  
    'A': [1, 2, 3, 4],  
    'B': [4.1, 3.5, 2.1, 5.7],  
    'C': ['one', 'two', 'three', 'four']  
}
```

```
df = pd.DataFrame(data)
```

```
df["변수명"].astype('category')
```

```
df['A'] = df['A'].astype(float)  
df['B'] = df['B'].astype(int)  
df['A'] = df['A'].astype('str')  
df['C'] = df['C'].astype('category')
```



df.dtypes → 변수별 type 확인

```
df['A'].dtype  
df['B'].dtype  
df['A'].dtype  
df['C'].dtype
```

```
df = df.rename(columns={'old_name': 'new_name'})
```

```
df = df.rename(columns={  
    'old_name1': 'new_name1',  
    'old_name2': 'new_name2',  
    'old_name3': 'new_name3'  
})
```

`df.dropna()`는 데이터프레임의 불완전한 데이터가 있는 행을 삭제함

`df.fillna(채워넣을 값)` 기본값을 제공하여 데이터 누락으로 인해 데이터에 대한 작업이 실패하지 않도록 하기 위해 자주 사용됨

→ `df.fillna(df.median())` #중앙값으로 대체

→ `df.fillna(df.mean())` #평균값으로 대체

```
data = {  
    'Numeric1': [1, np.nan, 3, 4],  
    'Numeric2': [5, 6, np.nan, 8.2],  
    'Category': ['dog', np.nan, 'cat', 'dog']  
}
```

```
df = pd.DataFrame(data)
```

```
A=df.select_dtypes(include=["number"]).columns #수치가 포함되어 있는 변수들만
```

```
print(df[A])
```

```
B=df.select_dtypes(include=[ ' object ' , ' category ' ]).columns #문자와 카테고리 변수들만
```

```
print(df[B])
```

```
df[A].fillna(df[A].median())
```

```
df[A].fillna(df[A]. mean())
```

```
df1 = pd.DataFrame({  
    'a': ['a0', 'a1', 'a2', 'a3'],  
    'b': ['b0', 'b1', 'b2', 'b3'],  
    'c': ['c0', 'c1', 'c2', 'c3']  
})
```

```
df2 = pd.DataFrame({  
    'a': ['a2', 'a3', 'a4', 'a5'],  
    'b': ['b2', 'b3', 'b4', 'b5'],  
    'c': ['c2', 'c3', 'c4', 'c5'],  
    'd': ['d2', 'd3', 'd4', 'd5']  
})
```

`pd.concat([df1, df2], axis=0) → 행기준으로 합쳐짐`

`pd.concat([df1, df2], axis=1) → 열기준으로 합쳐짐`

```
data = {  
    'Numeric1': [1, np.nan, 3, 4],  
    'Numeric2': [5, 6, np.nan, 8.2],  
    'Category': ['dog', np.nan, 'cat', 'dog']  
}
```

```
df = pd.DataFrame(data)
```

```
df[(df['Numeric2'] >= 6) & (df['Numeric2'] <= 10)]  
df[(df['Numeric2'] >= 6) | (df['Numeric2'] <= 10)]
```



```
data = {  
    'Numeric1': [1, np.nan, 3, 4],  
    'Numeric2': [5, 6, np.nan, 8.2],  
    'Category': ['dog', np.nan, 'cat', 'dog']  
}
```

```
df = pd.DataFrame(data)
```

```
df[df['Numeric2']>5] = "5보다 크다"
```



어떠한 문제가 발생함

```
data = {  
    '나이': [22, 35, 14, 55, 42]  
}
```

```
df = pd.DataFrame(data)
```

```
def categorize (data):  
    if 0 < data <= 10:  
        return "10대"  
    elif 10 < data <= 20:  
        return "20대"  
    elif 20 < data <= 30:  
        return "30대"  
    else:  
        return "30대 이상"
```

```
df['검사나이'] = df['나이'].apply(categorize)
```

apply : 데이터의 행단위로 무언가를 연속으로 진행할 경우

```
data = {  
    'Numeric1': [1, np.nan, 3, 4],  
    'Numeric2': [5, 6, np.nan, 8.2],  
    'Category': ['dog', np.nan, 'cat', 'dogg']  
}
```

```
df = pd.DataFrame(data)
```

```
filter_df = df[df['Category'] == 'dog']
```

```
filter_df_contains = df[df['Category'].str.contains('do', na=False)]
```

```
df = pd.DataFrame({  
    'variable': ['value1-value2-value3', 'value4-value5', 'value6']  
})
```

```
split_df = df['variable'].str.split('나누고 싶은 기준 문자', expand=True)  
#expand → 나눠서 각각 데이터프레임을 생성하도록 만드는 옵션
```

```
split_df.columns = ['var1', 'var2', 'var3']
```

```
df = pd.DataFrame({  
    'variable': ['value1-value2-value3', 'value4-value5', 'value6']  
})
```

`df['variable'] = df['variable'].str[0:5]` → 해당 위치에 포함되는 문자를 추출함

`df['variable'] = df['variable'].str[0:5] + df['variable'].str[6:12]`

```
df['var1'] = df['var1'].astype(str)  
df['var2'] = df['var2'].astype(str)  
df['var3'] = df['var3'].astype(str)
```

```
df['combined'] = df['var1'] + '-' + df['var2'] + '-' + df['var3']
```

```
text = "apple, "  
result = text.rstrip(", ")  
print(result)
```

```
text = "banana, , "  
result = text.rstrip(", ")  
print(result)
```

```
text = ",orange, pear"  
result = text.lstrip(", ")  
print(result)
```

```
df = pd.DataFrame({  
    '문항1': ['1', '2', '4', '2'],  
    '문항2': ['2', '2', '3', '2']  
})
```

```
df = pd.DataFrame(data)
```

```
df[ " 문항1 " ] = df[ " 문항1 " ].str.rstrip( " , " )  
df[ " 문항2 " ] = df[ " 문항2 " ].str.lstrip( " , " )
```

```
print(df)
```



```
data = {  
    '문항1': [0, 'A', 0, 'D'],  
    '문항2': ['B', 0, 'C', 0],  
    '문항3': [0, 'A', 0, 0],  
    '문항4': [0, 0, 'C', 0]  
}
```

```
df = pd.DataFrame(data)
```

```
df_replaced = df.replace(0, np.nan)
```

```
# 행단위로 작업을 수행함 : bfill(axis=1)
```

```
# 채운결과에서 각행의 첫번째 값을 가지고 온다는 의미 : iloc[:, 0]
```

```
df['하나의_항목'] = df[['문항1', '문항2', '문항3', '문항4']].bfill(axis=1).iloc[:, 0]
```

```
df = pd.DataFrame({  
    'variable': ['value1-value2-value3', "value3-value1-value5", "vvdvalue165656"]  
})
```

#특정 문자가 포함되어 있으면 전체를 해당 문자로 변경(앞에서→뒤로 이동하면서 찾음)

```
df["new_variable"]=df['variable'].str.replace(r'value1.*', 'value1', regex=True)
```

```
print(df["new_variable"])
```

```
df = pd.DataFrame({  
    'variable': ['value1-value2-value3', "value1-value4-value5", "vvdvalue165656"]  
})
```

#특정 문자가 포함되어 있으면 전체를 해당 문자로 변경(전체에서 찾을)

```
df["new_variable"]=df['variable'].str.extract(r'(value1)')  
print(df["new_variable"])
```

```
df["new_variable"]=df['variable'].str.extract(r'(value2)')  
print(df["new_variable"])
```

```
df = pd.DataFrame({  
    'variable': ['value1-value2-value3', 'value1-value4-value5', 'vvdvalue1']  
})
```



```
conti=df[df['variable'].str.contains('value2', na=False)]
```



```
conti['variable'].str.extract(r'(value2)')
```

```
data = {'date': ['202103151345', '20210315 1345', '0715', '07-15', '20210315', '1']}  
df = pd.DataFrame(data)
```

```
df['date']=pd.to_datetime(df['date'], format='%Y%m%d %H%M' , errors='coerce')
```

```
df['date']=pd.to_datetime(df['date'], format='%H%M' , errors='coerce').dt.time
```

```
df['date']=pd.to_datetime(df['date'], format='%Y%m%d', errors='coerce')
```

errors="raise" 변환이 실패하면 오류가 발생함

errors="coerce" 변환이 실패하면 값이 "NaT"로 변환됨

errors="ignore" 변환이 실패하면 원본 데이터의 값을 반환함

```
data = {'hour': ['1', '2', '12', '22', '11']}  
df = pd.DataFrame(data)
```

```
pd.to_datetime(df['date'], format='%H%M', errors='coerce').dt.time
```

```
df['hour'].str.rjust(문자의 길이, fillchar=채우고 싶은 문자) #왼쪽 부터 채움  
df['hour']=df['hour'].str.rjust(4, fillchar='0') #왼쪽 부터 채움
```

```
df['hour'].str.ljust(문자의 길이, fillchar=채우고 싶은 문자) #오른쪽 부터 채움  
df['hour']=df['hour'].str.ljust(4, fillchar='0')
```

```
df.to_csv('new_program.csv', index=False)
```



전처리한 데이터프레임



저장하고 싶은 경로



Index를 표시할지 하지 않을지

```
df.to_csv('new_program.csv', index=False, encoding='~~')
```



인코딩 할 언어를 지정함

肄뵼뵼	異뵼뵼?뵼	?뵼뵼	?뵼?뵼뵼뵼?뵼뵼	????
A	10:50:00	53.25	??95	12 25
B	12:05:00	54.75	??97	5 21
C	17:20:00	68.25	以?97	4 7
D	11:01:00	72.25	??95	2 4
E	12:20:00	59.75	以?94	7 3
F	14:25:00	53.25	??96	5 21



A	B	C	D	E	F	G
코드	출생시간	평균	평가분류	년도	월	일
A	10:50:00	53.25	하	95	12	25
B	12:05:00	54.75	하	97	5	21
C	17:20:00	68.25	중	97	4	7
D	11:01:00	72.25	상	95	2	4
E	12:20:00	59.75	중	94	7	3
F	14:25:00	53.25	하	96	5	21