



CSV (comma-separated values) CSV, TSV, SSV

CSV는 몇 가지 필드를 쉼표(,)로 구분한 텍스트 데이터 및 텍스트 파일이다.

확장자는 .csv이며 MIME 형식은 text/csv이다.

comma-separated variables라고도 한다.

MIME-Type, Content-Type이란 무엇인가?

우리가 클라이언트 브라우저로 어떤 자원을 보낼 때(어떤 형태의 파일이나 문서 등), 웹 서버는 일련의 HTTP 헤더로 파일이나 자원을 포함하는 바이트의 Stream을 앞에 보낸다. 이런 헤더는 클라이언트에게 웹 서버와 커뮤니케이션 세부사항을 묘사한다.

예를 들어, 헤더는 사용되고 있는 웹 서버의 소프트웨어의 타입, 서버의 날짜와 시간, HTTP 프로토콜, 사용 중인 커넥션 타입 등을 지정한다.

헤더는 또한 클라이언트가 이런 가상 패스나 도메인에 대해서 저장해야 할 쿠키를 포함한다.

이와 관련해서 가장 중요한 것은 헤더는 또한 보내지는 자원의 content 타입이 포함되는 것이다.

이것은 Content-Type 헤더에 의해 지정되는데, 이 값은 표준 MIME-Type의 하나이다.

MIME-Type을 살펴봄으로써 브라우저는 데이터를 나타내는데 어떤 종류의 파일 Stream인지를 알고 있다.

HTML 페이지에 MIME-Type을 가진다.

오래전부터 스프레드시트나 데이터베이스 소프트웨어에서 많이 쓰였으나 세부적인 구현은 소프트웨어에 따라 다르다. 그것들을 추가한 형태가 2005년 10월 RFC 4180에서 Informational(IESG의 외부에서 결정된 유용한 정보의 제공)로 사양이 문서화 되었다.

비슷한 포맷으로는 탭으로 구분하는 'tab-separated values'(TSV)나, 반각 스페이스로 구분하는 'space-separated values'(SSV) 등이 있으며, 이것들을 합쳐서 character-separated values (CSV), delimiter-separated values라고 부르는 경우가 많다.

csv 파일은 숫자나 문자열로 구성되어 있는 표 (혹은 스프레드시트) 형태의 데이터가 일반 텍스트plain-text 로 저장된다. 일반 텍스트로 저장되므로 이를 저장하거나 전송하고 처리할 수 있는 프로그램이 다양하다. 이점이 엑셀 파일과 비교했을 때 csv 파일의 가장 큰 장점 이다.

엑셀과 같은 스프레드시트 프로그램뿐만 아니라 워드프로세서 또는 간단한 텍스트 편집기로도 csv 파일을 처리할 수 있다. 반면에 엑셀 파일은 그것을 처리할 수 있는 프로그램이 매우 제한적이다.

엑셀은 그 자체로 강력한 도구지만, 엑셀 파일은 엑셀에서 수행할 수 있는 작업만이 가능하다면,

csv 파일은 자신이 원하는 작업에 적합한 툴로 데이터를 전송하거나 파이썬을 사용하여 직접 만든 툴에서도 사용 가능하다. 물론 CSV 파일로 작업할 때에는 엑셀 파일만의 편리함은 사라진다.

CSV 파일의 각 셀은 자료형이 없는 원시 데이터이다.

CSV 파일은 수식 없이 오직 데이터만 저장된다.

하지만 데이터 저장소인 CSV 파일과 데이터 처리인 파이썬 스크립트가 분리됨으로써 다른 데이터셋에 똑 같은 처리 과정을 보다 쉽게 적용할 수 있게 된다.

이러한 분리를 통해 데이터 자체와 데이터 처리 과정 모두에서 오류를 찾아내기가 더욱 쉬워지고, 오류가 전파되는 것은 더욱 어려워지게 된다.

실습] supplier_data.csv 파일 생성(엑셀) -> 내용 입력 -> csv 저장 (xls ->xlsx)

엑셀에 다음과 같은 내용을 입력한 후 Supplier_data.csv 파일로 저장한다.

Supplier Name	Invoice Number	Part Number	Cost	Purchase Date
Supplier X	001-1001	2341	\$500.00	1/20/19
Supplier X	001-1001	2341	\$500.00	1/20/19
Supplier X	001-1001	5467	\$750.00	1/20/19
Supplier X	001-1001	5467	\$750.00	1/20/19
Supplier Y	50-9501	7009	\$250.00	1/30/19
Supplier Y	50-9501	7009	\$250.00	1/30/19
Supplier Y	50-9505	6650	\$125.00	2019-03-14
Supplier Y	50-9505	6650	\$125.00	2019-03-14
Supplier Z	920-4803	3321	\$615.00	2019-03-14
Supplier Z	920-4804	3321	\$615.00	2019-10-14
Supplier Z	920-4805	3321	\$615.00	2/17/19
Supplier Z	920-4806	3321	\$615.00	2/24/19

이 파일은 단순한 텍스트 파일이고 각 행마다 다섯 개의 값이 쉼표로 구분되어 저장되어 있다.

csv 파일에서 구분자로 사용된 쉼표는 엑셀 파일의 열에 해당한다고 볼 수 있다.

=====

CSV 모듈을 사용하지 않는 기본 파이썬 코드

파일명 : 1csv_simple_parsing_and_write.py

```
#!/usr/bin/env python3
```

```
import sys
```

```
input_file = sys.argv[1]
```

```
output_file = sys.argv[2]
```

```
with open(input_file, 'r', newline='') as filereader :
```

```
    with open(output_file, 'w', newline='') as filewriter :
```

```
        header = filereader.readline()
```

```
        header = header.strip()
```

```
        header_list = header.split(',')
```

```
        print(header_list)
```

```
        filewriter.write(','.join(map(str,header_list))+'\n')
```

```
        for row in filereader:
```

```
            row = row.strip()
```

```
            row_list = row.split(',')
```

```
            print(row_list)
```

```
            filewriter.write(','.join(map(str, row_list))+'\n')
```

👉 스크립트 설명

1행은 다른 운영체제 간에 통용될 수 있는 스크립트를 만들게 해주는 셔뱅이다.

👉 `#!/usr/bin/env python3`

2행에서는 `sys` 모듈을 임포트한다.

파이썬에 기본으로 내장되어 있는 `sys` 모듈은 명령줄에서 추가적으로 입력된 인수를 스크립트로 넘겨준다.

👉 `import sys`

4행과 5행에서는 `sys` 모듈의 `argv`라는 인수를 사용한다.

이 인수는 명령 줄 실행 시에 추가적으로 입력되는 인수를 리스트 자료형으로 받는다.

👉 `input_file = sys.argv[1]`

👉 `output_file = sys.argv[2]`

다음은 윈도우 환경에서 입력되는 `csv` 파일을 읽고,

출력 `csv` 파일을 쓰는 데 사용되는 일반적인 형태의 명령 줄 인수이다.

python script_name.py "C:\path\to\input_file.csv" "C:\path\to\output_file.csv"

첫 번째 단어인 `python`은 파이썬을 사용하여 나머지 명령 줄 인수를 처리하도록 명령한다.

파이썬은 나머지 인수를 `argv`라는 특별한 리스트에 할당한다.

리스트의 첫 번째 위치인 `argv[0]`는 `script_name.py` 파일을 가리 킨다.

그 다음 명령 줄 인수는 입력 `csv` 파일의 경로와 파일명인 "C:\path\to\input_file.csv" 이다.

파이썬은 이 값을 `argv[1]`에 할당하므로 위에서 본 파이썬 스크립트의 4행에서 이 값을 `input_file` 변수에 할당한다.

마지막 명령 줄 인수인 "C:\path\to\output_file.csv"는 출력 csv 파일의 경로와 파일명 이다.

파이썬은 이 값 역시 argv[2]에 할당하고 5행에서 이 값을 output_file 변수에 할당한다.

7행은 input_file을 filereader라는 파일 객체로 열어주는 with 문이다.

open() 함수에서 'r' 은 읽기 모드를 할당한다.

이는 input file이 읽기 위해 열렸음을 의미한다.

 **with open(input_file, 'r', newline='') as filereader:**

8행은 output_file을 파일 객체인 filewriter로 여는 명령문이다.

'w' 는 쓰기 모드를 할당한다.

즉 output_file은 쓰기 위해 열렸음을 의미한다.


with 문은 with 문이 종료될 때 자동으로 파일 객체를 닫으므로 편리하다.

 **with open(output_file, 'w', newline='') as filewriter:**

9행에서는 filereader 객체의 readline() 함수를 사용하여 입력 파일의 첫 번째 행(헤더행)을 문자열로 읽고 이를 header라는 변수에 할당한다.

 **header = filereader.readline()**

10행에서는 strip() 함수를 사용하여 header에 있는 문자열의 def 끝에서 공백, 탭 및 개행 문자 (\n) 등을 제거한 뒤, header에 다시 할당한다.

 **header = header.strip()**

11행에서는 `split()` 함수를 사용하여 문자열을 쉼표 기준으로 구분하여 리스트에 할당한다.
리스트의 각 원소는 입력 파일의 각 열의 헤더이며 `header_list`라는 변수로 할당된다.

☞ `header_list = header.split(',')`

12행은 `header_list`의 값 (즉 헤더 행)을 화면에 출력하는 `Print()`문이다.

☞ `print(header_list)`

13행에서는 `filewriter` 객체의 `write()` 함수를 사용하여 `header list`의 각 값을 출력 파일에 쓴다.

`map()` 함수는 `header_list`의 각 원소에 `str()` 함수를 적용하여 각 원소를 문자열로 만든다.

그 다음 `join()` 함수는 `header_list`의 각 값 사이에 쉼표를 삽입하고 리스트를 문자열로 변환한다.

그 다음 개행문자를 문자열 끝에 추가한다.

끝으로 `filewriter` 객체는 그 문자열을 출력 파일의 첫 번째 행에 기록한다.

☞ `filewriter.write(','.join(map(str,header_list))+ '\n')`

14행에서는 `for` 반복문을 작성하여 입력 파일의 나머지 행을 반복한다.

☞ `for row in filereader:`

15행은 `strip()` 함수를 사용하여 `row`라는 변수의 문자열 양끝에서 공백, 탭 및 개행문자를 제거한 뒤 `row`에 다시 할당한다.

☞ `row = row.strip()`

16행에서는 `split()` 함수를 사용하여 문자열을 쉼표 기준으로 분리하여 리스트에 할당한다.

리스트의 값은 각 행의 열 값이고, `row_list` 라는 변수에 리스트가 할당된다.

☞ `row_list = row.split(',')`

17행에서는 row_list의 값을 화면에 출력하고,

```
print(row_list)
```

18행은 값을 출력 파일에 기록한다.

이 스크립트는 입력 파일의 모든 행에 대해서 15행에서 18행까지를 실행한다.

이 네 줄은 14행의 for 반복문 아래에 들여쓰기가 적용되어 있기 때문이다.

```
filewriter.write(','.join(map(str,row_list))+'\n')
```

명령 줄에서 실행한다.

```
# python 1csv_simple_parsing_and_write.py supplier_data.csv 1output.csv
```

 **PANDAS**

파일명 : **pandas_parsing_and_write.py**

팬더스를 이용해 csv 파일을 처리해보겠다.

```
#!/usr/bin/env python3
```

```
import sys
```

```
import pandas as pd
```

```
input_file = sys.argv[1]
```

```
output_file = sys.argv[2]
```

```
data_frame = pd.read_csv(input_file)
```

```
print(data_frame)
```

```
data_frame.to_csv(output_file, index=False)
```


스크립트 설명

팬더스 버전의 스크립트에서는 `data_frame`이라는 변수를 만들었다. 리스트, 딕셔너리, 튜플과 마찬가지로 데이터프레임도 데이터를 저장하는 하나의 방식이다. 데이터를 리스트의 목록으로 파싱할 필요 없이

'표' 형 태로 만들어 저장한다. 데이터 프레임은 팬더스 패키지의 자료형 중 하나이므로 스크립트에서 팬더스 패키지를 임포트해야 사용할 수 있다. 이 스크립트에서 `data_frame`이라는 변수명을 사용한 것은 리스트 자료형의 변수에 리스트라는 변수명을 사용하는 것과 같은 원리로, 학습 단계에서는 유용하지만 앞으로는 보다 설명적인 변수명을 사용하는 것이 좋다.

명령 줄에서 실행한다.

```
# python pandas_parsing_and_write.py supplier_data.csv 1pandas_output.csv
```

👉 기본 문자열 파싱이 실패하는 경우

supplier_data.csv 필드값 수정

D12 : \$615.00 -> \$6,015.00

D13 : \$615.00 -> \$1,006,015.00

스크립트는 헤더 행과 처음 10개의 행에는 데이터 값에 쉼표가 포함되어 있지 않으므로 올바르게 실행된다. 하지만 마지막 두 행에는 데이터 값 자체에 쉼표가 포함되어 있으므로 파싱이 실패할 것이다.

CSV 모듈을 사용하는 기본 파이썬 코드

파일명 : 2csv_reader_parsing_and_write.py

```
#!/usr/bin/env python3

import csv

import sys

input_file = sys.argv[1]

output_file = sys.argv[2]

with open(input_file, 'r', newline='') as csv_in_file:

    with open(output_file, 'w', newline='') as csv_out_file:

        filereader = csv.reader(csv_in_file, delimiter=',')

        filewriter = csv.writer(csv_out_file, delimiter=',')

        for row_list in filereader:

            filewriter.writerow(row_list)
```

2행에서 CSV 모듈을 임포트하여 내장 함수를 사용하여 입력 파일을 파싱하고 출력 파일에 쓸 수 있게 한다.

두 번째 with 문 다음에 있는 **10행은** csv 모듈의 reader() 함수를 사용하여 filereader라는 입력 파일을 읽는 객체를 만든다.

11행은 csv 모듈의 writer() 함수를 사용하여 출력 파일에 쓰는데 사용할 filewriter라는 객체를 만든다.

이 함수들의 두 번째 인수(delimiter= ' , ')는 기본값 인수이므로 입력 및 출력 파일이 쉼표로 구분된 경우라면 굳이 쓰지 않아도 된다. 다른 구분 기호, 예를 들어 세미콜론(;)이나 탭 (\t)으로 구분된 입력 파일을 읽거나 출력 파일을 작성하고 싶다면 구분 기호 인수를 지정해야 한다.

13행은 filewriter 객체의 writerow() 함수를 사용하여 각 행의 값을 리스트 자료형으로 출력 파일에 쓴다.

명령 줄에서 실행한다.

```
# python 2csv_reader_parsing_and_write.py supplier_data.csv 1output.csv
```

2CSV는 Pandas 파일 없음.

입력 파일에서 특정 행을 필터링하는 세 가지 방법을 소개한다.

- (1) 특정 조건을 충족하는 행을 필터링하기
- (2) 특정 집합의 값을 포함하는 행을 필터링하기
- (3) 정규 표현식을 활용해 필터링하기 # re (Regular Expression) 모듈

👉 특정 조건을 충족하는 행의 필터링 기본 파이썬 코드

파일명 : 3csv_reader_value_meets_condition.py

조건:

Supplier Name이 Supplier Z이거나 또는 Cost가 \$600.00 이상인 행만 필터링 하고
그 결과를 출력 파일에 기록한다.

```
#!/usr/bin/env python3
```

```
import csv
```

```
import sys
```

```

input_file = sys.argv[1]

output_file = sys.argv[2]

with open(input_file, 'r', newline='') as csv_in_file:

    with open(output_file, 'w', newline='') as csv_out_file:

        filereader = csv.reader(csv_in_file)

        filewriter = csv.writer(csv_out_file)

        header = next(filereader)

        filewriter.writerow(header)

        for row_list in filereader:

            supplier = str(row_list[0]).strip()

            cost = str(row_list[3]).strip('$').replace(',', '')

            if supplier == 'Supplier Z' or float(cost) > 600.0:

                filewriter.writerow(row_list)

```

12행은 csv 모듈의 next() 함수를 사용하여 입력 파일의 첫 번째 행(헤더행)을 header라는 리스트 변수로 할당한다.

13행은 그 헤더 행을 출력 파일에 쓴다.

15행은 각 행의 Supplier Name 열에 해당하는 값을 가져와서 supplier라는 변수에 할당한다.

리스트 인덱싱 (row_list[0])을 사용하여 각 행의 첫 번째 열(Supplier Name)의 값을 가져온 다음 str() 함수를 사용하여 값을 문자열로 변환한다.

그 다음 strip() 함수를 사용하여 문자열의 양끝에서 공백, 탭, 개행문자 등을 제거한다.

끝으로 이 문자열을 변수 supplier에 할당한다.

16행은 각 행의 Cost 열에 해당하는 값을 가져와서 cost라는 변수에 할당한다.

이를 위해 리스트 인덱싱 (row_list[3])을 통해 각 행의 네 번째 열 (Cost 열)의 값을 가져온 다음 str() 함수를 사용하여 값을 문자열로 변환한다.

그 다음 strip () 함수를 사용하여 문자열에서 달러(' \$ ') 기호를 제거한 뒤,

replace() 함수를 사용하여 쉼표 (', ') 도 제거한다,

마지막으로 이 문자열을 변수 cost에 할당한다.

17행은 각 행의 값들이 두 가지 조건에 부합하는지 판별하기 위해 if 문을 썼다.

즉 Supplier Name이 Supplier Z이거나 Cost가 \$600.00 이상인 행을 필터링한다.

첫 번째 조건은 if와 or 사이에 있고 supplier라는 변수의 값이 Supplier Z와 일치하는지를 판별한다.

두 번째 조건은 or와 : 사이에 있으며 부동소수점 숫자로 변환된 cost 변수의 값이 600.0보다 큰가를 판별한다.

18행에서는 filewriter의 writerow() 함수를 사용하여 조건을 만족하는 행을 출력 파일에 작성 한다.

명령 줄에서 실행한다.

```
# python 3csv_reader_value_meets_condition.py supplier_data.csv 3output.csv
```

 **Pandas**

파일명 : pandas_value_meets_condition.py

```
#!/usr/bin/env python3
```

```
import pandas as pd
```

```
import sys
```

```
input_file = sys.argv[1]
```

```
output_file = sys.argv[2]
```

```
data_frame = pd.read_csv(input_file)
```

```
data_frame['Cost'] = data_frame['Cost'].str.strip('$').astype(float)
```

```
data_frame_value_meets_condition = data_frame.loc[(data_frame['Supplier Name']₩  
.str.contains('Z')) | (data_frame['Cost'] > 600.0), :]
```

```
data_frame_value_meets_condition.to_csv(output_file, index=False)
```

팬더스는 특정 행과 열을 동시에 선택할 수 있는 loc() 함수를 제공한다.

선택표를 기준으로 앞에는 행을 필터링하는 조건을 지정하고 뒤에는 열을 필터링하는 조건을 지정하면 된다.

아래 스크립트는 loc() 함수를 이용해 Supplier Name 열의 값에 Z가 포함되거나 Cost 열의 값이 600.0보다 큰 모든 행을 필터링 조건으로 지정한다.

이 스크립트는 팬더스를 사용하여 csv 파일을 파싱하고,

조건을 충족하는 행을 필터링하여 출력 파일에 작성한다.

밑에서 둘째 줄을 보면 코드가 너무 길어서 역슬래시(₩)를 기준으로 텍스트를 두 줄로 나누었다.

명령 줄에서 스크립트를 하면, 데이터가 입력되고 출력 파일이 생성된다.

loc() 함수의 인수를 사용하면 데이터에서 다양한 선택이 가능하다.

명령 줄에서 실행한다.

```
# python pandas_value_meets_condition.py supplier_data.csv 3pandas_output.csv
```

👉 특정 집합의 값을 포함하는 기본 파이썬 코드

파일명 : 4csv_reader_value_in_set.py

```
#!/usr/bin/env python3
```

```
import csv
```

```
import sys
```

```
input_file = sys.argv[1]
```

```
output_file = sys.argv[2]
```

```
important_dates = ['1/20/14', '1/30/14']
```

```
with open(input_file, 'r', newline='') as csv_in_file:
```

```
    with open(output_file, 'w', newline='') as csv_out_file:
```

```
        filereader = csv.reader(csv_in_file)
```

```

filewriter = csv.writer(csv_out_file)

header = next(filereader)

filewriter.writerow(header)

for row_list in filereader:

    a_date = row_list[4]

    if a_date in important_dates:

        filewriter.writerow(row_list)

```

다음 실습은 행의 데이터 값이 어떤 집합의 원소로 포함되는지를 판별하고, 전체 행 중에서

그 값이 관심 집합의 값을 포함하는 부분 집합을 찾아내 출력 파일에 쓰는 방법을 보여준다.

이 실습에서는 Purchase Date 열의 구매 일자가 집합 {'1/20/14', '1/30/14'}를 포함하는 경우, 해당 행을 찾아서 출력 파일에 쓴다.

특정 집합의 값을 포함하는 행을 필터링하는 코드를 실습한다.

8행에서 관심 값에 해당하는 두 날짜가 포함된 important_dates라는 리스트 변수를 만들었다.

이 변수가 바로 관심 집합에 해당한다. 관심 값을 포함하는 변수를 만들고 코드에서 그 변수를 참조하도록 하는 것이 좋다.

이렇게 하면 관심 값이 달라지더라도 변수를 정의하는 위치에서 한 번만 수정하면 변경된 사항이 코드 전체에 적용되기 때문이다.

17행에서는 각 행에서 Purchase Date 열을 가져와서 그 값을 a_date라는 변수에 할당한다.

리스트 인덱싱 row_list[4]를 통해서 Purchase Date가 다섯 번째 열이란 것을 알 수 있다.

18번과 19행에서 if 문을 사용하여 a_date 변수에 들어 있는 Purchase Date 열의 데이터 값이 important dates에서 정의된 관심 집합에 포함되는지 여부를 판별한 뒤, 관심 집합에 포함된 값이라면 그 행을 출력 파일에 기록한다.

명령 줄에서 실행한다.

```
# python 4csv_reader_value_in_set.py supplier_data.csv 4output.csv
```

 **Pandas**

파일명 : pandas_value_in_set.py

```
#!/usr/bin/env python3
```

```
import pandas as pd
```

```
import sys
```

```
input_file = sys.argv[1]
```

```
output_file = sys.argv[2]
```

```
data_frame = pd.read_csv(input_file)
```

```
important_dates = ['1/20/14', '1/30/14']
```

```
data_frame_value_in_set = data_frame.loc[data_frame['Purchase Date']
```

```
.isin(important_dates), :]
```

```
data_frame_value_in_set.to_csv(output_file, index=False)
```

팬더스를 이용하여 관심 집합에 값이 포함되어 있는 행을 필터링 해보겠다.

이 스크립트는 csv 파일을 파싱하고 관심 집합에 포함된 값이 들어 있는 행을 출력 파일에 기록한다.

isin() 명령어는 "is in" 포함되어 있는가를 확인하는 명령어이다.

명령 줄에서 실행한다.

```
# python pandas_value_in_set.py supplier_data.csv 4pandas_output.csv
```

 CSV 모듈을 사용하지 않는 기본 파이썬 코드

파일명 : 5csv_reader_value_matches_pattern.py

```
#!/usr/bin/env python3
```

```
import csv
```

```
import re
```

```
import sys
```

```
input_file = sys.argv[1]
```

```
output_file = sys.argv[2]
```

```
pattern = re.compile(r'(?P<my_pattern_group>^001-.*)', re.I)
```

```
with open(input_file, 'r', newline='') as csv_in_file:
```

```
with open(output_file, 'w', newline='') as csv_out_file:
```

```
    filereader = csv.reader(csv_in_file)
```

```
    filewriter = csv.writer(csv_out_file)
```

```
    header = next(filereader)
```

```
    filewriter.writerow(header)
```

```
    for row_list in filereader:
```

```
        invoice_number = row_list[1]
```

```
        if pattern.search(invoice_number):
```

```
            filewriter.writerow(row_list)
```

이 스크립트는 어떤 행의 값이 특정 패턴과 일치하는지를 판별하고 패턴과 일치하는 값을 갖는 행의 하위 데이터 셋을 출력 파일에 작성 하는 방법을 보여준다.

이 실습에서는 Invoice Number 열의 데이터 값이 001- 로 시작하는 행을 선택하고 결과를 출력 파일에 쓴다.

이 패턴과 일치하는 값을 가진 행의 하위 데이터셋 필터링을 실습한다.

3행에서 re 모듈 (정규 표현식 모듈)의 함수에 접근할 수 있도록 re 모듈을 임포트했다.

9행에서는 re 모듈의 compile() 함수를 사용하여 pattern이라는 이름의 정규 표현식 변수를 생성한다.

여기서 r은 작은따옴표 사이의 패턴이 원시 문자열임을 나타낸다.

?P<my pattern_group> 라는 메타 문자는 <my_ pattern_group> 이라는 그룹에 포함되어 있는 하위 문자열을 찾아내 필요한 경우에 화면에 출력 하거나 파일에 기록할 수 있게 해준다.

우리가 찾는 실제 패턴은 A001-.* 이다. 캐럿(^)은 문자열이 시작하는 경우 패턴만 검색하는 특수 문자다. 즉 이 패턴은 001 로 시작하는 문자열만 찾는다.

마침표(.)는 개행을 제외한 임의의 문자 하나를 나타낸다.

따라서 개행을 제외한 모든 문자가 001- 뒤에 올 수 있다.

마지막으로 별표 (*)는 메타 문자 앞에 있는 문자가 0번 이상 일치해야 함을 나타낸다.

따라서 .* 조합은 개행을 제외한 모든 문자가 001 뒤에 몇 번이든 올 수 있다는 의미이다.

마지막으로 re.I 인수는 정규 표현식이 대/소문자를 구분하지 않고 찾도록 지시한다.

이 예에서는 패턴이 숫자이므로 이 인수가 덜 중요하지만, 패턴에 문자가 포함되어 있고 대/소문자를 구분하지 않고 찾는 경우 이 예제처럼 두 번째 인수로 지정 해야 한다.

18행에서는 리스트 인덱싱을 사용 하여 행에서 Invoice Number 열을 추출하고 이를 invoice_number라는 변수에 할당했다.

다음 행에서는 이 변수에 포함된 패턴을 찾을 것이다.

19 ~ 20행은 re 모듈의 search 함수를 사용하여 invoice number에 저장된 값의 패턴을 찾는다.

해당 패턴이 invoice number의 데이터 값에 들어 있으면 그 행을 출력 파일에 쓴다.

스크립트를 실행하려면 명령줄에 다음을 입력하고 엔터키를 누른다.

명령줄에 다음 명령을 실행한다.

python 5csv_reader_value_matches_pattern.py supplirdata.csv 5outputcs

 **Pandas**

파일명 : pandas_value_matches_pattern.py

#!/usr/bin/env python3

```
import pandas as pd

import sys

input_file = sys.argv[1]

output_file = sys.argv[2]

data_frame = pd.read_csv(input_file)

data_frame_value_matches_pattern = data_frame.loc[data_frame['Invoice Number']W

.str.startswith("001-"), :]

data_frame_value_matches_pattern.to_csv(output_file, index=False)
```

팬더스를 이용하여 특정 패턴과 일치하는 값을 가진 행을 필터링해본다.

이 스크립트에서는 startswith() 함수를 사용하여 정규표현식을 사용하지 않고 데이터를 찾아냈다.

ix 옵션은 현재 버전에서는 ioc, iloc 옵션으로 바꾸어 사용하도록 권장하고 있다.

명령줄에 다음 명령을 실행한다.

python pandas_value_matches_pattern.py supplirdata.csv 5pandas_output.csv

특정 열 선택하기

CSV 파일에서 특정 열을 선택하는 두 가지 방법이 있다.

- (1) 열의 인덱스 값을 사용하는 방법
- (2) 열의 헤더를 사용하는 방법

👉 열의 인덱스 값을 사용하여 특정 열을 선택하는 기본 파이썬 코드

파일명 : 6csv_reader_column_by_index.py

```
#!/usr/bin/env python3

import csv

import sys

input_file = sys.argv[1]
output_file = sys.argv[2]

my_columns = [0, 3]

with open(input_file, 'r', newline='') as csv_in_file:
    with open(output_file, 'w', newline='') as csv_out_file:
        filereader = csv.reader(csv_in_file)
        filewriter = csv.writer(csv_out_file)
```

```

for row_list in filereader:

    row_list_output = [ ]

    for index_value in my_columns:

        row_list_output.append(row_list[index_value])

    filewriter.writerow(row_list_output)

```

csv 파일에서 특정 열을 선택하는 첫 번째 방법은 열의 인덱스 값을 사용하는 것이다.

이 방법은 열의 인덱스 값을 쉽게 식별할 수 있거나 여러 개의 입력 파일을 처리할 때, 또는 모든 입력 파일에서 열의 위치가 변경 되지 않는 경우에 효과적이다.

예를 들어 첫 번째와 맨 마지막 열만 포함해야 하는 경우라면 row[0] 및 row[-1]을 사용하여 각 행의 첫 번째 및 마지막 데이터 값을 선택할 수 있다.

이 스크립트에서는 Supplier Name 및 Cost 열만 포함하려고 한다.

인덱스 값을 사용하여 이 두 열을 선택 해보겠다

8행에서 선택하려는 두 열의 인덱스 값을 포함하는 my_columns라는 리스트 변수를 생성했다.

이 실습 에서 이 두 인덱스 값은 Supplier Name 및 Cost 열에 해당한다.

다시 강조하지만 관심 인덱스 값을 포함하는 변수를 먼저 만든 다음 전체 코드에서 그 변수를 참조하도록 하는 것이 좋다. 이렇게 하면 관심 인덱스 값이 달라져도 한 곳 (여기서는 my_columns 변수 선언)만 변경 해도 my_columns를 참조하는 코드 전체에 변경된 사항이 적용되기 때문이다.

15 ~ 18행은 14행의 for 문 안쪽으로 들여쓰기가 적용되어 있으므로 입력 파일의 모든 행에서 반복적으로 실행된다. 15행 에서는 row_list_output 이라는 비어 있는 리스트 변수를 생성한다.

선택하려는 각 행의 값이 이 변수에 저 장될 것이다.

16행은 my_columns 에서 할당한 관심 인덱스 값을 반복하는 for 문이다.

17행은 리스트의 `append ()` 함수를 사용하여 `my_columns` 에서 각 행의 특정 인덱스의 데이터 값을 `row_list_output`에 추가한다.

이 세 줄의 코드 조합이 출력 파일에 쓰려는 각 행의 데이터 값을 포함하는 리스트를 만든다.

`filewriter`의 `write row()` 함수는 문자열이나 숫자의 시퀀스 자료형을 받으므로 리스트 변수 `row_list_output`를 만들어놓는 것이 유용하다.

18행은 `row_list_output`의 값을 출력 파일에 쓴다.

이 스크립트는 입력 파일의 모든 행에 대해 이 작업들을 반복해서 실행한다.

작업의 순서를 명확하게 살펴 보기 위해서 들여쓰기 바깥쪽의 `for` 문부터 살펴 보자.

입력 파일의 첫 번째 행 (헤더)에서부터 작업을 시작한다.

15행에서 비어 있는 리스트 변수인 `row list_output`을 생성한다.

16행은 `my_columns`의 값을 반복하는 `for` 문이다.

이 반복문은 처음으로 통과하는 `index_value`는 0 이므로,

17행에서 `append ()` 함수는 `row_list[0]` (문자열 `Supplier Name`) 을 `row_list_output`에 추가한다.

다음 코드는 16행의 `for` 문으로 돌아가고 이번에는 `index value`가 3이다.

`index_value`가 3이므로, 17행에서 `append()` 함수는 `row_list[3]` (문자열 `cost`)을 `row_list_output`에 추가한다.

`my columns`에 더 이상 값이 없으므로 16행의 `for`문이 완료되고 코드는 18행으로 이동한다.

18행은 row_list_output에 있는 데이터 값 리스트를 출력 파일에 쓴다. 그 다음, 코드는 입력 파일의 다음 행 처리를 시작하기 위해 14행에 있는 바깥쪽 for 문으로 되돌아간다.

명령 줄에서 실행한다.

```
# python 6csv_reader_column_by_index.py supplier_data .csv 6output .csv
```

 **Pandas**

파일명 : pandas_column_by_index.py

```
#!/usr/bin/env python3
```

```
import pandas as pd
```

```
import sys
```

```
input_file = sys.argv[1]
```

```
output_file = sys.argv[2]
```

```
data_frame = pd.read_csv(input_file)
```

```
data_frame_column_by_index = data_frame.iloc[:, [0, 3]]
```

```
data_frame_column_by_index.to_csv(output_file, index=False)
```

팬더스를 통해 인덱스 값을 사용하여 열을 선택해보겠다.

텍스트 편집기에 다음 코드를 입력하고 파일명을 pandas_column_by_index.py로 저장한다.

이 스크립트는 csv 파일을 읽고, 인덱스 값이 0과 3인 열을 화면에 출력하고 그 열을 출력 파일에 저장한

다. 여기서는 `iloc` 명령을 사용하여 인덱스 값 기반으로 열을 선택했다.

명령줄에 다음 명령을 실행한다.

```
# python pandas_column_by_index.py supplirdata.csv 6pandas_output.csv
```

👉 열의 헤더를 사용하여 특정 열을 선택하는 기본 파이썬 코드

파일명 : 7csv_reader_column_by_name.py

```
#!/usr/bin/env python3

import csv

import sys

input_file = sys.argv[1]
output_file = sys.argv[2]

my_columns = ['Invoice Number', 'Purchase Date']
my_columns_index = []

with open(input_file, 'r', newline='') as csv_in_file:

    with open(output_file, 'w', newline='') as csv_out_file:

        filereader = csv.reader(csv_in_file)

        filewriter = csv.writer(csv_out_file)

        header = next(filereader)

        for index_value in range(len(header)):
```

```

        if header[index_value] in my_columns:

            my_columns_index.append(index_value)

    filewriter.writerow(my_columns)

    for row_list in filereader:

        row_list_output = [ ]

        for index_value in my_columns_index:

            row_list_output.append(row_list[index_value])

    filewriter.writerow(row_list_output)

```

csv 파일에서 특정한 열을 선택하는 두 번째 방법은 인덱스 값 대신 열의 헤더 자체를 사용하는 것이다.

이 방법은 포함하려는 열 헤더를 식별하기 쉽거나, 여러 개의 입력 파일을 처리할 때 열의 헤더는 같지만 열의 위치가 입력 파일에 따라 다를 때 효과적이다.

예를 들어 Invoice Number 및 Purchase Date 열만 선택한다고 가정한 후 열의 이름을 사용하여 이 두 열을

선택해보겠다.

이 실습의 코드는 이 전의 코드보다 약간 길지만 익숙해져야 한다.

이 실습에서 더 많은 코드가 필요한 이유는 선택하고자 하는 열 헤더의 인덱스 값을 식별하려면 먼저 헤더 행을 별도로 처리 해야 하기 때문이다. 그 다음 이러한 인덱스 값을 사용하여 선택하려는 열의 헤더와 동일한 인덱스 값을 갖는 각 행의 데이터 값을 추출할 수 있다.

8행에서 my_columns 라는 이름의 리스트 변수를 생성한다.

이 변수에는 두 개의 문자열 값, 즉 포함하려는 두 개 열 헤더가 들어 있다.

9행에서는 my_columns_index라는 이름의 빈 리스트 변수를 만들었다.

뒤에서 이 리스트에 두 개의 관심 열의 인덱스 값을 채울 것이다.

15행은 filereader라는 객체의 next() 함수를 사용하여 입력 파일의 첫 번째 행을 header라는 리스트 변수로 읽어들인다.

16행은 열 헤더의 인덱스 값에 대해 for문을 시작한다.

17행에서는 if 문과 리스트 인덱싱을 사용하여 각 열의 헤더가 my_columns에 있는 값인지 판별한다.

예를 들어 for 문을 처음 실행할 때 index_value는 0 이므로 if 문은 header[0] (즉, 첫 번째 열의 헤더인 Supplier Name)이 my_columns에 포함되어 있는지 판별한다.

Supplier Name 이라는 문자열은 my_columns에 들어 있지 않으므로 **18행**은 실행되지 않는다.

다시 코드는 **16행의** for 문으로 돌아가 이번에는 index_value를 1 로 설정한다.

다음 **17행의** if 문은 header[1] (즉, 두 번째 열의 헤더인 Invoice Number)가 my_columns에 포함되어 있는지 판별한다.

Invoice Number는 my_columns 에 들어 있으므로 **18행이** 실행되고, 이 열의 인덱스 값이 my_columns index라는 리스트에 추가된다.

이런식으로 for 문이 계속 실행되고 마침내 Purchase Date 열의 인덱스 값이 my_columns_index에 추가된다.

for 문이 끝나면 **19행에서** my_columns의 두 문자열 (Invoice Number와 Purchase Date) 을 출력 파일에 쓴다.

21~24행의 코드는 입력 파일의 나머지 행에 대해서 작동한다,

21행은 row_list_output 이라는 빈 리스트를 작성하여 포함하려는 각 행의 데이터 값을 가져온다.

22행의 for 문은 my_columns_index의 인덱스 값을 반복하고,

23행은 이 인덱스 값에 해당하는 열의 데이터 값을 row_list_output 에 추가한다.

마지막으로 24행은 row_list_output의 값을 출력 파일에 쓴다.

명령줄에 실행한다.

```
# python 7csv_reader_column_by_name.py supplierdata.csv 7output.csv
```

 Pandas

파일명 : pandas_column_by_name.py

```
#!/usr/bin/env python3
```

```
import pandas as pd
```

```
import sys
```

```
input_file = sys.argv[1]
```

```
output_file = sys.argv[2]
```

```
data_frame = pd.read_csv(input_file)
```

```
data_frame_column_by_name = data_frame.loc[:, ['Invoice Number', 'Purchase Date']]
```

```
data_frame_column_by_name.to_csv(output_file, index=False)
```

팬더스를 사용해 열 헤더를 기반으로 열을 필터링 실습한다.

이 스크립트는 csv 파일을 읽고 Invoice Number 및 Purchase Date 열을 화면에 출력하고 그 열을 출력 파일에 저장한다.

명령줄에 다음 명령을 실행한다.

```
# python pandas_column_by_name.py supplirdata.csv 7pandas_output.csv
```

👉 여러 개의 CSV 파일 읽기 👈

지금까지는 하나의 csv 파일을 처리하는 방법들을 살펴 보았다.

실습들은 파이썬을 사용하여 프로그래밍을 통해서 파일을 처리하는 방법에 대한 아이디어를 제공해줬다.

파일이 하나뿐이라도 파일이 너무 커서 수동으로 처리 할 수 없는 경우, 프로그래밍으로 처리 하면 복사/붙여 넣기 오류 및 오타자 등 사람이 범하는 실수의 가능성이 줄어든다.

대부분의 경우에는 하나의 파일이 아닌 여러 개의 파일을 처리 해야 하므로 수동으로 이를 처리하는 것은 비효율적이거나 불가능하다. 이러한 상황에서 파이썬은 데이터 처리를 자동화 할 수 있으므로 데이터 분석의 범위를 확장할 수 있다.

이번 실습에서는 파이썬의 내장된 glob 모듈을 이용하여 앞서 소개한 실습들을 토대로 여러 개의 csv 파일을 처리하는 방법을 알아보겠다.

우선 여러 개의 csv 파일을 처리하려면 여러 개의 csv 파일을 만들어야 한다.

이번 실습에서는 **세 개의 파일만** 생성해서 사용할 것이다.

하지만 이 실습에서 소개되는 방법은 컴퓨터가 처리할 수 있는 수많은 파일로 확장할 수 있다.

=====

CSV 실습 파일명 : sales_january_2014, sales_february_2014, sales_march_2014

👉 **전체 파일 개수 및 각 파일의 행 및 열 개수 계산하는 기본 파이썬 코드**

파일명 : 8csv_reader_counts_for_multiple_files.py

행과 열의 개수를 세어 보는 간단한 작업부터 시작해보자.

이 작업은 매우 기본적 이지만 새로운 데이터 셋을 이해하는 좋은 방법이기도 하다.

경우에 따라서는 처리할 입력 파일의 내용을 미리 알 수도 있지만, 대부분의 경우 다른 사람에게 전달받은 일련의 파일을 처리 해야 하므로 그 파일의 내용에 대해 곧바로 알기는 어렵다.

이런 경우 처리 해야 하는 파일의 숫자를 확인하고, 각 파일의 행과 열의 개수를 계산해서 파악 해두면 큰 도움이 된다.

위에서 작성한 세 개의 csv 파일을 처리하는 실습을 한다.

(CSV 실습 파일명 : sales_january_2014, sales_february_2014, sales_march_2014)

3행과 4행에서 파이썬에 내장된 glob과 OS 모듈을 임포트했다.

이들 모듈의 함수를 사용하면 처리 할 파일의 경로명을 나열하고 파싱할 수 있다.

glob 모듈은 특정 패턴과 일치 하는 모든 경로명을 찾는다.

패턴에는 * 등 유닉스 쉘 스타일의 와일드카드 문자가 포함될 수 있다.

이 실습에서 찾으려는 패턴은 **sales_*** 로서 **sales_** 로 시작하는 이름을 가진 모든 파일을 찾는다.

밑줄 뒤에는 어떤 문자가 와도 상관이 없다.

앞서 세 개의 입력 파일을 만들 때 파일명을 모두 **sales_** 로 시작한 다음 밑줄 뒤에는 다른 SHELL 이름을 입력 했으므로, 코드에서 이 패턴을 사용하여 세 개의 입력 파일을 식별할 수 있다.

뒤에서는 **sales_** 로 시작하는 파일들이 아니라 폴더 속에 있는 모든 csv 파일을 처리할 것인데, 그렇게 하려면 이 스크립트 속의 패턴을 **sales_*** 에서 ***.csv**로 변경하기만 하면 된다.

csv는 모든 csv 파일의 이름 끝에 있는 패턴이므로 이 패턴을 통해서 모든 csv 파일을 효과적으로 찾을 수 있다.

OS 모듈은 경로명을 파싱 하는데 유용한 함수를 포함하고 있다.

os.path.basename (path)은 입력 받은 경로에서 파일명을 반환한다.

예를 들어

path가 "C:\Users\계정명\samole\my_input_file.csv" 라면 os.path.basename(path)는 my_input_file.csv를 반환한다.

10행은 여러 개의 입력 파일에서 데이터 처리를 위한 핵심이다.

glob 및 OS 모듈의 함수를 시용하여 처리할 입력 파일의 리스트를 만들어 for 문으로 일련의 입력 파일을 반복한다. 이 줄에는 많은 내용이 들어 있으므로 안쪽에서 바깥쪽으로 단계적으로 살펴본다.

OS 모듈의 OS.path.join() 함수는 괄호 안에 있는 두 개의 컴포넌트를 결합한다.

input_path는 입력 파일이 들어 있는 폴더의 경로이고 sales_는 sales 패턴으로 시작하는 파일의 이름을 나타낸다 .

glob 모듈의 glob.glob() 함수는 **sales ***의 별표 (*)를 실제 파일명으로 확장한다.

이 실습에서 glob.glob()과 os.path.join()은 세 개의 입력 파일이 들어 있는 리스트를 만든다.

```
['C\\Users\\계정명\\sample\\sales_january_2014.csv', 'C\\Users\\계정명\\sample\\sales_february_2014.csv',  
'C\\Users\\계정명\\sample\\sales_march_2014.csv']
```

그 다음 줄 시작 부분의 for 문은 이 리스트에 포함된 각 입력 파일에 대해서 그 줄 아래 들여 쓰기가 적용된 코드들을 실행한다.

17행은 각 입력 파일의 파일 이름, 파일의 행 과 열의 수를 출력하는 Print 문이다.

print 문에 탭 문자(\t)가 반드시 필요한 것은 아니지만 열 사이에 탭을 넣으면 세 개의 열을 정렬하는데 도움이 된다. 이 행은 { } 문자를 사용하여 세 가지 값을 print 문에 전달한다.

첫 번째 값은 os.path.basename() 함수를 사용하여 전체 경로명의 마지막 요소를 추출한다.

두 번째 값은 row_counter 변수를 사용하여 각 입력 파일에서 행의 개수를 계산한다.

세 번째 값은 내장된 len() 함수를 사용하여, 각 입력 파일의 열 헤더가 들어 있는 리스트 변수 header에 들어 있는 원소의 개수를 계산한다. 이 값을 각 입력 파일이 갖는 열의 개수로 사용한다.

17행에서 각 파일에 대한 정보를 출력한 뒤, 마지막으로 20행은 file_counter의 값을 사용하여 스크립트에서 처리한 파일의 숫자를 출력한다.

명령줄에서 실행한다.

```
# python 8csv_reader_counts_for_multiple_files.py "C:\Users\계정명\sample"
```

sales_february_2014.csv: 7 rows 5 columns

sales_january_2014.csv: 7 rows 5 columns

sales_march_2014.csv: 7 rows 5 columns

Number of files: 3

스크립트 파일 이름 뒤에 폴더 경로를 입력 해야 한다.

이전 실습에서는 이곳에 입력 파일의 이름을 넣었다.

이번 실습은 다수의 입력 파일을 처리 해야 하므로 모든 입력 파일이 들어 있는 폴더를 지정 해야 한다.

스크립트가 실행 되면 세 개의 입력 파일 이름과 각 파일의 행 및 열의 개수가 함께 화면에 출력된다.

세 개의 입력 파일에 대한 정보 밑에는 처리된 입력 파일의 총 개수가 표시 된다.

스크립트가 3 개의 파일을 처리 했고, 각 파일에는 7개의 행과 5 개의 열이 있음을 확인했다.

Pandas 없음

👉 여러 파일의 데이터를 합치는 기본 파이썬 코드

파일명 : 9csv_reader_concat_rows_from_multiple_files.py

유사한 데이터가 들어 있는 여러 개의 파일이 있는 경우 모든 데이터가 하나의 파일에 포함되도록 데이터를 합쳐야 하는 경우가 있다. 프로그래밍을 배우기 이전에는 각 파일을 수동으로 하나씩 열고 각 워크시트의 데이터를 복사하여 하나의 워크시트에 붙여 넣기 하는 식으로 작업했을 것이다. 이러한 수작업은 시간이 오래 걸리고 오류가 발생하기 쉽다. 또한 수작업이 불가능한 수준의 양 또는 크기의 파일들을 병합해야 할 수도 있다.

14행은 출력 파일을 여는 with 문이다.

지금까지의 실습에서 출력 파일을 쓸 때는 open() 함수의 인수로 w를 지정 했다.

이는 출력 파일을 쓰기 모드로 연다는 의미 였다.

이번 실습에서는 w 대신 a를 사용하여 추가모드에서 출력 파일을 열었다.

각 입력 파일의 데이터를 출력 파일에 계속해서 추가하려면 추가 모드를 사용해야 한다.

쓰기 모드를 사용하면 각 입력 파일의 데이터가 이전에 처리된 입력 파일의 데이터를 덮어쓰므로 출력 파일에는 마지막으로 처리된 입력 파일의 데이터만 남아 있게 된다.

17행에서 시작 하는 if-else 문은 **10행에서** 만든 first_file 변수를 사용하여 첫 번째 입력 파일과 후속 입력 파일을 구별한다. 헤더 행이 출력 파일에 한 번만 기록되도록 입력 파일을 구분해야 한다.

if 블록은 첫 번째 입력 파일을 처리 하고 헤더 행을 포함하여 모든 행을 출력 파일에 쓴다.

else 블록은 첫 번째를 제외한 나머지 모든 입력 파일을 처리하고 행을 출력 파일에 기록 하기 전에 next() 함수를 사용하여 각 파일의 헤더 행을 변수에 할당한다. (이렇게 해서 헤더 행을 처리 되지 않게 하는 것이다)

명령 줄에서 실행한다.

```
# python 9csv_reader_concat_rows_from_multiple_files.py "C:\Users\계정명\sample" 9output.csv
```

9output.csv

sales_february_2014.csv

sales_january_2014.csv

sales_march_2014.csv

화면에는 처리된 파일의 이름이 출력되고,

세 개의 입력 파일에 있는 데이터를 하나의 출력 파일인 9output.csv로 저장한다.

a를 W로 변경한 다음 스크립트를 저장하고 다시 실행하여 결과가 어떻게 변하는지 확인해보면 더 정확하게 이해할 수 있을 것이다.

마찬가지로 **if-else 문에서** if 부분을 제거 하면 출력 파일이 어떻게 변하는지 확인할 수 있다.

한 가지 명심할 점은 이 실습에서 사용한 패턴인 **sales_***가 상대적으로 구체적 이라는 점이다.

바탕화면에 세 개의 입력 파일 외에 **sales_**로 시작하는 다른 파일이 없다고 예상할 수 있는 상황이기 때문이다.

다른 상황에서는 ***.csv**와 같이 덜 구체적인 패턴을 사용하여 모든 csv 파일을 검색할 일이 더 많을 것이다.

이러한 상황에서는 입력 파일이 들어 있는 동일한 폴더에 출력 파일을 생성 하지 않는 것이 좋다.

입력 파일을 처리 하는 동안 스크립트에서 출력 파일을 열기때문에 패턴이 *.csv 이고 출력 파일도 csv 파일이면 스크립트는 출력 파일도 입력 파일 중 하나로 인식 하여 같은 방식으로 처리 하려고 시도할 것이고, 따라서 문제와 오류가 발생한다. 따라서 출력 파일을 다른 폴더에 저장하는 것이 좋다.

Pandas

파일명 : pandas_concat_rows_from_multiple_files.p

팬더스를 이용해 여러 파일의 데이터를 합치는 것은 더욱 쉽다.

기본 프로세스는 각 입력 파일을 데이터 프레임으로 읽어 들이고 all_data_frame 리스트에 추가한 다음 concat() 함수를 사용하여 모든 데이터 프레임을 하나의 데이터 프레임으로 합친다.

concat() 함수는 axis 인수를 통해서 데이터 프레임을 병합하는 방향을 수직(axis=0) 또는 수평(axis=1) 으로 지정할 수 있다.

이 코드는 데이터 프레임을 수직 방향으로 합친다.

수평 방향으로 연결하려면 concat() 함수에서 axis=1로 설정한다.

팬더스에는 데이터 프레임 외에도 시리즈라는 자료구조가 있다.

합치려는 객체가 시리즈인 경우에도 동일한 구문을 사용하면 된다.

때로는 단순히 데이터를 수직 또는 수평으로 합하는 대신에, SQL의 조인과 같이 키가 되는 열의 값을 기준으로 데이터 셋을 합해야 한다. 팬더스에서는 merge() 함수를 통해서 이런 작업이 가능하다.

SQL 조인에 익숙하다면 pd.merge(DataFrame1, DataFrame2, on='key', how= 'inner') 같이 merge() 함수의 구문을 쉽게 활용할 수 있을 것이다.

또 다른 파이썬 모듈인 넘파이도 데이터를 수직 및 수평으로 합치는 몇 가지 기능을 제공한다.

numpy를 np 라는 이름으로 임포트하는 것이 일반적이고,

데이터를 수직으로 합치려면 np.concatenate([array1, array2], axis= 0), np.vstack((array1, array2)) 또는 np.r_[array1, array2] 같은 함수를 사용할 수 있다.

마찬가지로 데이터를 수평으로 합치려면 np.concatenate([array1, array2], axis=1), np.hstack((array1, array2)) 또는 np.c_[array1, array2] 를 사용할 수 있다.

명령줄에서 실행한다.

```
# python pandas_concat_rows_from_multiple_files.py "C:\Users\계정명\sample" 9pandas_output.csv
```

 파일에서 데이터 값의 합계 및 평균을 계산하는 기본 파이썬 코드

파일명 : 10csv_reader_sum_average_from_multiple_files.py

때로는 여러 개의 입력 파일이 있을 때 각 입력 파일에 대해 몇 가지 통계 수치를 계산해야 한다.

이 스크립트에서는 앞에서 만든 세 개의 csv 파일을 사용하여, 각각의 입력 파일에서 열의 합계와 평균을 계산하는 실습을 한다.

11행에서 출력 파일의 열 헤더가 포함된 리스트를 만든다.

14행에서는 filewriter객체를 만들고 **15행에서** 출력 파일에 헤더 행을 쓴다.

20행에서 결과값을 저장할 비어 있는 리스트를 만든다.

각 입력 파일에 대한 합계와 평균을 계산해야 하므로, **16행에서** 입력 파일의 이름을 **output_list** 에 추가한다.

22행에서 next() 함수를 사용하여 각 입력 파일에서 헤더 행을 제거 한다.

23행에서 total_sales 라는 변수를 만들고 그 값을 0으로 설정한다.

24행에서 number_of_sales라는 변수를 만들고 그 값을 0으로 설정한다.

25행은 각 입력 파일의 데이터 행을 반복하는 for 문이다.

26행에서 리스트의 인덱스를 사용하여 Sale Amount 열의 데이터 값을 추출하고 이를 sales_amount 변수에 할당한다.

27행에서는 str() 함수를 사용하여 sales_amount 의 값이 문자열인지 확인한 다음 strip() 및 replace() 함수를 사용하여 데이터 값에서 달러 기호와 쉼표를 제거 한다. 그 다음 float() 함수를 사용하여 값을 부동소수점 숫자로 변환한 뒤, 이 값을 total_sales의 값에 더한다.

28행은 number_of_sales의 값을 1 씩 증가한다.

29행에서 total_sales의 값을 number_of_sales의 값으로 나누어 입력 파일의 평균 매출을 계산하고, 이 값을 소수점 이하 두 자리로 형식화하여 문자열로 변환한 뒤 average_sales 변수에 할당한다.

30행은 total_sales를 output_list의 두 번째 값으로 추가한다.

리스트의 첫 번째 값은 입력 파일의 이름으로서, 앞서 **21행에서** 리스트에 추가했다.

31행에서 average sales를 output_list의 세 번째 값으로 추가한다.

32행은 output_list의 값을 출력 파일에 쓴다.

스크립트는 각 입력 파일에 대해 이 코드를 실행하고 출력 파일에는 입력 파일명(file_name)열, 총 매출(total_sales) 열, 평균 매출(average_sales) 열이 포함된다.

명령줄에서 실행한다.

```
#python 10csv_reader_sum_average_from_multiple_files.py "C:\Users\계정명\sample" 10output.csv
```

 **Pandas**

파일명 : pandas_sum_average_from_multiple_files.py

팬더스는 행 및 열의 통계 수치를 계산하는데 사용할 수 있는 sum 및 mean 같은 요약 통계 함수를 제공한다.

다음 코드는 여러 입력 파일에서 특정 열에 대한 두 가지 통계(합계 및 평균)를 계산하고 각 입력 파일의 결과를 출력 파일에 쓰는 방법을 보여준다.

리스트 축약을 사용하여 Sale Amount 열의 문자열 값을 부동소수점 숫자로 바꾼 다음 DataFrame() 함수를 사용하여 개체를 데이터 프레임으로 변환하여 sum()과 mean() 함수를 사용하여 합과 평균을 계산했다.

출력 파일의 각 행에는 입력 파일 이름, 합계, 평균이 포함되어 있어야 하므로 이 세 가지 값을 데이터 프레임에 포함하고, concat() 함수를 사용하여 모든 데이터 프레임을 하나로 합친 다음 이 데이터 프레임을 출력 파일에 쓴다.

명령줄에서 실행한다.

```
# python pandas_sum_average_from_multiple_files.py "C:\Users\계정명\sample" 10pandas_output.csv
```

연속된 행 선택하기

때로는 분석해야 하는 파일에 맨 위 또는 맨 아래에 처리할 필요가 없는 내용이 포함되어 있다.

예를 들어 파일의 맨 위에 문서 제목과 작성자를 쓴 행이 있거나 파일의 맨 아래에 출처, 주의사항, 메모 등이 있는 경우가 있다. 대부분의 경우에 이런 내용은 처리할 필요가 없다.

csv 파일에서 연속된 행을 선택하는 방법을 살펴보기 위해 입력 파일에 필요 없는 행을 집어 넣겠다.

1) 스프레드시트에서 `supplier_data.csv`를 연다

2) 파일 맨 위에 세 개 행을 삽입한다.

즉 셀 A1 :A3에 I don't care about this line 식으로 필요 없는 텍스트를 추가한다.

3) 파일의 맨 아래, 즉 마지막 데이터 행 아래 세 행에도 마찬가지로 필요 없는 텍스트를 추가한다.

4) 입력 파일을 `supplier_data_unnecessary_header.footer.csv`라는 이름으로 저장한다.

입력 파일에 불필요한 머리말과 꼬리말 정보가 포함되었으므로 파이썬 스크립트를 수정 하여 이 정보를 가져오지 않도록 실습해본다.

연속된 행 선택하는 기본 파이썬 코드

파일명 : 11csv_reader_select_contiguous_rows.py

기본 파이썬으로 특정 행을 선택하기 위해 `row_counter`라는 변수를 추가하겠다.

이 변수로 행 번호를 추적하여 포함할 행을 식별하고 선택할 수 있다.

데이터 행 번호가 3 이상이고 15 이하인 행만 출력 파일에 쓴다.

if 문에 row_counter 변수를 사용하여 원하는 행만 선택하고 원하지 않는 헤더 및 푸터 내용은 건너 뛴다.

입력 파일의 처음 세 행은 row_counter가 3보다 작으므로 if 문 블록이 실행되지 않는다.

if 문 블록과 상관없이 매 행마다 row_counter의 값은 1씩 증가한다.

입력 파일의 마지막 세 행은 row_counter가 15보다 크므로 다시 if 문 블록이 실행되지 않는다.

선택하려는 행은 불필요한 헤더와 푸터 사이에 있다.

이 실습의 경우 선택할 row counter의 범위는 3행에서 15행까지이다.

if 문 블록은 이 행들만 처리해서 출력 파일에 쓴다. 리스트 축약과 strip() 함수를 사용하여 각 행의 값 양 끝에 서 공백, 탭, 개행문자를 제거한다.

위 코드에서 Writterow 문 위에 print(row_counter, [value.strip() for value in row]) 같은 식으로 print 문을 추가한다면 행 내용과 함께 변화되는 row_counter 변수의 값을 눈으로 확인할 수도 있다.

명령줄에서 실행한다.

```
# python 11csv_reader_select_contiguous_rows.py supplie_data_unnecessary_header_footer.csv  
11output.csv
```

 **Pandas**

파일명 : pandas_select_contiguous_rows.py

팬더스는 행의 인덱스 값 또는 열 헤더를 기반으로 행 또는 열을 삭제하기 위한 `drop()` 함수를 제공한다.

다음 스크립트에서 `drop()` 함수는 입력 파일에서 첫 세 행과 끝 세 행을 제거한다.

(즉, 인덱스 값이 0, 1, 2 인 행과 16, 17, 18인 행).`iloc()` 함수는 인덱스 값을 기반으로 단일 행을 선택하여 열 헤더 행 (`data_frame.columns`) 으로 사용할 수 있다.

마지막으로 하나 이상의 축을 새로운 인덱스에 맞추어 줄 수 있게 해주는 `reindex()` 함수를 사용한다.

팬더스를 가지고 열 헤더 행과 데이터 행을 선택하고 불필요한 헤더와 푸터 행을 제거 해보겠다.

명령줄에서 다음 명령을 실행한다.

```
#python pandas_select_contiguous_rows.py supplier_data_unnecessary_header_footer.csv 11pandas_output.csv
```

헤더 추가하기

때로는 스프레드시트에 헤더 행이 포함되지 않는 경우가 종종 있다.

이러한 경우에 파이썬 스크립트를 이용해 열 헤더를 추가할 수 있다.

실습을 위해 입력 파일을 수정한다.

- (1) 스프레드시트에서 원래 입력 파일인 `supplier_data.csv`를 연다.
- (2) 파일의 첫 번째 행(즉, 열 헤더가 포함되어 있는 헤더 행)을 삭제한다.
- (3) 수정한 파일을 `supplier_data_no_header_row.csv` 로 저장한다.

헤더를 추가하는 기본 파이썬 코드

파일명 : 12csv_reader_add_header_row.py

기본 파이썬에서 열 헤더를 추가해본다.

12행에서 각 열의 헤더에 해당하는 5개의 문자열 값이 들어 있는 header_list 라는 리스트 변수를 만들었다.

13행은 이 리스트의 값을 출력 파일의 첫 번째 행으로 쓴다.

마찬가지로 **15행은** 헤더 행 이후의 모든 데이터 값을 출력 파일에 쓴다.

명령줄에서 다음 명령을 실행한다.

```
# python 12csv_reader_add_header_row.py supplier_data_no_header_row.csv 12output.csv
```

Pandas

파일명 : pandas_add_header_rows.py

팬더스의 read_csv() 함수를 사용하면 헤더 행이 없는 입력 파일에 열 헤더를 할당하는 작업을 보다 손쉽게 할 수 있다.

입력 파일에 없는 헤더 행을 추가해본다.

명령줄에서 다음 명령을 실행한다.

python pandas_add_header_row.py supplier_data_no_header_row.csv 12pandas_output.csv

=====

※ 실습 파일 ※

1csv_simple_parsing_and_write

2csv_reader_parsing_and_write

3csv_reader_value_meets_condition

4csv_reader_value_in_set

5csv_reader_value_matches_pattern

6csv_reader_column_by_index

7csv_reader_column_by_name

8csv_reader_counts_for_multiple_files

9csv_reader_concat_rows_from_multiple_files

10csv_reader_sum_average_from_multiple_files

11csv_reader_select_contiguous_rows

12csv_reader_add_header_row

pandas_column_by_index.py

pandas_column_by_name.py

pandas_column_by_name_all_worksheets.py

pandas_concat_data_from_multiple_workbooks.py

pandas_parsing_and_write_keep_dates.py

pandas_sum_average_multiple_workbooks.py

pandas_value_in_set.py

pandas_value_matches_pattern.py

pandas_value_meets_condition.py

pandas_value_meets_condition_all_worksheets.py

pandas_value_meets_condition_set_of_worksheets.py

supplier_data.csv

supplier_data_no_header_row.csv

supplier_data_unnecessary_header_footer.csv

sales_february_2014.csv

sales_january_2014.csv

sales_march_2014.csv