
python

2-1. CSV 파일 읽고/쓰기(기본 파이썬 코드)

```
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')
```

basic01.py

2-2. 특정 행 필터링 (기본 csv 모듈)

- 특정 조건을 충족하는 행을 필터링
- 특정 집합의 값을 포함하는 행을 필터링
- 정규 표현식을 활용한 필터링

```
for   row   in   filerader:
      *** if 행에 있는 값이 특정한 규칙을 충족한다면 ***
            처리 1
      else:
            처리 2
```

2-2. 특정 행 필터링(기본 csv 모듈)

- 특정 조건을 충족하는 행을 필터링

  ex) 비용이 특정 값을 초과하는 모든 행을 선택하여 데이터셋으로 만들 경우

  ex) 구매 일자가 특정 날짜 이전인 모든 행을 데이터셋으로 만들 경우

- Supplier Name이 **Supplier Z** or **Cost가 $600.00 이상**인 행만 필터링 → 파일 출력

```
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)
```

```
Supplier Name,Invoice Number,Part Number,Cost,Purchase Date
Supplier X,001-1001,5467,$750.00 ,1/20/14
Supplier X,001-1001,5467,$750.00 ,1/20/14
Supplier Z,920-4803,3321,$615.00 ,2002-03-14
Supplier Z,920-4804,3321,$615.00 ,2002-10-14
Supplier Z,920-4805,3321,"$6,015.00",2/17/14
Supplier Z,920-4806,3321,"$1,006,015.00 ",2/24/14
```

basic02.py

2-2. 특정 행 필터링(기본 csv 모듈)

- 특정 집합의 값을 포함하는 행의 필터링
  - ***Supplier Name*** 열에서 집합 {**Supplier X, Supplier Y**} 중 한 값을 포함하는 모든 행
  - ***Purchase Date*** 열에서 구매 일자가 집합 *{'1/20/14', '1/30/14'}* 중 한 값을 포함하는 모든 행

```
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)
```

basic03.py

2-2. 특정 행 필터링(기본 csv 모듈)

- 패턴/정규 표현식을 활용한 필터링
  - ***Invoice Number*** 열의 데이터 값이 ***001-***로 시작하는 모든 행
  - ***Supplier Name*** 열의 데이터 값에 ***Y***가 포함되어 있는 모든 행

```
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)
```

basic04.py

2-3. 특정 열 선택(기본 csv 모듈)

- 열의 인덱스 값을 사용하는 방법
  - *row[0], row[-1]*
  - ***Supplier Name*** 및 ***Cost*** 열만 포함

```
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)
```

basic05.py

2-3. 특정 열 선택(기본 csv 모듈)

- 열의 헤더를 사용하는 방법
  - *Invoice Number* 및 *Purchase Date* 열만 포함

```
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)
```

basic06.py

2-4. 연속 된 행 선택(기본 csv 모듈)

- 분석에 필요 없는 맨 위 또는 맨 아래의 행 처리
  - csv 파일에 필요 없는 행 삽입 후 실행 (A1:A3, 제일 마지막 2행)

```
row_counter = 0
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 in filereader:
            if row_counter >= 3 and row_counter <= 10:
                filewriter.writerow([value.strip() for value in row])
            row_counter += 1
```

basic07.py

2-5. 여러 개의 CSV 파일 읽기 (기본 csv 모듈)

- 파이썬 내장 모듈인 ***glob*** 사용

```
file_counter = 0
for input_file in glob.glob(os.path.join(input_path,'sales_*')):
    row_counter = 1
    with open(input_file, 'r', newline='') as csv_in_file:
        filereader = csv.reader(csv_in_file)
        header = next(filereader)
        for row in filereader:
            row_counter += 1
    print('{0!s}: \t{1:d} rows \t{2:d} columns'.format(\
    os.path.basename(input_file), row_counter, len(header)))
    file_counter += 1
print('Number of files: {0:d}'.format(file_counter))
```

```
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
```

basic08.py

2-7. 여러 파일의 데이터 합치기 (기본 csv 모듈)

```
first_file = True
for input_file in glob.glob(os.path.join(input_path,'sales_*')):
    print(os.path.basename(input_file))
    with open(input_file, 'r', newline='') as csv_in_file:
        with open(output_file, 'a', newline='') as csv_out_file:
            filereader = csv.reader(csv_in_file)
            filewriter = csv.writer(csv_out_file)
            if first_file:
                for row in filereader:
                    filewriter.writerow(row)
                first_file = False
            else:
                header = next(filereader)
                for row in filereader:
                    filewriter.writerow(row)
```

sales_february_2014.csv
sales_january_2014.csv
sales_march_2014.csv

exercise

2-8. 파일에서 데이터 값의 합계 및 평균 계산(기본 csv 모듈)

```
output_header_list = ['file_name', 'total_sales', 'average_sales']

csv_out_file = open(output_file, 'a', newline='')
filewriter = csv.writer(csv_out_file)
filewriter.writerow(output_header_list)
```

```
        header = next(filereader)
        total_sales = 0.0
        number_of_sales = 0.0
        for row in filereader:
            sale_amount = row[3]
            total_sales += float(str(sale_amount).strip('$').replace(',',''))
            number_of_sales += 1.0
        average_sales = '{0:.2f}'.format(total_sales / number_of_sales)
```

basic10.py

Pandas

- 통계 분석을 위한 ***R***의 ***DataFrame*** 데이터 타입과 같은 ***Pandas DataFrame***을 사용
- Pandas DataFrame
  - ***테이블 형식의 데이터*** (tabular, rectangular grid 등으로 불림)를 다룰 때 사용
  - Column, Row(데이터), Index
  - Numpy의 ndarray, Pandas의 DataFrame, Series, Python의 dictionary, list 등으로 부터 생성 가능

```
df = pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6]]))
print(df.shape)
print(len(df.index))
print(list(df.columns))
```

```
(2, 3)
2
[0, 1, 2]
```

exercise

Pandas

- Dataframe에서 특정 컬럼이나 로우 선택
  - iloc, loc

```
 9  df = pd.DataFrame({"A": [1,4,7], "B": [2,5,8], "C":[3,6,9]})
10  print(df.loc[0])
11
12  print(df.loc[:, 'A'])
13  print(df.loc[1:, 'B'])
```

```
A   1
B   2
C   3
Name: 0, dtype: int64
```

```
0   1
1   4
2   7
Name: A, dtype: int64
```

```
1   5
2   8
Name: B, dtype: int64
```

exercise

2-1. CSV 파일 읽고/쓰기(Pandas)

```
data_frame = pd.read_csv(input_file)
print(data_frame)
data_frame.to_csv(output_file, index=False)
```

	Supplier Name	Invoice Number	Part Number	Cost	Purchase Date
0	Supplier X	001-1001	2341	$500.00	1/20/14
1	Supplier X	001-1001	2341	$500.00	1/20/14
2	Supplier X	001-1001	5467	$750.00	1/20/14
3	Supplier X	001-1001	5467	$750.00	1/20/14
4	Supplier Y	50-9501	7009	$250.00	1/30/14
5	Supplier Y	50-9501	7009	$250.00	1/30/14
6	Supplier Y	50-9505	6650	$125.00	2/3/14
7	Supplier Y	50-9505	6650	$125.00	2/3/14
8	Supplier Z	920-4803	3321	$615.00	2/3/14
9	Supplier Z	920-4804	3321	$615.00	2/10/14
10	Supplier Z	920-4805	3321	$615.00	2/17/14
11	Supplier Z	920-4806	3321	$615.00	2/24/14

exercise

2-2. 특정 행 필터링(Pandas)

- 특정 조건을 충족하는 행을 필터링
- loc()

  - 특정 행과 열을 동시에 선택

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

```
Supplier Name,Invoice Number,Part Number,Cost,Purchase Date
Supplier X,001-1001,5467,$750.00 ,1/20/14
Supplier X,001-1001,5467,$750.00 ,1/20/14
Supplier Z,920-4803,3321,$615.00 ,2002-03-14
Supplier Z,920-4804,3321,$615.00 ,2002-10-14
Supplier Z,920-4805,3321,$615.00 ,2/17/14
Supplier Z,920-4806,3321,$615.00 ,2/24/14
```

pandas02.py

2-2. 특정 행 필터링(Pandas)

- 특정 집합의 값을 포함하는 행의 필터링
  - ***Supplier Name*** 열에서 집합 {**Supplier X, Supplier Y**} 중 한 값을 포함하는 모든 행
  - ***Purchase Date*** 열에서 구매 일자가 집합 *{'1/20/14', '1/30/14'}* 중 한 값을 포함하는 모든 행

```
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)
```

pandas03.py

2-2. 특정 행 필터링(Pandas)

- 특정 집합의 값을 포함하는 행의 필터링
  - ***Supplier Name*** 열에서 집합 {**Supplier X, Supplier Y**} 중 한 값을 포함하는 모든 행
  - ***Purchase Date*** 열에서 구매 일자가 집합 *{'1/20/14', '1/30/14'}* 중 한 값을 포함하는 모든 행

```
data_frame = pd.read_csv(input_file)
data_frame_value_matches_pattern =
       data_frame.ix[data_frame['Invoice Number'].str.startswith("001-"), :]

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

pandas04.py

2-3. 특정 열 선택(Pandas)

- 열의 인덱스 값을 사용하는 방법
  - ***Supplier Name*** 및 ***Cost*** 열만 포함
  - iloc() ➔ 정수기반 위치 (Integer Location)

```
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)
```

pandas05.py

2-3. 특정 열 선택(Pandas)

- 열의 헤더를 사용하는 방법
  - *Invoice Number* 및 *Purchase Date* 열만 포함

```
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)
```

pandas06.py

2-4. 연속 된 행 선택(Pandas)

- 분석에 필요 없는 맨 위 또는 맨 아래의 행 처리
  - csv 파일에 필요 없는 행 삽입 후 실행 (A1:A3, 제일 마지막 2행)
- ***drop()***
  - 행 또는 열 삭제 함수
- ***iloc()***
  - 열 헤더 행 선택 ➔ data_frame.columns
- ***reindex()***
  - 새로운 인덱스에 맞추는 함수

```
data_frame = pd.read_csv(input_file, header=None)

data_frame = data_frame.drop([0,1,2,11,12])
data_frame.columns = data_frame.iloc[0]
data_frame = data_frame.reindex(data_frame.index.drop(3))

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

pandas07.py

2-5. 여러 파일의 데이터 합치기 (Pandas)

- 각 입력 파일을 데이터프레임으로 읽어 들이고 all_data_frame에 추가 ➔ concat() 함수 사용
- concat()
  - axis 인수를 통해 데이터프레임 병합
    - axis=0(수직), axis=1(수평)

```
all_files = glob.glob(os.path.join(input_path,'sales_*'))

all_data_frames = []
for file in all_files:
    data_frame = pd.read_csv(file, index_col=None)
    all_data_frames.append(data_frame)
data_frame_concat = pd.concat(all_data_frames, axis=0,
ignore_index=True)

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

pandas09.py

2-6. 파일에서 데이터 값의 합계 및 평균 계산(Pandas)

- sum() 및 mean() 같은 통계 함수 제공

```
all_files = glob.glob(os.path.join(input_path,'sales_*'))
all_data_frames = []
for input_file in all_files:
    data_frame = pd.read_csv(input_file, index_col=None)

    total_sales = pd.DataFrame([float(str(value).strip('$').replace(',','')) \
                                for value in data_frame.loc[:, 'Sale Amount']]).sum()

    average_sales = pd.DataFrame([float(str(value).strip('$').replace(',','')) \
                                for value in data_frame.loc[:, 'Sale Amount']]).mean()

    data = {'file_name': os.path.basename(input_file),
            'total_sales': total_sales,
            'average_sales': average_sales}

    all_data_frames.append(pd.DataFrame(data, columns=['file_name', 'total_sales', 'average_sales']))

data_frames_concat = pd.concat(all_data_frames, axis=0, ignore_index=True)
```

pandas10.py