☞ Excel

마이크로소프트 엑셀은 매우 다양한 비즈니스 상화에서 활용되는 도구이다.

고객, 재고, 직원 등에 대한 데이터를 저장하고 운영, 판매, 재무 상태를 추적하고 관리하는 데 엑셀이 사용된다. 엑셀은 업무에서 필수적으로 사용되는 도구이므로, 파이썬으로 엑셀 파일을 처리하고 데이터를 분석하는 방법을 익힌다면 다른 사람들로부터 데이터를 제공받아 보다 편하게 그 결과를 공유할수 있을 것이다.

파이썬에는 csv 파일을 처리를 위한 csv 모듈은 있지만 엑셀 파일 (즉 .xls 또는 . xlsx 확장자를 가진 파일) 처리를 위한 표준 모듈은 없다. 이번 실습을 위해서 xlrd 및 xlwt 패키지를 먼저 설치해야 한다.

xlrd 및 xlwt 패키지는 모든 운영 체제에서 파이썬으로 엑셀 파일을 처리할 수 있게 해주고, 날짜 형식을 지원한다.

cmd> pip install xlrd --upgrade

cmd> pip install xlwt

cmd> pip install pandas

cmd> pip install xlsxwriter

cmd> pip install matplotlib

엑셀 파일은 적어도 두 가지 중요한 점에서 csv 파일과 다르다.

첫째, 엑셀 파일은 csv 파일과 달리 일반 텍스트 파일이 아니므로 텍스트 편집기에서 파일을 열어 데이터를 볼 수 없다.

둘째, 엑셀 통합 문서는 csv 파일과 달리 여러 개의 워크시트를 포함하도록 설계 되었다.

단일 엑셀 통합 문서에 여러 개의 워크시트가 포함될 수 있으므로 수동으로 통합 문서를 열지 않고도 통합 문서의 모든 워크시트를 살펴보고 검토하는 방법을 알아야 한다.

※ 실습 파일 ※

sales_2013.xlsx

sales_2014.xlsx

sales_2015.xlsx

sales_march_2014.xlsx

supplier_data.xlsx

supplier_data_no_header_row.xlsx

supplier_data_unnecessary_header_footer.xlsx

==========

☞ 엑셀 통합 문서 내부 살펴보기

파일명 : 1excel_introspect_workbook.py

통합 문서의 워크시트 개수, 워크시트 이름 및 각 워크시트의 행과 열 수를 확인한다.

```
#!/usr/bin/env python3
import sys
from xlrd import open_workbook

input_file = sys.argv[1]

workbook = open_workbook(input_file)
print('Number of worksheets:', workbook.nsheets)
for worksheet in workbook.sheets():
        print("Worksheet name:", worksheet.name, "₩tRows:", ₩
                worksheet.nrows, "₩tColumns:", worksheet.ncols)
```

☞ **스크립트 설명**

3행은 xlrd 모듈에서 open_workbook() 함수를 임포트해 엑셀 파일을 읽고 파싱하는데 사용할 수 있게 한다.

7행은 open_workbook() 함수를 사용하여 엑셀 입력 파일을 workbook 이라는 이름의 객체로 연다.

workbook 객체에는 엑셀 통합 문서에 대한 사용 가능한 모든 정보가 포함되어 있으므로 개별 워크시트를 검색하는데 사용할 수 있다.

8행은 workbook에 있는 워크시트 수를 출력한다.

9행은 for 문으로 workbook의 모든 워크시트를 반복한다.

workbook 개체의 sheets( ) 함수는 엑셀 통합 문서 내의 개별 워크시트를 식별하는데 사용한다.

10행은 각 워크시트의 이름과 각 워크시트의 행 및 열 수를 화면에 출력한다.

print( ) 문은 워크시트 객체의 name 속성을 사용하여 각 워크시트의 이름을 식별한다.

nrows 및 ncols 속성을 사용하여 각 워크시트의 행 및 열 수를 식별한다.

명령 줄에서 실행한다.

 # python 1excel_introspect_workbook.py sales_2013.xlsx

```
C:₩UsersWTJ₩sample>python 1excel_introspect_workbook.py sales_2013.xlsx
Number of worksheets: 3
Worksheet name: january_2013    Rows: 7         Columns: 5
Worksheet name: february_2013   Rows: 7         Columns: 5
Worksheet name: march_2013      Rows: 7         Columns: 5
C:₩UsersWTJ₩sample>
```

첫 번째 출력행은 엑셀 입력 파일 sales_2013. xlsx 에 총 세 개의 워크시트가 있음을 보여준다.

다음 세 줄은 3개 워크시트의 이름이 january_2013. february_2013.march_2013임을 보여준다.

또한 각 작업 시트에는 헤더 행을 포함해 7개 행과 5개 열이 있음을 알 수 있다.

파일명 : 2excel_parsing_and_write.py

☞ 단일 워크시트 처리

하나의 워크시트를 파싱하는 방법을 실습한다.

기본 파이썬과 xlrd, xlwt 모듈을 사용하여 엑셀 파일을 읽고 작성한다.

```
#!/usr/bin/env python3
import sys
from xlrd import open_workbook
from xlwt import Workbook

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

output_workbook = Workbook()
output_worksheet = output_workbook.add_sheet('jan_2013_output')

with open_workbook(input_file) as workbook:
        worksheet = workbook.sheet_by_name('january_2013')
        for row_index in range(worksheet.nrows):
                for column_index in range(worksheet.ncols):
                        output_worksheet.write(row_index, column_index, worksheet.cell_value(row_index,
column_index))
output_workbook.save(output_file)
```

3행에서 xlrd의 open_workbook( ) 함수를 임포트하고 4행에서 xlwt의 Workbook 객체를 임포트한다.

9행에서 xlwt의 Workbook 객체를 인스턴스화하여 결과를 출력하여 엑셀 통합 문서에 쓸 수 있다.

10행은 xlwt의 add_sheet( ) 함수를 사용하여,

출력하는 엑셀 통합 문서 안에 jan_2013_output 이라는 워크시트를 추가한다.

12행은 xlrd의 open_workbook( ) 함수를 사용하여 입력되는 엑셀 통합 문서를 Workbook 객체로 연다.

13행에서는 workbook 객체에 sheet_by_name() 함수를 사용하여 january_2013 이라는 이름의 워크시트에 연결한다.

14 ~ 15행은 range( ) 함수와 worksheet 객체의 nrows 및 ncols 속성을 사용하여 행 및 열 인덱스 값에 대한 반복문으로서 워크시트의 각 행과 열에서 작업을 반복할 수 있게 한다.

16행에서 xlwt의 write ( ) 함수와 행 및 열 인덱스를 사용하여 모든 셀 값을 출력 파일의 워크시트에 기록한다.

17행은 출력 파일을 저장하고 닫는다.

명령 줄에서 실행한다.

 # python 2excel_parsing_and_write.py sales_2013.xlsx 2output.xls

출력 파일에서 Purchase Date 열인 E열의 날짜가 날짜 대신 숫자로 표시 된다는 것을 확인할 수 있다.

엑셀은 날짜와 시간을 1900-Jan-0부터 일 수를 나타내는 부동소수점 숫자와 하루의 24시간을 분수로 할당하여 저장한다.

예를 들어 숫자 1 은 1900-Jan-0부터 하루가 지난 1900-Jan-1을 나타낸다.

따라서 이 열의 숫자는 날찌를 나타내지만 날짜 형식은 아닌 것이다.

xlrd 패키지는 날짜와 유사한 값의 포멧팅하는 추가 기능을 제공한다.

날짜 형식 할당

이번 실습에서는 xlrd를 사용하여 입력되는 엑셀 파일과 마찬가지로 날짜 형식을 유지하는 방법에 대해 실습한다.

예를 들어 엑셀 워크시트 속 데이터 값이 날짜인 1/1 9/2000 인 경우, 일반적으로 1/19/2000 또는 다른 날짜 형식으로 이를 출력 파일에 쓰고자 할 경우, 현재 코드에서는 출력 파일에 36544. 0 이라는 숫자가 표시된다.

이 값은 1/0/1900과 1/19/2000 사이의 일 수를 의미한다.

파일명 : 3excel_parsing_and_write_keep_dates.py

☞ 특정 조건을 충족하는 행의 필터링 기본 파이썬 코드

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

import sys

from datetime import date

from xlrd import open_workbook, xldate_as_tuple

from xlwt import Workbook
```

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

output_workbook = Workbook()
output_worksheet = output_workbook.add_sheet('jan_2013_output')

with open_workbook(input_file) as workbook:
        worksheet = workbook.sheet_by_name('january_2013')
        for row_index in range(worksheet.nrows):
                row_list_output = []
                for col_index in range(worksheet.ncols):
                        if worksheet.cell_type(row_index, col_index) == 3:
                                date_cell = xldate_as_tuple(worksheet.cell_value₩
                                        (row_index, col_index),workbook.datemode)
                                date_cell = date(*date_cell[0:3]).strftime₩
                                        ('%m/%d/%Y')
                                row_list_output.append(date_cell)
                                output_worksheet.write(row_index, col_index, date_cell)
                        else:
                                non_date_cell = worksheet.cell_value₩
                                        (row_index,col_index)
                                row_list_output.append(non_date_cell)
                                output_worksheet.write(row_index, col_index,₩
                                        non_date_cell)
output_workbook.save(output_file)
```

3행은 datetime 모듈에서 date( ) 함수를 임포트해서 뒤에서 값을 날짜로 변환하고 날짜 형식으로 포멧팅할 수 있게 한다.

4 행은 xlrd 모듈에서 두 개의 함수를 임포트한다.

xldate_as_tuple( ) 함수를 사용하면 날짜 시간, 날짜 + 시간을 나타내는 엑셀의 숫자를 튜플로 변환할 수 있다.

숫자를 튜플로 변환하면 특정 날짜 요소(예를 들면 년 , 월, 일)를 추출하여 다른 닐짜 형식 (예를 들어 1/1/2 0 10 또는 January 1, 20 10)으로 포멧팅할 수 있다.

18행은 if-else 문으로 셀 유형이 3번인지 판별한다.

xlrd 모듈의 문서 (http://xlrd.readthedocs.io/en/latest/api.html?highlight=cell_type#xlrd.sheet.Cell) 를 보면

셀 유형 3은 날짜 셀임을 의미한다. 따라서 if-else 문은 각 셀에 날짜가 들어 있는지 판별하고, 들어 있다면 if 블록의 코드를 실행하고 아니 면 else 블록의 코드가 실행한다. 이 예제의 경우 날짜가 마지 막 열에 있으므로 if 블록은 마지 막 열에 대해서만 실행된다.

9~20행은 worksheet 객체의 cell_value() 함수와 행/열 인덱싱을 사용하여 셀의 값에 접근한다. 또는 cell( ).value 함수를 사용할 수도 있다. 두 버전 모두 동일한 결과를 제공한다. 이 셀 값은 **xldate_as_tuple** 함수의 첫 번째 인수로 들어가 부동소수점 숫자를 날짜로 나타내는 튜플로 변환한다.

이 함수에 두 번째 인수인 workbook.datemode가 필요한 이유는 날짜가 1900년 기반인지 1904년 기반인지 확인하여 올바르게 튜플로 변환하기 위해서다. (맥용 엑셀 일부 버전은 1900년이 아니라 1904년 1월 1일부터 날짜를 계산한다. 이에 대한 정보는 마이크로소프트의 문서 (https://support.microsoft.com/ko-us/kb/21433@) 를 참고) .

이 함수의 결과는 date_cell 이라는 튜플 변수에 할당된다.

21 ~22 행에서는 튜플 인덱싱을 사용하여 date_cell 튜플의 첫 세 요소 (연도, 월, 일)에 접근하고 date( ) 함수의 인수로 전달한다. 이 함수는 값을 날짜 객체로 변환한다.

다음으로 strftime( ) 함수는 날짜 객체를 할당된 날짜 형식의 문자열로 변환한다.

%m/%d /%Y 형식은 March 15, 2014 같은 날짜가 03/15/2014 처럼 표시 되어야 함을 지정한다.

포멧팅된 날짜 문자열은 date_cell 이라는 변수에 다시 할당된다.

23행에서는 리스트의 append( ) 함수를 사용하여 date_cell의 값을 row_list_output 이라는 출력 리스트에 추가한다.

26~27행은 Worksheet 객체의 cell_value( ) 함수와 행 및 열 인덱싱을 사용하여 셀의 값에 접근하여 non_date_cell 이라는 변수에 할당한다.

28행은 리스트의 append() 함수를 사용하여 non_date_cell의 값을 row_list_output에 추가한다.

이 두 줄은 입력 파일의 각 행에서 처음 네 열의 값을 그대로 추출하여 row_list_output에 추가할 것이다.

각 행의 모든 열을 처리하고 모든 행을 처리한 다음에

31행에서 출력 워크시트를 지정된 출력 파일에 기록한다.

명령 줄에서 실행한다.

 # python 3excel_parsing_and_write_keep_dates.py sales_2013.xlsx 3output.xls

☞ **Pandas**

파일명 : 3pandas_parsing_and_write_keep_dates.py

팬더스는 엑셀 파일을 읽고 쓰는 데 특화되어 있는 명령어들이 있다.

이 스크립트는 입력된 엑셀 파일을 읽고 화면에 내용을 출력한 다음,

그 내용을 출력 엑셀 파일에 기록한다.

명령 줄에서 실행한다.

 # python 3pandas_parsing_and_write_keep_dates.py sales_2013.xlsx 3pandas_output.xls

---

특정 행 필터링 하기

파이썬을 이용하여 필요하지 않은 행은 걸러내고 필요한 행을 유지할 수 있다.

데이터가 너무 커서 열 수도 없는 엑셀 파일과 수동으로 처리하는데 너무 많은 시간이 소요되는 다수의 엑셀 워크 시트들을 파이썬으로 처리할 수 있다.

파일명 : 4excel_value_meets_condition.py

☞ 특정 조건을 충족하는 행의 필터링하는 기본 파이썬 코드

Sale Amount 열의 데이터 값이 $1,400.00 보다 큰 행을 하위 데이터셋으로 선택하고자 한다.

이 조건을 만족하는 행을 필터링한다.

16행에서 data라는 빈 리스트를 만들었다. 출력 파일에 쓰고자 하는 입력 파일의 모든 행을 이 리스트에 채울 것이다.

17행은 헤더 행의 값을 추출한다. 헤더 행을 필터링 조건으로 판별하는 것은 의미가 없으므로

18행에서는 헤더 행을 있는 그대로 data에 추가한다.

21 행에서 Sale Amount 열의 데이터 값을 담을 sale_amount 라는 변수를 만들었다.

cell_value( ) 함수는 13 행 에서 정의된 sales_amount_column_index의 인덱스를 사용하여 Sale Amount 열을 찾는다.

판매 금액이 $1,400,00보다 큰 행을 필터링하고 싶은 싱황이므로 sale_amount 변수를 사용하여 조건을 판별한다.

23행은 Sale Amou nt 열의 데이터 값이 1400. 0보다 큰 행만 처리하는 for 문이다.

조건이 일치하는 행에서 각 셀의 값을 cell_value 변수에 추출하고 셀의 유형 을 cell_type 변수에 추출한다.

다음으로 각 행의 값이 날짜 인지 판별한 뒤 그렇다면 값을 날짜 형식으로 포멧팅한다.

제대로 포멧팅된 값을 포함하는 행을 생성하기 위해서 20행에서 row_list라는 빈 리스트를 만든 다음

29행과 31행에서 행의 날짜 및 날짜가 아닌 값을 row_list에 추가한다.

입력 파일의 모든 데이터 행에 대해 빈 row_list를 만들지만 이 중에서 판매 금액 열의 값이 1400. 0보다 큰 행의 경우만 값으로 채운다.

따라서 32 행은 입력 파일의 각 행에 대해서 row_list가 비어 있는지 여부를 확인하고 row_list가 비어 있지 않은 경우에만 row_list를 data에추가한다.

마지막으로 35 ~ 36행은 data와 각 리스트의 값을 출력 파일에 기록하기 위한 반복문이다.

보유하려는 행을 새로운 리스트인 data에 추가한 이유는 연속하는 행의 인덱스 값을 새로 얻기 위해서다.

이렇게 해야 출력 파일에 행을 쓸 때 행 사이에 간격이 없는 연속된 행으로 나타난다.

이렇게 하지 않으면 xlwt의 write() 함수는 입력 파일의 기존 행 인덱스 값을 사용하므로 행 사이에 간격을 두고 출력 파일에 행을 쓸 것이다.

명령 줄에서 실행한다.

 # python 4excel_value_meets_condition.py sales_2013.xlsx 4output.xls

☞ **Pandas**

파일명 : 4pandas_value_meets_condition.py

팬더스에서는 데이터 프레임의 이름 뒤에 대괄호 ([ ])를 쓰고 선택하려는 열의 이름과 특정 조건을 지정하면 조건에 맞는 행을 쉽게 필터링할 수 있다.

예를 들어,

다음 스크립트에 표시된 조건은 Sale Amount 열의 값이 $1,400.00보다 큰 모든 행을 원한다는 뜻이다.

여러 조건을 동시에 적용해야 하는 경우,

조건을 괄호 안에 넣고 적용할 조건부 논리에 따라 엠퍼샌드(&) 또는 파이프 (|) 와 함께 쓴다.

명령 줄에서 실행한다.

 # python 4pandas_value_meets_condition.py sales_2013.xlsx 4pandas_output.xls

---

파일명 : 5excel_value_in_set.py

☞ 특정 집합의 값을 포함하는 행의 필터링 기본 파이썬 코드

이 스크립트는 조건에 맞는 행을 필터링 하는 스크립트와 매우 유사하다.

차이 점은 13 , 23 , 25행에 있다.

13 행은 관심 있는 날짜가 들어 있는 important_dates 라는 리스트를 작성한다.

23 행에서는 purchase_date라는 변수에 Purchase Date 열의 값을 저장하되 important dates의 날찌들과 같은 형식으로 포멧팅 된 값을 담는다.

25행은 행의 날짜가 important_dates의 날짜 중 하나와 같은지 여부를 판별한다.

일치하면 해당 행을 처리하여 출력 파일에 쓴다.

명령줄에 다음 명령을 실행한다.

 # python 5excel_value_in_set.py sales_2013.xlsx 5output.xls

☞ Pandas

파일명 : 5pandas_value_in_set.py

이 스크립트에서는 Purchase Date 열의 값이 01/24/2013 혹은 01/31/2013인 행을 필터링 하려고 한다.

팬더스는 특정 값이 리스트에 있는지 확인하는 데 사용할 수 있는 isin( ) 함수를 제공한다.

명령줄에 다음 명령을 실행한다.

 # python 5pandas_value_in_set.py sales_2013.xlsx 5pandas_output.xls

---

파일명 : 6excel_value_matches_pattern.py

☞ 패턴을 활용한 필터링 기본 파이썬 코드

이번 스크립트에서는 Customer Name 열이 대문자 J로 시작하는 행을 필터링한다.

2 행 에서 re 모듈을 임포트한다.

14행은 re 모듈의 compile( ) 함수를 사용하여 pattern 이라는 정규식을 만든다.

r은 작은따옴표 사이의 패턴이 원시 문자열임을 의미 한다.

?P<my_pattern> 라는 메타 문자는 <my_pattern> 이라는 그룹에서 일치 하는 하위 문자열을 캡처하여 필요할 경우 화면에 출력 하거나 파일에 쓸 수 있게 한다. 실제 패턴은 ^J.*이다.

캐럿 (^) 은 '문자열의 시작 부분'을 의미 하는 특수문자이다.

즉, 문자열은 대문자 J로 시작해야 한다는 뜻이다.

마침표(.)는 개행문자를 제외한 모든 문자와 일치하므로 개행문자를 제외한 모든 문자가 J 뒤에 올 수 있다.

마지막으로 별표 (*) 는 앞의 문자를 0번 이상 반복하는 것을 의미 한다.

따라서 .* 조합은 개행문자를 제외한 모든 문자가 J 다음에 여러 번 나타날 수 있음을 의미 한다.

24행 에서는 re 모듈의 search( ) 함수를 사용하여 Customer Name 열의 패턴을 찾고 일치하는 항목이 있는지 판별한다. 일치하는 항목을 찾으면 행의 값을 row_list에 추가한다.

31행은 날짜 값을 row_list 에 추가하고,

33행은 날짜가 아닌 값을 row_list 에 추가한다.

34행은 리스트가 비어 있지 않은지 판별하고, 비어 있지 않으면 row_list 의 각 값 리스트를 data에 추가한다.

37~38행의 두 for 문은 data 속의 리스트를 반복하여 출력 파일에 행을 기록한다.

명령 줄에서 실행한다.

 # python 6excel_value_matches_pattern.py sales_2013.xlsx 6output.xls

☞ **Pandas**

파일명 : pandas_value_matches_pattern.py

팬더스에서도 마찬가지로 Customer Name 열이 대문자 J로 시작하는 행을 필터링하려고 한다.

팬더스는 startswith , endswith , match , search 등 텍스트의 하위 문자열과 패턴을 식별할 수 있는 다양한 문자열 및 정규 표현식 함수를 제공한다.

명령줄에 다음 명령을 실행한다.

 # python pandas_value_matches_pattern.py sales_2013.xlsx 6pandas_output.xls

---

파일명 : 7excel_column_by_index.py

☞ 열의 인덱스 값을 사용하여 특정 열을 선택하는 기본 파이썬 코드

워크시트에서 특정 열을 선택하는 방법은 필요한 열의 인덱스 값을 사용하는 것이다.

이 방법은 관심 있는 열의 인덱스 값을 쉽게 식별할 수 있거나 여러 개의 입력 파일을 처리할 때 ,

열의 위치가 모든 입력 파일에서 일관성이 있을 때 효과적이다.

예를 들어, Customer Name과 Purchase Date 열만 선택한다고 가정 해보자.

13행에서 정수 1과 4를 포함하는 my_columns 라는 리스트 변수를 만든다.

이 두 숫자는 Customer Name 및 Purchase Date 열의 인덱스 값을 나타낸다.

20행은 my_columns에 들어 있는 두 개의 열 인덱스 값을 반복하는 for문이다.

반복문을 돌 때마다 해당 열의 데이터 값과 형식을 추출하고 값이 날짜인지 여부를 확인하고 그에 따라 셀 값을 처리한 다음 row_list 에 값을 추가한다.

29행은 row list 의 값을 data에 추가한다.

31~32행의 두 개의 for 문은 data에 포함되어 있는 리스트를 반복하여 출력 파일에 값을 쓴다.

명령줄에 실행한다.

 # python 7excel_column_by_index.py sales_2013.xlsx 7output.xls

☞ **Pandas**

파일명 : 7pandas_column_by_index.py

팬더스로 특정 열을 선택히는 두 가지 방법이 있다.

첫 번째 방법은 데이터 프레임을 할당하고 대괄호 ([ ]) 안에 선택할 열의 인덱스 값 또는 열 헤더를 나열하는 것이다.

두 번째 방법은 iloc( ) 함수로 데이터 프레임을 지정하는 것이다.

iloc( ) 함수는 특정 행과 열을 동시에 선택할 수 있으므로 유용하다.

iloc( ) 함수를 사용하여 열을 선택하는 경우에는 열 인덱스 값 리스트 앞에 콜론 (:) 을 넣어야 한다.

이렇게 하지 않으면 iloc( ) 함수는 해당 인덱스 값의 행을 필터링할 것이다.

명령줄에 다음 명령을 실행한다.

 # python 7pandas_column_by_index.py sales_2013.xlsx 7pandas_output.xls

---

파일명 : 8excel_column_by_name.py

☞ 열 헤더를 사용하여 특정 열을 선택하는 기본 파이썬 코드

워크시트에서 특정한 열의 집합을 선택히는 두 번째 방법은 열 헤더를 사용하는 것이다.

이 방법 은 선택하려는 열의 이름을 쉽게 식별할 수 있을 때 효과적이다.

또한 여러 입력 파일을 처리할 때 열 헤더 내용 자체는 입력 파일 전체에서 일관되지만 열의 위치는 일치하지 않는 경우 유용하다.

Customer ID 및 Purchase Date 열을 선택해보겠다.

13행 에서 포함하려는 열 헤더 2개를 담은 my_columns라는 리스트 변수를 만든다.

이것은 출력 파일에 쓰려고 하는 열의 헤더이므로

17행의 data 라는 출력 리스트에 직접 추가한다.

20행의 for 문은 header_list의 열 헤더 인덱스 값을 반복해서 처리한다.

21행에서는 리스트 인덱싱을 사용하여 각 열 헤더가 my_columns에 포함되어 있는지 판별한다.

포함되어 있다면

22행에서 열 헤더의 인덱스 값을 header_index_list에 추가한다.

29행에서 이 리스트에 있는 인덱스 값을 사용하여 출력 파일에 쓰려는 열만 처리할 것이다.

25행 for문은 header_index_list의 열 인덱스 값을 반복 처리한다.

header_index_list를 사용하여 my_columns 에 나열된 열만 처리한다.

명령줄에서 실행한다.

 # python 8excel_column_by_name.py sales_2013.xlsx 8output.xls

☞ **Pandas**

파일명 : 8pandas_column_by_name.py

팬더스에서 열 헤더를 기반으로 특정 열을 선택하려면 데이터 프레임 뒤에 대괄호 ([ ]) 안에 열 헤더를 문자열로 넣으면 된다. 또는 loc( ) 함수를 사용할 수 있다.

iloc( ) 함수를 사용하여 열을 선택하는 경우에는 열 인덱스 값 리스트 앞에 콜론 (:) 을 넣어야 한다.

이렇게 하지 않으면 iloc( ) 함수는 해당 인덱스 값의 열을 필터링할 것이다.

명령줄에 다음 명령을 실행한다.

 # python 8pandas_column_by_name.py sales_2013.xlsx 8pandas_output.xls

---

파일명 : 9excel_value_meets_condition_all_worksheets.py

☞ 모든 워크시트에서 특정 행 필터링하는 기본 파이썬 코드

기본 파이썬 코드로 모든 워크시트에서 Sale Amount 열의 값이 $2,000.00 이상인 모든 행을 필터링 한다.

13행에서는 sales_column_index라는 변수에 Sale Amount 열의 인덱스 값을 할당한다.

14행에서는 원하는 Sale Amount 열을 선택하기 위해서 threshold 라는 변수를 만든다.

Sale Amount 열의 각 데이터 값을 이 값(2000.0)과 비교하여 출력 파일에 쓸 행을 결정할 것이다.

19행은 통합 문서의 모든 워크시트를 반복 처리하는데 사용하는 for 문이다.

workbook 객체의 sheets 속성을 사용하여 통합 문서의 모든 워크시트에 접근할 수 있다.

16 행에서 first_worksheet를 True로 할당했으므로 20행 if는 21~23행을 실행한다.

즉, 첫 번째 워크시트에서 헤더 행을 추출하여 data에 추가한 다음, first_worksheet를 False로 설정한다.

이후 그다음 코드가 계속 진행되어 Sale Amount 열의 데이터 값이 특정 값(threshold)보다 큰 행을 처리한다.

두 번째 이후 워크시트의 경우 , first_worksheet가 False 이므로 20행 if 블록은 실행되지 않고,

스크립트는 24행으로 이동한다.

range() 함수가 0 대신 1에서 시작되는 점을 보면 헤더 행을 제외한 나머지 데이터 행을 처리한다는것을 알 수 있다.

명령 줄에서 실행한다.

 # 9excel_value_meets_condition_all_worksheets.py sales_2013.xlsx 9output.xls

☞ **Pandas**

파일명 : 9pandas_value_meets_condition_all_worksheets.py sales_2013.xlsx 9pandas_output.xls

팬더스를 사용하면 read_excel() 함수에서 sheet name = None으로 한 번에 통합 문서의 모든 워크시트를 읽을 수 있다.

팬더스는 모든 워크시트를 데이터프레임으로 구성 된 딕셔너리 자료형으로 읽어 들인다.

여기서 키 key는 워크시트의 이름이고, 값은 데이터 프레임 즉, 워크시트 안에 있는 데이터이다.

따라서 딕셔너리의 키와 값을 반복해서 처리하면 통합 문서 내의 모든 데이터를 확인할 수 있다.

각 데이터 프레임에서 특정 행을 필터링하면 그 결과 역시 필터링된 새로운 데이터 프레임이 되므로 이러한 필터링된 데이터 프레임의 리스트를 만든 다음 최종적으로 하나의 데이터 프레임으로 합칠 수 있다.

팬더스를 이용해 모든 워크시트에서 Sale Amount 열의 데이터 값이 $2,000.00 이상인 모든 행을 필터링한다.

명령줄에서 실행한다.

 # python 9pandas_value_meets_condition_all_worksheets.py sales_2013.xlsx 9pandas_output.xls

---

파일명 : 10excel_column_by_name_all_worksheets.py

☞ 모든 워크시트에서 특정 열을 선택하는 기본 파이썬 코드

엑셀 통합 문서에는 여러 개의 워크시트가 포함되어 있으며 각 워크시트에 필요한 것보다 많은 열이 포함되어 있을 수 있다. 이러한 경우 파이썬을 사용하여 모든 워크시트를 읽고, 불필요한 열을 필터링하고, 필요한 열만 선택할 수 있다.

워크시트에서 특정 열의 하위 데이터셋을 선택하는 방법에는 인덱스 값을 이용하는 방법과 열 헤더를 이용하는 방법 두 가지가 있다.

기본 파이썬을 사용하여 모든 워크시트에서 Customer Name 및 Sale Amount 열만 선택한다

13행에서 포함하려는 두 개 열의 이름이 들어 있는 my_columns라는 리스트 변수를 생성한다.

17행은 my_columns를 data 리스트의 첫 번째 원소로 할당한다.

이것이 출력 파일에 저장하려는 특정 열의 헤더이기 때문이다.

18행에서는 Customer Name 및 Sale Amount 열의 인덱스 값을 담을 index_of_cols_to_keep이라는 빈 리스트를 만든다.

20행에서 if 문을 이용해 첫 번째 워크시트를 처리한다.

첫 번째 워크시트에서 Customer Name 및 S ale Amount 열의 인덱스 값을 식별하여 index_of_cols_to_keep에 추가한 다음, first_worksheet를 False로 설정한다. 스크립트는 계속 실행되어 나머지 데이터 행을 계속 처리하고,

28행에서는 Customer Name 및 Sale Amount 열의 값만 처리한다.

모든 후속 워크시트의 경우 first_worksheet는 False이므로 스크립트는 20행 if 문 블록을 건너뛰고 26행으로 이동하여 각 워크시트의 데이터 행을 처리한다. 후속 워크시트의 경우 index_of_cols_to_keep에 들어 있는 인덱스 값에 해당하는 열만 처리한다. 이 열 중 한 값이 날짜인 경우에는 날짜 형식으로 포매팅한다.

그리고 출력 파일에 쓰고 싶은 행의 값을 모은 다음 37행에서 data 리스트에 이를 추가한다.

명령줄에서 실행한다.

 # python 10excel_column_by_name_all_worksheets.py sales_2013.xlsx 10output.xls

☞ **Pandas**

파일명 : 10pandas_column_by_name_all_worksheets.py

팬더스에서는 read_excel( ) 함수로 모든 워크시트를 덕셔너리 자료형으로 읽는다.

그 다음 loc() 함수를 사용하여 각 워크시트의 특정 열을 선택하여 필터링된 데이터 프레임의 리스트를 만들고,

그 데이터프레임들을 하나의 최종적인 데이터 프레임으로 합친다.

이 스크립트에서도 모든 워크시트에서 Customer Name 및 Sale Amount 열을 선택할 것이다.

명령줄에서 실행한다.

 # python 10pandas_column_by_name_all_worksheets.py sales_2013.xlsx 10pandas_output.xls

---

※ 엑셀 통합 문서에서 워크 시트 집합 읽기 ※

지금까지는 하나의 워크시트에서 특정 행과 열을 필터링하는 방법과 통합 문서의 모든 워크시트에서 특정 행과 열을 필터링하는 방법을 실습했다.

하지만 어떤 경우에는 통합 문서에서 워크시트의 일부분만 처리 해야 할 수도 있다.

예를 들어,

통합 문서 내에 수십 개의 워크시트가 포함되어 있지만 그 중에서 필요한 20개의 워크시트만 처리 해야 할 수도 있다.

이런 상황에서는 sheet_by_index( ) 또는 sheet_by_name( ) 함수를 사용하여 통합 문서 내에서 워크시트의 하위 데이터셋을 처리할 수 있다.

이번 실습에서는 엑셀 통합 문서 내의 특정 한 워크시트 집합에서 특정 행을 필터링하는 방법을 실습한다.

파일명 : 11excel_value_meets_condition_set_of_worksheets.py

☞ 워크시트 집합에 걸쳐서 특정 행 필터링하는 기본 파이썬 코드

이번 스크립트에서는 첫 번째와 두 번째 워크시트에서 Sale Amount 열의 데이터 값이 $1,900.00 보다 큰 행만 필터링하고자 한다. 기본 파이썬을 이용해 첫 번째와 두 번째 워크시트에서 특정 행을 선택하는 코드는 다음과 같다.

13 행에서는 처리해야 하는 워크시트의 인덱스 값을 나타내는 두 개의 정수(0과 1)가 포함된 my_s heets 라는 리스트 변수를 만들었다.

20 행에서는 통합 문서의 모든 워크시트에 대한 인덱스 값을 만들고 인덱스 값에 대해 for 문을 적용한다.

21 행에서는 for 문의 현재 인덱스 값이 my_sheets의 인덱스 값 중 하나인지 여부를 판별하여 원하는 워크시트만 처리하도록 한다.

22행에서는 sheet_by_index( ) 함수에 현재 인덱스(sheet_index)를 전달해 현재 처리할 워크시트에 접근한다.

첫 번째 워크시트의 경우 23행은 True 이므로 헤더 행을 data에 추가하고 first_worksheet를 False로 설정한다.

그 다음 앞의 예제들과 마찬가지로 나머지 데이터 행을 비슷한 방식으로 처리한다.

후속 워크시트들은 이 if 블록을 건너뛰고 27행으로 이동하여 나머지 데이터 행을 처리한다.

명령줄에서 실행한다.

 # python 11excel_value_meets_condition_set_of_worksheets.py sales_2013.xlsx 11output.xls

☞ **Pandas**

파일명 : 11pandas_value_meets_condition_set_of_worksheets.py

팬더스에서는 통합 문서에서 특정 워크시트를 쉽게 선택할 수 있다.

read_excel( ) 함수에서 워크시트의 인덱스 번호나 이름을 지정하기만 하면 된다.

이 실습에서는 my_sheets라는 인덱스 번호 리스트를 만든 다음,

read_excel( ) 함수내에서 sheetname을 my_sheets과 동일하게 설정하겠다.

팬더스로 워크시트의 하위 집합을 선택해보겠다.

명령줄에서 다음 명령을 실행한다.

 # python 11pandas_value_meets_condition_set_of_worksheets.py sales_2013.xlsx 11pandas_output.xls

---

※ 여러 개의 통합 문서 처리 ※

이번 실습을 위해 3개의 통합 문서 파일을 사용한다.

파일명 : sales_2013.xlsx, sales_2014.xlsx, sales_2015.xlsx

파일명 : 12excel_introspect_all_workbooks.py

☞ **여러 개의 통합 문서를 처리하는 기본 파이썬 코드**

2행과 3행에서 파이썬에 내장된 glob과 OS 모듈을 임포트해서 처리할 파일의 경로를 식별하고 파싱하는 함수를 사용할 수 있게 했다.

10행에서는 파이썬에 내장된 glob과 OS 모듈을 사용하여 처리하려는 입력 파일 리스트를 만들고 for문을 적용한다.

이를 통해 모든 통합 문서를 반복 처리할 수 있다.

12 ~ 16행은 각 통합 문서에 대한 정보를 화면에 출력한다.

12행은 통합 문서의 이름을, 13행은 통합 문서 내의 워크시트 개수를 출력한다.

15 ~16행은 통합 문서 내의 워크시트 이름과 각 워크시트 내의 행과 열의 개수를 출력 한다.

명령줄에서 다음 명령을 실행한다.

 # python 12excel_introspect_all_workbooks.py "C:₩Users₩계정명₩sample"

결과는 다음과 같다.

Workbook: sales_2013.xlsx

Number of worksheets: 3

Worksheet name: january_2013 Rows: 7 Columns: 5

Worksheet name: february_2013 Rows: 7 Columns: 5

Worksheet name: march_2013 Rows: 7 Columns: 5

Workbook: sales_2014.xlsx

Number of worksheets: 3

Worksheet name: january_2014 Rows: 7 Columns: 5

Worksheet name: february_2014 Rows: 7 Columns: 5

Worksheet name: march_2014 Rows: 7 Columns: 5

Workbook: sales_2015.xlsx

Number of worksheets: 3

Worksheet name: january_2015 Rows: 7 Columns: 5

Worksheet name: february_2015 Rows: 7 Columns: 5

Worksheet name: march_2015 Rows: 7 Columns: 5

파일명 : 13excel_concat_data_from_multiple_workbooks.py

☞ 여러 개의 통합 문서를 합치는 기본 파이썬 코드

기본 파이썬을 사용해 여러 통합 문서 내의 모든 워크시트의 데이터를 하나의 출력 파일로 수직 방향으로 합쳐 보겠다.

16행에서는 첫 번째 워크시트와 이후 처리할 모든 후속 워크시트를 구분하는 데 사용하는 first_worksheet 라는 불리언(True/ False) 변수를 만든다.

첫 번째 워크시트의 경우, 21행이 True 이므로 헤더 행을 데이터에 추가한 다음 first_worksheet를 False로 설정한다.

그 다음 첫 번째 워크시트와 나머지 모든 후속 워크시트에 대해 헤더 이후 나머 지 데이터 행을 처리한다.

25행의 range() 함수가 0 대신 1 에서 시작하므로 두 번째 행부터 시작한다는 것을 알 수 있다.

명령줄에서 다음 명령을 실행한다.

 # python 13excel_concat_data_from_multiple_workbooks.py "C:₩Users₩계정명₩sample" 13output.xls

☞ Pandas

파일명 : 13pandas_concat_data_from_multiple_workbooks.py

팬더스는 데이터 프레임을 결합하는 concat( ) 함수를 제공한다.

데이터 프레임을 수직으로 결합하려면 axis=0, 수평으로 결합하려면 axis=1 을 사용한다.

또는 키가 되는 열을 기반으로 데이터 프레임을 조인해야 하는 경우에는 팬더스의 merge( ) 함수가 SQL의 조인과 유사한 기능을 제공한다.

팬더스를 이용해 여러 개의 통합 문서의 워크시트 속 모든 데이터를 수직 방향으로 하나의 출력 파일로 결합해보겠다.

명령줄에서 다음 명령을 실행한다.

 # python 13pandas_concat_data_from_multiple_workbooks.py "C:₩Users₩계정명₩sample" 13pandas_output.xls

---

파일명 : 14excel_sum_average_multiple_workbooks.py

☞ 통합 문서 및 워크시트별 합계 및 평균을 구하는 기본 파이썬 코드

여러 개의 통합 문서에서 워크시크 및 통합 문서별 통계를 계산해보는 실습을 한다.

15 행 에서 all_data 라는 빈 리스트를 만들었다. 출력 파일에 기록할 모든 행을 여기 담을 것이다.

16행에서는 sales_column_index라는 변수를 만들어 Sale Amount 열의 인텍스 값을 담는다.

18~19행은 출력 파일의 열 헤더 리스트를 만들고,

20행에서는 이 헤더 리스트의 값을 all_data 에 추가한다.

24~26행 에서는 3 개의 리스트를 생성한다.

list_of_tota ls에는 통합 문서 내의 모든 워크시트별 총 판매 금액을 담을 것이다.

list_of_numbe rs 에는 통합 문서의 모든 워크시트별 총 판매 금액을 계산하든데 사용된 판매 금액의 개수를 담을 것이 다. 세 번째 리스트인 workbook output은 출력 파일에 출력할 모든 리스트를 담을 것이다.

30행 에서는 각 워크시트에 대한 모든 정보를 담을 worksheet_list 리스트를 만든다.

31 ~32행 에서 통합 문서의 이름과 워크시트의 이름을 worksheet_list 에 추가한다.

41 ~42행에서는 Sale Amount 열의 합계 및 평균을 worksheet_list 에 추가한다.

45행에서는 이 workheet_list를 workbook_output 에 추가하여 통합 문서 차원의 정보를 저장한다 .

43~44 행에서는 워크시트 차원에서 Sale Amount 열의 합계와 개수를 각각 list_of_totals와 list_of_numbers 에 추가한다. 이렇게 함으로써 모든 워크시트에 걸쳐 각 워크시트의 정보를 저장할 수 있다.

46~47행에서는 이 리스트를 이용하여 통합 문서 차원에서 Sale Amount 열의 총 합계와 평균을 계산한다.

48 ~50행은 workbook_output 리스트를 반복 처리하여 (각 통합 문서에는 세 개의 워크시트가 있으므로 각 통합 문서 당 세 개의 리스트가 있다) 각 리스트에 통합 문서 차원의 합계 및 평균을 추가한다.

통합 문서에서 얻으려는 모든 정보 (즉 각 워크시트별 하나씩 세 개의 리스트) 가 확보되면 그 리스트를 all_data로 확장한다. 이 때 append( ) 함수 대신 extend( ) 함수를 사용하여 workbook_ output의 각 리스트가 all_data의 개별 원소가 되게 했다.

이렇게 하면 세 개의 모든 통합 문서를 처리한 후 이 all_data는 9개의 원소를 가진 리스트가 되며 , 이 각 원소 역시

리스트이다. 이렇게 하지 않고 대신 append( ) 를 사용한다면 all_data 에는 세 개의 원소만 들어가고, 각 원소는 리스트의 리스트가 될 것이다.

명령줄에서 다음 명령을 실행한다.

 # python 14excel_sum_average_multiple_workbooks.py "C:₩Users₩계정명₩sample" 14output.xls

☞ **Pandas**

파일명 : 14pandas_sum_average_multiple_workbooks.py

팬더스에서는 여러 개의 통합 문서에 대해 워크시트 및 통합 문서 차원에서 통계를 계산하는 것이 상대적으로 간단하다.

이 스크립트에서는 통합 문서의 각 워크시트에 대한 통계를 계산하고 그 결과를 데이터 프레임으로 결합한다.

그 다음 통합 문서 차원의 통계를 계산하여 데이터 프레임으로 변환한다.

이 두 개의 데이터 프레임을 통합 문서 이름을 기준으로 레프트 조인하여 결합한 데이터 프레임을 리스트에 추가한다.

통합 문서 차원의 모든 데이터 프레임을 리스트에 포함한 다음에는 이를 하나의 데이터 프레임으로 결합하여 출력 파일에 쓴다.

팬더스를 이용해 여러 통합 문서에서 워크시트 및 통합 문서 별 통계를 계산해보겠다.

명령줄에서 다음 명령을 실행한다.

 # python 14pandas_sum_average_multiple_workbooks.py.py "C:₩Users₩계정명₩sample" 14pandas_output.xls

==========

※ 실습 파일 ※

1excel_introspect_workbook

2excel_parsing_and_write

3excel_parsing_and_write_keep_dates

4excel_value_meets_condition

5excel_value_in_set

6excel_value_matches_pattern

7excel_column_by_index

8excel_column_by_name

9excel_value_meets_condition_all_worksheets

10excel_column_by_name_all_worksheets

11excel_value_meets_condition_set_of_worksheets

12excel_introspect_all_workbooks

13excel_concat_data_from_multiple_workbooks

14excel_sum_average_multiple_workbooks

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

sales_2013.xlsx

sales_2014.xlsx

sales_2015.xlsx