Working with Excel files using `openpyxl` package

Disclaimer: These notes and examples are an adaptation of the references listed at the end. They are compiled to fit the scope of this specific course.

Introduction

As a student/graduate of a business school, you are required to know - at least - the basics of MS Excel, the most widely used software for analytical purposes in the business world. Fortunately, Python has numerous packages available to help you create, read, and edit Excel files. Complete Python packages available to work with Excel are listed in https://www.python-excel.org/. Some of these packages, such as `xlrd` and `xlwt` work only with older Excel files with `.xls` file extension.

This handout will cover the basics of `openpyxl` package. `openpyxl` is the recommended package for:

- Reading the Excel files
- Writing data to an Excel file and creating `xlsx` and `xlsm` files
- Updating existing Excel files

`openpyxl` can help you automate repetitive tasks such as preparing/updating weekly/monthly reports in Excel files. Additionally, `numpy` , `pandas` , and many other Python libraries and packages are easily integrated with `OpenpyxL` to automate tasks and to collect and analyze data.

Note 1: Some of the tasks we will do with `openpyxl` can also be done with libraries such as `pandas` which we will learn later.

Note 2: Note that `openpyxl` is a very elaborate library with numerous methods for performing advanced tasks such as plotting Excel graphs. This handout covers the absolute minimum necessities. Complete documentation of this package is available at https://openpyxl.readthedocs.io/en/stable/

Version & Installation

At the time of preparing this handout, `openpyxl` was in `version 3.0.10`, which was released on `May 19, 2022`.

Install this package by Package installation can be done by running the following snippet in an empty cell of `JupyterLab`.

```
pip install openpyxl
```

Why not `csv` files?

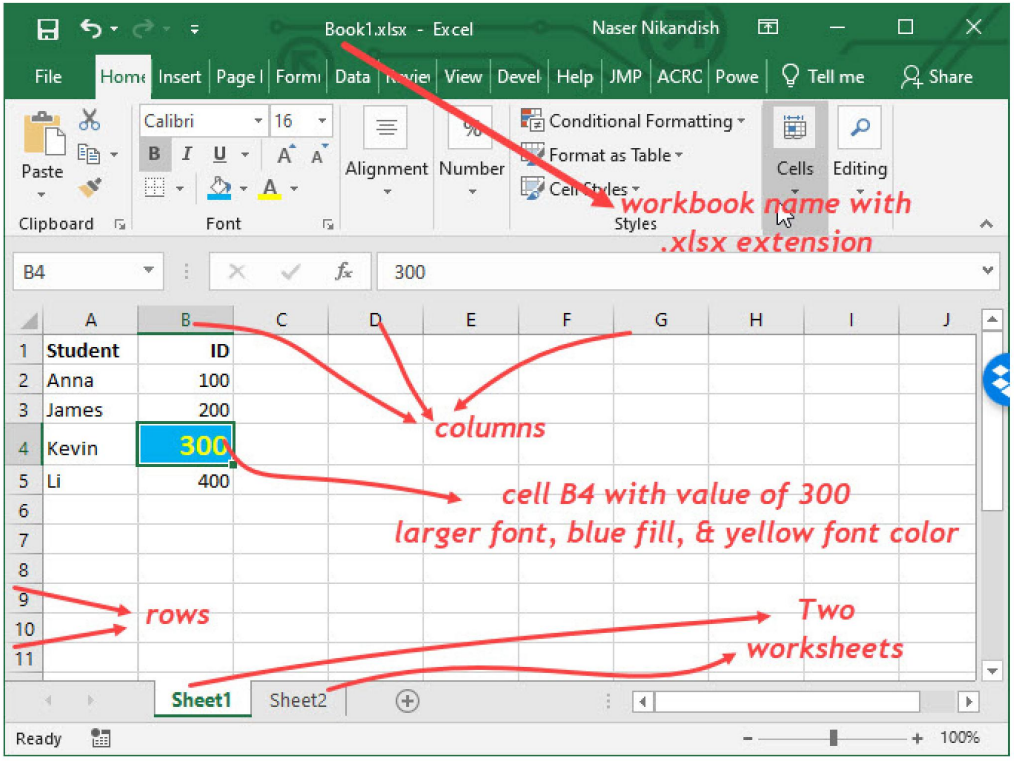
You have learned to easily work with `csv (comma-separated values)` files in other classes and may wonder why one would spend time dealing with Excel files. Why not use `csv` files instead? Among many reasons, these are just a few:

- `csv` files do not save formulas.
- `csv` file type offers only 2 data types: Strings and Numbers. Any other data type such as dates, monetary values, etc cannot be saved in a `csv` file. Graphs cannot be saved in a `csv` file either.
- There is no generally accepted standard around the world regarding separating values. One country may use `,` while another uses `-` for separating values.

Both `csv` files and `xlsx` files are helpful; one cannot be a 100% replacement for the other.

Excel Basics

An Excel file is usually referred to as a *`workbook`*. The default file extension for an Excel *`workbook`* is `xlsx`. Each *`workbook`* may contain one or more *`worksheets`* or *`sheets`*, in short; and the sheet the user is currently working on (or last viewed before saving and closing the Excel file) is known as *`active sheet`*. Each Excel *`sheet`* is a giant two-dimensional table with *`columns`* (denoted by capital letters starting at *`A`*) and *`rows`* (denoted by numbers starting from *`1`*). A column and row combination (intersection) is known as a *`cell`*. A *`cell`* is referenced by its *`column`* letter followed by its *`row`* number such as *`A1`* or *`B3`*. Here is a visual guide:



Common Methods used for reading `openpyxl`

After installing `openpyxl`,

```
import openpyxl
```

will import `openpyxl` and will make all its capabilities available to you in your current session. The following are most commonly used methods of `openpyxl`:

Common methods used for reading an Excel file

Open an Excel file

- `workbook_identifier = openpyxl.load_workbook('workbook_name.xlsx')` will open the `workbook_name.xlsx` file and link the variable `workbook_identifier` to this workbook object. **Important note:** `load_workbook` loads the latest saved Excel file into the computer's memory. If you change & save the Excel file after loading it, changes will not be reflected in the computer's memory. You need to reload the file using `load_workbook` to see the latest version.
- `workbook_identifier.active` will return the name of the active worksheet. `active` returns a `list`.

Get worksheet names

- `workbook_identifier.sheetnames` will return names of worksheets in this workbook. `sheetnames` returns a `list`.

Get a worksheet from a workbook

- `my_sheet = workbook_identifier['sheet_name']` will point `my_sheet` to `sheet_name` worksheet.
- `my_sheet.title` will print the name of `my_sheet`.

Get worksheet info

- `my_sheet.max_row` and `my_sheet.max_column` will return the maximum number of rows and columns in `my_sheet`, respectively.

Accessing a row and column

- `my_sheet['column_letter']` returns the whole column of `column_letter` as a tuple
- `my_sheet[row_number]` returns the whole row of `row_number` as a tuple

Accessing a cell value

There are two common ways of accessing a cell value.

- Use `my_sheet['cell_reference'].value` to get the value stored in cell `cell_reference`. `cell_reference` is the cell address in the form of `A1`, `B4`, etc.
- Use the `.cell(row = row_number, column = col_number)` method to get the cell. Both `row_number` and `col_number` must be integer numbers starting from 1. for example, cell `B4` has `row_number = 4` and `col_number = 2`. Do not forget `.value` to

get the cell value.

Note: For consistency and better readability purposes, it is always a good practice to choose one method and use it consistently.

Common methods used for editing Excel file

Creating a new worksheet & deleting an existing one

- `workbook_identifier.create_sheet(title = 'my_new_worksheet')` will add a new worksheet with the name `my_new_worksheet`.
- `del workbook_identifier['sheet_name']` will delete `sheet_name` worksheet.

Writing a value in a cell

- `my_sheet.cell(row = row_number, column = column_number).value = target_value` will write `'target_value'` in cell row `row_number` and column `column_number`.

Saving the Excel file

- `workbook_identifier.save('file_name.xlsx')` will save the workbook `workbook_identifier` with the file name `filename.xlsx'`. It is always a good practice to save your work under a new meaningful name.

Closing the Excel file

- `workbook_identifier.close()` will close the workbook `workbook_identifier`.

An example

```
import openpyxl


# Reading the excel file
# ################################################################################
wb = openpyxl.load_workbook('new.xlsx')    # load the workbook new.xlsx
print(wb.sheetnames)                       # print all worksheets names in a list
sheet = wb['sheet_name']                   # define sheet to work with
wb.active = sheet                          # design sheet as the active worksheet
```

```
print(sheet.max_row)                      # print max row numbers
print(sheet.max_column)                   # print max column numbers
x = sheet.cell(row = 2, column = 1).value  # assign value of cell A2 to variable x
# ###############################################################################


# writing to Excel file
# ###############################################################################
wb.create_sheet('results')                    # create a new worksheet
sheet2 = wb['results']                        # define sheet2 to work with
sheet2.cell(row = 1, column = 3).value = 100  # write in cell C1
# ###############################################################################


# save the workbook
# ###############################################################################
wb.save('updated_new.xlsx')  # save under new name
###############################################################################
```

Two more elaborate methods of `openpyxl`

Now that we are familiar with fundamental methods of `openpyxl`, I would like to point your attention to two handy methods for looping through a worksheet's rows and columns.

- `iter_rows()` let's you iterate through rows.
- `iter_cols()` let's you iterate through columns.

Here is a quick example

```
import openpyxl

wb = openpyxl.load_workbook('new.xlsx')
ws = wb['Sheet1']

for row in ws.iter_rows():
    print(row[0].value)
```

will print all values in the first column of all rows.

> similarly,
>
> ```
> import openpyxl
>
> wb = openpyxl.load_workbook('new.xlsx')
> ws = wb['Sheet1']
>
> for col in ws.iter_cols():
>     print(col[1].value)
> ```
>
> prints all values in the second row of all columns.

Please consult the package documentation at https://openpyxl.readthedocs.io/en/stable/ for more details.

References

1- *Python online documentation*; available at https://docs.python.org/3/

2- *A Byte of Python*; available at https://Python.swaroopch.com/

3- *Introduction to programming in Python*; available at https://introcs.cs.princeton.edu/Python/home

4- *Introduction to Computation and Programming Using Python: With Application to Understanding Data*, John Guttag, The MIT Press, 2nd edition, 2016

5- *Introduction to Python for Computer Science and Data Science*, Paul J. Deitel, Harvey Deitel, Pearson, 1st edition, 2019

6- *Data Mining for Business Analytics*, Galit Shmueli, Peter C. Bruce, Peter Gedeck, Nitin R. Patel, Wiley, 2020

7- *Introduction to Programming Using Java*; available at http://math.hws.edu/javanotes/

8- https://betterprogramming.pub/

9- https://www.learnpython.org/

10- https://www.python-excel.org/

11- https://ryanflynndev.medium.com/how-to-easily-automate-excel-spreadsheetswith-python-and-openpyxl-1f502400976