

A PYTHON PROJECT

ON

PANDA AND OPENPYXL

COMPLETED BY TEAM TORNADO

GUIDE: MR. SHOBHIT NIGAM

Submitted By:

Dolly Bagaria
Devvrat Vaidya
Ishant Tiwari
Pramodh Narayan L

PROBLEM STATEMENT

To design a python application which will be able to open up 2 Excel files. These files will both be data tables laid out in a similar format as the below mentioned examples.

| Unique ID | Data Point | 2020 | 2021 | 2022 | 2023 |
|-----------|------------|------|------|------|------|
| Abc | Var_a | 1 | 43 | 543 | 42 |
| Abc | Var_b | 42 | 7654 | 786 | 63 |
| Def | Var_a | 234 | 543 | 654 | 432 |
| Def | Var_b | 4 | 54 | 653 | 2 |
| Ghi | Var_a | 43 | 342 | 0 | 5 |
| Ghi | Var_b | 43 | 543 | 65 | 123 |
| Jkl | Var_a | 432 | 432 | 432 | 432 |
| Jkl | Var_b | 54 | 2 | 65 | 872 |
| Mno | Var_a | 981 | 234 | 32 | 98 |
| Mno | Var_b | 5342 | 6543 | 875 | 2456 |

Fig 1 : Table 1 for comparison

| Unique ID | Data Point | 2020 | 2021 | 2022 | 2023 |
|-----------|------------|------|------|------|------|
| Abc | Var_a | 1 | 43 | 500 | 42 |
| Abc | Var_b | 42 | 7654 | 700 | 63 |
| Def | Var_a | 234 | 543 | 654 | 432 |
| Def | Var_b | 4 | 54 | 653 | 10 |
| Ghi | Var_a | 43 | 342 | 4 | 5 |
| Ghi | Var_b | 43 | 600 | 65 | 123 |
| Jkl | Var_a | 400 | 432 | 432 | 432 |
| Jkl | Var_b | 54 | 2 | 75 | 872 |
| Mno | Var_a | 981 | 234 | 32 | 100 |
| Mno | Var_b | 5342 | 6543 | 0 | 2456 |

Fig 2 : Table 2 for comparison

It should be able to tell for each ID what Data Point what has changed with respect to each other. We will be highlighting all the updated rows and columns using a special formatting (in this case the color changes to red).

| Unique ID | Data Point | 2020 | 2021 | 2022 | 2023 |
|-----------|------------|------|------|------|------|
| Abc | Var_a | 0 | 0 | 43 | 0 |
| Abc | Var_b | 0 | 0 | 86 | 0 |
| Def | Var_a | 0 | 0 | 0 | 0 |
| Def | Var_b | 0 | 0 | 0 | -8 |
| Ghi | Var_a | 0 | 0 | -4 | 0 |
| Ghi | Var_b | 0 | -57 | 0 | 0 |
| Jkl | Var_a | 32 | 0 | 0 | 0 |
| Jkl | Var_b | 0 | 0 | -10 | 0 |
| Mno | Var_a | 0 | 0 | 0 | -2 |
| Mno | Var_b | 0 | 0 | 875 | 0 |

Fig 3 : Comparison Table

TECHNOLOGIES USED

The project uses **Python 3** as the coding platform and uses two libraries i.e

1. **Pandas** - an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming
2. **OpenPYXL** - is a Python library to read/write Excel 2010 xlsx/xlsm/xltx/xltn files. It was born from lack of existing library to read/write natively from Python

SCOPE OF THE PROJECT

The project will revolve around comparing two excel files and highlighting out the differences to compare values in multiple scenarios, and making a real world application out of it.

SOURCE CODE

IMPLEMENTATION USING PANDAS

a) Import

Importing essential libraries pandas and numpy

```
# importing panda and numpy libraries
import pandas as pd
import numpy as np
```

b) Assigning the excel sheets as inputs in python

Assigning old and new excel sheets to old and new respectively

```
# importing excel files as dataframes:old, new and final are the dataframes of excel file
old=pd.read_excel('excel_old.xlsx')
new=pd.read_excel('excel_new.xlsx')
final=pd.read_excel('excel_new.xlsx')
```

c) A boolean matrix is generated using the comparison in-built function between the two generated DataFrames

```
#Comparing old and new values of the excel sheets.
#If values are same, it will return true.
#Else false
comparison_values = old.values == new.values
print (comparison_values)
```

```
[[ True  True  True False False  True]
 [ True  True  True False False  True]
 [ True  True False False False False]
 [ True  True False False False  True]
 [ True  True False False False  True]
 [ True  True False False False False]
 [ True  True  True  True  True  True]]
```

Fig 4 : Boolean Matrix

d) Looking for rows and columns with 'False' values

```
#Putting the indexes of row and columns who have value as false int "rows" and "cols"
rows,cols=np.where(comparison_values==False)
```

e) After subtracting the false column value

“iloc” is used to fetch the row of a dataframe

```
#Subtracting the row,column values of new file from old file
for item in zip(rows,cols):
    final.iloc[item[0], item[1]] = '{}'.format(new.iloc[item[0], item[1]]-old.iloc[item[0], item[1]])
```

| | unique id | data point | 2020 | 2021 | 2022 | 2023 |
|---|-----------|------------|------|------|------|------|
| 0 | abc | var_a | 11 | 21 | 48 | 87 |
| 1 | abc | var_b | 8 | 21 | -45 | 76 |
| 2 | def | var_a | 53 | 11 | 200 | 12 |
| 3 | def | var_b | 45 | -23 | 33 | 88 |
| 4 | ghi | var_a | 12 | 11 | 32 | 50 |
| 5 | ghi | var_b | 1 | 19 | 28 | 39 |
| 6 | jkl | var_a | 10 | 11 | 12 | 13 |

Fig 5: modified excel sheet

f) Looking for 'True' values

```
#Putting the indexes of row and columns who have value as true int "rows" and "cols"
rows1,cols1=np.where(comparison_values==True)
```

```
#Putting zero value where non-string values were same in old and new file
for item in zip(rows1,cols1):
    if (type(old.iloc[item[0],item[1]])!=str):
        final.iloc[item[0], item[1]] = '{}'.format(old.iloc[item[0], item[1]]-new.iloc[item[0], item[1]])
```

```
#Displaying the final dataframe after operations
final
```

| | unique id | data point | 2020 | 2021 | 2022 | 2023 |
|---|-----------|------------|------|------|------|------|
| 0 | abc | var_a | 0 | 21 | 48 | 0 |
| 1 | abc | var_b | 0 | 21 | -45 | 0 |
| 2 | def | var_a | 53 | 11 | 200 | 12 |
| 3 | def | var_b | 45 | -23 | 33 | 0 |
| 4 | ghi | var_a | 12 | 11 | 32 | 0 |
| 5 | ghi | var_b | 1 | 19 | 28 | 39 |
| 6 | jkl | var_a | 0 | 0 | 0 | 0 |

Fig 6: Modified excel sheet

g) Colouring the values

```
#Changing the value colour to red if the value has changed
def color_negative_red(value):
    color = 'red' if (value!=0 and type(value)!=str) else 'black'
    return 'color: %s' %color
```

```
df_display=final.style.applymap(color_negative_red)
```

```
df_display
```

```
#Storing the dataframe as excel file
df_display.to_excel('Final_panda.xlsx', index=False)
```

| Home Insert Page Layout Formulas Data | | | | | | |
|---------------------------------------|-----------|-----------------|------|----------------|------|------|
| Paste | | Cut Copy Format | | Calibri (Body) | 11 | A A |
| A1 | | fx | | unique id | | |
| | A | B | C | D | E | F |
| 1 | unique id | data point | 2020 | 2021 | 2022 | 2023 |
| 2 | abc | var_a | 0 | 21 | 48 | 0 |
| 3 | abc | var_b | 0 | 21 | -45 | 0 |
| 4 | def | var_a | 53 | 11 | 200 | 12 |
| 5 | def | var_b | 45 | -23 | 33 | 0 |
| 6 | ghi | var_a | 12 | 11 | 32 | 0 |
| 7 | ghi | var_b | 1 | 19 | 28 | 39 |
| 8 | jkl | var_a | 0 | 0 | 0 | 0 |
| 9 | | | | | | |
| 10 | | | | | | |
| 11 | | | | | | |
| 12 | | | | | | |

Fig 7: Modified excel sheet

CODE:

```
#importing necessary modules
import pandas as pd
import numpy as np
#reading old new final excel files through panda
old=pd.read_excel('excel_old.xlsx')
new=pd.read_excel('excel_new.xlsx')
final=pd.read_excel('excel_new.xlsx')
#list of columns for both the files
lista=list(old.columns)
listb=list(new.columns)
#comparing columns , sorting them and calling diff() to compute difference
if lista==listb:
    old=old.sort_values(by=['unique id','data point'])
    new=new.sort_values(by=['unique id','data point'])
    diff()
    df_display=final.style.applymap(color_negative_red)
```

```
def diff():
    #compute comparison matrix
    comparison_values = old.values == new.values
    print (comparison_values)
    rows,cols=np.where(comparison_values==False)
    #iterating rows and columns and formating the excel accordingly
    for item in zip(rows,cols):
        final.iloc[item[0], item[1]] = '{}'.format(new.iloc[item[0], item[1]]-old.iloc[item[0], item[1]])
    rows1,cols1=np.where(comparison_values==True)
    for item in zip(rows1,cols1):
        if (type(old.iloc[item[0],item[1]])!=str):
            final.iloc[item[0], item[1]] = '{}'.format(old.iloc[item[0], item[1]]-new.iloc[item[0], item[1]])
```

#function for highlighting the non zero values

```
def color_negative_red(value):
    color = 'red' if (value!=0 and type(value)!=str) else 'black'
    return 'color: %s' %color
```

df_display

| | unique id | data point | 2020 | 2021 | 2022 | 2023 |
|---|-----------|------------|------|------|------|------|
| 0 | abc | var_a | 0 | 21 | 48 | 0 |
| 1 | abc | var_b | 0 | 21 | -45 | 0 |
| 2 | def | var_a | 53 | 11 | 200 | 12 |
| 3 | def | var_b | 45 | -23 | 33 | 0 |
| 4 | ghi | var_a | 12 | 11 | 32 | 0 |
| 5 | ghi | var_b | 1 | 19 | 28 | 39 |
| 6 | jkl | var_a | 0 | 0 | 0 | 0 |

```
#converting dataframe to excel for final output
df_display.to_excel('Final_panda',index=False)
```

IMPLEMENTATION USING OPENPYXL

a) Importing pyxl libraries

Importing the necessary libraries

```
# importing pyxl libraries
from openpyxl import Workbook
from openpyxl import load_workbook
from openpyxl.styles import Fill,Font,Color,colors
```

b) Creating a workbook object from old excel sheet

```
#Creating a workbook object for old excel sheet
wold = load_workbook('excel_old.xlsx')
```

c) Creating a workbook object from new excel sheet

```
#Creating a workbook object for new excel sheet
wnew = load_workbook('excel_new.xlsx')
```

d) Creating a workbook object for final excel sheet

```
#Creating a workbook object for final excel sheet
wfinal = Workbook()
```

e) Activating the worksheet and giving the detail as sheet1

```
#Activating the wold and wnew sheets as sheet1 and sheet2 respectively
sheet1=wold.active
sheet2=wnew.active
```

f) Finding the maximum rows and columns of old sheets

```
#Activating the wold and wnew sheets as sheet1 and sheet2 respectively
sheet1=wold.active
sheet2=wnew.active
```

```
#Finding the maximum row and columns of old sheet and assigning it to variable.
max_row1=sheet1.max_row
max_column1=sheet1.max_column
```

g) Getting the headers of old sheet and putting it in final sheet and saving it

```
# Getting the headers of old sheet and putting it in final sheet
for i in range(1,2):
    for j in range(1,max_column1+1):
        cell_obj1=sheet1.cell(row=i,column=j)
        wfinalsheet.cell(row=i, column=j).value = cell_obj1.value
#Saving the final workbook
wfinal.save('final_excel.xlsx')
```

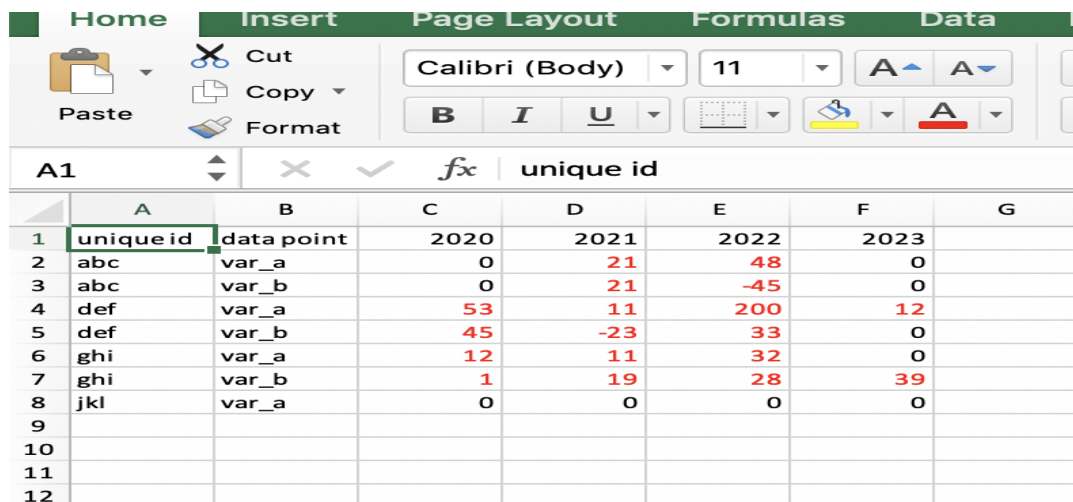

h) When the values are non-string, adding the values of new excel from old excel to final sheet

```
#Adding the subtracted value of new excel from old excel to final sheet...
#...where the values are not string type
for i in range(2,max_row1+1):
    for j in range(1,max_column1+1):
        cell_obj1=sheet1.cell(row=i,column=j)
        cell_obj2=sheet2.cell(row=i,column=j)
        if type(cell_obj1.value) == str and type(cell_obj2.value) == str :
            wfinalsheet.cell(row=i, column=j).value = cell_obj1.value
        else:
            temp=cell_obj2.value-cell_obj1.value
            if(temp!=0):
                wfinalsheet.cell(row=i, column=j).font = Font(color=colors.RED)
            wfinalsheet.cell(row=i, column=j).value = temp
```

i) Saving a file

```
#Saving the file
wfinal.save('final_excel.xlsx')
```

Fig 9 : excel_file_old.xlsx



| | A | B | C | D | E | F | G |
|----|-----------|------------|------|------|------|------|---|
| 1 | unique id | data point | 2020 | 2021 | 2022 | 2023 | |
| 2 | abc | var_a | 0 | 21 | 48 | 0 | |
| 3 | abc | var_b | 0 | 21 | -45 | 0 | |
| 4 | def | var_a | 53 | 11 | 200 | 12 | |
| 5 | def | var_b | 45 | -23 | 33 | 0 | |
| 6 | ghi | var_a | 12 | 11 | 32 | 0 | |
| 7 | ghi | var_b | 1 | 19 | 28 | 39 | |
| 8 | jkl | var_a | 0 | 0 | 0 | 0 | |
| 9 | | | | | | | |
| 10 | | | | | | | |
| 11 | | | | | | | |
| 12 | | | | | | | |

Fig 10 : excel_file_diff.xlsx

Result

The files have been compared using execution of two technologies and a new file has been generated in all instances.

