Add New PCF to Seita

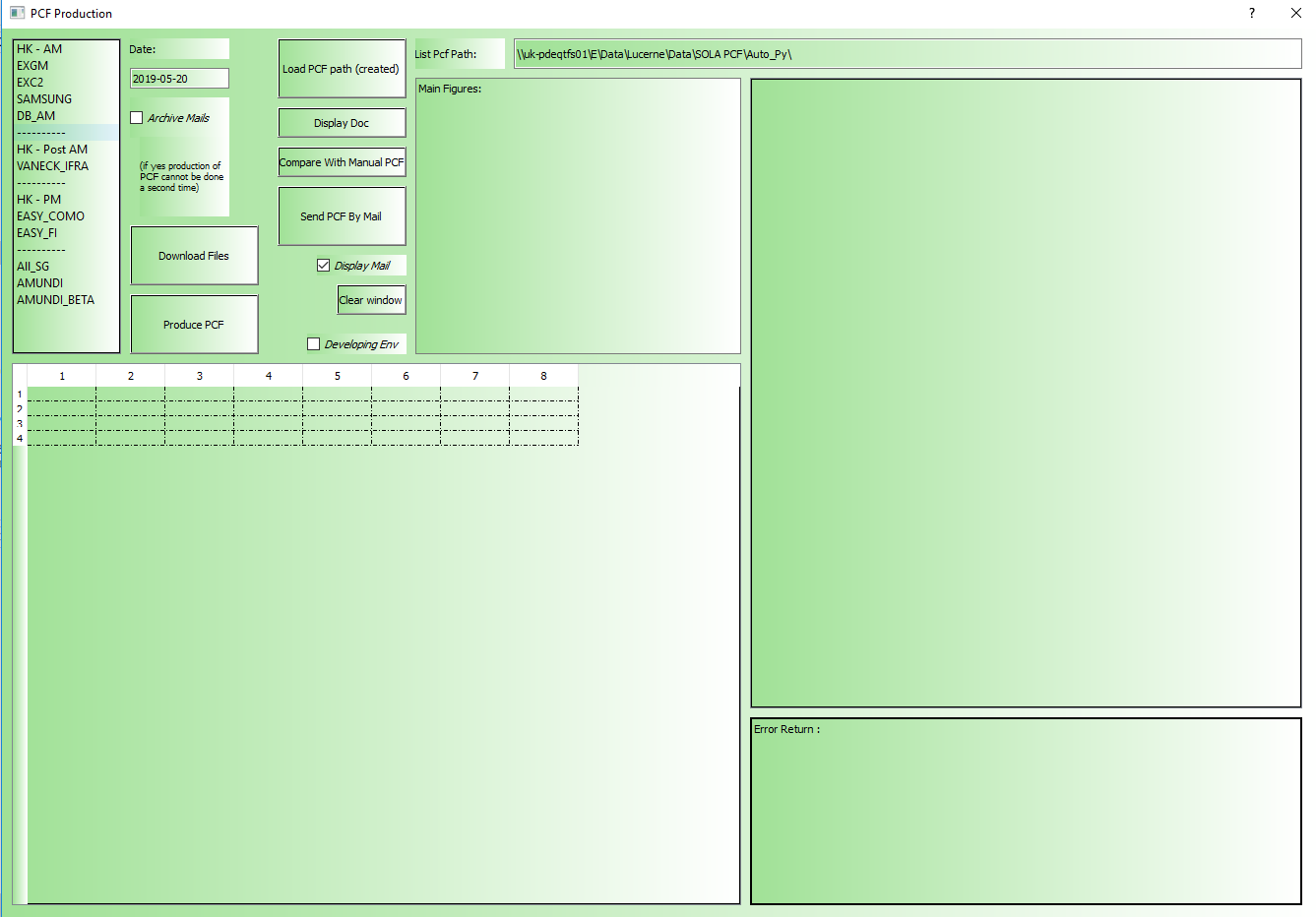
## What is Seita?

Seita is a program developed in Python to automatize your PCF creation that would still be manual.

There is 2 ways of using it:

* The App
* Automatic launch with windows Scheduler and receives the file by Mail

# The App



## Where to find it ?

[\\uk-pdeqtfs01\E\SolaBI\Python\_Installer\](file:///\\uk-pdeqtfs01\E\SolaBI\Python_Installer\)

You can read the UserGuide.docx to understand how it works

Do not hesitate to contact me for more information and specific demo: [laurent.tupin@ihsmarkit.com](mailto:laurent.tupin@ihsmarkit.com)

## What can you do to add more PCF to it?

You need to contact me as I would be able to help you. But there is some coding you can do to make it very fast.

I will try in this part to show you as clearly as possible what you can provide me to implement a new PCF.

## Parameters File

First there is some files you need to download as input. You can parameter that in a CSV file as follow:

Name:

* Seita\_Param.csv

Columns:

* **PCF**
  + Name of your PCF
  + **Examples:**
    - EXGM
    - SAMSUNG
    - VANECK\_IFRA
    - EASY\_COMO
* **FileDownloadMode**
  + Type of download
  + **Examples:**
    - OUTLOOK
    - FTP
    - SFTP\_Paramiko
    - FOLDER
* **ID**
  + Find an ID that would be unique
  + You will need to re-use it on the code as follow in the code:
    - df\_EXNAV = dic\_df ['**OUTLOOKEXNAV**']
  + The file you download in csv / txt will be used as a dataframe (you need to know about pandas) to create the PCF
* **URL**
  + **Used Only if FileDownloadMode = HTML\_...**
  + It is a URL that include an array we can save in a csv to re-use it later
  + We can use the string: {fileDate} to be replace by the value in the column **fileDate**
* **url\_keyword**
  + It is used to find the array within the webpage by its name
* **FTP\_server**
  + **Used only if FileDownloadMode IN (FTP, SFTP\_Paramiko)**
  + Download File from an FTP
  + **Examples:**
    - data.ftse.com
    - edx.standardandpoors.com
* **FTP\_uid**
  + UID of the FTP
* **FTP\_pwd**
  + Password of the FTP
* **FTP\_directory**
  + Path of the file we want to look for
  + **Examples:**
    - /data/goldmines/Open\_Constituents
    - /data/custom/fcnacl2v/history
    - /data/Developed\_Core\_Infrastructure\_50\_50/Open\_Constituent
* **FTP\_exactName**
  + In case of Seita does not find the file, we will search for the file with a date d-1, then d-2 by default
  + If you input TRUE in the column, it will ***not*** search a d-1 or d-2, just the exact name
* **outlook\_Acct**
  + **Used only if FileDownloadMode = OUTLOOK**
  + It is the account where you are receiving the mail
  + **Examples:**
    - sola@ihsmarkit.com
* **outlook\_mailBox**
  + First folder, I always use ‘Inbox’, if you have another case, let me know
* **outlook\_folder**
  + Folder in Inbox where the mails would be
  + You may need to have a rule on Outlook to move the mails in that folder
* **outlook\_folderToArchive**
  + By default, the option of moving the mail after the treatment is deactivated
  + If it is activated, the mail will be move to the folder you would fill in that cell
* **outlook\_subject**
  + The App will search the mails with the subject that include the string you will put in that cell
  + By default, we will choose the last mail in that list of mails
* **outlook\_fileType**
  + We will short the previous list by checking the name of the enclosed file that would include what you fill in that cell
  + By default, we will choose the last mail in that list of mails
* **outlook\_To**
  + I do not fill this, except to put ‘empty’ to discriminate
  + But you can discriminate the list of mails this way
* **outlook\_Cc**
  + Same with CC
* **outlook\_fileNameCheck**
  + We can shorten the list even more by searching the exact name of the enclosed file, date included ‘fileName’
  + By default, it is FALSE, and it is useful when the time is in the file name for example
  + I will be renamed by the ‘fileName’ in that case
* **filename**
  + Is common to all kind of download
  + It is possible to include {fileDate} to be replaced by the date
* **fileDate**
  + Format of the Date to replace in the cell ‘fileName’ or URL
  + It is the format as it is for Python
* **DateOffset**
  + It is possible to put an offset to the Date: D-1, D-2…
* **Dir\_Source**
  + **Used only if FileDownloadMode = FOLDER**
  + It is a simple Copy Paste from Dir\_Source to Dir\_Dest
* **Dir\_Dest**
  + Folder where the file will be saved
  + The folder can include a date in the format of ‘folderDate’
* **folderDate**
  + Format of the Date to replace in the cell ‘DirDest’

## Produce Output by Python

I will treat this part by the example of EXGM, I will display the code and comment it:

## Function Call



* str\_floderRoot:
  + It is the folder root defined in the application:
    - By default [\\uk-pdeqtfs01\E\SolaBI\Python\_Installer\](file:///\\uk-pdeqtfs01\E\SolaBI\Python_Installer\)
* dte\_date:
  + Date defined in the application: Today by default
* bl\_ArchiveMails
  + TRUE / FALSE: False by default
* bl\_devEnv
  + TRUE / FALSE: False by default

## Download Files

# Download Files

try:

str\_resultFigures, dic\_df = dwl.fDic\_downloadFiles(str\_floderRoot, dte\_date, bl\_ArchiveMails, bl\_devEnv, 'EXGM', True, False)

except: return 'ERROR: EXGM - Download files', []

# 0. Dataframe

try:

df\_EXGM = dic\_df['OUTLOOKEXGM']

df\_EXNAV = dic\_df['OUTLOOKEXNAV']

df\_GMO = dic\_df['FTP']

str\_folder = dic\_df['Folder']

except: return 'ERROR: EXGM - 0. Dataframe', []

For EXGM, I defined 3 rows in the parameters file with the ID: **OUTLOOKEXGM**, **OUTLOOKEXNAV**, **FTP**.

These files will be downloaded and will be stored in the dataframe: **df\_EXGM**, **df\_EXNAV**, **df\_GMO** as arrays.

**str\_folder** will be the folder where the PCF will be produced.

## Define Name of PCF

# Name of the PCF

l\_pcfFileName = ['PCF\_EXGM.' + dte\_date.strftime('%Y%m%d') + '.csv']

At your convenience…

## SQL request

# Get SQL

try:

dte\_dm1 = (dte\_date-BDay(1)).strftime("%Y%m%d")

str\_sqlFolder = str\_floderRoot + 'SQL\_Req\\'

str\_sqlFileName = 'fx\_' + str(dte\_dm1) + '.csv'

try:

df\_FX = pd.read\_csv(str\_sqlFolder + str\_sqlFileName)

except:

df\_FX = pp.fDf\_sqlFx(["FXRateSetID = '1'", "FromCurrencyCode = 'USD'", "AsAtDate = '" + str(dte\_dm1) + "'"])

pp.fL\_CreateTxtFile\_RetunPath(str\_sqlFolder, [str\_sqlFileName], df\_FX, True)

df\_mapping = pp.fDf\_sqlMapping('vwSecurityListing', 'Isin, Sedol, Ric, SecurityName, CurrencyCode',

['IsPrimaryListing=1', 'Isin IS NOT NULL', 'Sedol IS NOT NULL', 'Ric IS NOT NULL'

, "Isin IN ('" + "','".join(df\_GMO['ISIN']) + "')"], 'CurrencyCode')

except: return 'ERROR: EXGM - SQL request', []

Two requests:

* Mapping: You can read the request as follow

SELECT Isin, Sedol, Ric, SecurityName, CurrencyCode

FROM vwSecurityListing

WHERE IsPrimaryListing=1

AND Isin IS NOT NULL

AND Sedol IS NOT NULL

AND Ric IS NOT NULL

AND Isin IN (\*\*list defined in the dataframe df\_GMO\*\*)

ORDER BY CurrencyCode

\*\*The result of this request will be stored in the dataframe : ***df\_mapping***

* Forex request

Will be stored in the dataframe : ***df\_FX***

But as it is used for everything, for performance purpose, I saved it as a csv to be requested only once.

If you do not have any use for that, you can just use the ***fDf\_sqlMapping*** function to make any request.

str\_server\_default = '10.233.6.147'

str\_database\_default = 'SolaDBServer'

str\_uid\_default = 'pcfReporting'

str\_pwd\_default = '\*\*\*\*'

If you want to change these db connexion ID, please contact: [laurent.tupin@ihsmarkit.com](mailto:laurent.tupin@ihsmarkit.com)

## Define Variables

# 1. Variables

try:

flt\_mgtFees = 0.006

int\_nbDay = int((dte\_date.date() - (dte\_date-BDay(1)).date()).days)

flt\_numberSharePerUnit = 500000

flt\_numberShareFund = df\_EXNAV[df\_EXNAV['Share Class ID'] =='GM1HK']['Units Outstanding'].values[0]

flt\_NavPerShare = df\_EXNAV[df\_EXNAV['Share Class ID'] =='GM1HK']['Accounting NAV per Share'].values[0]

flt\_sumIf\_mktValyeVCY\_Equity = df\_EXGM[df\_EXGM['Position Type'] =='EQUITY']['Accounting Market Value (VCY)'].sum()

flt\_unequitizedCashPerUnit = flt\_numberSharePerUnit \* (flt\_NavPerShare - flt\_sumIf\_mktValyeVCY\_Equity/flt\_numberShareFund)

flt\_estimatedCachPerunit = round(flt\_unequitizedCashPerUnit - (flt\_NavPerShare \* flt\_numberSharePerUnit \* flt\_mgtFees \* int\_nbDay / 365),4)

except: return 'ERROR: EXGM - 1. Variables', []

You can define Constant like Management fees, number of days between 2 working day (=3 if we are on Monday), NAV depending of one of the DataFrame…

## Reshape Data frame

# 2. Df

try:

df\_mapping.rename(index = str, columns = {'Isin':'ISIN'}, inplace = True)

df\_result = df\_GMO.merge(df\_mapping, on='ISIN')

df\_result = df\_result[['SecurityName','CurrencyCode','ISIN', 'Sedol', 'Ric', 'Price',

'% wght in FTSE Gold Mines Index']].sort\_values('ISIN', ascending = True)

df\_result.rename(index = str, columns = {'% wght in FTSE Gold Mines Index':'weight\_FTSE'}, inplace = True)

except: return 'ERROR: EXGM - 2. Df -- A', []

try:

df\_result.rename(index = str, columns = {'CurrencyCode':'ccy'}, inplace = True)

df\_result.loc[df\_result['ccy'] == 'ZAC', 'Price'] = df\_result['Price'] \* 100

df\_result.loc[df\_result['ccy'] == 'GBP', 'Price'] = df\_result['Price'] \* 100

df\_result.reset\_index(drop = True, inplace = True)

df\_result['ind'] = df\_result.index + 1

except: return 'ERROR: EXGM - 2. Df -- B', []

try:

df\_FX['flt\_hkdccy'] = df\_FX['Value'] / df\_FX[df\_FX['ToCurrencyCode'] =='HKD']['Value'].values[0]

df\_FX = df\_FX[['ToCurrencyCode','flt\_hkdccy']]

df\_FX.rename(index = str, columns = {'ToCurrencyCode':'ccy'}, inplace = True)

df\_result = df\_result.merge(df\_FX, on='ccy').sort\_values('ind', ascending = True)

df\_result['weight\_FTSE2'] = pd.to\_numeric(df\_result['weight\_FTSE'].str.replace('%','')) / 100

df\_result['Shares'] = round (((flt\_NavPerShare\*flt\_numberSharePerUnit - flt\_estimatedCachPerunit)

\* df\_result['weight\_FTSE2'] \* df\_result['flt\_hkdccy'] / df\_result['Price']) + epsilon, 0)

except: return 'ERROR: EXGM - 2. Df -- C', []

try:

df\_result['Shares'] = df\_result['Shares'].astype(int)

df\_result['Price'] = round(df\_result['Price'], 4)

df\_result['Price'] = df\_result['Price'].astype(int).astype(str).where(

df\_result['Price'].astype(str) == df\_result['Price'].astype(int).astype(str) + '.0'

, df\_result['Price'])

df\_result = df\_result[['ind','SecurityName','ccy', 'Price', 'Shares', 'ISIN', 'Sedol', 'Ric']]

except: return 'ERROR: EXGM - 2. Df -- D', []

You have different operations to produce a last PCF file from the input DataFrame:

* Rename column
* Merge 2 DataFrame and merge on ISIN
* Sort values
* Create new columns
* Multiply a column of value if the ccy column is ZAC or GBP
* Add a column of Index (1, 2, 3…)
* Create a column of Forex of CCY vs HKD
* Keep only some columns in a dataframe at the end

## Add a Header DataFrame

# 3. Add rows of title

try:

myCol = df\_result.columns

df\_header = pd.DataFrame(columns=myCol)

df\_header.loc[len(df\_header)] = ['','XIE Shares ETFs','','','','','','']

df\_header.loc[len(df\_header)] = ['','','','','','','','']

df\_header.loc[len(df\_header)] = ['','Indicative Creation / Redemption Basket Composition for Trade Date',

dte\_date.strftime('%d/%m/%Y'),'','','','','']

df\_header.loc[len(df\_header)] = ['','','','','','','','']

df\_header.loc[len(df\_header)] = ['','FUND INFORMATION','','','','','','']

df\_header.loc[len(df\_header)] = ['','Fund Name','XIE Shares FTSE Gold Miners ETF','','','','','']

df\_header.loc[len(df\_header)] = ['','ISIN','HK0000313210','','','','','']

df\_header.loc[len(df\_header)] = ['','BBG Ticker','3116 HK','','','','','']

df\_header.loc[len(df\_header)] = ['','Reuters Code','3116.HK','','','','','']

df\_header.loc[len(df\_header)] = ['','Fund Currency','HKD','','','','','']

df\_header.loc[len(df\_header)] = ['','NAV per Share',round(flt\_NavPerShare, 4),'','','','','']

df\_header.loc[len(df\_header)] = ['','Number of Fund Shares in Issue',flt\_numberShareFund,'','','','','']

df\_header.loc[len(df\_header)] = ['','Total NAV of Fund per Creation / Redemption Basket',

int(flt\_NavPerShare \* flt\_numberSharePerUnit),'','','','','']

df\_header.loc[len(df\_header)] = ['','Actual Cash per Creation / Redemption Basket',round(flt\_unequitizedCashPerUnit,2),'','','','','']

df\_header.loc[len(df\_header)] = ['','Estimated Cash per Creation / Redemption Basket', round(flt\_estimatedCachPerunit, 2),'','','','','']

df\_header.loc[len(df\_header)] = ['','Number of Fund Shares per Creation / Redemption Basket',flt\_numberSharePerUnit,'','','','','']

df\_header.loc[len(df\_header)] = ['','','','','','','','']

df\_header.loc[len(df\_header)] = ['','Company Name','Currency','Price','Shares','ISIN','SEDOL','RIC']

except: return 'ERROR: EXGM - 3. Add rows of title', []

Very simple way of creating rows of value at the beginning of the DataFrame

# 4. FINAL OUTPUT

try:

df\_result = df\_header.append(df\_result)

except: return 'ERROR: EXGM - 4. FINAL OUTPUT', []

And merge them to the dataframe

## Create the PCF file from the DataFrame

# 5.Create Pcf

try:

l\_pathAttach = pp.fL\_CreateTxtFile\_RetunPath(str\_folder, l\_pcfFileName, df\_result)

if not l\_pathAttach: return 'ERROR: Could not create the PCF EXGM', []

l\_pathAttach = [path.replace(str\_floderRoot, '') for path in l\_pathAttach]

except: return 'ERROR: EXGM - 5. Create Pcf', []

This function will take the DataFrame: df\_result

Will create the file define above: l\_pcfFileName

And save it in the folder: str\_folder

## Display a return message in the App

If you want to have some main figures to be displayed in the App, it is possible by doing so:

# 6. Build the return message

try:

str\_resultFigures += '\n' + str(round(flt\_NavPerShare, 4)) + ' NAV per Share'

str\_resultFigures += '\n' + str(flt\_numberShareFund) + ' Number of Fund Shares in Issue'

str\_resultFigures += '\n' + str(int(flt\_NavPerShare \* flt\_numberSharePerUnit)) + ' Total NAV of Fund per Creation / Redemption Basket'

str\_resultFigures += '\n' + str(round(flt\_unequitizedCashPerUnit, 2)) + ' Actual Cash per Creation / Redemption Basket'

str\_resultFigures += '\n' + str(round(flt\_estimatedCachPerunit, 2)) + ' Estimated Cash per Creation / Redemption Basket'

str\_resultFigures += '\n' + str(flt\_numberSharePerUnit) + ' Number of Fund Shares per Creation / Redemption Basket'

str\_resultFigures += '\n' + '------------------------------------------------'

## Return of the function

return str\_resultFigures, l\_pathAttach

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Is that all needed to be done?

Yes… Almost.

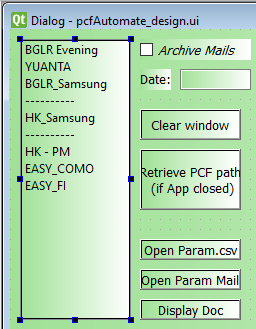
I will need to add some constant to integrate this code to the global App, but that would be very quick.

Also, I need to add your PCF to the list of PCF available in the App designer

At the end, I will need to deliver the solution and you will be able to use it from the path:

[\\uk-pdeqtfs01\E\SolaBI\Python\_Installer \](file:///\\uk-pdeqtfs01\E\SolaBI\Python_Installer%20\)

# ANNEXES - Other Example: BGLR\_SAMSUNG (UK)



### Add the perimeter in the .UI file

You need to open the file “pcfAutomate\_design.ui” with designer.exe

*I normally take care of that part for Synergy matters and so you can no install design.exe.*

I add the field: BGLR\_Samsung

### Download the Input files

The files are from a ZIP downloaded online with a token code. In that context, it does not seem possible to automate the downloading of these files.

The only thing we can automatize is the extraction of the files, so the users only have to Download the Zip anywhere if they update the Dir of the ZIP in the CSV file ***Seita\_Param***.csv

It is fillable this way:

|  |  |
| --- | --- |
| PCF | BGLR\_Samsung |
| FileDownloadMode | ZIP |
| ID | ZIP\_SHLP00845164\_SR0490XS *(ANY ID you choose,* ***must be unique in the CSV****)* |
| fileName | SHLP00845164\_SR0490XS\_{fileDate}\_1\_{XXXXXX}.CSV *({X} can be replaced by any character (File Name include Time)* |
| fileDate | %Y%m%d |
| DateOffset | 0 |
| Dir\_Source | \\uk-pdeqtfs01\E\Data\Lucerne\Data\SOLA PCF\BGLR\_Samsung\Input\HSBC\_BULKREPORTS\_{folderDate}{XXXX}.zip |
| Dir\_Dest | \\uk-pdeqtfs01\E\Data\Lucerne\Data\SOLA PCF\BGLR\_Samsung\Input |
| folderDate | %d%m%Y |
| folderDateOffset | 0 |

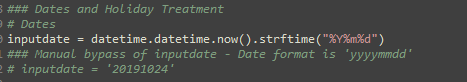
The process is extracting the files in the ZIP which path is defined in Dir\_Source to the folder defined in Dir\_Dest.

In this case folderDate and folderDateOffset are useless because there is no date in Dir\_Dest.

### Point on Initial code



Is ***no longer*** needed. The App will download the file, and I will explain to you how to ‘put’ these CSV into DataFrame



Date is also ***not*** needed.

The date is the first argument the user can see in the App.

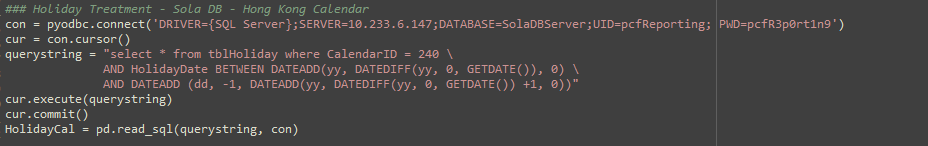
### SQL request

It is also possible to store request into CSV file, as if you were downloading a file.

Main reason for that is low performance of SQL, so if the user wishes to generate the PCF several time, only the first time will be SQL request, the rest will be taking the data from CSV. Also, it is empowering user so they can fill the SQL request in the ***Seita\_Param***.csv file as follow:

|  |  |
| --- | --- |
| PCF | BGLR\_Samsung |
| FileDownloadMode | SQL |
| ID | sql\_Calendar |
| sqlReq | SELECT \* FROM tblHoliday WHERE CalendarID = 240 AND HolidayDate BETWEEN… |
| sqlDateFormat | %Y%m%d *(Useless here but put by default)* |
| sqlDateOffset | 0 *(Useless here but put by default)* |
| fileName | Calendar\_sql |
| fileDate | %Y%m%d *(Useless here but put by default)* |
| DateOffset | 0 *(Useless here but put by default)* |
| Dir\_Dest | \\uk-pdeqtfs01\E\Data\Lucerne\Data\SOLA PCF\BGLR\_Samsung\Input |
| folderDate | %d%m%Y *(Useless here but put by default)* |
| folderDateOffset | 0 *(Useless here but put by default)* |

### Point on Initial code



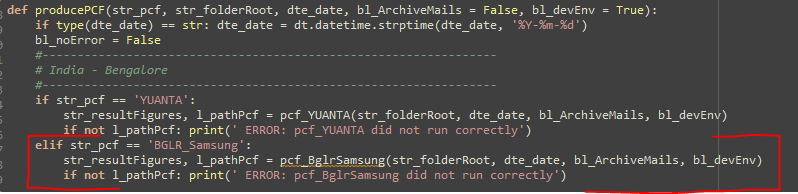
Is ***no longer*** needed. The App will download the file, and I will explain to you how to ‘put’ these CSV into DataFrame (HolidayCal in the present example)

### Final: NEW code

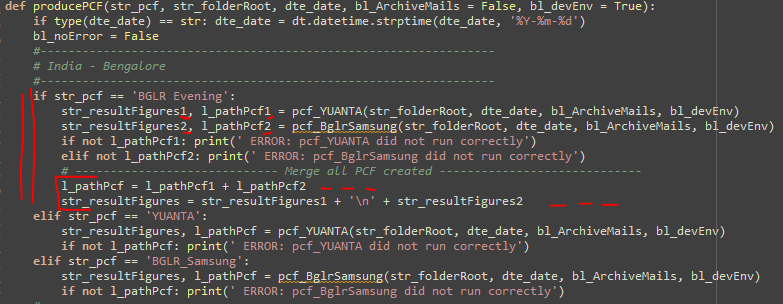
As I said before, most of things are automatized with the App, but you still need to create a dedicated function to create the PCF.

**First**:

Add a condition in the function ***producePCF*** (file: **PCF\_creater.py**)

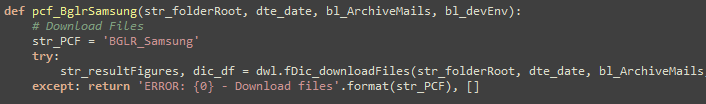


If the PCF belongs to a list of PCF, we need to add the following code (Let’s put YAUNTA and BGLR\_Samsung into 1 category: Bengalore Evening):



**Second**:

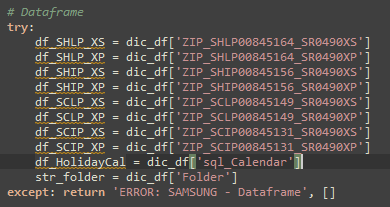
Create the function ***pcf\_BglrSamsung*** (file: **PCF\_creater.py**)



This simple code will find the Input file. If the files are not found, they will be downloaded.

These files will be stored into a dictionary of DataFrame, the ID defined in the ***Seita\_Param***.csv file.

You will get them back into customized DataFrame as follow:



**Third**:

You can now copy the process already written (more or less):

*Except, I replaced the name of Variables to have more consistency with the rest of the file…*

**Last**:

I made some change to adapt the code and be able to send the PCF at the end of the process:

I put all the path of the PCF produced in output in the list: ***l\_pathAttach***

For this list to be return by the function ***producePCF***

And be displayed in the APP.