Table of Contents

[Section 1: Flow chart of the exclusion transformation process 1](#_Toc58586315)

[Section 2: Data Lineage Diagram 1](#_Toc58586316)

[Section 3: Export analysis G 2](#_Toc58586317)

[Section 4: EXPORT\_analysis\_RAG 3](#_Toc58586318)

[Section 5: ERROR LOG FILES 6](#_Toc58586319)

[Section 6: EXPORT ANALYSIS RAG ACTION 7](#_Toc58586320)

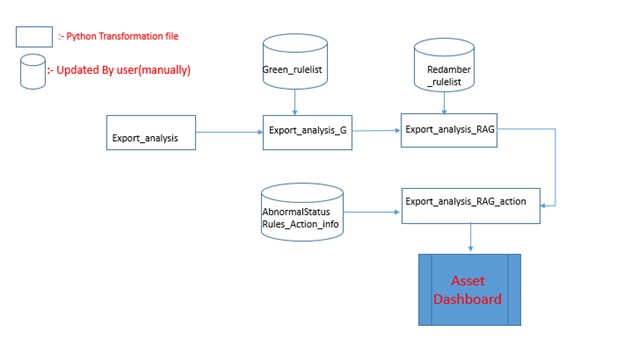
[AP 1: export\_analysis\_RAG.py 1](#_Toc58586321)

[AP 2: export\_analysis\_RAG\_actions.py 4](#_Toc58586322)

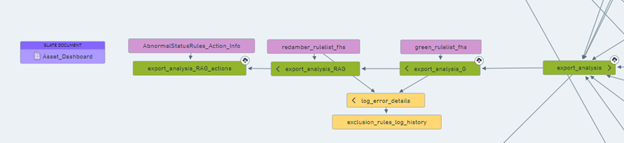
[AP 3: Computation\_logs.py 6](#_Toc58586323)

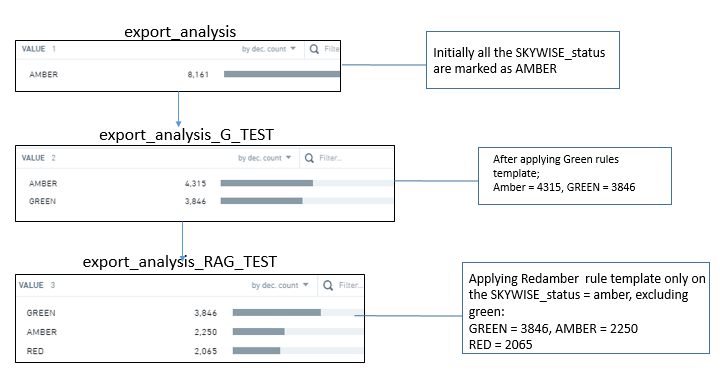
[AP 4:incremental\_log\_computation.py 10](#_Toc58586324)

## Section 1: Flow chart of the exclusion transformation process



## Section 2: Data Lineage Diagram





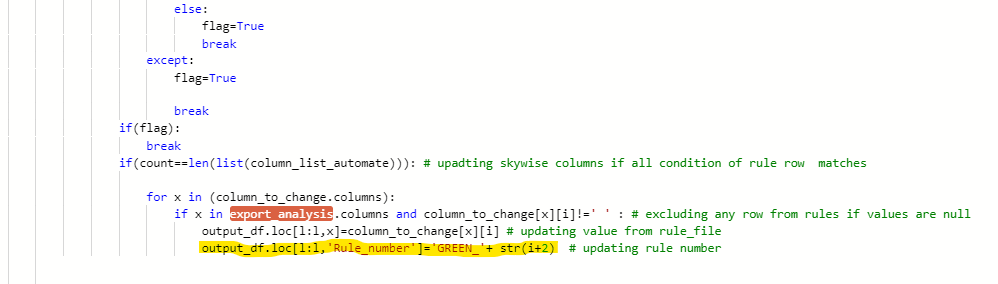
## Section 3: Export analysis G

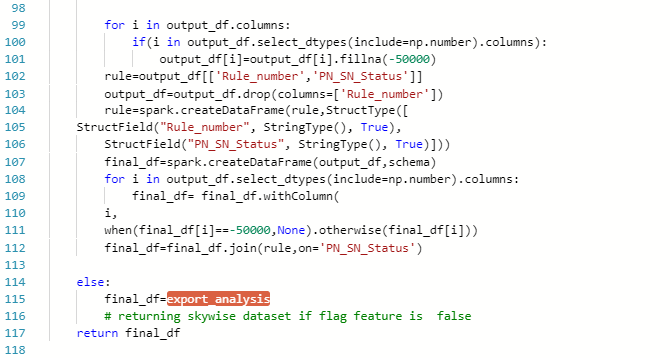
In this transformation, no logic is modified.

Only rule\_number is modified to a string type, earlier it contains integer type as we are just printing the rule number.

Now, with the pipeline containing rules for redamber and green unlike before, thus, there was a need to uniquely identify the error. Therefore, now a prefix named Green\_’%rule\_no’ is there, to differentiate between redamber and green.

Following change in the original code:





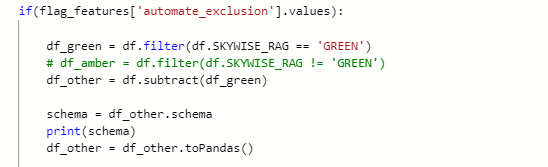
## Section 4: EXPORT\_analysis\_RAG

Primary Focus:

1. Here, the focus was to dynamically update the entire record present in the export\_analysis\_G based on the rules present in the redamber rules template.
2. The concerned columns which are being parsed from the template and compared with the data in export\_analysis\_G:

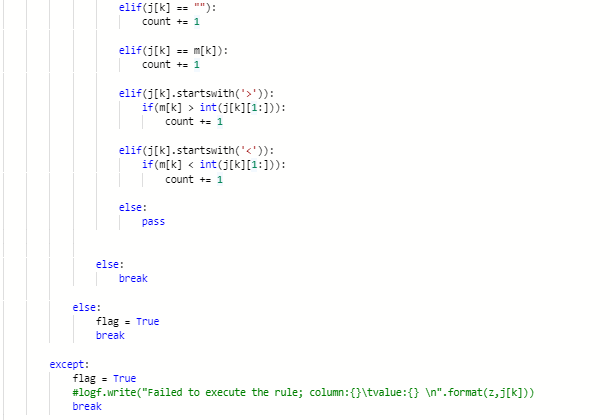
column\_list\_automate=['company','amasis\_status\_code','SKYWISE\_status\_code','owner\_code','storage\_location\_code','internal\_location\_code','computed\_number\_of\_days\_in\_situation', 'SKYWISE\_RAG']

1. Before iterating through the data frame and applying the algorithm, I subtract all the records with green status from the export\_analysis\_G. In this way, the algorithm can only be applied on the records which do not contain green status and have the default ruleno = ‘-1’.



1. The step before starting the iterating process is to change the pyspark dataframe into pandas dataframe. In this transformation file, only export\_analysis\_G needs to be converted into pandas data frame.

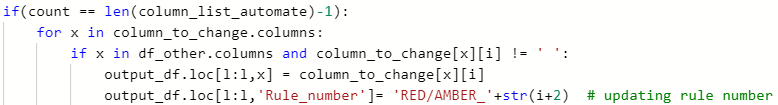




1. After applying the the algorithm, the records update accordingly

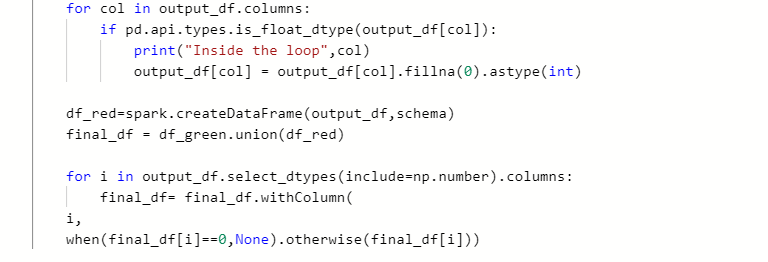
When: 

It basically means all the column data is validated and does not contain any null or any invalid values, and the code can now update the record.



Else it will throw an error, which I handled through a pipeline running from a repository named ‘ERROR\_LOG\_REPO’.

1. Also, I simultaneously updated the rule\_number column with a prefix ‘RED/AMBER\_’ which is mentioned in the above code snippet.
2. Finally, I performed a few type conversions which include handling na values and joined the redamber transformed with green transformed records.

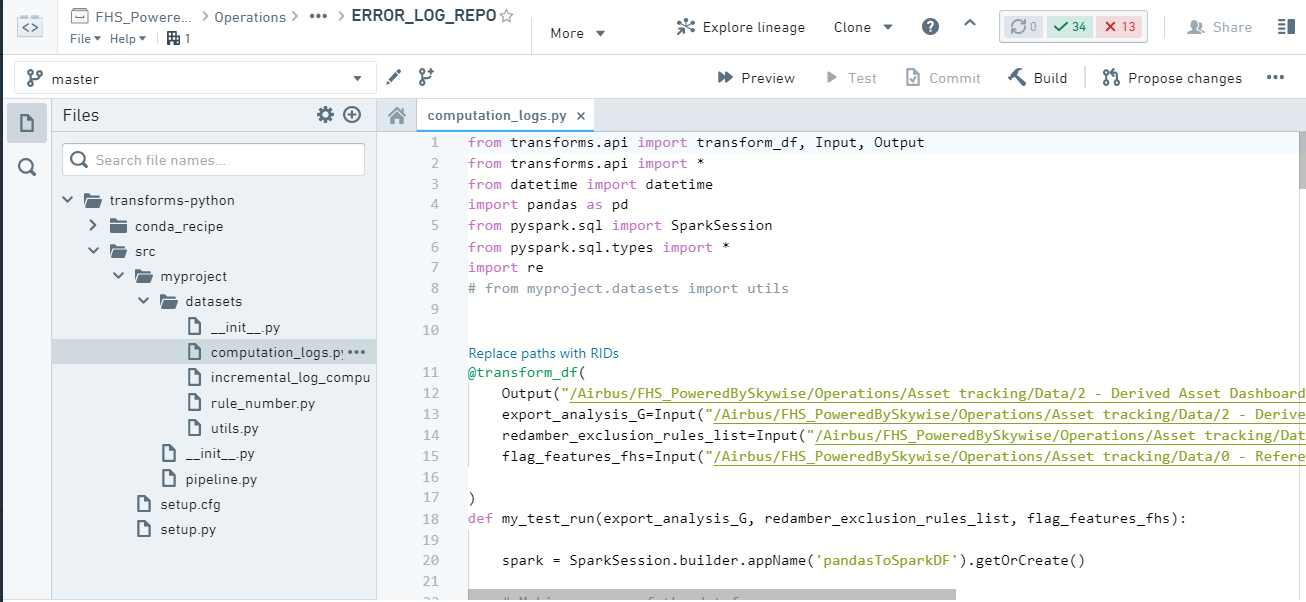


1. The entire **‘export\_analysis\_RAG.py’** is in the **appendix**.

## Section 5: ERROR LOG FILES

To catch the error logs, I developed a pipeline which will generate a log\_error\_details file.

Environment where the code is executed and the pipelined builded:



How does it work ?

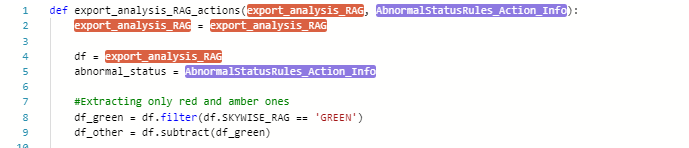
Whenever, user enters a null value, space or any special characters other than which are being handled in the code, that rule(record) will not execute and the algorithm will capture the rule number along with the respective column where the incorrect value is entered.

* The file which is being executed to capture logs is ‘**computed\_logs.py’** and it generates an output file named **‘log\_error\_details’.**
* There is, moreover, another log file which is capturing all the error logs with its history is ‘**exclusion\_rules\_log\_history\_prod’** and it is being handled by **‘incremental\_log\_computation.py’**
* All the codes are present in the appendix section.

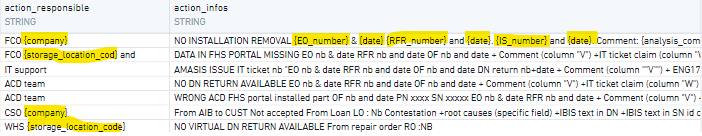
## Section 6: EXPORT ANALYSIS RAG ACTION

Transformation involved:

* Here, in this transformation there are two major tasks. The first one is to update records based on a template where the user will populate the corresponding columns.
* On the basis of ‘SKYWISE\_status\_code’, we are updating these following columns(target): ['action', 'action\_holder', 'action\_accountable', 'action\_responsible', 'action\_infos']. The data of the target columns are being parsed from the input file named **AbnormalStatusRules\_Action\_Info.**
* Structure of the function and the input files:

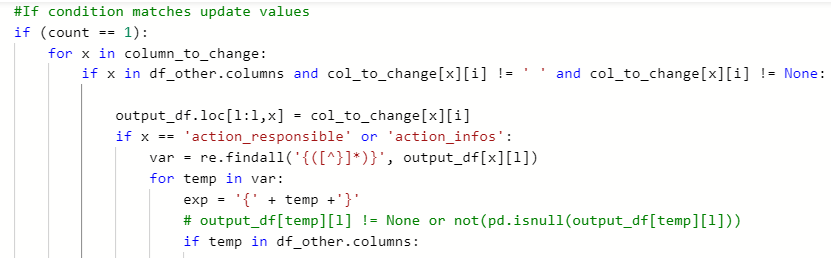


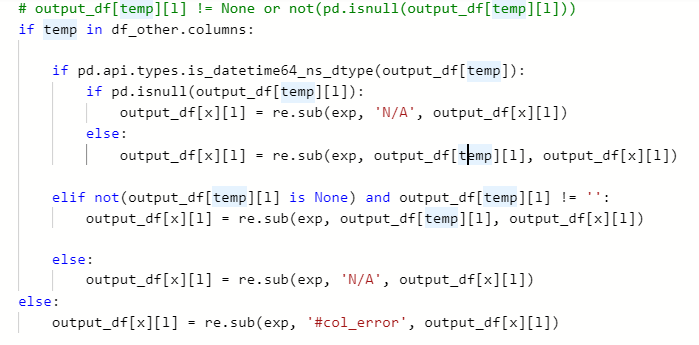
* The second task is to dynamically populate certain mentioned fields in ‘action\_responsible’ and ‘action\_info’ cells such as:



The cells which contain curly braces will be parsed by the algorithm, and updated accordingly. Also, most of the iterating process would be quite similar to what is being done in the previous transformations.

Below is the algorithm for this specific transformation:





## AP 1: export\_analysis\_RAG.py

def export\_analysis\_RAG(export\_analysis\_G, redamber\_rulelist\_fhs, flag\_features\_fhs):

exlusion\_rule\_ls\_red\_Sheet1 = redamber\_rulelist\_fhs

exclusionauto\_fhs\_1\_TEST = export\_analysis\_G

import numpy as np

# from pyspark.sql.types import IntegerType()

automate = exlusion\_rule\_ls\_red\_Sheet1

df = exclusionauto\_fhs\_1\_TEST

# df = df.withColumn("Rule\_number",df["Rule\_number"].cast(StringType()))

flag\_features = flag\_features\_fhs

#logf = open("exclusion\_rules\_error.log", "w")

if(flag\_features['automate\_exclusion'].values):

df\_green = df.filter(df.SKYWISE\_RAG == 'GREEN')

# df\_amber = df.filter(df.SKYWISE\_RAG != 'GREEN')

df\_other = df.subtract(df\_green)

schema = df\_other.schema

print(schema)

df\_other = df\_other.toPandas()

column\_list\_automate=['company','amasis\_status\_code','SKYWISE\_status\_code','owner\_code','storage\_location\_code','internal\_location\_code','computed\_number\_of\_days\_in\_situation', 'SKYWISE\_RAG']

output\_df = df\_other.copy()

#output\_df = df\_other.copy()

#automate = automate[column\_list\_automate[:-1]]

subset = df\_other[column\_list\_automate[:-1]]

# print(automate.columns, '\n')

column\_to\_change=automate[['SKYWISE\_RAG']]

"""

print("data types of automate",'\n')

print(automate.dtypes)

print("data types of subset",'\n')

print(subset.dtypes)

"""

for i, j in automate.iterrows():

flag = False

#print("inside automate"," ",i)

for l,m in subset.iterrows():

count = 0

#print("inside subset"," ",l)

for k,z in enumerate(column\_list\_automate[:-1]):

#or (not(re.search(r'^[><]\d+$', j[k])) and z=='computed\_number\_of\_days\_in\_situation')

try:

if not(j[k]==None or (j[k].isdigit() and z=='computed\_number\_of\_days\_in\_situation')):

if(isinstance(j[k], str)):

if(j[k] == '\*'):

count += 1

elif(j[k].startswith('\*') and len(j[k])>1):

if(m[k].endswith(j[k][1:])):

count+=1

elif(j[k].endswith('\*') and len(j[k])>1):

if(m[k].startswith(j[k][0:-1])):

count+=1

elif(re.search(r'\bSI\*\b', j[k])):

if(m[k].startswith(j[k][0:2])):

count += 1

elif(j[k] == ""):

count += 1

elif(j[k] == m[k]):

count += 1

elif(j[k].startswith('>')):

if(m[k] > int(j[k][1:])):

count += 1

elif(j[k].startswith('<')):

if(m[k] < int(j[k][1:])):

count += 1

else:

pass

else:

break

else:

flag = True

break

except:

flag = True

#logf.write("Failed to execute the rule; column:{}\tvalue:{} \n".format(z,j[k]))

break

if(flag):

break

if(count == len(column\_list\_automate)-1):

for x in column\_to\_change.columns:

if x in df\_other.columns and column\_to\_change[x][i] != ' ':

output\_df.loc[l:l,x] = column\_to\_change[x][i]

output\_df.loc[l:l,'Rule\_number']= 'RED/AMBER\_'+str(i+2) # updating rule number

for col in output\_df.columns:

if pd.api.types.is\_float\_dtype(output\_df[col]):

print("Inside the loop",col)

output\_df[col] = output\_df[col].fillna(0).astype(int)

df\_red=spark.createDataFrame(output\_df,schema)

final\_df = df\_green.union(df\_red)

for i in output\_df.select\_dtypes(include=np.number).columns:

final\_df= final\_df.withColumn(

i,

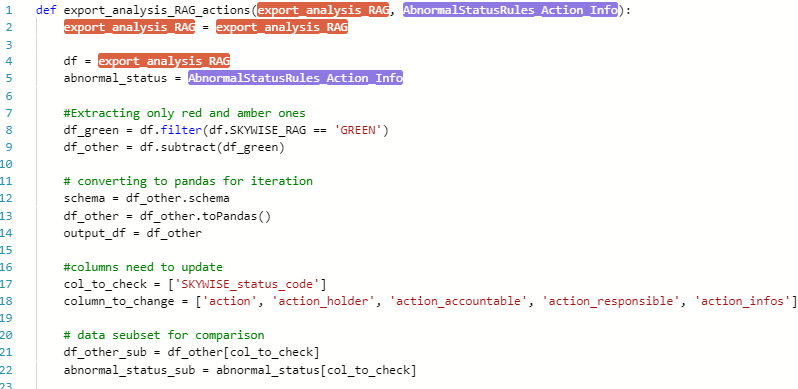
when(final\_df[i]==0,None).otherwise(final\_df[i]))

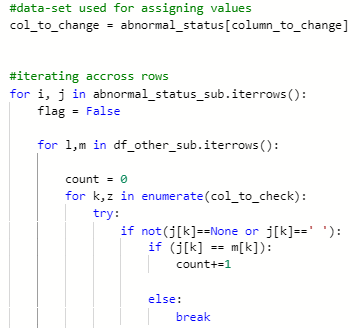
else:

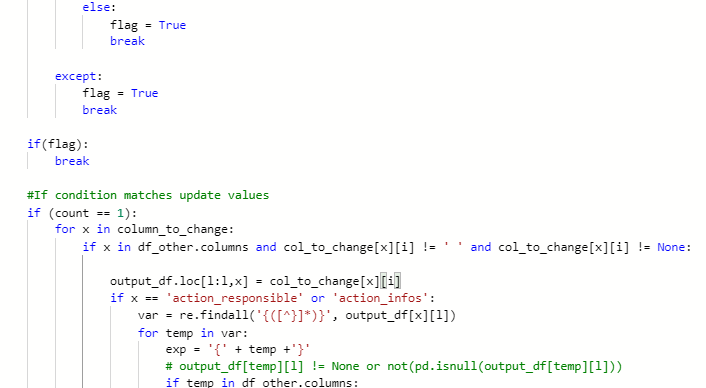
final\_df = df

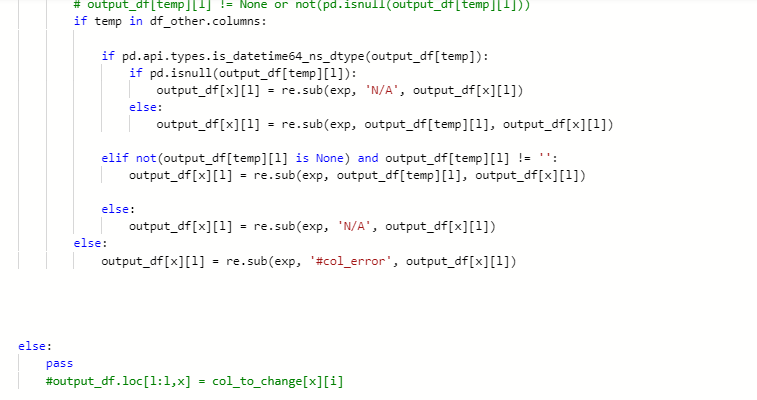
return final\_df

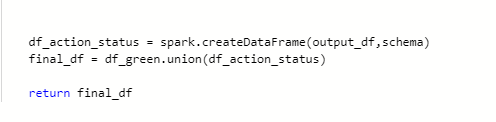
## AP 2: export\_analysis\_RAG\_actions.py











## AP 3: Computation\_logs.py

from transforms.api import transform\_df, Input, Output

from transforms.api import \*

from datetime import datetime

import pandas as pd

from pyspark.sql import SparkSession

from pyspark.sql.types import \*

import re

# from myproject.datasets import utils

@transform\_df(

Output("/Airbus/FHS\_PoweredBySkywise/Operations/Asset tracking/Data/2 - Derived Asset Dashboard/log\_error\_details"),

export\_analysis\_G=Input("/Airbus/FHS\_PoweredBySkywise/Operations/Asset tracking/Data/2 - Derived Asset Dashboard/export\_analysis\_G"),

redamber\_exclusion\_rules\_list=Input("/Airbus/FHS\_PoweredBySkywise/Operations/Asset tracking/Data/0 - Reference/Rule List Asset Dashboard/redamber\_rulelist\_fhs"),

flag\_features\_fhs=Input("/Airbus/FHS\_PoweredBySkywise/Operations/Asset tracking/Data/0 - Reference/Rule List Asset Dashboard/flag\_features\_fhs")

)

def my\_test\_run(export\_analysis\_G, redamber\_exclusion\_rules\_list, flag\_features\_fhs):

spark = SparkSession.builder.appName('pandasToSparkDF').getOrCreate()

# Making a copy of the dataframe

automate = redamber\_exclusion\_rules\_list.toPandas()

df = export\_analysis\_G

flag\_features\_sheet1 = flag\_features\_fhs

flag\_features\_sheet1 = flag\_features\_sheet1.toPandas()

# Variable declaration

rule\_number\_errored\_out = []

error\_type = []

no\_of\_rows\_updated = 0

row\_list = []

if(flag\_features\_sheet1['automate\_exclusion'].values):

df\_green = df.filter(df.SKYWISE\_RAG == 'GREEN')

# df\_amber = df.filter(df.SKYWISE\_RAG != 'GREEN')

df\_other = df.subtract(df\_green)

schema = df\_other.schema

df\_other = df\_other.toPandas()

column\_list\_automate = ['company', 'amasis\_status\_code', 'SKYWISE\_status\_code', 'owner\_code', 'storage\_location\_code', 'internal\_location\_code',

'computed\_number\_of\_days\_in\_situation', 'SKYWISE\_RAG']

output\_df = df\_other.copy()

subset = df\_other[column\_list\_automate[:-1]]

# print(automate.columns, '\n')

column\_to\_change=automate[['SKYWISE\_RAG']]

for i, j in automate.iterrows():

flag = False

#print("inside automate"," ",i)

for l,m in subset.iterrows():

count = 0

#print("inside subset"," ",l)

for k,z in enumerate(column\_list\_automate[:-1]):

# or (not(re.search(r'^[><]\d+$', j[k])) and z=='computed\_number\_of\_days\_in\_situation')

#print('k: {} ; z: {}'.format(k,z))

try:

if not(j[k]==None or (j[k].isdigit() and z=='computed\_number\_of\_days\_in\_situation')):

if (isinstance(j[k], str)):

if(j[k] == '\*'):

count += 1

elif(j[k].startswith('\*') and len(j[k])>1):

if(m[k].endswith(j[k][1:])):

count+=1

elif(j[k].endswith('\*') and len(j[k])>1):

if(m[k].startswith(j[k][0:-1])):

count+=1

elif(re.search(r'\bSI\*\b', j[k])):

if(m[k].startswith(j[k][0:2])):

count += 1

elif(j[k] == ""):

count += 1

elif(j[k] == m[k]):

count += 1

elif(j[k].startswith('>')):

if(m[k] > int(j[k][1:])):

count += 1

elif(j[k].startswith('<')):

if(m[k] < int(j[k][1:])):

count += 1

else:

pass

else:

break

else:

flag = True

if j[k] == None:

val = str(z)

error\_type.append(val)

elif j[k].isdigit():

val = str(z)

error\_type.append(val)

else:

val = str(z)

error\_type.append(val)

break

except:

flag = True

break

if(flag):

rule\_number\_errored\_out.append(i+1)

break

# this line will update the status column if each conditions are satisfied

if(count == len(column\_list\_automate)-1):

if (l not in row\_list):

no\_of\_rows\_updated += 1

row\_list.append(l)

dateTimeObj = datetime.now()

data = {'timestamp': dateTimeObj, 'total\_records': export\_analysis\_G.toPandas().shape[0], 'number\_of\_records\_updated': no\_of\_rows\_updated}

data = pd.DataFrame(data, index=[0])

if(rule\_number\_errored\_out!=[]):

s\_rule\_number\_error = []

count\_1 = 0

while count\_1 < len(rule\_number\_errored\_out):

x= 'Rule:'+str(rule\_number\_errored\_out[count\_1]) +' ;'+str(error\_type[count\_1])

s\_rule\_number\_error.append(x)

count\_1 += 1

rule\_number\_errored\_out = [str(i) for i in s\_rule\_number\_error]

data['rule\_number\_errored\_out'] = ' , '.join(rule\_number\_errored\_out)

else:

data['rule\_number\_errored\_out'] = ''

mySchema = StructType([

StructField('timestamp', TimestampType(), True),

StructField('total\_records', IntegerType(), True),

StructField('number\_of\_records\_updated', IntegerType(), True),

StructField('rule\_number\_errored\_out', StringType(), True)

])

data = spark.createDataFrame(data, schema = mySchema)

return data

## AP 4:incremental\_log\_computation.py

from transforms.api import transform, incremental, Input, Output

@incremental(snapshot\_inputs=['error\_input'], require\_incremental = True)

@transform(

out = Output("/Airbus/FHS\_PoweredBySkywise/Operations/Asset tracking/Data/2 - Derived Asset Dashboard/exclusion\_rules\_log\_history\_prod"),

error\_input = Input("/Airbus/FHS\_PoweredBySkywise/Operations/Asset tracking/Data/2 - Derived Asset Dashboard/log\_error\_details")

)

def incremental\_compute\_log(error\_input, out):

data=error\_input.dataframe('added')

out.write\_dataframe(data)