import sys  
import os  
import urllib3  
  
urllib3.disable\_warnings(urllib3.exceptions.InsecureRequestWarning)  
current\_path = os.path.dirname(os.path.abspath(\_\_file\_\_))  
print(current\_path)  
root\_path = os.path.abspath(os.path.join(current\_path,'..','..','..'))  
print(root\_path)  
sys.path.append(root\_path)  
  
import shutil  
from openpyxl.styles import Font, PatternFill  
from openpyxl import load\_workbook  
from datetime import datetime  
from collections import defaultdict  
import numpy as np  
# import dataBuilderUtils as utils  
import sys  
import os  
import pandas as pd  
import requests  
import random  
import xml.dom.minidom as md2  
import warnings  
import logging  
from Xcro.Utils.Escrow.PropertyRead import Property  
from Xcro.DataBuilder.New\_Deal import dataBuilderUtils as utils  
from Xcro.DataBuilder.New\_Deal.accClassBuilder import CreateAccounts  
  
logging.basicConfig(filename=Property.LOG\_FILE, level=logging.INFO,  
 format='%(asctime)s - %(name)s - %(levelname)s - %(message)s')  
logger = logging.getLogger(\_\_name\_\_)  
  
# Clear the logger  
file\_path = Property.LOG\_FILE  
try:  
 with open(file\_path, "w") as log\_file:  
 # Clear the content of the file  
 log\_file.truncate(0)  
 print(f"Log file '{file\_path}' has been cleared.")  
except FileNotFoundError:  
 print(f"Log file '{file\_path}' not found.")  
except Exception as e:  
 print(f"An error occurred while clearing the log file: {e}")  
  
# Cleaning up the files and the folders  
files\_to\_delete = [Property.DATABUILDER\_OUTPUT,f"{Property.ACCOUNTS\_CSV}{str(datetime.now().date())}.csv"]  
folders\_to\_delete = [Property.CUSTOMER\_XML\_FILES,Property.CASA\_XML\_FILES]  
  
for file in files\_to\_delete:  
 if os.path.exists(file):  
 os.remove(file)  
 print(f"Deleted file: {file}")  
 logger.info(f"Deleted file: {file}")  
 else:  
 print(f"File not found: {file}")  
 logger.error(f"File not found: {file}")  
  
# Delete folders  
for folder in folders\_to\_delete:  
 if os.path.exists(folder):  
 shutil.rmtree(folder)  
 print(f"Deleted folder: {folder}")  
 logger.info(f"Deleted folder: {folder}")  
 else:  
 print(f"Folder not found: {folder}")  
 logger.error(f"Folder not found : {folder}")  
  
warnings.filterwarnings("ignore")  
account\_numbers = []  
  
createacc = CreateAccounts()  
  
try:  
 create\_deal\_Sheet = utils.excel\_read(pathname=Property.PAYMENT\_THEME,sheetname=Property.CREATE\_DEAL\_SHEET)  
 new\_Deal\_Sheet = utils.excel\_read(pathname=Property.PAYMENT\_THEME,sheetname=Property.NEW\_DEAL\_SHEET)  
 product\_Deal\_Sheet = utils.excel\_read(pathname=Property.PAYMENT\_THEME,sheetname=Property.PRODUCT\_SHEET)  
 businessSegment\_Deal\_Sheet = utils.excel\_read(pathname=Property.PAYMENT\_THEME,sheetname=Property.BUSINESS\_SEGMENT\_SHEET)  
 attribute\_Sheet = utils.excel\_read(pathname=Property.PAYMENT\_THEME,sheetname="ATTRIBUTES")  
 transactionCategories\_Sheet = utils.excel\_read(pathname=Property.PAYMENT\_THEME,sheetname="TRANSACTION\_CATEGORIES")  
 account\_Sheet = utils.excel\_read(pathname=Property.PAYMENT\_THEME,sheetname="ACCOUNTS")  
 party\_Sheet = utils.excel\_read(pathname=Property.PAYMENT\_THEME,sheetname="PARTIES")  
 scheduledFees\_Sheet = utils.excel\_read(pathname=Property.PAYMENT\_THEME,sheetname="SCHEDULED\_FEES")  
 schedule\_Sheet = utils.excel\_read(pathname=Property.PAYMENT\_THEME,sheetname="PAYMENTS")  
 partyAccounts\_sheet = utils.excel\_read(pathname=Property.PAYMENT\_THEME,sheetname="PARTY\_ACCOUNT")  
 budget\_sheet = utils.excel\_read(pathname=Property.PAYMENT\_THEME,sheetname="BUDGET")  
 link\_sheet = utils.excel\_read(pathname=Property.PAYMENT\_THEME,sheetname="LINK\_ADHOC")  
 adhoc\_sheet = utils.excel\_read(pathname=Property.PAYMENT\_THEME,sheetname="ADHOC")  
 bulk\_sheet= utils.excel\_read(pathname=Property.PAYMENT\_THEME,sheetname="BULK\_UPLOAD")  
  
except Exception as e:  
 create\_deal\_Sheet = ""  
 new\_Deal\_Sheet = ""  
 product\_Deal\_Sheet = ""  
 businessSegment\_Deal\_Sheet = ""  
 attribute\_Sheet = ""  
 transactionCategories\_Sheet = ""  
 account\_Sheet = ""  
 party\_Sheet = ""  
 scheduledFees\_Sheet = ""  
 schedule\_Sheet = ""  
 partyAccounts\_sheet = ""  
 bulk\_sheet = ""  
  
ignore\_columns = ["S\_ACC\_NUM","SR NO","EXECUTE\_SCHEDULE","EXECUTION\_TYPE","PAY\_RETENTION","SOURCE\_ACC","L\_PURPOSE","L\_BALANCE","AMOUNT\_TOGGLE","DESTINATION\_TOGGLE", "L\_SCHEDULE", "L\_HOLIDAY\_ACTION","SP\_PURPOSE", "SP\_BALANCE", "SP\_SPLIT", "SP\_AMT", "SP\_SPECIFY", "SP\_BULK", "SP\_BULK\_LOCATION", "SP\_CURR", "SP\_AMT", "SP\_PYMT\_MODE", "SP\_ACC\_TYPE", "SP\_IFSCCODE", "L\_INSTRUMENT", "L\_BULK", "L\_BULK\_LOCATION", "S\_ACC\_NUMBER","S\_AMOUNT", "S\_IFSC\_CODE", "S\_ACCOUNT\_TYPE","BUDGET\_NAME","BUDGET MONTH","BUDGET\_YEAR","BUDGET\_PURPOSE","BUDGET\_INTERVAL","BUD\_YEAR","BUDGET\_AMOUNT","BULK\_LOCATION","BUDGET","BUDGET\_NAME","PAY\_RETENTION\_FULL","SS\_ACC\_NO","SS\_AMT","SS\_ACC\_TYPE","SF\_ACCTYPE","SS\_IFSC\_CODE","APPLICATION","SUBMODULE","BAU/CR","THEME ID","MECE\_THEME\_ID","THEME\_ID","UNIQUE\_ID","EXECUTE","TC\_TYPE","EVENT","ENTITY","SHORT\_NAME","PROCESSING\_UNIT(S)","COUNTRY","TIMEZONE","TRANSACTIONS\_TO\_NON-REGISTERED\_BENEFICIARIES","TRANSACTION\_PURPOSE","TRANSACTION\_CHECKLIST","PARTY\_RESPONSIBILITIES","T\_GROUPS","External","IS\_NEUTRAL\_PARTY","RESPONSIBILITY","COUNTRY","ECOMMERCE\_PARTY","SF\_ACCTYPE","STATUS","KYC\_COMPLETED","ACCOUNT\_NO","ON\_PARTY\_DEACTIVATION","ON\_PARTY\_ACTIVATION","OVERRIDE\_DEAL\_RESERVE\_POLICY","PARTY\_TYPE","P\_COUNTRY1","P\_COUNTRY2","INSTRUCTION\_NAME","CURR","TAX\_SETTING","BRANCH\_CODE","STATE","UPFRONT","UPFRONT\_AMT","FEE\_MODE","FEE\_TYPE","TXN\_BASED","RECURRING\_AMT","HOLIDAY\_UNITS","HOLIDAY\_ACTION","TAX\_CAT","CONTRIBUTE\_AS","TDS\_APPLICABLE","TDS %","TDS AMOUNT","TDS DEBIT ACCOUNT","SF\_PARTY","FEE DEBIT ACCOUNT","FEE\_CONTRIBUTION","TAX\_AMT","SAME\_AS\_SCHEDULED\_DATE","INVOICE\_GENERATION\_TYPE","INVOICE\_DUE\_ DATE","FEE\_DUE\_DATE","SCHEDULE\_AT","FREQUENCY","REPEAT EVERY\_MONTH","SUB\_TYPE","DAY","WEEK","SELECT\_DAY\_OFWEEK","DEBIT\_ACC","CREDIT\_ACC","AMT","NARRATION","S\_NAME","PURPOSE","SRC ACC","BALNC\_CONSIDERATION","SPLIT","SPECIFY\_AMT","AMT\_VALUE","PARTIAL\_PAYMENT","S\_HOLIDAY\_UNITS","REPEATING","ALL\_SUB-INSTR","S\_SCHEDULE\_AT","S\_HLDY\_ACTION","S\_FREQUENCY","S\_REPEAT\_EVERY\_DAY","ENABLE\_AUTO\_RETRY","NOTIFICATION\_ALERTS","S\_CREDIT REMIT\_INFO","S\_DEBIT\_REMIT\_INFO","Priority Dependency","PAYMENT\_SYSTEM","PAYMENT\_MODE","BENE\_ACC\_NAME","IFSC\_CODE","LEI\_TYPE","LEI\_UNIQUE\_ID","LEI\_VALIDITY","BENE\_COUNTRY\_OF\_INCOR","BENE\_COUNTRY","BENE\_CUR","DESCRIPTION","DEAL\_STATUS","BALANCE\_UPDATE\_STATUS","ERROR","CHECKER\_STATUS","PARTIAL\_PAYMENT","SETTLE\_LATER","S\_PAYMENT\_MODE","S\_AMT","S\_ACC\_TYPE","S\_IFSC CODE","L\_ACC\_NUM","L\_ACC\_NAME","L\_AMT","L\_IFSC\_CODE","L\_ACC\_TYPE"]  
  
ignore\_columns2 = ["S.no","Unique\_ID","NEW\_DEAL\_ID","BUDGET\_ID","PRODUCT\_ID","BUSINESS\_SEGMENT\_ID","ATTRIBUTES\_ID","TRANSACTION\_CATEGORIES\_ID","ACCOUNTS\_ID","PARTY\_ID","PARTY\_ACCOUNT\_ID","SCHEDULED\_FEES\_ID","LINK\_ADHOC\_ID","ADHOC\_ID","RETENTION\_ID","RETENTION\_SURPLUS\_ID","PAYMENT\_RETENTION\_SURPLUS\_ID","WATERFALL\_ID","PAYMENT\_SURPLUS\_ID","PAYMENTS\_ID","UPLOAD\_ID","PAYMENT\_RETENTION\_ID","PAYMENT\_RETENTION\_ID"]  
expected = {"Deal\_ID":[],"Deal\_Name":[]}  
  
columns = list(create\_deal\_Sheet.columns)  
filtered\_data = utils.filter\_records(create\_deal\_Sheet)  
print(f"this is filtered {filtered\_data}")  
  
# Number of accounts required  
accounts\_req = len(filtered\_data)  
print(filtered\_data.columns)  
  
# destination  
# destination = createacc.getAccounts(1, accounts\_req, [10000] \*accounts\_req)  
# print(destination)  
  
## Customer and account creation  
accounts\_list = []  
balance\_list =[]  
  
filtered\_data\_acc = utils.convert\_to\_listofdict(data=filtered\_data,columns = columns)  
for record in filtered\_data\_acc:  
 new\_deal\_row = ""  
  
 for key,val in record.items():  
 if key == "ACCOUNTS":  
 try:  
 accounts\_row = account\_Sheet[account\_Sheet["ACCOUNTS\_ID"] == val]  
 accounts\_row = utils.convert\_to\_listofdict(accounts\_row, account\_Sheet.columns)  
 except Exception as e:  
 print(str(e))  
 accounts\_row = ""  
  
 for record in accounts\_row:  
 balancer = []  
 cust = 0  
 accs = 0  
 for accountkey,accountval in record.items():  
 if accountkey == "A\_CUST\_ID":  
 val, counts = utils.extract\_attributes(accountval)  
 cust = counts  
 if accountkey == "ACCOUNT\_NUMBER":  
 val, counts = utils.extract\_attributes(accountval)  
 accs = counts  
 if accountkey == "BALANCE":  
 val,counts = utils.extract\_attributes(str(accountval))  
 for v in val:  
 balancer.append(v)  
 balance\_list.append(v)  
 account\_numbers = createacc.getAccounts(cust,accs,balancer)  
 merged = ';'.join(map(str,account\_numbers))  
 accounts\_list.append(merged)  
 print(accounts\_list)  
 # accounts\_list = 60\*['123']  
  
  
output = defaultdict(list)  
x = 0  
mapper =[]  
  
def extract\_and\_save\_rows\_by\_key(input\_file,sheet,key):  
  
 # Read the Excel file into a DataFrame  
 df = pd.read\_excel(input\_file,sheet\_name=sheet)  
  
 # Filter rows based on the given key  
 try:  
 filtered\_df = df[df['UPLOAD\_ID'] == key]  
 filename = f"{Property.BULK\_FILE\_PATH}bulk\_upload\_20240823\_152827.xlsx"  
 except:  
 filtered\_df = df[df['UPLOAD\_BUDGET\_ID'] == key]  
 filename = f"{Property.BULK\_FILE\_PATH}bulk\_upload\_20240823\_152829.xlsx"  
 print("filter\_df :- ", filtered\_df)  
  
 # Create a new filename with datetime appended  
 now = datetime.now().strftime("%Y%m%d\_%H%M%S")  
 new\_filename = f"{Property.BULK\_FILE\_PATH}bulk\_upload\_{now}.xlsx"  
  
 # Save the filtered rows to a new Excel file  
 filtered\_df.to\_excel(new\_filename, index=False)  
  
 # new\_filename = f"bulk\_upload\_20240809\_120958.xlsx"  
 # Get the absolute path of the new file  
 # abs\_path = os.path.abspath(new\_filename)  
 abs\_path = os.path.abspath(new\_filename)  
  
 return abs\_path  
  
filtered\_data = utils.convert\_to\_listofdict(data=filtered\_data,columns = columns)  
for record in filtered\_data:  
 new\_deal\_row = ""  
  
 for key,val in record.items():  
 if key in ignore\_columns:  
 try:  
 output[key].append(val)  
 except Exception as e:  
 print(str(e))  
  
 if key == "NEW\_DEAL":  
 try:  
 new\_deal\_row = new\_Deal\_Sheet[new\_Deal\_Sheet['NEW\_DEAL\_ID'] == val]  
 new\_deal\_row = utils.convert\_to\_listofdict(new\_deal\_row,new\_Deal\_Sheet.columns)  
 except Exception as e:  
 print(str(e))  
 new\_deal\_row = ""  
 for newdealrecord in new\_deal\_row:  
 product\_deal\_row = ""  
 businesssegment\_row = ""  
  
 for newdealkey,newdealval in newdealrecord.items():  
 if newdealkey in ignore\_columns and newdealkey not in ignore\_columns2:  
 try:  
 output[newdealkey].append(newdealval)  
 except Exception as e:  
 print(str(e))  
 try:  
 product\_deal\_row = product\_Deal\_Sheet[product\_Deal\_Sheet['PRODUCT\_ID'] == newdealval]  
 businesssegment\_row = businessSegment\_Deal\_Sheet[businessSegment\_Deal\_Sheet['BUSINESS\_SEGMENT\_ID'] == newdealval]  
 except:  
 product\_deal\_row = ""  
 businesssegment\_row = ""  
  
 if newdealkey == 'NAME' and newdealval == '<User\_Data>':  
 name = ""  
 try:  
 name = utils.generate\_random\_name(length=10)  
 output[newdealkey].append(name)  
 except:  
 output[newdealkey].append(name)  
  
 if newdealkey == "PRODUCT":  
 try:  
 output[newdealkey].append(product\_deal\_row.iloc[0]['PRODUCT'])  
 except Exception as e:  
 print(str(e))  
  
 if newdealkey in ["ENDS\_ON","STARTS\_ON"]:  
 try:  
 date = utils.calculate\_date(newdealval)  
 output[newdealkey].append(date)  
 except Exception as e :  
 print(str(e))  
  
 if newdealkey == "BUSINESS\_SEGMENT":  
 try:  
 output[newdealkey].append(businesssegment\_row.iloc[0]["BUSINESS\_SEGMENT"])  
 except Exception as e:  
 print(str(e))  
  
 if key == "ATTRIBUTE":  
 values, count = utils.extract\_attributes(val)  
 if count == 1:  
 try:  
 attribute\_row = attribute\_Sheet[attribute\_Sheet['ATTRIBUTES\_ID'] == val]  
 if not attribute\_row.empty:  
 attribute\_row = attribute\_row.to\_dict(orient='records')[0]  
 else:  
 print(f"No data found for 'Unique\_ID' {val}")  
 attribute\_row = {}  
 except Exception as e:  
 print(str(e))  
 attribute\_row = ""  
  
 for attributekey, attributeval in attribute\_row.items():  
 if attributekey in ignore\_columns and attributekey not in ignore\_columns2:  
 try:  
 output[attributekey].append(attributeval)  
 except Exception as e:  
 print(str(e))  
  
 if attributekey == "A\_Groups" :  
 if attributeval == "<User\_Data>":  
 try:  
 group = utils.generate\_random\_name()  
 output[attributekey].append(group)  
 # adding the count column  
 # output["CountAttributes"].append(count)  
 except Exception as r:  
 # output[attributekey].append(np.nan)  
 print(str(r))  
 else:  
 output[attributekey].append(attributeval)  
  
 if attributekey in ["ATTRIBUTES\_NAME","ATTRIBUTES\_VALUE"]:  
 if attributeval == "<User\_Data>":  
 try:  
 name = utils.generate\_random\_name()  
 output[attributekey].append(name)  
 except Exception as e:  
 output[attributekey].append(np.nan)  
 print(str(e))  
 else:  
 output[attributekey].append(attributeval)  
  
 else:  
 group\_string =""  
 name\_string = ""  
 value\_string = ""  
 for index,attr in enumerate(values):  
 try:  
 attribute\_row = attribute\_Sheet[attribute\_Sheet['ATTRIBUTES\_ID'] == attr]  
 if not attribute\_row.empty:  
 attribute\_row = attribute\_row.to\_dict(orient='records')[0]  
 else:  
 print(f"No data found for 'Unique\_ID' {attr}")  
 attribute\_row = {}  
 except Exception as e:  
 print(str(e))  
 attribute\_row = ""  
  
 for attributekey, attributeval in attribute\_row.items():  
 if attributekey == "A\_Groups":  
 if attributeval == "<User\_Data>":  
 try:  
 group = utils.generate\_random\_name()  
 if group\_string:  
 group\_string += ";" + group  
 else:  
 group\_string += group  
 if index == len(values) -1:  
 output[attributekey].append(group\_string)  
 except Exception as r:  
 print(str(r))  
 else:  
 try:  
 group = attributeval  
 if group\_string:  
 group\_string += ";" + group  
 else:  
 group\_string += group  
 if index == len(values) - 1:  
 output[attributekey].append(group\_string)  
 except Exception as r:  
 print(str(r))  
  
 if attributekey == "ATTRIBUTES\_NAME" :  
 if attributeval == "<User\_Data>":  
 try:  
 name = utils.generate\_random\_name()  
 if name\_string:  
 name\_string += ";" + name  
 else:  
 name\_string += name  
 if index == len(values) - 1:  
 output[attributekey].append(name\_string)  
 except Exception as e:  
 print(str(e))  
 else:  
 try:  
 name = attributeval  
 if name\_string:  
 name\_string += ";" + name  
 else:  
 name\_string += name  
 if index == len(values) - 1:  
 output[attributekey].append(name\_string)  
 except Exception as e:  
 print(str(e))  
  
 if attributekey == "ATTRIBUTES\_VALUE" :  
 if attributeval == "<User\_Data>":  
 try:  
 value = utils.generate\_random\_name()  
 if value\_string:  
 value\_string += ";" + value  
 else:  
 value\_string += value  
 if index == len(values) - 1:  
 output[attributekey].append(value\_string)  
 # output["CountAttributes"].append(count)  
 except Exception as e:  
 print(str(e))  
 else:  
 try:  
 value = attributeval  
 if value\_string:  
 value\_string += ";" + value  
 else:  
 value\_string += value  
 if index == len(values) - 1:  
 output[attributekey].append(value\_string)  
 # output["CountAttributes"].append(count)  
 except Exception as e:  
 print(str(e))  
  
 if key == "TRANSACTION\_CATEGORIES":  
 try:  
 transaction\_row = transactionCategories\_Sheet[transactionCategories\_Sheet["TRANSACTION\_CATEGORIES\_ID"]==val]  
 transaction\_row = utils.convert\_to\_listofdict(transaction\_row,transactionCategories\_Sheet.columns)  
 except Exception as e:  
 print(str(e))  
 transaction\_row = ""  
  
 for newtransaction in transaction\_row:  
 for transactionkey,transactionval in newtransaction.items():  
 if transactionkey in ignore\_columns and transactionkey not in ignore\_columns2:  
 try:  
 output[transactionkey].append(transactionval)  
 except Exception as e:  
 print(str(e))  
  
 if key == "BUDGETS":  
 if pd.isna(val) :  
 budget\_row = budget\_sheet[budget\_sheet["BUDGET\_ID"] == "BUDGET\_1"]  
 budget\_row = utils.convert\_to\_listofdict(budget\_row, budget\_sheet.columns)  
  
 for newbudget in budget\_row:  
 for budkey,budval in newbudget.items():  
 if budkey in ignore\_columns and budkey not in ignore\_columns2:  
 try:  
 output[budkey].append(None)  
 except Exception as e:  
 print(str(e))  
 else:  
 try:  
 budget\_row = budget\_sheet[budget\_sheet["BUDGET\_ID"]==val]  
 budget\_row = utils.convert\_to\_listofdict(budget\_row,budget\_sheet.columns)  
 except Exception as e:  
 print(str(e))  
 budget\_row = ""  
  
 for newbudget in budget\_row:  
 for budkey,budval in newbudget.items():  
 if budkey in ignore\_columns and budkey not in ignore\_columns2:  
 try:  
 output[budkey].append(budval)  
 except Exception as e:  
 print(str(e))  
  
  
 if key == "ACCOUNTS":  
 try:  
 accounts\_row = account\_Sheet[account\_Sheet["ACCOUNTS\_ID"] == val]  
 accounts\_row = utils.convert\_to\_listofdict(accounts\_row, account\_Sheet.columns)  
 except Exception as e:  
 print(str(e))  
 accounts\_row = ""  
  
 for record in accounts\_row:  
 for accountkey,accountval in record.items():  
 if accountkey in ignore\_columns and accountkey not in ignore\_columns2:  
 try:  
 output[accountkey].append(accountval)  
 except Exception as r:  
 print(str(r))  
  
 if accountkey == "A\_CUST\_ID":  
 try:  
 output[accountkey].append(accountval)  
 except Exception as r:  
 print(str(r))  
  
 if accountkey == "ACCOUNT\_NUMBER":  
 try:  
 output[accountkey].append(accountval)  
 except Exception as r:  
 print(str(r))  
  
 if key == "PARTIES":  
 try:  
 party\_row = party\_Sheet[party\_Sheet["PARTY\_ID"]==val]  
 # print(party\_row)  
 party\_row = utils.convert\_to\_listofdict(party\_row,party\_Sheet.columns)  
 except Exception as e:  
 print(str(e))  
 party\_row = ""  
  
 for record in party\_row:  
 for partykey,partyval in record.items():  
 if partykey in ignore\_columns and partykey not in ignore\_columns2:  
 try:  
 output[partykey].append(partyval)  
 except Exception as e:  
 print(str(e))  
  
 if partykey in ["P\_CUSTOMER\_ID","PARTICIPANT ID"]:  
 if partyval == "<User\_Data>":  
 try:  
 cid = utils.generate\_random\_digits()  
 output[partykey].append(cid)  
 except Exception as r:  
 print(str(r))  
 output[partykey].append("")  
 else :  
 try:  
 output[partykey].append(partyval)  
 except Exception as e:  
 print(str(e))  
  
 if partykey in ["PARTY\_NAME","REMARKS","ADDR\_LINE\_1","ADDR\_LINE\_2","ADDR\_LINE\_3","ADDR LINE 4"]:  
 if partyval == "<User\_Data>":  
 try:  
 partyname = utils.generate\_random\_name()  
 output[partykey].append(partyname)  
 except Exception as r:  
 print(str(r))  
 output[partykey].append("")  
 else :  
 try:  
 output[partykey].append(partyval)  
 except Exception as e:  
 print(str(e))  
  
 if partykey in ["VALID\_FROM","VALID\_UNTIL","EFFECTIVE\_DATE"]:  
 try:  
 date = utils.calculate\_date(partyval)  
 output[partykey].append(date)  
 except Exception as e:  
 print(str(e))  
 output[partykey].append("")  
  
 if partykey == "PARTY\_ACCOUNT":  
 values,counts = utils.extract\_attributes(partyval)  
 # print(values)  
 if counts == 1:  
 try:  
 partyacc\_row = partyAccounts\_sheet[partyAccounts\_sheet['PARTY\_ACCOUNT\_ID'] == partyval]  
 if not partyacc\_row.empty:  
 partyacc\_row = partyacc\_row.to\_dict(orient='records')[0]  
 else:  
 print(f"No data found for 'Unique\_ID' {val}")  
 partyacc\_row = {}  
 except Exception as e:  
 print(str(e))  
 partyacc\_row = ""  
  
 for partyacckey,partyaccval in partyacc\_row.items():  
 if partyacckey in ignore\_columns and partyacckey not in ignore\_columns2:  
 try:  
 output[partyacckey].append(partyaccval)  
 except Exception as e:  
 print(str(e))  
  
 if partyacckey == "BENE\_ACC\_ NUM" :  
 if partyaccval == "<User\_Data>":  
 try:  
 num = utils.generate\_random\_digits(length=10)  
 output[partyacckey].append(num)  
 except Exception as e:  
 print(str(e))  
 else:  
 try:  
 output[partyacckey].append(partyaccval)  
 except Exception as e:  
 print(str(e))  
  
 else:  
 payment\_system = ""  
 payment\_mode = ""  
 bene\_accno = ""  
 beneacc\_name = ""  
 ifsccode = ""  
 benecountincop = ""  
 benecountry = ""  
 benecurrency = ""  
 desc = ""  
 # print("checkpoint1")  
 for index,attr in enumerate(values):  
 try:  
 # print(attr)  
 partyacc\_row = partyAccounts\_sheet[partyAccounts\_sheet['PARTY\_ACCOUNT\_ID'] == attr]  
 if not partyacc\_row.empty:  
 partyacc\_row = partyacc\_row.to\_dict(orient='records')[0]  
 else:  
 print(f"No data found for 'Unique\_ID' {attr}")  
 partyacc\_row = {}  
 except Exception as e:  
 print(str(e))  
 partyacc\_row = ""  
  
 for partyacckey, partyaccval in partyacc\_row.items():  
  
 if partyacckey == "PAYMENT\_SYSTEM":  
 try:  
 if payment\_system:  
 payment\_system += ";" + partyaccval  
 else:  
 payment\_system = partyaccval  
 if index == len(values) - 1:  
 output[partyacckey].append(payment\_system)  
 except Exception as e:  
 print(str(e))  
  
 if partyacckey == "PAYMENT\_MODE":  
 try:  
 if payment\_mode:  
 payment\_mode += ";" + partyaccval  
 else:  
 payment\_mode = partyaccval  
 if index == len(values) - 1:  
 output[partyacckey].append(payment\_mode)  
 except Exception as e:  
 print(str(e))  
  
 if partyacckey == "BENE\_ACC\_ NUM" :  
 if partyaccval == "<User\_Data>":  
 try:  
 if bene\_accno:  
 bene\_accno += ";" + str(utils.generate\_random\_digits(length=10))  
 else:  
 bene\_accno = str(utils.generate\_random\_digits(length=10))  
 if index == len(values) -1:  
 output[partyacckey].append(bene\_accno)  
 except Exception as e:  
 print(str(e))  
 else:  
 try:  
 if bene\_accno:  
 bene\_accno += ";" + str(partyaccval)  
 else:  
 bene\_accno = str(partyaccval)  
 if index == len(values) - 1:  
 output[partyacckey].append(bene\_accno)  
 except Exception as e:  
 print(str(e))  
  
 if partyacckey == "BENE\_ACC\_NAME":  
 try:  
 if beneacc\_name:  
 beneacc\_name += ";" + partyaccval  
 else:  
 beneacc\_name = partyaccval  
 if index == len(values) - 1:  
 output[partyacckey].append(beneacc\_name)  
 except Exception as e:  
 print(str(e))  
  
 if partyacckey == "IFSC\_CODE":  
 try:  
 if ifsccode:  
 ifsccode += ";" + str(partyaccval)  
 else:  
 ifsccode = str(partyaccval)  
 if index == len(values) - 1:  
 output[partyacckey].append(ifsccode)  
 except Exception as e:  
 print(str(e))  
  
 if partyacckey == "BENE\_COUNTRY\_OF\_INCOR":  
 try:  
 if benecountincop:  
 benecountincop += ";" + partyaccval  
 else:  
 benecountincop = partyaccval  
 if index == len(values) - 1:  
 output[partyacckey].append(benecountincop)  
 except Exception as e:  
 print(str(e))  
  
 if partyacckey == "BENE\_COUNTRY":  
 try:  
 if benecountry:  
 benecountry += ";" + partyaccval  
 else:  
 benecountry = partyaccval  
 if index == len(values)-1:  
 output[partyacckey].append(benecountry)  
 except Exception as e:  
 print(str(e))  
  
 if partyacckey == "BENE\_CUR":  
 try:  
 if benecurrency:  
 benecurrency += ";" + partyaccval  
 else:  
 benecurrency = partyaccval  
 if index == len(values)-1:  
 output[partyacckey].append(benecurrency)  
 except Exception as e:  
 print(str(e))  
  
 if partyacckey == "DESCRIPTION":  
 try:  
 if desc:  
 desc += ";" + partyaccval  
 else:  
 desc = partyaccval  
 if index == len(values) - 1:  
 output[partyacckey].append(desc)  
 except Exception as e:  
 print(str(e))  
  
 if partyacckey in ["LEI\_TYPE","LEI\_UNIQUE\_ID","LEI\_VALIDITY"]:  
 try:  
 if index == len(values)-1:  
 output[partyacckey].append(partyaccval)  
 else:  
 pass  
 except Exception as e:  
 print(str(e))  
  
 if key == "LINKED\_ADHOC":  
 if pd.isna(val):  
 link\_row = link\_sheet[link\_sheet["LINK\_ADHOC\_ID"] == "LINK\_ADHOC\_1"]  
 link\_row = utils.convert\_to\_listofdict(link\_row, link\_sheet.columns)  
  
 for record in link\_row:  
 for lkey, lval in record.items():  
 if lkey in ignore\_columns and lkey not in ignore\_columns2:  
 try:  
 output[lkey].append(None)  
 except Exception as e:  
 print(str(e))  
 output[lkey].append("")  
  
 if lkey == "L\_SCHEDULE\_TIME":  
 try:  
 output[lkey].append(None)  
 except Exception as e:  
 print(str(e))  
 output[lkey].append("")  
  
 if lkey == "L\_EXECUTION\_DATE":  
 if lval == "SYSDATE":  
 try:  
 output[lkey].append(None)  
 except Exception as e:  
 print(str(e))  
 output[lkey].append("")  
 else:  
 output[lkey].append(None)  
  
 else:  
 try:  
 link\_row = link\_sheet[link\_sheet["LINK\_ADHOC\_ID"]==val]  
 link\_row = utils.convert\_to\_listofdict(link\_row,link\_sheet.columns)  
 except Exception as e:  
 print(str(e))  
 link\_row = ""  
  
 for record in link\_row:  
 for lkey,lval in record.items():  
 if lkey in ignore\_columns and lkey not in ignore\_columns2:  
 try:  
 output[lkey].append(lval)  
 except Exception as e:  
 print(str(e))  
 output[lkey].append("")  
  
 if lkey == "L\_SCHEDULE\_TIME":  
 try:  
 xtime = utils.add\_minutes\_to\_systime(lval)  
 output[lkey].append(xtime)  
 except Exception as e:  
 print(str(e))  
 output[lkey].append("")  
  
 if lkey == "L\_EXECUTION\_DATE":  
 if lval == "SYSDATE":  
 try:  
 date = utils.calculate\_date(str(lval))  
 output[lkey].append(date)  
 except Exception as e:  
 print(str(e))  
 output[lkey].append("")  
 else:  
 output[lkey].append("")  
  
 if key == "ADHOC":  
 if pd.isna(val):  
 adhoc\_row = adhoc\_sheet[adhoc\_sheet["ADHOC\_ID"] == "ADHOC\_1"]  
 adhoc\_row = utils.convert\_to\_listofdict(adhoc\_row, adhoc\_sheet.columns)  
  
 for record in adhoc\_row:  
 for lkey, lval in record.items():  
 if lkey in ignore\_columns and lkey not in ignore\_columns2:  
 try:  
 output[lkey].append(None)  
 except Exception as e:  
 print(str(e))  
 output[lkey].append("")  
  
 if lkey == "SP\_ACC":  
 if lval == "<USER\_DATA>":  
 try:  
 output[lkey].append(None)  
 except Exception as e:  
 print(str(e))  
 output[lkey].append("")  
 else:  
 output[lkey].append(None)  
  
 if lkey == "SP\_ACC\_NAME":  
 if lval == "<USER\_DATA>":  
 try:  
 output[lkey].append(None)  
 except Exception as e:  
 print(str(e))  
 output[lkey].append("")  
 else:  
 output[lkey].append("")  
  
  
 else:  
 try:  
 adhoc\_row = adhoc\_sheet[adhoc\_sheet["ADHOC\_ID"] == val]  
 adhoc\_row = utils.convert\_to\_listofdict(adhoc\_row, adhoc\_sheet.columns)  
 except Exception as e:  
 print(str(e))  
 adhoc\_row = ""  
  
 for record in adhoc\_row:  
 for lkey, lval in record.items():  
 if lkey in ignore\_columns and lkey not in ignore\_columns2:  
 try:  
 output[lkey].append(lval)  
 except Exception as e:  
 print(str(e))  
 output[lkey].append("")  
  
 if lkey == "SP\_ACC":  
 if lval == "<USER\_DATA>":  
 try:  
 data = utils.generate\_random\_digits(length=10)  
 output[lkey].append(data)  
 except Exception as e:  
 print(str(e))  
 output[lkey].append("")  
 else:  
 output[lkey].append(lval)  
  
 if lkey == "SP\_ACC\_NAME":  
 if lval == "<USER\_DATA>":  
 try:  
 data = utils.generate\_random\_name()  
 output[lkey].append(data)  
 except Exception as e:  
 print(str(e))  
 output[lkey].append("")  
 else:  
 output[lkey].append(lval)  
  
 if key == "SCHEDULED\_FEES":  
 try:  
 scheduledFees\_row = scheduledFees\_Sheet[scheduledFees\_Sheet["SCHEDULED\_FEES\_ID"]==val]  
 scheduledFees\_row = utils.convert\_to\_listofdict(scheduledFees\_row, scheduledFees\_Sheet.columns)  
 except Exception as e:  
 print(str(e))  
 scheduledFees\_row = ""  
  
 for record in scheduledFees\_row:  
 for sfkey,sfval in record.items():  
 if sfkey in ignore\_columns and sfkey not in ignore\_columns2:  
 try:  
 output[sfkey].append(sfval)  
 except Exception as e:  
 print(str(e))  
 output[sfkey].append("")  
  
 if sfkey in ["SF\_REMARKS","INVOICE\_TO"]:  
 if sfval == "<User\_Data>":  
 try:  
 data = utils.generate\_random\_name()  
 output[sfkey].append(data)  
 except Exception as e:  
 print(str(e))  
 output[sfkey].append("")  
 else:  
 try:  
 output[sfkey].append(sfval)  
 except Exception as e:  
 print(str(e))  
  
 if sfkey == "TA\_ REF\_NO":  
 if sfval == "<User\_Data>":  
 try:  
 data = utils.generate\_random\_digits()  
 output[sfkey].append(data)  
 except Exception as e:  
 print(str(e))  
 output[sfkey].append("")  
 else:  
 try:  
 output[sfkey].append(sfval)  
 except Exception as e:  
 print(str(e))  
  
 if sfkey == "SF\_TIME":  
 try:  
 xtime = utils.add\_minutes\_to\_systime(sfval)  
 output[sfkey].append(xtime)  
 except Exception as e:  
 print(str(e))  
 output[sfkey].append("")  
  
 if sfkey in ["END\_DATE","START\_DATE","SCH\_CURRDATE"]:  
 try:  
 date = utils.calculate\_date(sfval)  
 output[sfkey].append(date)  
 except Exception as e:  
 print(str(e))  
 output[sfkey].append("")  
  
 if key =="INSTRUCTION\_TYPE":  
 if pd.isna(val):  
 schedule\_Sheet = utils.excel\_read(pathname=Property.PAYMENT\_THEME, sheetname="PAYMENTS")  
 schedule\_row = schedule\_Sheet[schedule\_Sheet["PAYMENTS\_ID"] == "PAYMENTS\_1"]  
 schedule\_row = utils.convert\_to\_listofdict(schedule\_row, schedule\_Sheet.columns)  
 mapper.append("")  
 for records in schedule\_row:  
 for sdkey, sdval in records.items():  
 if sdkey in ignore\_columns and sdkey not in ignore\_columns2:  
 try:  
 output[sdkey].append(None)  
 except Exception as e:  
 print(str(e))  
  
 if sdkey == "S\_REMARKS":  
 if sdval == "<User\_Data>":  
 try:  
 output[sdkey].append(None)  
 except Exception as e:  
 print(str(e))  
 output[sdkey].append("")  
 else:  
 try:  
 output[sdkey].append(None)  
 except Exception as e:  
 print(str(e))  
  
 if sdkey in ["S\_START\_DATE", "S\_END\_DATE", "PREV\_DATE", "CUR\_DATE", "FUT\_DATE"]:  
 try:  
 output[sdkey].append(None)  
 except Exception as e:  
 print(str(e))  
 output[sdkey].append("")  
  
 if sdkey == "SCHEDULED\_TIME":  
 try:  
 output[sdkey].append(None)  
 except Exception as e:  
 print(str(e))  
 output[sdkey].append("")  
  
 if sdkey in ["S\_ACC\_NAME","SS\_ACC\_NAME"]:  
 try:  
 output[sdkey].append(None)  
 except Exception as e:  
 print(str(e))  
 output[sdkey].append("")  
  
 if sdkey == "BULK":  
 try:  
 output[sdkey].append(None)  
 output["BULK\_LOCATION"].append(None)  
 except Exception as e:  
 print(str(e))  
 output[sdkey].append("")  
 output["BULK\_LOCATION"].append("")  
  
  
 else:  
 svalues,scounts = utils.extract\_attributes(val)  
 print(svalues)  
 print(scounts)  
 # Single Instructions  
 if scounts == 1:  
 try:  
 # removing last 2 characters  
 sheetname = str(val[:-2])  
 dsheetname = str(val[:-3])  
 if val[-1] == "n":  
 mapper.append("")  
 else:  
 mapper.append(sheetname)  
 try:  
 schedule\_Sheet = utils.excel\_read(pathname=Property.PAYMENT\_THEME, sheetname=sheetname)  
 schedule\_row = schedule\_Sheet[schedule\_Sheet[f'{sheetname}\_ID'] == val]  
 except:  
 schedule\_Sheet = utils.excel\_read(pathname=Property.PAYMENT\_THEME, sheetname=dsheetname)  
 schedule\_row = schedule\_Sheet[schedule\_Sheet[f'{dsheetname}\_ID'] == val]  
 # schedule\_row = schedule\_Sheet[schedule\_Sheet["Unique\_ID"] == val]  
  
 schedule\_row = utils.convert\_to\_listofdict(schedule\_row, schedule\_Sheet.columns)  
 except Exception as e:  
 print(str(e))  
 schedule\_row = ""  
  
 for recordd in schedule\_row:  
 for sdkey, sdval in recordd.items():  
 if sdkey in ignore\_columns and sdkey not in ignore\_columns2:  
 try:  
 output[sdkey].append(sdval)  
 # print(sdkey)  
 # print(sdval)  
 except Exception as e:  
 print(str(e))  
  
 if sdkey == 'SS\_ACC\_NAME':  
 if sdval == "<User\_Data>":  
 try:  
 name = utils.generate\_random\_name()  
 output[sdkey].append(name)  
 except Exception as e:  
 print(str(e))  
 output[sdkey].append("")  
 else:  
 output[sdkey].append("")  
  
 if sdkey == 'S\_ACC\_NAME':  
 if sdval == "<User\_Data>":  
 try:  
 name = utils.generate\_random\_name()  
 output[sdkey].append(name)  
 except Exception as e:  
 print(str(e))  
 output[sdkey].append("")  
  
 elif sdval=="<User\_Data>;<User\_Data>":  
 try:  
 nam1=utils.generate\_random\_name()  
 nam2 = utils.generate\_random\_name()  
 name = f"{nam1};{nam2}"  
 output[sdkey].append(name)  
 except Exception as e:  
 print(str(e))  
 output[sdkey].append("")  
  
 else:  
 output[sdkey].append(sdval)  
  
  
 if sdkey == "S\_REMARKS":  
 if sdval == "<User\_Data>":  
 try:  
 remark = utils.generate\_random\_name()  
 output[sdkey].append(remark)  
 except Exception as e:  
 print(str(e))  
 output[sdkey].append("")  
 else:  
 try:  
 output[sdkey].append(sdval)  
 except Exception as e:  
 print(str(e))  
  
 if sdkey in ["S\_START\_DATE", "S\_END\_DATE", "PREV\_DATE", "CUR\_DATE", "FUT\_DATE"]:  
 try:  
 date = utils.calculate\_date(sdval)  
 output[sdkey].append(date)  
 except Exception as e:  
 print(str(e))  
 output[sdkey].append("")  
  
 if sdkey == "SCHEDULED\_TIME":  
 try:  
 xtime = utils.add\_minutes\_to\_systime(sdval)  
 output[sdkey].append(xtime)  
 except Exception as e:  
 print(str(e))  
 output[sdkey].append("")  
  
 if sdkey == "BULK":  
 if pd.isna(sdval):  
 output[sdkey].append(None)  
 output["BULK\_LOCATION"].append(None)  
 else:  
 if sdval == 'UPLOAD\_1':  
 abs\_path = extract\_and\_save\_rows\_by\_key(Property.PAYMENT\_THEME, "UPLOAD", sdval)  
 else:  
 abs\_path = extract\_and\_save\_rows\_by\_key(Property.PAYMENT\_THEME, "UPLOAD\_BUDGET", sdval)  
 # output[sdkey].append(abs\_path)  
 output[sdkey].append("YES")  
 output["BULK\_LOCATION"].append(abs\_path)  
  
 if key == "DOCUMENTS":  
 try:  
 output[key].append(Property.UPLOAD\_FILE)  
 except Exception as e:  
 print(str(e))  
 output[key].append("")  
  
# Append the mapper list  
output['Mapper'] = mapper  
output['Balance'] = balance\_list  
  
# Update the Accounts Number Sheet  
try:  
 output['ACCOUNT NUMBER'] = accounts\_list  
 # output['Destination\_Account'] = destination  
except Exception as e:  
 print(str(e))  
  
# Accounts Mapping  
payments\_acc\_list = accounts\_list[0].split(';')[0:]  
payments\_accounts = [';'.join(payments\_acc\_list)]  
  
#update the payments source accounts  
# output['Source Accounts'] = payments\_accounts  
  
print(payments\_accounts)  
print(output)  
  
  
# Save to excel file  
utils.excel\_write(Property.DATABUILDER\_OUTPUT,output,Property.INITIATE\_SHEET)  
  
# Styling  
wb = load\_workbook(Property.DATABUILDER\_OUTPUT)  
  
header\_font = Font( bold=True, color='FFFFFF',b=True)  
header\_fill = PatternFill(start\_color='000080', end\_color='000080', fill\_type='solid')  
  
sheet1 = wb['Initiate']  
  
for cell in sheet1['1']:  
 cell.font = header\_font  
 cell.fill = header\_fill  
  
# Save the modified workbook  
wb.save(Property.DATABUILDER\_OUTPUT)  
#  
# # Expected Part  
  
file\_to\_delete = Property.EXPECTED\_FILE  
if os.path.exists(file\_to\_delete):  
 os.remove(file\_to\_delete)  
 print("Deleted the previous data file")  
else:  
 print("File not found")  
  
try:  
 expected\_sheet = utils.excel\_read(pathname=Property.PAYMENT\_THEME,sheetname=Property.EXPECTED\_SHEET)  
 databuilder\_sheet = utils.excel\_read(pathname=Property.DATABUILDER\_OUTPUT,sheetname=Property.INITIATE\_SHEET)  
except Exception as e:  
 print(str(e))  
 expected\_sheet = ""  
  
expected\_sheet = utils.filter\_records(expected\_sheet)  
  
expected\_filtered\_data = utils.convert\_to\_listofdict(data=expected\_sheet,columns=expected\_sheet.columns)  
  
eignore\_columns = ["SR NO","APPLICATION","SUBMODULE","EXECUTION\_TYPE","BUDGET\_NAME","INSTRUCTION\_TYPE","INSTRUCTION\_END\_DATE","INSTRUCTION\_PURPOSE","LINKED\_PURPOSE","LINKED\_TYPE","BAU/CR","THEME ID","MECE\_THEME\_ID","EXECUTE","TC\_TYPE","Event","ENTITY","DEAL\_ID","DEAL\_SEGMENT","DEAL\_PRODUCT", "DEAL\_STATUS",  
"PRODUCT","BUSINESS SEGMENT","DEAL\_TIMEZONE","DEAL\_COUNTRY","ENTITLEMENTS", "ATTRIBUTES", "TRANSACTION\_LIMIT", "TRANSACTION\_PURPOSES",  
 "TRANSACTION\_CHECKLIST", "DEAL\_CHECKLIST","BUDGETS","SCHEDULES","LINKED","ENTITLEMENTS", "TRANSACTION\_CATEGORIES",  
 "PARTY\_RESPONSIBILITY", "PROCESSING\_UNITS", "NOTIFICATIONS", "CONTACTS",  
 "DOCUMENTS", "EXECUTION", "ATTRIBUTES\_GROUPS", "ATTRIBUTE\_NAMES",  
 "ATTRIBUTE\_VALUES", "TRANSACTIONS\_PURPOSE", "TRANSACTION\_CHECKLIST\_MAKER",  
 "TRANSACTION\_CHECKLIST\_CHECKER", "TRANSACTION\_CATEGORY", "PARTY\_RES",  
 "PROCESSING\_UNIT", "CONTACT\_NAME", "CONTACT\_EMAIL", "CONTACT\_NUMBER",  
 "CONTACT\_TYPE", "EXECUTION\_GROUP\_PAYEMNTS", "EXECUTION\_DEPENDENCIES",  
 "EXECUTION\_VERIFY\_BALANCES", "EXECUTION\_HOLIDAY\_ACTION", "EXECUTION\_INACTIVE",  
 "EXECUTION\_REATTEMPT", "EXECUTION\_REATTEMPE\_INTERVAL",  
 "EXECUTION\_BALANCE\_CONSIDERATION", "EXECUTION\_WORK\_ITEMS", "EXECUTION\_OFFER\_TO",  
 "EXECUTION\_AUTO\_INACTIVATE", "EXECUTION\_PARTY\_DEACTIVATION",  
 "EXECUTION\_PARTY\_ACTIVATION", "EXECUTION\_SCHEDULE\_ON",  
 "EXECUTION\_SCHEDULE\_TIMEZONE", "EXECUTION\_SCHEDULE\_TIME", "EXECUTION\_RESERVE\_TYPE",  
 "STATUS", "EXECUTION\_RESERVE\_CAP\_LIMIT", "EXECUTION\_RESERVE\_PERCENTAGE",  
 "EXECUTION\_PAST\_RANGE", "EXECUTION\_CUSTOMER\_REFERENCE", "EXECUTION\_FUTURE\_RANGE","PARTY\_NAME","RESPONSIBILITY","COUNTRY","DEBIT\_ACCOUNTS\_TAGGED","RESERVE\_ATTRIBUTES","RESERVE\_CONFIG","PARTY\_ATTRIBUTES","ACCOUNTS","DOCS","PARTY\_STATUS","KYC\_COMPLETED","ON\_PARTY\_DEACTIVATION","ON\_PARTY\_ACTIVATION",'ADDRESS1', 'ADDRESS2', 'ADDRESS3', 'ADDRESS4', 'ECOM\_NATIONALITY', 'ECOM\_IDENTIFICATION\_DOC', 'ECOM\_DOC\_NUMBER', 'ECOM\_ISSUE\_DATE', 'ECOM\_EXPIRY\_DATE', 'ECOM\_TAX\_ID', 'ECOM\_EMAIL\_ID', 'ECOM\_PHONE\_NUMBER']  
  
eoutput = defaultdict(list)  
  
for record in expected\_filtered\_data:  
  
 for key,val in record.items():  
 if key in eignore\_columns:  
 try:  
 eoutput[key].append(val)  
 except Exception as e:  
 print(str(e))  
  
 if key == "STARTS\_ON":  
 if val == "<User\_Data>":  
 try:  
 initiate\_row = databuilder\_sheet[databuilder\_sheet['THEME\_ID']== record['UNIQUE\_ID']]  
 date = initiate\_row.iloc[0]['STARTS\_ON']  
 date = utils.convert\_date\_format(date)  
 eoutput[key].append(date)  
 except Exception as e:  
 print(str(e))  
 eoutput[key].append(None)  
  
 if key == "ENDS\_ON":  
 if val == "<User\_Data>":  
 try:  
 initiate\_row = databuilder\_sheet[databuilder\_sheet['THEME\_ID']== record['UNIQUE\_ID']]  
 date = initiate\_row.iloc[0]['ENDS\_ON']  
 date = utils.convert\_date\_format(date)  
 eoutput[key].append(date)  
 except Exception as e:  
 print(str(e))  
 eoutput[key].append(None)  
  
 if key == "INSTRUCTION\_START\_DATE":  
 try:  
 if val == "<User\_Data>":  
 initiate\_row = databuilder\_sheet[databuilder\_sheet['THEME\_ID'] == record['UNIQUE\_ID']]  
 date = initiate\_row.iloc[0]['S\_START\_DATE']  
 date = utils.convert\_date\_format(date)  
 eoutput[key].append(date)  
 else:  
 eoutput[key].append(None)  
 except Exception as e:  
 print(str(e))  
 eoutput[key].append(None)  
  
 if key == "ACCOUNT\_NO":  
 if val == "<User\_Data>":  
 try:  
 initiate\_row = databuilder\_sheet[databuilder\_sheet['THEME\_ID']== record['UNIQUE\_ID']]  
 # dest\_row = databuilder\_sheet[databuilder\_sheet['UNIQUE\_ID']==record["UNIQUE\_ID"]]  
 # desti = initiate\_row.iloc[0]['Destination\_Account']  
 sourceacc = initiate\_row.iloc[0]['ACCOUNT NUMBER']  
 eoutput[key].append(sourceacc)  
 except Exception as e:  
 print(str(e))  
 eoutput[key].append(None)  
  
 if key == "ACCOUNT\_NAME":  
 if val == "<User\_Data>":  
 try:  
 import xml.etree.ElementTree as ET  
  
 # Replace 'path/to/your/xml\_file.xml' with the actual file path  
 current\_date = datetime.now().strftime('%Y-%m-%d')  
 # xml\_file\_path = f"{Property.ACCOUNT\_NO\_API\_FILE}{current\_date}/UNIQUE\_CUST\_112.xml"  
 xml\_file\_path = Property.CUSTOMER\_XML  
 # xml\_file\_path = "customer.xml"  
  
 # Read the XML content from the file  
 with open(xml\_file\_path, 'r') as file:  
 xml\_data = file.read()  
  
 # Parse the XML data  
 root = ET.fromstring(xml\_data)  
  
 # Find the first and last names  
 first\_name = root.find('.//{http://TCS.BANCS.Adapter/BANCSSchema}CustFirstName').text  
 last\_name = root.find('.//{http://TCS.BANCS.Adapter/BANCSSchema}CustLastName').text  
  
 print("First Name:", first\_name)  
 print("Last Name:", last\_name)  
 eoutput[key].append(f"Mr. {first\_name} H {last\_name}")  
 except Exception as e:  
 print(str(e))  
  
 if key =="PARTICIPANT\_ID":  
 if val == "<User\_Data>":  
 try:  
 initiate\_row = databuilder\_sheet[databuilder\_sheet['THEME\_ID'] == record['UNIQUE\_ID']]  
 eoutput[key].append(initiate\_row.iloc[0]['PARTICIPANT ID'])  
 except Exception as e:  
 print(str(e))  
 else:  
 eoutput[key].append("N.A")  
  
 if key == "CUSTOMER\_ID":  
 if val == "<User\_Data>":  
 try:  
 initiate\_row = databuilder\_sheet[databuilder\_sheet['THEME\_ID'] == record['UNIQUE\_ID']]  
 eoutput[key].append(initiate\_row.iloc[0]['P\_CUSTOMER\_ID'])  
 except Exception as e:  
 print(str(e))  
 eoutput[key].append(None)  
 else:  
 eoutput[key].append(None)  
  
 if key =="VALID\_FROM":  
 if val == "<User\_Data>":  
 try:  
 initiate\_row = databuilder\_sheet[databuilder\_sheet['THEME\_ID'] == record['UNIQUE\_ID']]  
 date = initiate\_row.iloc[0]['VALID\_FROM']  
 date = utils.convert\_date\_format(date)  
 eoutput[key].append(date)  
 except Exception as e:  
 print(str(e))  
 else:  
 eoutput[key].append(None)  
  
 if key =="VALID\_UNTIL":  
 if val == "<User\_Data>":  
 try:  
 initiate\_row = databuilder\_sheet[databuilder\_sheet['THEME\_ID'] == record['UNIQUE\_ID']]  
 date = initiate\_row.iloc[0]['VALID\_UNTIL']  
 date = utils.convert\_date\_format(date)  
 eoutput[key].append(date)  
 except Exception as e:  
 print(str(e))  
 else:  
 eoutput[key].append(None)  
  
 if key == 'ECOM\_DOB':  
 if val == "<User\_Data>":  
 try:  
 currdate = datetime.now()  
 currdate = currdate.strftime("%d-%b-%Y")  
 eoutput[key].append(currdate)  
 except Exception as e:  
 print(str(e))  
 else:  
 eoutput[key].append(None)  
  
 if key == "DEAL\_SHORTNAME":  
 try:  
 initiate\_row = databuilder\_sheet[databuilder\_sheet['THEME\_ID'] == record['UNIQUE\_ID']]  
 shortname = initiate\_row.iloc[0]['NAME']  
 # print(str(shortname.lower()))  
 eoutput[key].append(str(shortname.lower()))  
 except Exception as e:  
 print(str(e))  
 eoutput[key].append(None)  
  
print(eoutput)  
df2 = pd.DataFrame(eoutput)  
path = Property.EXPECTED\_FILE  
  
try:  
 utils.excel\_write(path,eoutput,"Expected\_Data")  
except Exception as e:  
 print(str(e))  
  
  
## Expected Report Start  
  
# those which are yes  
# FOR EACH:  
# from their theme id check if split,non split,bulk using the deal create  
# also keep a global flag for lenght if split and bulk  
# if non split directly add the sp amount into the amount column same with others  
# else : first find the then  
# if split then from dealcreate.xlsx : for that theme id first divide the 3 cols  
# [amt,destiacc,destiname, just copy src twice] then: calc for % and for amt simple copy  
# is bulk: read the bulk file path from the dealcreate  
# open the bulk file and read amt,destinationacc,name from the sheet and append the same  
# just copy the src acc as many times required in the report  
# update that theme's report dataframe with the fetched values  
# create the final report excel for the same with the theme id as the report name  
# do the same for others  
  
  
  
# Jenkins Path  
# XCRO UI Interface