**Os,rename,del,create,write,Join,split,loc,iloc,lambda,excel dict all operations**

**pdExcelfile,excelwrite,preadexcel with sheet as none forall sheet**

**then dict.itmes with sheetname and df as key valye pairs**

**Set root path**

current\_path = os.path.dirname(os.path.abspath(\_\_file\_\_))  
print(current\_path)  
root\_path = os.path.abspath(os.path.join(current\_path,'..','..','..'))

**Appending data into new sheet if the file exists if not then creating one**

def excel\_write(path, output, sheetname):  
 try:  
 data = pd.DataFrame(output)  
  
 # Check if the file exists  
 if not os.path.exists(path):  
 # If the file doesn't exist, create it with the first sheet  
 with pd.ExcelWriter(path, engine='openpyxl') as writer:  
 data.to\_excel(writer, sheet\_name=sheetname, index=False)  
 print(f"{sheetname} created successfully!!!")  
 else:  
 # If the file exists, append the data as a new sheet  
 with pd.ExcelWriter(path, engine='openpyxl', mode='a') as writer:  
 data.to\_excel(writer, sheet\_name=sheetname, index=False)  
 print(f"{sheetname} created successfully!!!")  
 except Exception as r:  
 print(str(r))  
 print("Cannot write to excel file")

**FILTERING**

def filter\_records(dataframe):  
 if 'EXECUTE' in dataframe.columns:  
 records = dataframe.loc[dataframe['EXECUTE'] == 'Y']  
 elif 'Execute' in dataframe.columns:  
 records = dataframe.loc[dataframe['Execute'] == 'Y']  
 else:  
 records = dataframe  
 return records

**CONVERTING LIST OF DICT FOR LOOPING**

def convert\_to\_listofdict(data,columns):  
 records = []  
 for row in range(len(data)):  
 dictionary = {}  
 for col in range(len(columns)):  
 dictionary[columns[col]] = data.iloc[row][col]  
 records.append(dictionary)  
  
 return records

Each row is one dict inside the list

**CALCULATION OF DATE AND TIME BASED ON STANDARD INPUT FORMAT**

def calculate\_date(input\_string):  
 try:  
 parts = input\_string.split('+')  
 if parts[0] == 'SYSDATE' or parts[0] == '<SYSDATE>':  
 days\_to\_add = int(parts[1]) if len(parts) == 2 else 0  
 # else:  
 # raise ValueError("Invalid input format. Please use 'SYSDATE' or 'SYSDATE+X' where X is the number of days to add.")  
  
 current\_date = datetime.now()  
 new\_date = current\_date + timedelta(days=days\_to\_add)  
  
 return new\_date.strftime('%d-%m-%Y')  
  
 except :  
 try:  
 parts = input\_string.split('-')  
 if parts[0] == 'SYSDATE' or parts[0] == '<SYSDATE>':  
 days\_to\_add = int(parts[1]) if len(parts) == 2 else 0  
 # else:  
 # raise ValueError(  
 # "Invalid input format. Please use 'SYSDATE' or 'SYSDATE+X' where X is the number of days to add.")  
  
 current\_date = datetime.now()  
 days\_to\_add = -days\_to\_add  
 new\_date = current\_date + timedelta(days=days\_to\_add)  
  
 return new\_date.strftime('%d-%m-%Y')  
 except Exception as e:  
 return ""

def add\_minutes\_to\_systime(inputTime):  
 try:  
 parts = inputTime.split('+')  
 if parts[0] == 'SYSTIME' or parts[0] == '<SYSTIME>':  
 minutes\_to\_add = int(parts[1]) if len(parts) == 2 else 0  
 else:  
 raise ValueError(  
 "Invalid input format. Please use 'SYSTIME' or 'SYSTIME+X' where X is the number of minutes to add.")  
  
 current\_time = datetime.now()  
 new\_time = current\_time + timedelta(minutes=minutes\_to\_add)  
 formatted\_time = new\_time.strftime('%H:%M%p')  
  
 return formatted\_time  
  
 except ValueError as v:  
 print(str(v))

**Extract Attributes count and values divided by some symbol**

def extract\_attributes(attribute\_str):  
 attribute\_str = str(attribute\_str)  
 attributes = attribute\_str.split(';')  
 values = [attr.strip() for attr in attributes if attr.strip()]  
 count = len(attributes)  
  
 return values, count

**EXCEL TO DICT**

def excel\_to\_dict(dataframe):  
 *'''Convert Excel to dict without unique id and return'''* columns\_name = []  
 for col in dataframe.to\_dict():  
 columns\_name.append(col)  
 data = []  
 for row in range(0, len(dataframe)):  
 elm = {}  
 for col in range(len(dataframe.columns)):  
 if columns\_name[col] != "UNIQUE\_ID":  
 elm[columns\_name[col]] = dataframe.iloc[row][col]  
 data.append(elm)  
 return data

**EXCEL TO DICT WITH UNIQUE ID**

def excel\_to\_dict\_with\_unique\_id(dataframe):  
 *'''Convert Excel to dict with unique id and return'''* columns\_name = []  
 for col in dataframe.to\_dict():  
 columns\_name.append(col)  
 data = []  
 for row in range(0, len(dataframe)):  
 elm = {}  
 for col in range(len(dataframe.columns)):  
 elm[columns\_name[col]] = dataframe.iloc[row][col]  
 data.append(elm)  
 return data

**Generate random key**

def generate\_random\_key(length=10,alphadigits=8):  
 # Generate random string  
 random\_string = ''.join(random.choice(string.ascii\_letters) for i in range(alphadigits))  
  
 # Generate random digits  
 random\_digits = ''.join(random.choice(string.digits) for i in range(length-alphadigits))  
  
 # Combine the random string and digits  
 random\_key = random\_string + random\_digits  
 return random\_key.upper()

**File names based on now**

now = datetime.now().strftime("%Y%m%d\_%H%M%S")  
new\_filename = f"{Property.BULK\_FILE\_PATH}bulk\_upload\_{now}.xlsx"

**Decorator for creating primary key in a df with apply**

def create\_unique\_key(df,name):  
 key\_count = {}  
  
 def generate\_key(row):  
 base\_key = f"{row['Beneficiary Name']}\_{row['Deal Name']}"  
  
 if base\_key in key\_count:  
 key\_count[base\_key] += 1  
 return f"{base\_key}\_{key\_count[base\_key]}"  
 else:  
 key\_count[base\_key] = 0  
 return base\_key  
  
 df['PrimaryKey'] = df.apply(generate\_key, axis=1)  
 df.to\_excel(f"{name}.xlsx", index=False, sheet\_name=f"{name}\_sheet")

**DELETE SHEET FROM EXCEL**

# Function to delete the first sheet from an Excel file

def delete\_first\_sheet(excel\_file):

    # Load the workbook

    wb = openpyxl.load\_workbook(excel\_file)

    # Get the first sheet name

    first\_sheet = wb.sheetnames[0]

    # Remove the first sheet

    wb.remove(wb[first\_sheet])

    # Save the workbook (you can save it with a different name if desired)

    wb.save(excel\_file)

# List of Excel files

excel\_files = ['file1.xlsx', 'file2.xlsx', 'file3.xlsx']

# Iterate through each file and delete the first sheet

for file in excel\_files:

    delete\_first\_sheet(file)

print("First sheet removed from all Excel files.")

**DELETING SHEETS USING SHEET NAMES**

# Function to delete a sheet with a given name from an Excel file

def delete\_sheet\_by\_name(excel\_file, sheet\_name):

    # Load the workbook

    wb = openpyxl.load\_workbook(excel\_file)

    # Check if the sheet exists in the workbook

    if sheet\_name in wb.sheetnames:

        # Remove the sheet

        wb.remove(wb[sheet\_name])

        # Save the workbook after removing the sheet

        wb.save(excel\_file)

        print(f"Sheet '{sheet\_name}' deleted from {excel\_file}")

    else:

        print(f"Sheet '{sheet\_name}' not found in {excel\_file}")

# List of Excel files

excel\_files = ['file1.xlsx', 'file2.xlsx', 'file3.xlsx']

sheet\_name\_to\_delete = 'Sheet1'  # Specify the sheet name you want to delete

# Iterate through each file and delete the specified sheet

for file in excel\_files:

    delete\_sheet\_by\_name(file, sheet\_name\_to\_delete)

if os.path.exists(actual\_reports):  
 existing\_df = pd.read\_excel(actual\_reports)  
  
 # Creating new dataframe from the captured data  
 new\_Df = {  
 "DEAL\_ID":length \*[ str(dealId)],  
 "AMOUNT": amounts,  
 "SOURCE\_ACCOUNTS": srcaccounts,  
 "SETTLED AMOUNT":settledamounts,  
 "ORIGINAL AMOUNT": orgamounts,  
 "BENEFICIARY ACCOUNT NUMBER":beneaccounts,  
 "BENEFICIARY NAMES":benenames  
 }  
  
 newdf = pd.DataFrame(new\_Df)  
 concat\_df = pd.concat([existing\_df,newdf],ignore\_index=True)  
 concat\_df.to\_excel(actual\_reports,"Actual\_Reports")