**Utility\_functions**

color\_coding\_red = PatternFill(start\_color='E84603', end\_color='E84603', fill\_type='solid')

color\_coding\_green = PatternFill(start\_color='9DEC67', end\_color='9DEC67', fill\_type='solid')

color\_coding\_yellow = PatternFill(start\_color='EAE93F', end\_color='EAE93F', fill\_type='solid')

color\_coding\_blue = PatternFill(start\_color='ADD8E6', end\_color='ADD8E6', fill\_type='solid')

* Setting up color coding using the openpyxl.

def find\_header\_string\_position(sheet\_name, str\_to\_find):

    position = [0, 0]

    for i in range(1, sheet\_name.max\_row + 1):

        for j in range(1, sheet\_name.max\_column + 1):

            if (str\_to\_find.strip().upper() in str(sheet\_name.cell(i, j).value).strip().upper()):

#                print("Found the str - ", str\_to\_find, " at : ", i, " , ", j)

                position = [i, j]

                return position

    print("String not Found : ", str\_to\_find)

    return None

**def find\_header\_string\_position** searches for a specific string in an Excel sheet and returns its position if found.

1. **sheet\_name**: Represents the Excel sheet.
2. **str\_to\_find**: String you want to find.
3. position: It's initialized as [0, 0] and will be updated to store the row and column position of the first occurrence of the **str\_to\_find** in the sheet.  
     
   When this function finds the string (**str\_to\_find**), When the function finds the string it will update the position with row and column and if does not find the string it will print “String not Found” and return None

def find\_last\_col\_position(sheet\_name, str\_to\_find):

    last\_col\_position = find\_header\_string\_position(sheet\_name, str\_to\_find)

    if (last\_col\_position == None) :

        print("String not Found : ", str\_to\_find)

        return None

    last\_col\_start\_pos = last\_col\_position[1]

    for i in range(last\_col\_start\_pos + 1, sheet\_name.max\_column + 1):

        if (str\_to\_find.strip().upper() in str(sheet\_name.cell(last\_col\_position[0], i).value).strip().upper()):

            last\_col\_position[1] = i

    return last\_col\_position

**def find\_last\_col\_position** find the last occurrence of a specified string in a column and return the position of this string. It relies on **find\_header\_string\_position**, to find the initial occurrence of the string and then iterates through the row from that point to identify the last occurrence.

1. **find\_last\_col\_position** takes in the **sheet\_name** and the **str\_to\_find** string as arguments.
2. It uses the **find\_header\_string\_position** function to locate the first occurrence of the specified string.
3. If the initial string position is found, it continues the search for the string in the same row. It updates the column value in the **last\_col\_position** list whenever it finds the string, thus recording the last occurrence.
4. It returns the **last\_col\_position**, which contains the row and column positions of the last occurrence of the specified string.

def format\_header\_cells(file\_sheet, cell\_position, no\_of\_rows\_to\_merge, no\_of\_columns\_to\_merge, color\_coding,

                        column\_width):

    file\_sheet.merge\_cells(start\_row=cell\_position[0], start\_column=cell\_position[1],

                           end\_row=cell\_position[0] + no\_of\_rows\_to\_merge,

                           end\_column=cell\_position[1] + no\_of\_columns\_to\_merge)

    file\_sheet.cell(cell\_position[0], cell\_position[1]).fill = color\_coding

    file\_sheet.cell(cell\_position[0], cell\_position[1]).alignment = Alignment(wrap\_text=True, vertical='center')

    file\_sheet.column\_dimensions[get\_column\_letter(cell\_position[1])].width = column\_width

**def format\_header\_cells**, is used to format header cells in an Excel file.

**Arguments**:

* + **file\_sheet**: The Excel sheet where the formatting is to be applied.
  + **cell\_position**: Contains the row and column position.
  + **no\_of\_rows\_to\_merge**: Number of rows to merge.
  + **no\_of\_columns\_to\_merge**: Number of columns to merge.
  + **color\_coding**: The color to be applied to the header cells.
  + **column\_width**: The width of the header columns.

def load\_df\_from\_excel\_file(file\_path, sheet\_name\_to\_read, no\_of\_rows\_to\_skip, levels\_of\_index, no\_of\_data\_rows\_to\_drop):

    raw\_df = pd.DataFrame()

    # =========== Read file path data provided in config file =======================

    raw\_df = pd.read\_excel(file\_path, sheet\_name=sheet\_name\_to\_read, skiprows=no\_of\_rows\_to\_skip, header=list(range(0,levels\_of\_index)))

    # =========== Merge the headers into one header =======================

    if(levels\_of\_index > 1) :

        new\_header = []

        headers\_count\_dict = {}

        for col in raw\_df.columns:

            temp\_header = ""

            for i in range(0,levels\_of\_index) :

                if not (col[i].startswith('Unnamed:') or col[i] == '') :

                    temp\_header = temp\_header + ' ' + col[i]

            if(temp\_header not in headers\_count\_dict) :

                headers\_count\_dict.update({temp\_header : 0})

            else :

                headers\_count\_dict.update({temp\_header : (headers\_count\_dict[temp\_header] + 1)})

                temp\_header = temp\_header + ' ' + str(headers\_count\_dict[temp\_header])

            new\_header.append(temp\_header)

        raw\_df.columns = pd.MultiIndex.from\_tuples(

            list(zip(new\_header, raw\_df.columns.get\_level\_values(levels\_of\_index-1))))

        raw\_df.columns = raw\_df.columns.droplevel(level = 1)

    # =========== drop irrelevant rows after headers from the df and reset the index =======================

    raw\_df = raw\_df.drop(index = list(range(0, no\_of\_data\_rows\_to\_drop)))

    raw\_df = raw\_df.reset\_index(drop = True)

    raw\_df.columns = raw\_df.columns.str.strip()

    return raw\_df

**def load\_df\_from\_excel\_file**, processes an Excel file using Pandas and manipulate the DataFrame

**Arguments**:

* + **no\_of\_rows\_to\_skip**: Number of rows to skip at the beginning.
  + **levels\_of\_index**: Number of header levels in the DataFrame.
  + **no\_of\_data\_rows\_to\_drop**: Number of rows to drop after the header.

**levels\_of\_index > 1**

It creates a new header by combining the different levels and checks for any 'Unnamed' or empty column entries.

also manages duplicate headers by appending counts

def load\_df\_from\_excel\_file\_backup(file\_path, sheet\_name\_to\_read, no\_of\_rows\_to\_skip, levels\_of\_index, no\_of\_data\_rows\_to\_drop):

    raw\_df = pd.DataFrame()

    # =========== Read file path data provided in config file =======================

    raw\_df = pd.read\_excel(file\_path, sheet\_name=sheet\_name\_to\_read, skiprows=no\_of\_rows\_to\_skip, header=list(range(0,levels\_of\_index)))

    # =========== Merge the headers into one header =======================

    if(levels\_of\_index > 1) :

        new\_header = []

        for col in raw\_df.columns:

            temp\_header = ""

            for i in range(0,levels\_of\_index) :

                if not (col[i].startswith('Unnamed:') or col[i] == '') :

                    temp\_header = temp\_header + ' ' + col[i]

            new\_header.append(temp\_header)

        raw\_df.columns = pd.MultiIndex.from\_tuples(

            list(zip(new\_header, raw\_df.columns.get\_level\_values(levels\_of\_index-1))))

        raw\_df.columns = raw\_df.columns.droplevel(level = 1)

    # =========== drop irrelevant rows after headers from the df and reset the index =======================

    raw\_df = raw\_df.drop(index = list(range(0, no\_of\_data\_rows\_to\_drop)))

    raw\_df = raw\_df.reset\_index(drop = True)

    # ============ Remove all white spaces from raw\_df data frame as well as from column headers of all fields ============

#    for column in raw\_df.columns:

#        raw\_df[column] = raw\_df[column].apply(lambda x: str(x).strip())

    raw\_df.columns = raw\_df.columns.str.strip()

    return raw\_df

Backup version of the **def load\_df\_from\_excel\_file** function,

1. **Header Consolidation**:
   * It retains the functionality to merge the headers into one header, ensuring that for **levels\_of\_index** greater than 1, the header is combined into a single header.
2. **Dropping Irrelevant Rows**:
   * It drops the specified number of irrelevant rows after the header and resets the DataFrame's index.
3. **Strip Whitespace**:
   * A modification is made in the way whitespace stripping is applied. Instead of individually applying the strip() function to each column, it uses the **str.strip()** method directly to remove leading and trailing whitespaces from all column headers.

Merge the headers into a single header in case the DataFrame has multiple levels **(levels\_of\_index > 1)**.

The key differences between the key difference between the main function and backup function is how the functions are handling duplicate headers, Backup function concatenates the headers without accounting for duplicate headers. It will result in a single header but may not manage duplicates, potentially causing loss of information. Main function uses dictionary **headers\_count\_dict** to handle and count duplicate headers. It appends a count suffix to the duplicate headers

def add\_row\_no\_in\_dataframe(file\_name, sheet\_name, final\_data\_frame, column\_header\_in\_excel\_sheet, column\_header\_in\_dataframe):

    file\_wb = openpyxl.load\_workbook(file\_name)

    file\_sheet = file\_wb[sheet\_name]

    column\_header\_in\_excel\_sheet\_pos = find\_header\_string\_position(file\_sheet, column\_header\_in\_excel\_sheet)

    list\_of\_sr\_no = []

    list\_of\_row\_no = []

    for i in range(column\_header\_in\_excel\_sheet\_pos[0] + 1, file\_sheet.max\_row + 1):

        if (file\_sheet.cell(i, column\_header\_in\_excel\_sheet\_pos[1]).value == None):

            continue

        list\_of\_row\_no.append(i)

        list\_of\_sr\_no.append(file\_sheet.cell(i, column\_header\_in\_excel\_sheet\_pos[1]).value)

    row\_no\_df = pd.DataFrame(list(zip(list\_of\_sr\_no, list\_of\_row\_no)), columns=['Sr. No.', 'Row No in Excel Sheet'])

    for column in row\_no\_df.columns:

        row\_no\_df[column] = row\_no\_df[column].apply(lambda x: str(x).strip())

    for column in final\_data\_frame.columns:

        final\_data\_frame[column] = final\_data\_frame[column].apply(lambda x: str(x).strip())

    final\_data\_frame = final\_data\_frame.merge(row\_no\_df, left\_on="Sr. No.", right\_on="Sr. No.")

    final\_data\_frame['Sr. No.'] = final\_data\_frame['Sr. No.'].apply(lambda x: int(x))

    final\_data\_frame['Row No in Excel Sheet'] = final\_data\_frame['Row No in Excel Sheet'].apply(lambda x: int(x))

    return final\_data\_frame

**def add\_row\_no\_in\_dataframe** add row numbers from an Excel file to a provided Pandas DataFrame

1. **Find Header Position**:
   * It finds the position of a specific column header (**column\_header\_in\_excel\_sheet**) in the Excel sheet using a function **find\_header\_string\_position**.
2. **Iterate Through Rows**:
   * It iterates through the rows of the specified column in the Excel sheet to collect row numbers (**list\_of\_row\_no**) and associated serial numbers (**list\_of\_sr\_no**) present in the Excel file.
3. **Create DataFrame**:
   * It constructs a DataFrame (**row\_no\_df**) from the collected row numbers and serial numbers.
4. **Data Cleaning**:
   * It applies string stripping to all columns in both the DataFrame and the newly created **row\_no\_df**.
5. **Merging DataFrames**:
   * It merges the provided DataFrame (**final\_data\_frame**) with the **row\_no\_df** DataFrame based on the 'Sr. No.' column, effectively adding row numbers from the Excel file to the provided DataFrame.
6. **Data Type Conversion**:
   * It converts the 'Sr. No.' and 'Row No in Excel Sheet' columns to integer type.
7. **Return**:
   * Returns the updated **final\_data\_frame** with the added row numbers from the Excel file.

def write\_alerts\_compliance\_status\_in\_excel\_sheet(file\_name, sheet\_name, final\_data\_frame, compliance\_headers\_dict, last\_col\_str, no\_of\_empty\_cols\_after\_last\_col):

    file\_wb = openpyxl.load\_workbook(file\_name)

    file\_sheet = file\_wb[sheet\_name]

    last\_column\_loc = find\_last\_col\_position(file\_sheet, last\_col\_str)

    compliance\_headers\_list = list(compliance\_headers\_dict.keys())

    for compliance\_header\_str in compliance\_headers\_list :

        if compliance\_header\_str in final\_data\_frame.columns:

            file\_sheet.cell(last\_column\_loc[0],

                            last\_column\_loc[1] + no\_of\_empty\_cols\_after\_last\_col +

                            compliance\_headers\_dict[compliance\_header\_str]).value = compliance\_header\_str

            format\_header\_cells(file\_sheet, [last\_column\_loc[0],

                                             last\_column\_loc[1] + no\_of\_empty\_cols\_after\_last\_col +

                                             compliance\_headers\_dict[compliance\_header\_str]], 1, 0, color\_coding\_blue, 15)

            for index, row in final\_data\_frame.iterrows():

                if (row[compliance\_header\_str] not in [None, 'None']) :

                    file\_sheet.cell(row['Row No in Excel Sheet'],

                                    last\_column\_loc[1] + no\_of\_empty\_cols\_after\_last\_col +

                                    compliance\_headers\_dict[compliance\_header\_str]).value = row[compliance\_header\_str]

**def write\_alerts\_compliance\_status\_in\_excel\_sheet** update an Excel sheet with compliance information

1. **Find Position of Last Column**:
   * It identifies the location of the last column in the Excel sheet, based on the provided string (**last\_col\_str**), with the function **find\_last\_col\_position**.
2. **Processing Compliance Headers**:
   * It iterates through the compliance headers list (**compliance\_headers\_list**) and checks if these headers exist in the columns of the provided **final\_data\_frame**.
3. **Updating Excel Sheet**:
   * For each compliance header present in **final\_data\_frame**, it writes the compliance header to the Excel sheet at a specified position relative to the last column.
   * It formats the header cell using the **format\_header\_cells** function with specific parameters (**color\_coding\_blue** and a column width of 15).
4. **Updating Compliance Information**:
   * It iterates through the rows of **final\_data\_frame** and updates the compliance information in the Excel sheet for the corresponding row numbers and compliance headers.

The function effectively populates the Excel sheet with compliance information from the provided data frame, aligning it with the specific row numbers and compliance headers, as detailed in the **compliance\_headers\_dict**.