import pandas as pd

import numpy as np

import datacompy

import xlsxwriter

from datetime import datetime

import os

import zipfile

from io import BytesIO

import logging

from typing import Dict, List, Optional, Tuple

from backend.profile\_utils import (

    generate\_distribution\_plot,

    generate\_frequency\_plot,

    generate\_comparison\_plot,

    generate\_comparison\_rows,

    calculate\_column\_stats

)

logger = logging.getLogger(\_\_name\_\_)

def generate\_mismatch\_html(comparison):

    """Generate HTML for mismatches with proper error handling."""

    try:

        mismatches\_html = []

        # Handle source-only rows

        if hasattr(comparison, 'df1\_unq\_rows') and not comparison.df1\_unq\_rows.empty:

            mismatches\_html.append("<h3>Rows Only in Source</h3>")

            mismatches\_html.append(comparison.df1\_unq\_rows.head(5).to\_html())

        # Handle target-only rows

        if hasattr(comparison, 'df2\_unq\_rows') and not comparison.df2\_unq\_rows.empty:

            mismatches\_html.append("<h3>Rows Only in Target</h3>")

            mismatches\_html.append(comparison.df2\_unq\_rows.head(5).to\_html())

        if mismatches\_html:

            return "\n".join(mismatches\_html)

        return "<p>No mismatches found</p>"

    except Exception as e:

        logger.warning(f"Error generating mismatch HTML: {str(e)}")

        return f"<p>Error generating mismatch details: {str(e)}</p>"

def generate\_datacompy\_report(source\_df: pd.DataFrame, target\_df: pd.DataFrame,

                            join\_columns: List[str], mapping\_df: pd.DataFrame,

                            join\_mappings: Dict[str, str]) -> Tuple[BytesIO, BytesIO]:

    """Generate a DataCompy comparison report."""

    try:

        # Get excluded columns

        excluded\_columns = mapping\_df[mapping\_df['Exclude from Comparison']]['Source Column'].tolist()

        # Create mapping dictionary

        column\_mapping = dict(zip(

            mapping\_df['Source Column'],

            mapping\_df['Target Column']

        ))

        # Filter out unmapped and excluded columns

        valid\_columns = {

            src: tgt for src, tgt in column\_mapping.items()

            if tgt and not pd.isna(tgt) and src not in excluded\_columns

        }

        # Get all columns that need to be processed

        source\_cols = list(valid\_columns.keys())

        target\_cols = [valid\_columns[src] for src in source\_cols]

        # Create unique column names for comparison

        source\_rename = {}

        target\_rename = {}

        final\_join\_columns = []

        # First, handle join columns to ensure they're unique

        for src\_col in join\_columns:

            tgt\_col = join\_mappings[src\_col]

            unique\_join\_name = f"join\_{src\_col}"

            source\_rename[src\_col] = unique\_join\_name

            target\_rename[tgt\_col] = unique\_join\_name

            final\_join\_columns.append(unique\_join\_name)

        # Then handle comparison columns

        for src\_col, tgt\_col in valid\_columns.items():

            if src\_col not in join\_columns:  # Skip if it's already a join column

                unique\_col\_name = f"compare\_{src\_col}"

                source\_rename[src\_col] = unique\_col\_name

                target\_rename[tgt\_col] = unique\_col\_name

        # Create copies of DataFrames with only needed columns

        source\_compare = source\_df[list(set(source\_cols + join\_columns))].copy()

        target\_compare = target\_df[list(set(target\_cols + [join\_mappings[src] for src in join\_columns]))].copy()

        # Rename columns to ensure uniqueness

        source\_compare.rename(columns=source\_rename, inplace=True)

        target\_compare.rename(columns=target\_rename, inplace=True)

        # Update join columns for the comparison

        join\_columns = final\_join\_columns

        logger.info(f"Source columns after renaming: {source\_compare.columns.tolist()}")

        logger.info(f"Target columns after renaming: {target\_compare.columns.tolist()}")

        logger.info(f"Join columns for comparison: {join\_columns}")

        # Create comparison object with proper configuration

        try:

            comparison = datacompy.Compare(

                df1=source\_compare,

                df2=target\_compare,

                join\_columns=join\_columns,

                df1\_name='Source',

                df2\_name='Target',

                on\_index=False

            )

        except Exception as e:

            logger.warning(f"Initial comparison failed: {str(e)}, trying with string conversion")

            # Convert all columns to string type for comparison

            source\_compare = source\_compare.astype(str)

            target\_compare = target\_compare.astype(str)

            comparison = datacompy.Compare(

                df1=source\_compare,

                df2=target\_compare,

                join\_columns=join\_columns,

                df1\_name='Source',

                df2\_name='Target',

                on\_index=False

            )

        # Generate Excel report

        excel\_output = BytesIO()

        with pd.ExcelWriter(excel\_output, engine='xlsxwriter') as writer:

            # Write summary

            summary\_data = {

                'Metric': [

                    'Rows in Source',

                    'Rows in Target',

                    'Rows in Common',

                    'Rows Only in Source',

                    'Rows Only in Target',

                    'Columns Match',

                    'All Row Values Match'

                ],

                'Value': [

                    len(source\_compare),

                    len(target\_compare),

                    comparison.intersect\_rows,

                    len(comparison.df1\_unq\_rows) if hasattr(comparison, 'df1\_unq\_rows') else 0,

                    len(comparison.df2\_unq\_rows) if hasattr(comparison, 'df2\_unq\_rows') else 0,

                    comparison.all\_columns\_match(),

                    comparison.matches()

                ]

            }

            pd.DataFrame(summary\_data).to\_excel(writer, sheet\_name='Summary', index=False)

            # Write column stats

            if hasattr(comparison, 'column\_stats'):

                comparison.column\_stats.to\_excel(writer, sheet\_name='Column Stats', index=True)

            # Write sample mismatches with proper error handling

            try:

                mismatches\_df = pd.DataFrame()

                # Get source-only rows

                if hasattr(comparison, 'df1\_unq\_rows') and not comparison.df1\_unq\_rows.empty:

                    source\_mismatches = comparison.df1\_unq\_rows.head(5).copy()

                    source\_mismatches['Match Type'] = 'Source Only'

                    mismatches\_df = pd.concat([mismatches\_df, source\_mismatches])

                # Get target-only rows

                if hasattr(comparison, 'df2\_unq\_rows') and not comparison.df2\_unq\_rows.empty:

                    target\_mismatches = comparison.df2\_unq\_rows.head(5).copy()

                    target\_mismatches['Match Type'] = 'Target Only'

                    mismatches\_df = pd.concat([mismatches\_df, target\_mismatches])

                if not mismatches\_df.empty:

                    mismatches\_df.to\_excel(writer, sheet\_name='Sample Mismatches', index=True)

                else:

                    pd.DataFrame({'Status': ['No mismatches found']}).to\_excel(

                        writer, sheet\_name='Sample Mismatches', index=False)

            except Exception as e:

                logger.warning(f"Error generating sample mismatches: {str(e)}")

                pd.DataFrame({'Error': [f'Failed to generate sample mismatches: {str(e)}']}).to\_excel(

                    writer, sheet\_name='Sample Mismatches', index=False)

        # Generate HTML report

        html\_output = BytesIO()

        html\_report = f"""

        <html>

        <head>

            <title>DataCompy Comparison Report</title>

            <style>

                body {{ font-family: Arial, sans-serif; margin: 20px; }}

                table {{ border-collapse: collapse; width: 100%; margin-bottom: 20px; }}

                th, td {{ border: 1px solid #ddd; padding: 8px; text-align: left; }}

                th {{ background-color: #f2f2f2; }}

                .pass {{ color: green; }}

                .fail {{ color: red; }}

                .section {{ margin-bottom: 30px; }}

            </style>

        </head>

        <body>

            <h1>DataCompy Comparison Report</h1>

            <div class="section">

                <h2>Summary</h2>

                <p>Source rows: {len(source\_compare)}</p>

                <p>Target rows: {len(target\_compare)}</p>

                <p>Rows in common: {comparison.intersect\_rows}</p>

                <p>Rows only in source: {len(comparison.df1\_unq\_rows) if hasattr(comparison, 'df1\_unq\_rows') else 0}</p>

                <p>Rows only in target: {len(comparison.df2\_unq\_rows) if hasattr(comparison, 'df2\_unq\_rows') else 0}</p>

                <p>Columns match: <span class="{'pass' if comparison.all\_columns\_match() else 'fail'}">{comparison.all\_columns\_match()}</span></p>

                <p>All rows match: <span class="{'pass' if comparison.matches() else 'fail'}">{comparison.matches()}</span></p>

            </div>

            <div class="section">

                <h2>Column Statistics</h2>

                {comparison.column\_stats.to\_html() if hasattr(comparison, 'column\_stats') else '<p>No column statistics available</p>'}

            </div>

            <div class="section">

                <h2>Sample Mismatches</h2>

                {generate\_mismatch\_html(comparison)}

            </div>

        </body>

        </html>

        """

        html\_output.write(html\_report.encode('utf-8'))

        html\_output.seek(0)

        excel\_output.seek(0)

        return excel\_output, html\_output

    except Exception as e:

        logger.error(f"Error generating DataCompy report: {str(e)}")

        raise Exception(f"Failed to generate DataCompy report: {str(e)}")

# Rest of the file remains unchanged...