import os

import pandas as pd

from dateutil.parser import parse

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import LabelEncoder

def preprocess\_csv\_folder(input\_folder, output\_folder=r"Preprocced Output Folder Path", handle\_missing\_values=True, dominance\_threshold=0.8,

                          date\_format=None, high\_cardinality\_threshold=100):

    """

    Reads all CSV files in a folder, processes them, and stores the cleaned files in an output folder.

    Args:

        input\_folder (str): Path to the folder containing CSV files.

        output\_folder (str, optional): Path to the folder where cleaned files will be stored.

            Defaults to "cleaned".

        handle\_missing\_values (bool, optional): Whether to impute missing values (default: True).

            Defaults to True.

        dominance\_threshold (float, optional): Threshold for considering a data type dominant

            in a mixed-type column (0.0 to 1.0). Defaults to 0.8.

        date\_format (str, optional): Specific date format to use for parsing dates (e.g., '%Y-%m-%d').

            Defaults to None (automatic detection).

        high\_cardinality\_threshold (int, optional): Threshold for high cardinality categorical variables.

            Defaults to 100.

    """

    if not os.path.exists(output\_folder):

        os.makedirs(output\_folder)

    for filename in os.listdir(input\_folder):

        if filename.endswith(".csv"):

            filepath = os.path.join(input\_folder, filename)

            df = pd.read\_csv(filepath)

            # Dynamic data type detection and row removal for mixed types

            for col in df.columns:

                if df[col].dtype == 'object':

                    try:

                        df[col] = df[col].apply(parse)

                        df[col] = pd.to\_datetime(df[col], errors='coerce')

                    except (ValueError, TypeError):

                        pass

                value\_counts = df[col].apply(lambda x: type(x)).value\_counts()

                if len(value\_counts) > 1:  # Check for mixed data types

                    dominant\_dtype = value\_counts.idxmax()

                    if value\_counts.iloc[0] / len(df) > dominance\_threshold:

                        df = df[df[col].apply(lambda x: type(x)) == dominant\_dtype]

                        print(f"Removed rows with non-{dominant\_dtype} data type in column '{col}' in file '{filename}'.")

                    else:

                        print(f"Column '{col}' in file '{filename}' has mixed data types. Consider manual cleaning or conversion.")

            # Detect and convert date columns

            potential\_date\_cols = []

            for col in df.columns:

                if df[col].dtype == 'datetime64[ns]':

                    potential\_date\_cols.append(col)

            if potential\_date\_cols:

                if date\_format:

                    for col in potential\_date\_cols:

                        df[col] = df[col].dt.strftime(date\_format)

                else:

                    first\_format = df[potential\_date\_cols[0]].dt.strftime('%Y-%m-%d')[0]

                    for col in potential\_date\_cols[1:]:

                        if df[col].dt.strftime('%Y-%m-%d')[0] != first\_format:

                            print(f"Warning: Inconsistent date formats detected in columns {', '.join(potential\_date\_cols)} in file '{filename}'.")

            # Handle missing values (optional)

            if handle\_missing\_values:

                imputers = {

                    'int64': SimpleImputer(strategy='mean'),

                    'float64': SimpleImputer(strategy='mean'),

                    'object': SimpleImputer(strategy='most\_frequent')

                }

                missing\_cols = df.columns[df.isna().any()]

                for col in missing\_cols:

                    imputer = imputers.get(str(df[col].dtype), None)

                    if imputer:

                        df[[col]] = imputer.fit\_transform(df[[col]])

            # Identify and encode low cardinality categorical variables

            for col in df.select\_dtypes(include=['object']).columns:

                if df[col].nunique() <= high\_cardinality\_threshold:

                    encoder = LabelEncoder()

                    encoded\_col\_name = f"encoding\_{col}"

                    df[encoded\_col\_name] = encoder.fit\_transform(df[col])

                    print(f"Encoded column '{col}' as '{encoded\_col\_name}' in file '{filename}'.")

            output\_filepath = os.path.join(output\_folder, filename)

            df.to\_csv(output\_filepath, index=False)

            print(f"Processed and saved cleaned file '{output\_filepath}'.")

# Example usage

input\_folder = r"Input folder path"

preprocess\_csv\_folder(input\_folder)