import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler, StandardScaler

from sklearn.impute import SimpleImputer

from datetime import datetime

def load\_data(tnpcb\_path, imd\_path=None, supplementary\_path=None):

"""Loads air quality and meteorological data."""

tnpcb\_df = pd.read\_csv(tnpcb\_path, parse\_dates=['Timestamp']) # Assuming 'Timestamp' column

tnpcb\_df = tnpcb\_df.set\_index('Timestamp')

print(f"TNPCB data loaded with shape: {tnpcb\_df.shape}")

imd\_df = None

if imd\_path:

imd\_df = pd.read\_csv(imd\_path, parse\_dates=['Timestamp']) # Assuming 'Timestamp' column

imd\_df = imd\_df.set\_index('Timestamp')

print(f"IMD data loaded with shape: {imd\_df.shape}")

supplementary\_df = None

if supplementary\_path:

# Adjust loading based on the type of supplementary data (e.g., CSV, GeoJSON)

supplementary\_df = pd.read\_csv(supplementary\_path, parse\_dates=['Timestamp'] if 'Timestamp' in pd.read\_csv(supplementary\_path).columns else None)

if 'Timestamp' in supplementary\_df.columns:

supplementary\_df = supplementary\_df.set\_index('Timestamp')

print(f"Supplementary data loaded with shape: {supplementary\_df.shape}")

return tnpcb\_df, imd\_df, supplementary\_df

def preprocess\_missing\_values(df, strategy='interpolate'):

"""Handles missing values using different strategies."""

if df is None:

return None

print(f"Handling missing values using strategy: {strategy}")

if strategy == 'interpolate':

df = df.interpolate(method='linear')

elif strategy == 'mean':

imputer = SimpleImputer(strategy='mean')

df = pd.DataFrame(imputer.fit\_transform(df), index=df.index, columns=df.columns)

elif strategy == 'median':

imputer = SimpleImputer(strategy='median')

df = pd.DataFrame(imputer.fit\_transform(df), index=df.index, columns=df.columns)

df = df.dropna() # Remove any remaining NaNs after interpolation

print(f"Shape after handling missing values: {df.shape}")

return df

def merge\_data(tnpcb\_df, imd\_df=None, supplementary\_df=None):

"""Merges air quality and meteorological data based on timestamp."""

merged\_df = tnpcb\_df

if imd\_df is not None:

merged\_df = pd.merge(merged\_df, imd\_df, left\_index=True, right\_index=True, how='left')

print(f"Merged with IMD data, new shape: {merged\_df.shape}")

if supplementary\_df is not None:

merged\_df = pd.merge(merged\_df, supplementary\_df, left\_index=True, right\_index=True, how='left')

print(f"Merged with supplementary data, new shape: {merged\_df.shape}")

return merged\_df

def create\_time\_features(df):

"""Creates time-based features."""

df['hour'] = df.index.hour

df['day\_of\_week'] = df.index.dayofweek

df['day\_of\_year'] = df.index.dayofyear

df['month'] = df.index.month

df['year'] = df.index.year

df['sin\_hour'] = np.sin(2 \* np.pi \* df['hour'] / 24)

df['cos\_hour'] = np.cos(2 \* np.pi \* df['hour'] / 24)

df['sin\_day'] = np.sin(2 \* np.pi \* df['day\_of\_year'] / 365)

df['cos\_day'] = np.cos(2 \* np.pi \* df['day\_of\_year'] / 365)

return df

def create\_lagged\_features(df, pollutants, lags):

"""Creates lagged features for specified pollutants."""

for pollutant in pollutants:

for lag in range(1, lags + 1):

df[f'{pollutant}lag{lag}'] = df[pollutant].shift(lag)

df = df.dropna() # Drop rows with NaN due to lags

return df

def preprocess\_data(tnpcb\_path, imd\_path=None, supplementary\_path=None, missing\_strategy='interpolate', lags=24, pollutants\_to\_lag=['PM2.5', 'PM10', 'O3', 'NO2', 'SO2', 'CO']):

"""Orchestrates the data loading and preprocessing steps."""

tnpcb\_df, imd\_df, supplementary\_df = load\_data(tnpcb\_path, imd\_path, supplementary\_path)

tnpcb\_df = preprocess\_missing\_values(tnpcb\_df, missing\_strategy)

imd\_df = preprocess\_missing\_values(imd\_df, missing\_strategy)

supplementary\_df = preprocess\_missing\_values(supplementary\_df, missing\_strategy)

merged\_df = merge\_data(tnpcb\_df, imd\_df, supplementary\_df)

merged\_df = create\_time\_features(merged\_df)

merged\_df = create\_lagged\_features(merged\_df, pollutants\_to\_lag, lags)

return merged\_df

def split\_data(df, target\_pollutant, test\_size=0.2, shuffle=False):

"""Splits the data into training and testing sets."""

X = df.drop(columns=[target\_pollutant])

y = df[target\_pollutant]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=test\_size, shuffle=shuffle, random\_state=42) # For reproducibility

return X\_train, X\_test, y\_train, y\_test

def scale\_data(X\_train, X\_test, scaler\_type='standard'):

"""Scales the features using MinMaxScaler or StandardScaler."""

if scaler\_type == 'minmax':

scaler = MinMaxScaler()

elif scaler\_type == 'standard':

scaler = StandardScaler()

else:

raise ValueError("Invalid scaler\_type. Choose 'minmax' or 'standard'.")

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

return X\_train\_scaled, X\_test\_scaled, scaler

if \_name\_ == '\_main\_':

# Example usage: Replace with your actual file paths and target pollutant

tnpcb\_file = 'salem\_air\_quality.csv'

imd\_file = 'salem\_weather.csv'

processed\_df = preprocess\_data(tnpcb\_file, imd\_file)

print("Processed DataFrame head:")

print(processed\_df.head())

target = 'PM2.5'

X\_train, X\_test, y\_train, y\_test = split\_data(processed\_df, target)

X\_train\_scaled, X\_test\_scaled, scaler = scale\_data(X\_train, X\_test)

print(f"X\_train shape: {X\_train\_scaled.shape}")

print(f"X\_test shape: {X\_test\_scaled.shape}")

print(f"y\_train shape: {y\_train.shape}")

print(f"y\_test shape: {y\_test.shape}")