This code is designed to process, transform, and cluster time series data from .csv files in a specified folder, testing\_data\_trial\_2. It specifically works with VRM (Red Phase Voltage) data, performing both Fourier and LSTM-based transformations, and clustering based on combined features. Here’s a line-by-line breakdown of the code:

**Import Libraries**

python

Copy code

import pandas as pd

import numpy as np

import os

from sklearn.cluster import KMeans

from sklearn.preprocessing import MinMaxScaler

from keras.\_tf\_keras.keras.models import Sequential

from keras.\_tf\_keras.keras.layers import Dense,LSTM

* **Pandas and Numpy**: For data manipulation and numerical operations.
* **OS**: To handle file paths and directory operations.
* **sklearn.cluster.KMeans**: For clustering processed data.
* **sklearn.preprocessing.MinMaxScaler**: For scaling the VRM data.
* **Keras**: Used to create and train the LSTM model.

**Preprocess CSV Files**

python

Copy code

def preprocess\_csv(file\_path):

try:

df = pd.read\_csv(file\_path)

vrm\_data = df['VRM'].values.reshape(-1, 1)

* **pd.read\_csv**: Loads the CSV file.
* **df['VRM']**: Extracts the VRM column (Red Phase Voltage) and reshapes it to be 2D, required for scaling.

**Handle NaN Values**

python

Copy code

if np.any(np.isnan(vrm\_data)):

print(f"NaN values found in {file\_path}.")

if np.count\_nonzero(~np.isnan(vrm\_data)) > 0:

mean\_value = np.nanmean(vrm\_data)

print(f"Filling NaNs with the mean: {mean\_value}")

vrm\_data = np.nan\_to\_num(vrm\_data, nan=mean\_value)

else:

print(f"All values are NaN in {file\_path}. Skipping this file.")

return None

* **np.isnan**: Checks for any NaN values in the VRM data.
* **np.nanmean**: Calculates the mean of non-NaN values if there are some valid values in the column.
* **np.nan\_to\_num**: Replaces NaN values with the calculated mean. If all values are NaN, the function returns None and skips that file.

**Scaling**

python

Copy code

scaler = MinMaxScaler()

vrm\_data\_scaled = scaler.fit\_transform(vrm\_data)

return vrm\_data\_scaled

except Exception as e:

print(f"Error processing {file\_path}: {e}")

return None

* **MinMaxScaler**: Scales VRM data between 0 and 1 for normalization.
* **vrm\_data\_scaled**: Returns the scaled VRM data for further processing.

**Fourier Transform Function**

python

Copy code

def fourier\_transform(data):

fft\_data = np.fft.fft(data)

fft\_data = np.abs(fft\_data)

return fft\_data

* **np.fft.fft**: Computes the Fourier transform of the data to capture frequency-based features.
* **np.abs**: Converts complex Fourier output to its magnitude, allowing the use of real values for clustering.

**Process Each CSV File**

python

Copy code

folder\_path = 'testing\_data\_trial\_2'

processed\_data = []

file\_names = []

for filename in os.listdir(folder\_path):

if filename.endswith('.csv'):

file\_path = os.path.join(folder\_path, filename)

data = preprocess\_csv(file\_path)

if data is not None:

processed\_data.append(data)

file\_names.append(filename)

* **os.listdir**: Lists all files in the folder.
* **if filename.endswith('.csv')**: Ensures only .csv files are processed.
* **preprocess\_csv(file\_path)**: Calls the preprocessing function on each file and appends the processed data and filename.

**Sequence Preparation for LSTM Model**

python

Copy code

sequence\_length = 10

X = []

for data in processed\_data:

for i in range(len(data) - sequence\_length + 1):

X.append(data[i:i + sequence\_length])

X = np.array(X)

* Defines the **sequence\_length** as 10.
* Creates sequences of length 10 by sliding a window across the VRM data to feed into the LSTM model.

**Check and Exit for NaN Values**

python

Copy code

if np.any(np.isnan(X)):

print("NaN values found in X. Exiting.")

exit()

* Ensures that X does not contain any NaN values before feeding into the model.

**Define and Train LSTM Model**

python

Copy code

model = Sequential()

model.add(LSTM(50, activation='relu', input\_shape=(sequence\_length, 1)))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mse')

model.fit(X, X, epochs=10, batch\_size=32)

* **Sequential Model**: Defines a model structure.
* **LSTM Layer**: Adds an LSTM layer with 50 units and a ReLU activation function.
* **Dense Layer**: Adds an output dense layer with 1 unit.
* **model.fit**: Trains the LSTM model on sequences X.

**Extract Features Using LSTM and Fourier Transform**

python

Copy code

lstm\_features = model.predict(X)

lstm\_features = lstm\_features.reshape(lstm\_features.shape[0], -1)

fourier\_features = [fourier\_transform(data.flatten()) for data in processed\_data]

* **lstm\_features**: Extracts features from the trained LSTM model predictions.
* **fourier\_features**: Computes Fourier transform features for each sequence.

**Combine LSTM and Fourier Features**

python

Copy code

combined\_features = []

for lstm\_f, fourier\_f in zip(lstm\_features, fourier\_features):

combined\_features.append(np.concatenate((lstm\_f, fourier\_f[:10])))

* **combined\_features**: Concatenates the LSTM features and the first 10 Fourier features.

**Clustering with KMeans**

python

Copy code

kmeans = KMeans(n\_clusters=10, random\_state=0)

kmeans.fit(combined\_features)

labels = kmeans.labels\_

* **KMeans**: Clusters the combined features into 10 clusters.
* **labels**: Stores the cluster label for each file.

**Create Folders and Move Files by Cluster**

python

Copy code

os.makedirs('Signature Fault Clusters/VRM', exist\_ok=True)

for i in range(10):

os.makedirs(os.path.join('Signature Fault Clusters/VRM', f'VRM Cluster {i}'), exist\_ok=True)

for i, filename in enumerate(file\_names):

cluster\_label = labels[i]

source\_path = os.path.join(folder\_path, filename)

destination\_path = os.path.join('Signature Fault Clusters/VRM', f'VRM Cluster {cluster\_label}', filename)

os.rename(source\_path, destination\_path)

* **os.makedirs**: Creates folders for each cluster.
* **os.rename**: Moves each CSV file to its respective cluster folder based on its assigned cluster label.

This code clusters VRM data from CSV files based on time-series patterns detected by an LSTM model and frequency patterns detected by Fourier transformation. It organizes the files by cluster for easy analysis.