Here’s a detailed explanation of the provided code and its components:

**1. Importing Required Libraries**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import os

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

* **pandas**: For handling CSV files and tabular data.
* **numpy**: For numerical operations like adding noise.
* **matplotlib**: For optional visualization.
* **os**: For file and directory operations.
* **sklearn.metrics**: For calculating classification metrics.

**2. Reading the Dataset**

def read\_csv(file\_path):

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

print(f"Data loaded successfully from {file\_path}. Columns: {list(data.columns)}")

return data

This function reads the dataset from a CSV file and displays its column names. It returns the loaded data as a pandas DataFrame.

**3. Adding Gaussian Noise**

def add\_noise(data, columns, snr\_db):

noisy\_data = data.copy()

for column in columns:

if column in noisy\_data.columns:

signal\_power = np.mean(data[column] \*\* 2)

noise\_power = signal\_power / (10 \*\* (snr\_db / 10))

noise = np.random.normal(0, np.sqrt(noise\_power), len(data))

noisy\_data[column] += noise

print(f"Added noise to {column} with SNR: {snr\_db} dB")

else:

print(f"Column {column} not found in dataset. Skipping noise addition.")

return noisy\_data

* **Purpose**: Simulates real-world measurement errors by adding Gaussian noise to specified columns.
* **SNR (Signal-to-Noise Ratio)**: A higher SNR adds less noise, while a lower SNR introduces more distortion.

**Steps**:

1. Calculate the power of the signal in the column.
2. Derive the noise power based on the SNR.
3. Add noise to the signal.

**4. Introducing Anomalies**

def introduce\_anomalies(data, anomalies):

anomalous\_data = data.copy()

anomalous\_data['Label'] = 0 # Normal data has label 0

for anomaly in anomalies:

if anomaly['column'] in anomalous\_data.columns:

start, end = anomaly['start'], anomaly['end']

if anomaly['type'] == 'drop':

anomalous\_data.loc[start:end, anomaly['column']] = np.nan

anomalous\_data.loc[start:end, 'Label'] = 1 # Anomalous data labeled as 1

elif anomaly['type'] == 'spike':

anomalous\_data.loc[start:end, anomaly['column']] += anomaly['magnitude']

anomalous\_data.loc[start:end, 'Label'] = 1 # Anomalous data labeled as 1

return anomalous\_data

* **Purpose**: Introduces controlled anomalies into the data.
* **Types of Anomalies**:
  1. **Drop**: Sets data to NaN within a specified range.
  2. **Spike**: Adds a large value (magnitude) to simulate sudden changes.
* **Labels**:
  1. Normal data is labeled as 0.
  2. Anomalous data is labeled as 1 in the Label column.

**5. Evaluating Classification**

def evaluate\_classification(data):

true\_labels = data['True\_Label']

predicted\_labels = data['Label']

metrics = {

'Accuracy': accuracy\_score(true\_labels, predicted\_labels),

'Precision': precision\_score(true\_labels, predicted\_labels, zero\_division=0),

'Recall': recall\_score(true\_labels, predicted\_labels, zero\_division=0),

'F1-Score': f1\_score(true\_labels, predicted\_labels, zero\_division=0)

}

for metric, value in metrics.items():

print(f"{metric}: {value:.4f}")

return metrics

* **Purpose**: Measures how well the introduced anomalies (labels) align with ground truth (True\_Label).
* **Metrics**:
  + **Accuracy**: Correct predictions / Total predictions.
  + **Precision**: Proportion of correctly identified anomalies among all predicted anomalies.
  + **Recall**: Proportion of actual anomalies correctly identified.
  + **F1-Score**: Harmonic mean of precision and recall.

**6. Saving Processed Data**

def save\_csv(data, file\_path):

data.to\_csv(file\_path, index=False)

print(f"Processed data saved to {file\_path}")

Saves the processed dataset with added noise and anomalies to a new CSV file.

**7. Processing All Files**

def process\_all\_files(input\_dir, output\_dir, snr, anomalies):

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

os.makedirs(output\_dir)

for file\_name in os.listdir(input\_dir):

if file\_name.endswith('.csv'):

file\_path = os.path.join(input\_dir, file\_name)

output\_path = os.path.join(output\_dir, file\_name.replace('.csv', '\_processed.csv'))

data = read\_csv(file\_path)

data['True\_Label'] = 0

numeric\_columns = data.select\_dtypes(include=[np.number]).columns.tolist()

noisy\_data = add\_noise(data, numeric\_columns, snr)

anomalous\_data = introduce\_anomalies(noisy\_data, anomalies)

metrics = evaluate\_classification(anomalous\_data)

save\_csv(anomalous\_data, output\_path)

* **Purpose**: Automates processing for multiple CSV files in a directory.
* **Steps**:
  1. Reads each file.
  2. Adds noise and introduces anomalies.
  3. Evaluates classification performance.
  4. Saves the processed file.

**8. Example Usage**

if \_\_name\_\_ == "\_\_main\_\_":

input\_directory = "PMU\_Data\_with\_Anomalies and Events"

output\_directory = "Processed\_PMU\_Data"

signal\_to\_noise\_ratio = 50

anomaly\_list = [

{'type': 'drop', 'column': 'Voltage', 'start': 100, 'end': 200},

{'type': 'spike', 'column': 'Frequency', 'start': 300, 'end': 350, 'magnitude': 0.5}

]

process\_all\_files(input\_directory, output\_directory, signal\_to\_noise\_ratio, anomaly\_list)

* **Input**: Directory with raw PMU datasets.
* **Output**: Directory with processed datasets.
* **Parameters**:
  + SNR for noise addition.
  + List of anomalies with types, columns, ranges, and magnitudes.

**Workflow**

1. Read raw data.
2. Add noise to numeric columns.
3. Introduce anomalies with specified types and ranges.
4. Generate labels for normal and anomalous data.
5. Evaluate classification accuracy.
6. Save the processed files.

This setup provides a comprehensive pipeline for data simulation and anomaly detection evaluation. Let me know if you need additional explanations!