Flow of Code for Fuel Level and Signal Processing

Summary of Workflow:

1. **Mount Google Drive**: Access files from Google Drive.
2. **Import Libraries**: Load required libraries for data processing.
3. **Check Latest Stable Value** (optional): Determines stable fuel level over time.
4. **Butterworth Low-Pass Filter**: Smooths noisy signals by filtering out high frequencies.
5. **Load Data from Files**: Load fuel level and speed data from JSON files into a DataFrame.
6. **Assign Folder Path**: Locate and list all JSON files in subdirectories.
7. **Generate DataFrame**: Convert loaded JSON data into a Pandas DataFrame.
8. **Set Timestamp Index**: Set the timestamp column as the DataFrame index for time-based operations.
9. **Resample Data**: Resample the data into 1-minute intervals, using forward fill to handle gaps.
10. **Signal Smoothing**:
    - Apply the **Savitzky-Golay filter** to reduce high-frequency noise.
    - Apply the **Butterworth low-pass filter** for further noise reduction.
11. **Plot the Results**: Visualize the smoothed fuel level and speed over time.

1. Mount Google Drive

Purpose: Mounting Google Drive is essential to access and load JSON files stored in the Google Drive directory. This enables you to work with the data directly from your drive.

Steps:

- Use the following code to mount Google Drive:

```
from google.colab import drive
drive.mount('/content/drive')
```

This will allow you to interact with the files stored in your Google Drive from your notebook environment.

2. Import Relevant Libraries

Purpose: Import necessary Python libraries to handle the JSON files, process the data, apply filters, and visualize the results.

Key Libraries:

- **os, glob**: Used for file handling and locating JSON files in directories.
- **json**: Parses JSON files.
- **pandas**: For creating and manipulating DataFrames.
- **matplotlib.pyplot**: For plotting the processed data.
- **scipy.signal**: For applying signal processing filters (Savitzky-Golay and Butterworth).
- **timedelta**: To handle time-based data.

```
import os
import json
import glob
import pandas as pd
import matplotlib.pyplot as plt
from scipy.signal import savgol_filter, butter, filtfilt
from datetime import timedelta
```

3. Create Function to Check the Latest Stable Value (Optional)

Purpose: Although this function is not currently in use, it is designed to identify the most recent "stable" fuel level, i.e., when the fuel level has remained within a certain threshold for a specified duration.

Code:

```
def get_latest_stable_fuel_level(df, stability_threshold=10,
stability_duration_minutes=10):
    """
    Identifies the latest stable fuel level based on the given
stability threshold and duration.
    """
    df['timestamp'] = pd.to_datetime(df.index)
    stability_duration = timedelta(minutes=stability_duration_minutes)
    stable_level = df['smoothed_fuel_level'].iloc[0]
    stable_start_time = df['timestamp'].iloc[0]
    latest_stable_level = stable_level

    for i, row in df.iterrows():
        current_time = row['timestamp']
        current_level = row['smoothed_fuel_level']
        if abs(current_level - stable_level) > stability_threshold:
            if current_time - stable_start_time >= stability_duration:
                latest_stable_level = stable_level
            stable_level = current_level
            stable_start_time = current_time
    return latest_stable_level
```

4. Function to Execute Low-Pass Butterworth Filter

Purpose: This function applies the **Butterworth low-pass filter** to smooth noisy data by allowing low-frequency signals to pass while reducing high-frequency noise.

Code:

```
def butter_lowpass_filter(data, cutoff, fs, order=4):
    """
    Applies a Butterworth low-pass filter to the data to remove
high-frequency noise.
    """
    nyquist = 0.5 * fs
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    y = filtfilt(b, a, data)

    return y
```

5. Function to Load Data from JSON Files

Purpose: This function reads JSON files from the directory, extracts relevant information (`fuelLevelE2`, `speed`, and timestamps), and stores it in a DataFrame.

Code:

```
def load_data_from_files(json_files):
    """
    Loads data from JSON files and returns a flattened dataset.
    """
    all_data = []
    for file_path in json_files:
        with open(file_path, 'r') as file:
            try:
                data = json.load(file)
                geo_data = data["geoData"]
                for item in geo_data:
                    speed = float(item["speed"])
                    fuel_level =
int(item["fuelLevelE2"]["$numberInt"])
                    flattened_item = {
                        "timestamp":
pd.to_datetime(item["timeStamp"]["$date"]["$numberLong"], unit="ms"),
                        "fuelLevelE2": fuel_level,
                        "speed": speed,
                    }
                    all_data.append(flattened_item)
            except json.JSONDecodeError as e:
                print(f"Error decoding JSON in file {file_path}: {e}")
            except KeyError as e:
                print(f"Missing key in JSON data: {e}")
    return all_data
```

---

6. Assign Folder Path and Get All JSON Files in the Subdirectories

Purpose: Set the folder path and use `glob` to search for all JSON files recursively in the directory.

Code:

```
folder_path = '/content/drive/MyDrive/json_files/'  # Update this to your actual directory

json_files = glob.glob(os.path.join(folder_path, '**', '*.json'), recursive=True)
```

---

7. Generate DataFrame

Purpose: Convert the JSON data into a Pandas DataFrame for easier data manipulation.

Code:

```
df = pd.DataFrame(load_data_from_files(json_files))
```

---

8. Set Timestamp as Index

Purpose: Set the `timestamp` column as the index of the DataFrame to enable time-based operations.

Code:

```
df.set_index("timestamp", inplace=True)
```

9. Resample the DataFrame to 1-Minute Intervals and Forward Fill Missing Data

Purpose: Resample the DataFrame to ensure data is aligned at 1-minute intervals and use `ffill()` to propagate the last valid data point forward, filling any missing values.

Code:

```
resampled_df = df.resample("60S").mean().ffill()
```

- **Resampling**: Aggregates the data into 1-minute intervals.
- **Forward Filling (`ffill`)**: Fills in missing or NaN values by carrying the last valid data point forward.

10. Signal Smoothing Process

A. Savitzky-Golay Filter

Purpose: To apply the **Savitzky-Golay filter**, which fits a polynomial to a sliding window of data points, smoothing the data while preserving key features.

Code:

```
resampled_df['fuelLevelE2_smoothed_savgol'] =
savgol_filter(resampled_df['fuelLevelE2'], window_length=5,
polyorder=3)
```

- **`window_length=5`**: Defines the size of the sliding window.
- **`polyorder=3`**: Specifies the degree of the polynomial.

B. Butterworth Low-Pass Filter

Purpose: To apply the **Butterworth low-pass filter** to further smooth the already processed data from the Savitzky-Golay filter, reducing high-frequency noise.

Code:

```
cutoff_frequency = 0.01
sampling_rate = 2
order = 3



resampled_df['fuelLevelE2_smoothed_butter'] =
butter_lowpass_filter(resampled_df['fuelLevelE2_smoothed_savgol'],
cutoff_frequency, sampling_rate, order)

resampled_df['smoothed_fuel_level'] =
resampled_df['fuelLevelE2_smoothed_butter']
```

11. Plot the Graph

Purpose: Visualize the processed fuel level and speed data over time, with the option to include both raw and smoothed data for comparison.

Code:

```
plt.figure(figsize=(15, 10))
plt.plot(resampled_df.index, resampled_df['smoothed_fuel_level'],
color='blue', label='Smoothed Fuel Level (Liters)')
plt.plot(resampled_df.index, resampled_df['speed'], color='orange',
label='Speed')

plt.xlabel('Time')
plt.ylabel('Fuel Level (Liters)', color='blue')
plt.legend(loc='upper left')
plt.grid(True)
plt.show()
```