**eda\_peaks.py**

* The eda\_peaks function acts as a specialized "wave analyst" for your EDA data.
* It's about the phasic signal – the fast-changing "waves" that ride on top of the slow-moving "tide" (the tonic signal).
* **The function's main job is to:**
  1. Find all the significant waves (the Skin Conductance Responses, or SCRs).
  2. Clean up the findings to make sure they are valid (e.g., every peak has a starting point).
  3. Measure the key properties of each wave: its height (amplitude), how fast it rises (rise time), and how long it takes to fade away (recovery time).
* **It returns two things:**
  1. A "timeline" DataFrame showing where the peaks are for easy plotting.
  2. A detailed "report" dictionary containing all the measurements for each SCR.

**Imports:**

**import pandas as pd**

**from ..misc import find\_closest**

**from ..signal import signal\_formatpeaks**

**from .eda\_findpeaks import eda\_findpeaks**

**from .eda\_fixpeaks import eda\_fixpeaks**

* **pd**: For handling data in tables (DataFrames).
* **find\_closest**: A helper function to find the number in a list that is closest to a target value. This will be used for finding the recovery time.
* **signal\_formatpeaks**: A helper function that takes a list of peak locations and turns it into a full-length signal (a timeline with flags on it).
* eda\_findpeaks: The "spotter" function that does the initial, rough search for peaks.
* eda\_fixpeaks: The "quality control" function that cleans up the initial findings from the spotter.

The Main eda\_peaks Function

def eda\_peaks(eda\_phasic, sampling\_rate=1000, method="neurokit", amplitude\_min=0.1):

* **eda\_phasic**: The input phasic EDA signal (the "waves").
* **method**: The specific algorithm to use for finding peaks. Different methods exist based on different scientific papers.
* **amplitude\_min**: A threshold to ignore tiny fluctuations that aren't meaningful responses. Think of it as, "don't count ripples, only look for real waves."

**"""Find Skin Conductance Responses (SCR) in EDA"""**

* This is a docstring that explains what the function does, its parameters, and provides examples. It also importantly lists the keys for the output dictionary, like "SCR\_Amplitude" and "SCR\_Onsets".

if isinstance(eda\_phasic, (pd.DataFrame, pd.Series)):

try:

eda\_phasic = eda\_phasic["EDA\_Phasic"]

except KeyError:

eda\_phasic = eda\_phasic.values```

* This is the data formatting step. It makes sure that the input eda\_phasic is a simple list-like array of numbers, even if you passed in a more complex pandas DataFrame or Series object.

# Get basic features

info = eda\_findpeaks(eda\_phasic, sampling\_rate=sampling\_rate,

method=method, amplitude\_min=amplitude\_min)

info = eda\_fixpeaks(info)

* **info = eda\_findpeaks(...)**: It first calls the eda\_findpeaks "spotter" function. This function does the first pass over the signal and identifies the potential onsets (start of a wave) and peaks (top of a wave). The results are stored in the info dictionary.
* **info = eda\_fixpeaks(info)**: The initial results might have errors (e.g., a peak with no start, or a start with no peak). This line calls the eda\_fixpeaks "quality control" function to clean up these issues and ensure that we only have valid onset-peak pairs.

# Get additional features (rise time, half-recovery time, etc.)

info = \_eda\_peaks\_getfeatures(info, eda\_phasic, sampling\_rate,

recovery\_percentage=0.5)

* Now that we have a clean list of valid SCRs, this line calls the main "measurement" helper function, \_eda\_peaks\_getfeatures. This function takes the info dictionary and the signal, and calculates all the detailed metrics like amplitude, rise time, and recovery time, adding them to the info dictionary.

# Prepare output

peak\_signal = signal\_formatpeaks(info,

desired\_length=len(eda\_phasic),

peak\_indices=info["SCR\_Peaks"],

other\_indices=info["SCR\_Recovery"])

info["sampling\_rate"] = sampling\_rate # Add sampling rate to dict info

return peak\_signal, info

This block finalizes the output.

* **peak\_signal = signal\_formatpeaks(...)**: It calls a helper to create the peak\_signal DataFrame. This is essentially a timeline of the same length as your input signal, with 1s placed at the location of each peak and recovery point, and 0s everywhere else. It is very useful for plotting.
* **return peak\_signal, info**: It returns the two final products: the plottable timeline and the detailed report dictionary.

The "Measurement" Helper: \_eda\_peaks\_getfeatures

def \_eda\_peaks\_getfeatures(info, eda\_phasic, sampling\_rate=1000,

recovery\_percentage=0.5):

**1. Sanity checks & cleanup**

if len(onsets) != len(peaks):

raise ValueError(...)

* This section performs a final, critical cleanup. It ensures that every peak has a corresponding onset and vice-versa.
* The if len(onsets) != len(peaks): is a very crucial safety check. If the numbers don't match, it is impossible to calculate amplitude correctly, so the program stops with an error.

2. Calculate Amplitude & Rise Time:

# Amplitudes

amplitude[valid\_peaks] = info["SCR\_Height"][valid\_peaks] - eda\_phasic[onsets]

# Rise Times (in seconds)

risetime[valid\_peaks] = (peaks - onsets) / sampling\_rate

* **Amplitude**: For each valid peak, the amplitude is calculated as the value at the peak minus the value at the onset. This is the true height of the "wave".
* **Rise Time**: The rise time is calculated by (time of peak - time of onset). This result is divided by the sampling rate to convert it from "number of samples" into "seconds".

**3. Calculate Recovery Time:**

# Initialize arrays...

recovery\_values = eda\_phasic[onsets] + amplitude[valid\_peaks] \* recovery\_percentage

* It first calculates the target recovery value for each peak. If recovery\_percentage is 0.5 (for half-recovery), this value is:
  + the height of the signal at the onset + half of the amplitude. This is the height on the "downslope" that we are trying to find.

for i, peak\_index in enumerate(peaks):

* It loops through each peak one by one.

# Get segment between peak and next peak

segment = eda\_phasic[peak\_index : peaks[i + 1]]

* For the current peak, it creates the segment of the signal that starts at the peak and ends before the next peak begins. This is the "recovery zone".

# Adjust segment (cut when it reaches minimum)

segment = segment[0 : np.argmin(segment)]

* This prevents the code from accidentally finding the "recovery" point on the rising edge of the next SCR.

# Find recovery time

recovery\_value = find\_closest(...)

* It uses the find\_closest helper to search within this clean segment for the data point that is nearest to the target recovery time value.

# Detect recovery points for only if there are data points below recovery\_value

if np.min(segment) < recovery\_value:

* If the signal did recover, it finds the exact sample number and calculates the recovery time in seconds, storing both.