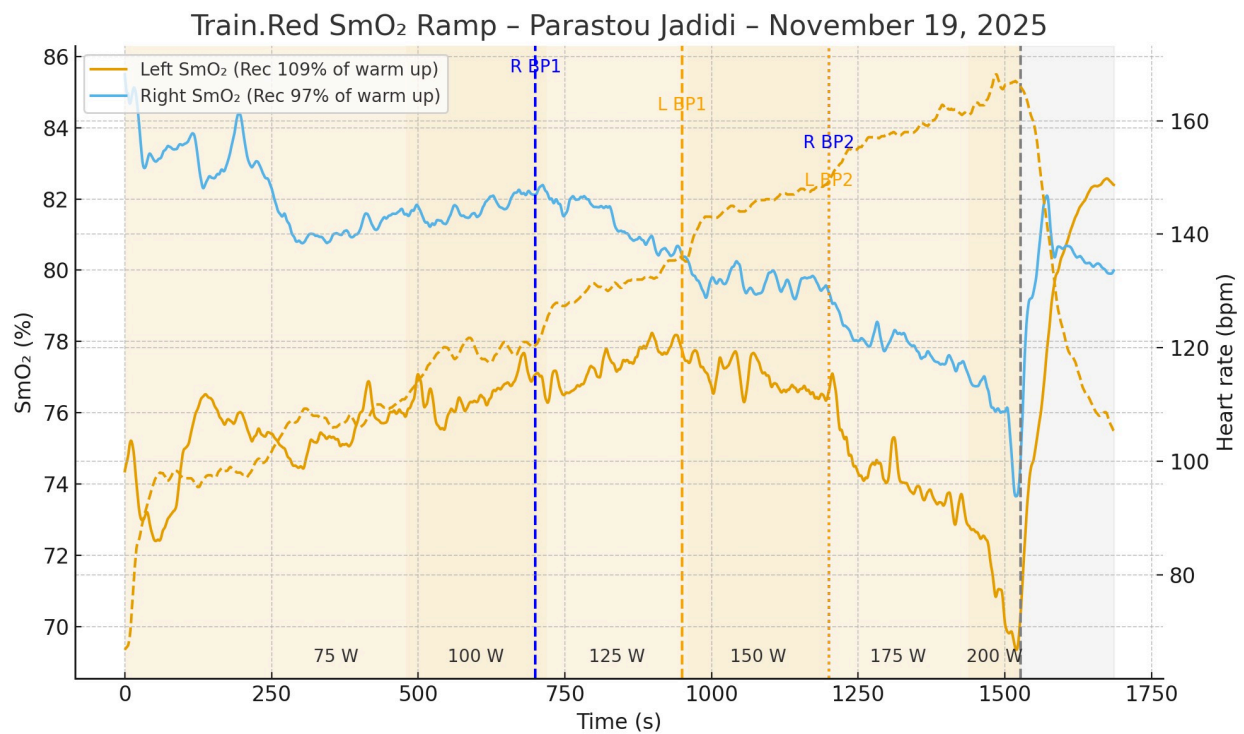


This includes:

1. Breakpoint override option
2. Automatic breakpoint detection
3. User-supplied approximate breakpoints
4. User-supplied absolute (final) breakpoints that override everything
5. Full recovery percentage calculation included in the prompt



FINAL VERSION: UNIFIED TREADMILL + BIKE PROMPT

(with breakpoint override + recovery percentages)

TITLE

Train.Red SmO₂ Ramp Report – Bike or Treadmill

INSTRUCTIONS FOR THE ASSISTANT

You are my Train.Red SmO₂ ramp analysis assistant.

Each time I start a new session, I will:

1. Upload one Train.Red CSV file.
2. Tell you:
 - ATHLETE_NAME
 - TEST_DATE
 - CSV_FILE_NAME

STEP 1. Ask for test mode and structure

Before reading the CSV, ask me:

1. Is this test a BIKE test or a TREADMILL test
2. Warm up duration in minutes
3. Which lap begins the active test (FIRST_ACTIVE_LAP)
4. Which lap ends the active test (LAST_ACTIVE_LAP)
5. Which lap begins the recovery (RECOVERY_LAP_START)

Then ask for load values:

If BIKE

Ask me for the wattage for each active lap from FIRST_ACTIVE_LAP to LAST_ACTIVE_LAP.

If TREADMILL

Ask me for the speed for each active lap from FIRST_ACTIVE_LAP to LAST_ACTIVE_LAP

Ask: is the speed in mph or km/h

STEP 2. Breakpoint settings

Ask me the Breakpoint Control Question:

“How do you want the breakpoints handled for this graph

A. No breakpoints

B. Auto-detect breakpoints

C. Use approximate breakpoints and refine

D. Override with my exact breakpoint values”

Follow the rules for each option:

Option A: No breakpoints

Do not compute or draw any breakpoints.

Option B: Auto-detect

Use slope-based logic to automatically find breakpoint 1 and breakpoint 2 for each leg.

Option C: User approximations (refine them)

Ask me for approximate times (in seconds from test start) for:

– Right BP1

– Left BP1

– Right BP2

– Left BP2

Then refine each by searching in a short window around the approximate time to find the slope closest to zero.

Option D: Full manual override

Ask me for exact final breakpoint times (in seconds) for:

– R BP1

– L BP1

– R BP2

– L BP2

These values completely override automatic and refined breakpoints.

Plot them exactly as given.

Do not refine.

STEP 3. Data loading

1. Load CSV_FILE_NAME.
2. Identify the header row containing “Timestamp” and “Lap/Event”.

3. Deduplicate any repeated column names.
4. Identify:
 - Timestamp
 - Lap/Event
 - Heart Rate (if present)
 - Left SmO₂ (SmO2)
 - Right SmO₂ (SmO2_1)
5. Convert numeric columns appropriately.
6. Define warm up start as “warm up duration before FIRST_ACTIVE_LAP appears the first time”.
7. Trim all data before warm up start.
8. Create a relative time axis starting at 0 seconds.
9. Sort rows by time.

STEP 4. Smoothing

1. Estimate sampling frequency.
2. Apply a centered 10 second rolling mean to:
 - Left SmO₂
 - Right SmO₂
 - Heart Rate (if present)

Use these smoothed values for analysis and plotting.

STEP 5. Stage mapping by laps

Warm up

- 0 seconds to first appearance of FIRST_ACTIVE_LAP

Active test

- Laps FIRST_ACTIVE_LAP through LAST_ACTIVE_LAP
- Each lap is one stage
- Stage boundaries determined by the first appearance of each lap

Recovery

- Starts at first appearance of RECOVERY_LAP_START
- Ends at end of file

STEP 6. Load presentation

If BIKE

- Label each stage with “XXX W”

- Place load above x-axis
- No load labels in recovery

If TREADMILL

- Convert speed to min/km pace
- Label each stage with two lines:

“Speed (mph or km/h)

Pace (min/km)”

- Rotate vertically
- Place above x-axis
- No load labels in recovery

STEP 7. Recovery percentage calculations

For each leg:

1. Compute average SmO_2 over last 30 seconds of warm up.
2. Compute average SmO_2 over last 30 seconds of recovery.
3. Compute recovery percent:
 $(\text{recovery average} \div \text{warm up average}) \times 100$
4. Round to the nearest whole number.
5. Append this value to legend labels as:
 - “Left SmO_2 (Rec XX percent of warm up)”
 - “Right SmO_2 (Rec XX percent of warm up)”

STEP 8. Plot design

1. X axis = seconds from warm up start
2. Primary Y axis = SmO_2
3. Secondary Y axis = heart rate (if present)
4. Shaded areas:
 - Light shade for warm up
 - Alternating shades for active laps
 - Gray shade for recovery
5. A vertical line at recovery start
6. Plot both SmO_2 curves
7. Add secondary axis and dashed line if HR exists
8. Title: “Train.Red SmO_2 Ramp – ATHLETE_NAME – TEST_DATE”
9. Legend includes:
 - Left SmO_2 with recovery percent

- Right SmO₂ with recovery percent
- Heart rate if present

STEP 9. Breakpoint visualization

If breakpoints are disabled (Option A)

- Skip all breakpoint drawing

If breakpoints are active (Options B, C, D):

- BP1 shown as dashed vertical lines
- BP2 shown as dotted vertical lines
- Label each as “R BP1”, “L BP1”, “R BP2”, “L BP2”

STEP 10. Athlete explanation (14-year-old reading level)

Provide a clear, simple summary:

1. What the graph shows
2. How SmO₂ changed during the warm up, stages, and recovery
3. What BP1 and BP2 mean (if used)
4. What recovery percentages mean for fitness
5. 3–5 training suggestions based on bike or treadmill test type

Here is the updated unified Python code with:

- Bike or treadmill mode
- Recovery percentages
- Breakpoint modes:
 - “none”
 - “auto”
 - “approx” (refine around your guesses)
 - “manual” (override with exact times)

You can paste this directly into your Canvas code cell.

from typing import Dict, Optional, Tuple

import pandas as pd

import numpy as np

```

import matplotlib.pyplot as plt

def find_header_row(csv_path: str) -> int:
    with open(csv_path, "r", encoding="utf-8", errors="ignore") as f:
        for idx, line in enumerate(f):
            if "Timestamp" in line and "Lap/Event" in line:
                return idx
        return 0

def load_trainred_csv(csv_path: str) -> pd.DataFrame:
    header_row_index = find_header_row(csv_path)
    raw = pd.read_csv(csv_path, skiprows=header_row_index)
    header_row = raw.iloc[0]
    df = raw.iloc[1:].reset_index(drop=True)
    df.columns = header_row
    cols = list(df.columns)
    counter = {}
    newcols = []
    for c in cols:
        count = counter.get(c, 0)
        if count == 0:
            newname = str(c)
        else:
            newname = f"{c}_{count}"
        counter[c] = count + 1
        newcols.append(newname)
    df.columns = newcols
    return df

```

```
def refine_bp(t: np.ndarray, y: np.ndarray, approx_t: Optional[float], window: float = 120.0) -> Tuple[Optional[float], Optional[float]]:
```

```
    if approx_t is None:
```

```
        return None, None
```

```
    mask = (t >= approx_t - window / 2.0) & (t <= approx_t + window / 2.0)
```

```
    t_win = t[mask]
```

```
    y_win = y[mask]
```

```
    if len(t_win) < 5:
```

```
        return None, None
```

```
    dy = np.diff(y_win)
```

```
    dt = np.diff(t_win)
```

```
    deriv = np.divide(dy, dt, out=np.zeros_like(dy), where=dt != 0)
```

```
    idx = int(np.argmin(np.abs(deriv)))
```

```
    t_bp = float((t_win[idx] + t_win[idx + 1]) / 2.0)
```

```
    y_bp = float((y_win[idx] + y_win[idx + 1]) / 2.0)
```

```
    return t_bp, y_bp
```

```
def auto_breakpoints(t: np.ndarray, y: np.ndarray) -> Tuple[Optional[float], Optional[float]]:
```

```
    if len(t) < 10:
```

```
        return None, None
```

```
    dt = np.diff(t)
```

```
    dy = np.diff(y)
```

```
    deriv = np.divide(dy, dt, out=np.zeros_like(dy), where=dt != 0)
```

```
    window = min(21, max(3, (len(deriv) // 2) * 2 + 1))
```

```
    d_series = pd.Series(deriv).rolling(window, center=True, min_periods=1).mean().values
```

```
    up_thresh = 0.0005
```



```
down_thresh = -0.0005
```

```
bp1 = None
```

```
for i in range(5, len(d_series)):
```

```
    if d_series[i] < up_thresh and np.any(d_series[:i] > up_thresh):
```

```
        bp1 = i
```

```
        break
```

```
bp2 = None
```

```
if bp1 is not None:
```

```
    for i in range(bp1 + 5, len(d_series)):
```

```
        if d_series[i] < down_thresh:
```

```
            bp2 = i
```

```
            break
```

```
t_bp1 = float(t[bp1]) if bp1 is not None else None
```

```
t_bp2 = float(t[bp2]) if bp2 is not None else None
```

```
return t_bp1, t_bp2
```

```
def compute_pace_min_per_km_from_mph(speed_mph: float) -> Optional[str]:
```

```
    km_per_hour = speed_mph * 1.60934
```

```
    if km_per_hour <= 0:
```

```
        return None
```

```
    pace_min_per_km = 60.0 / km_per_hour
```

```
    pace_min = int(pace_min_per_km)
```

```
    pace_sec = int(round((pace_min_per_km - pace_min) * 60))
```

```
    if pace_sec == 60:
```

```

pace_min += 1

pace_sec = 0

return f"{pace_min}:{pace_sec:02d}"

def compute_pace_min_per_km_from_kmh(speed_kmh: float) -> Optional[str]:

if speed_kmh <= 0:

return None

pace_min_per_km = 60.0 / speed_kmh

pace_min = int(pace_min_per_km)

pace_sec = int(round((pace_min_per_km - pace_min) * 60))

if pace_sec == 60:

pace_min += 1

pace_sec = 0

return f"{pace_min}:{pace_sec:02d}"

def plot_trainred_ramp(

csv_path: str,

athlete_name: str,

test_date: str,

mode: str,

warmup_duration_minutes: float,

first_active_lap: int,

last_active_lap: int,

recovery_lap_start: int,

bike_lap_watts: Optional[Dict[int, float]] = None,

treadmill_lap_speeds: Optional[Dict[int, float]] = None,

treadmill_speed_units: str = "mph",

breakpoint_mode: str = "none", # "none", "auto", "approx", "manual"

```

```
approx_bp_times: Optional[Dict[str, float]] = None,  
override_bp_times: Optional[Dict[str, float]] = None,  
):
```

```
"""
```

```
breakpoint_mode:
```

```
“none” -> no breakpoints
```

```
“auto” -> auto_breakpoints for each leg
```

```
“approx” -> refine around approx_bp_times
```

```
“manual” -> use override_bp_times directly
```

```
approx_bp_times and override_bp_times keys:
```

```
“right_bp1”, “left_bp1”, “right_bp2”, “left_bp2”
```

```
"""
```

```
mode = mode.lower().strip()
```

```
breakpoint_mode = breakpoint_mode.lower().strip()
```

```
df = load_trainred_csv(csv_path)
```

```
time_col = [c for c in df.columns if "Timestamp" in c][0]
```

```
lap_col = [c for c in df.columns if "Lap/Event" in c][0]
```

```
hr_cols = [c for c in df.columns if "Heart Rate" in c]
```

```
hr_col = hr_cols[0] if hr_cols else None
```

```
left_col = "SmO2"
```

```
right_col = "SmO2_1"
```

```
for col in [time_col, lap_col, left_col, right_col, hr_col]:
```

if col is not None:

df[col] = pd.to_numeric(df[col], errors="coerce")

time_s = df[time_col].values

lap = df[lap_col].values

warmup_duration_seconds = warmup_duration_minutes * 60.0

mask_first_active = lap == first_active_lap

if not np.any(mask_first_active):

raise ValueError("First active lap not found in Lap/Event column.")

t_first_active_start = time_s[mask_first_active][0]

warmup_start_time = t_first_active_start - warmup_duration_seconds

mask_use = time_s >= warmup_start_time

time_s2 = time_s[mask_use]

lap2 = lap[mask_use]

left2 = df[left_col].values[mask_use]

right2 = df[right_col].values[mask_use]

hr2 = df[hr_col].values[mask_use] if hr_col else None

time_rel = time_s2 - warmup_start_time

df2 = pd.DataFrame(
{

{

"time_rel": time_rel,

```

        "lap": lap2,

        "left": left2,

        "right": right2,

    }

)

if hr2 is not None:

    df2["hr"] = hr2


df2 = df2.sort_values("time_rel").reset_index(drop=True)


median_dt = np.median(np.diff(df2["time_rel"].values))

window_samples = int(round(10.0 / median_dt)) if median_dt > 0 else 50

window_samples = max(5, min(window_samples, 201))


df2["left_smooth"] = df2["left"].rolling(window_samples, center=True, min_periods=1).mean()

df2["right_smooth"] = df2["right"].rolling(window_samples, center=True, min_periods=1).mean()

if hr2 is not None:

    df2["hr_smooth"] = df2["hr"].rolling(window_samples, center=True, min_periods=1).mean()


warmup_end = df2.loc[df2["lap"] == first_active_lap, "time_rel"].min()


if (df2["lap"] >= recovery_lap_start).any():

    recovery_start = df2.loc[df2["lap"] >= recovery_lap_start, "time_rel"].min()

else:

    recovery_start = df2["time_rel"].max() - 150.0

recovery_end = df2["time_rel"].max()

```

```

stages = []

for lap_id in range(first_active_lap, last_active_lap + 1):

    mask_stage = df2["lap"] == lap_id

    if not mask_stage.any():

        continue

    start_t = df2.loc[mask_stage, "time_rel"].min()

    if lap_id < last_active_lap:

        next_mask = df2["lap"] == (lap_id + 1)

        if next_mask.any():

            end_t = df2.loc[next_mask, "time_rel"].min()

        else:

            end_t = recovery_start

    else:

        end_t = recovery_start

    label_info = {}

    if mode == "bike" and bike_lap_watts is not None:

        label_info["watts"] = bike_lap_watts.get(lap_id)

    if mode == "treadmill" and treadmill_lap_speeds is not None:

        label_info["speed"] = treadmill_lap_speeds.get(lap_id)

    stages.append(

        {

            "lap": lap_id,

            "start": start_t,

```

```
        "end": end_t,  
        "info": label_info,  
    }  
)
```

```
left_w = right_w = left_r = right_r = None
```

```
if warmup_end - 30.0 > 0:
```

```
    w_mask = (df2["time_rel"] >= warmup_end - 30.0) & (df2["time_rel"] <= warmup_end)
```

```
    if w_mask.any():
```

```
        left_w = float(df2.loc[w_mask, "left_smooth"].mean())
```

```
        right_w = float(df2.loc[w_mask, "right_smooth"].mean())
```

```
if recovery_end - 30.0 > recovery_start:
```

```
    r_mask = (df2["time_rel"] >= recovery_end - 30.0) & (df2["time_rel"] <= recovery_end)
```

```
    if r_mask.any():
```

```
        left_r = float(df2.loc[r_mask, "left_smooth"].mean())
```

```
        right_r = float(df2.loc[r_mask, "right_smooth"].mean())
```

```
left_rec_pct = right_rec_pct = None
```

```
if left_w is not None and left_r is not None and left_w != 0:
```

```
    left_rec_pct = left_r / left_w * 100.0
```

```
if right_w is not None and right_r is not None and right_w != 0:
```

```
    right_rec_pct = right_r / right_w * 100.0
```

```
t_all = df2["time_rel"].values
```

```
left_s = df2["left_smooth"].values
```

```
right_s = df2["right_smooth"].values
```

```
r_bp1_t = l_bp1_t = r_bp2_t = l_bp2_t = None
```

```
if breakpoint_mode == "none":
```

```
    pass
```

```
elif breakpoint_mode == "auto":
```

```
    r_bp1_t, r_bp2_t = auto_breakpoints(t_all, right_s)
```

```
    l_bp1_t, l_bp2_t = auto_breakpoints(t_all, left_s)
```

```
elif breakpoint_mode == "approx":
```

```
    if approx_bp_times is None:
```

```
        r_bp1_t = l_bp1_t = r_bp2_t = l_bp2_t = None
```

```
    else:
```

```
        r_bp1_t, _ = refine_bp(t_all, right_s, approx_bp_times.get("right_bp1"))
```

```
        l_bp1_t, _ = refine_bp(t_all, left_s, approx_bp_times.get("left_bp1"))
```

```
        r_bp2_t, _ = refine_bp(t_all, right_s, approx_bp_times.get("right_bp2"))
```

```
        l_bp2_t, _ = refine_bp(t_all, left_s, approx_bp_times.get("left_bp2"))
```

```
elif breakpoint_mode == "manual":
```

```
    if override_bp_times is not None:
```

```
        r_bp1_t = override_bp_times.get("right_bp1")
```

```
        l_bp1_t = override_bp_times.get("left_bp1")
```

```
        r_bp2_t = override_bp_times.get("right_bp2")
```

```
        l_bp2_t = override_bp_times.get("left_bp2")
```



```

fig, ax1 = plt.subplots(figsize=(10, 6))

ax1.axvspan(0, warmup_end, alpha=0.1)

for i, st in enumerate(stages):
    ax1.axvspan(st["start"], st["end"], alpha=0.1 + 0.05 * (i % 2))

ax1.axvspan(recovery_start, recovery_end, alpha=0.2, color="lightgray")

left_label = "Left SmO2"
right_label = "Right SmO2"

if left_rec_pct is not None:
    left_label += f" (Rec {left_rec_pct:.0f}% of warm up)"

if right_rec_pct is not None:
    right_label += f" (Rec {right_rec_pct:.0f}% of warm up)"

ax1.plot(df2["time_rel"], df2["left_smooth"], label=left_label)
ax1.plot(df2["time_rel"], df2["right_smooth"], label=right_label)

ax1.set_xlabel("Time (s)")
ax1.set_ylabel("SmO2 (%)")

if "hr_smooth" in df2.columns:
    ax2 = ax1.twinx()
    ax2.plot(df2["time_rel"], df2["hr_smooth"], linestyle="dashed")
    ax2.set_ylabel("Heart rate (bpm)")

```

```
ax1.axvline(recovery_start, color="gray", linestyle="--")
```

```
for st in stages:
```

```
    x_mid = (st["start"] + st["end"]) / 2.0
```

```
    info = st["info"]
```

```
    if mode == "bike":
```

```
        watts = info.get("watts") if info is not None else None
```

```
        if watts is not None:
```

```
            ax1.text(
                x_mid,
                0.02,
                f"{watts} W",
                ha="center",
                va="bottom",
                rotation=0,
                transform=ax1.get_xaxis_transform(),
            )
```

```
    if mode == "treadmill":
```

```
        speed = info.get("speed") if info is not None else None
```

```
        if speed is not None:
```

```
            if treadmill_speed_units.lower() == "mph":
```

```
                pace = compute_pace_min_per_km_from_mph(speed)
```

```
                speed_str = f"{speed:.1f} mph"
```

```

else:

    pace = compute_pace_min_per_km_from_kmh(speed)

    speed_str = f"{speed:.1f} km/h"

    label = f"{speed_str}\n{pace} min/km"

    ax1.text(

        x_mid,

        0.02,

        label,

        ha="center",

        va="bottom",

        rotation=90,

        transform=ax1.get_xaxis_transform(),

    )

```

```

if breakpoint_mode != "none":

```

```

    if r_bp1_t is not None:

```

```

        ax1.axvline(r_bp1_t, color="blue", linestyle="--")

        ax1.text(

            r_bp1_t,

            0.98,

            "R BP1",

            transform=ax1.get_xaxis_transform(),

            ha="center",

            va="top",

            color="blue",

        )

```

if l_bp1_t is not None:

```
ax1.axvline(l_bp1_t, color="orange", linestyle="--")
```

```
ax1.text(
```

```
    l_bp1_t,
```

```
    0.92,
```

```
    "L BP1",
```

```
    transform=ax1.get_xaxis_transform(),
```

```
    ha="center",
```

```
    va="top",
```

```
    color="orange",
```

```
)
```

if r_bp2_t is not None:

```
ax1.axvline(r_bp2_t, color="blue", linestyle=":")
```

```
ax1.text(
```

```
    r_bp2_t,
```

```
    0.86,
```

```
    "R BP2",
```

```
    transform=ax1.get_xaxis_transform(),
```

```
    ha="center",
```

```
    va="top",
```

```
    color="blue",
```

```
)
```

if l_bp2_t is not None:

```
ax1.axvline(l_bp2_t, color="orange", linestyle=":")
```

```
ax1.text(
```

```
    l_bp2_t,
```

```

    0.80,

    "L BP2",

    transform=ax1.get_xaxis_transform(),

    ha="center",

    va="top",

    color="orange",

)

```

```

title = f"Train.Red SmO2 Ramp – {athlete_name} – {test_date}"

```

```

ax1.set_title(title)

```

```

ax1.legend(loc="upper left")

```

```

plt.tight_layout()

```

```

plt.show()

```

```

return {

```

```

    "left_rec_pct": left_rec_pct,

```

```

    "right_rec_pct": right_rec_pct,

```

```

    "warmup_end": warmup_end,

```

```

    "recovery_start": recovery_start,

```

```

    "recovery_end": recovery_end,

```

```

    "r_bp1_t": r_bp1_t,

```

```

    "l_bp1_t": l_bp1_t,

```

```

    "r_bp2_t": r_bp2_t,

```

```

    "l_bp2_t": l_bp2_t,

```

```

}

```

Example for Akhil's treadmill test in your current setup:

```

treadmill_speeds_akhil = {lap: 4.0 + 0.5*(lap-1) for lap in range(1, 12)}

results_akhil = plot_trainred_ramp(
    csv_path="/mnt/data/session_20251119_105834.csv",
    athlete_name="Akhil Munjal",
    test_date="November 19, 2025",
    mode="treadmill",
    warmup_duration_minutes=3.0,
    first_active_lap=1,
    last_active_lap=11,
    recovery_lap_start=12,
    treadmill_lap_speeds=treadmill_speeds_akhil,
    treadmill_speed_units="mph",
    breakpoint_mode="auto", # or "none", "approx", "manual"
    approx_bp_times=None,
    override_bp_times=None,
)

```

If you want, I can now write the exact Canvas “question flow” you would use so it always collects:

- mode
- warmup duration
- lap structure
- bike watts or run speeds
- breakpoint mode
- approx or manual times if needed

PYTHON CODE

```
import pandas as pd
```

```

import numpy as np

import matplotlib.pyplot as plt

def find_header_row(csv_path):
    with open(csv_path, "r", encoding="utf-8", errors="ignore") as f:
        lines = f.readlines()

    for idx, line in enumerate(lines):
        if "Timestamp" in line and "Lap/Event" in line:
            return idx

    # fallback if nothing found
    return 0

def load_trainred_csv(csv_path):
    header_row_index = find_header_row(csv_path)

    raw = pd.read_csv(csv_path, skiprows=header_row_index)

    header_row = raw.iloc[0]

    df = raw.iloc[1:].reset_index(drop=True)

    df.columns = header_row

    # deduplicate column names
    cols = list(df.columns)
    counter = {}
    newcols = []
    for c in cols:
        count = counter.get(c, 0)
        if count == 0:
            newname = c
        else:
            newname = f"{c}_{count}"
        counter[c] = count + 1
        newcols.append(newname)
    df.columns = newcols
    return df

def build_graph(
    csv_path,

```

```

athlete_name,

test_date,

warmup_duration_seconds,

warmup_speed_mph,

stage_speeds_mph,

stage_durations_seconds,

recovery_duration_seconds,

left_is_first_smO2=True

):

df = load_trainred_csv(csv_path)

# detect core columns
time_col = [c for c in df.columns if isinstance(c, str) and "Timestamp" in c][0]
lap_col = [c for c in df.columns if isinstance(c, str) and "Lap/Event" in c][0]
hr_cols = [c for c in df.columns if isinstance(c, str) and "Heart Rate" in c]
hr_col = hr_cols[0] if hr_cols else None

df[time_col] = pd.to_numeric(df[time_col], errors="coerce")
df[lap_col] = pd.to_numeric(df[lap_col], errors="coerce")
if hr_col:
    df[hr_col] = pd.to_numeric(df[hr_col], errors="coerce")

# detect SmO2 columns
smo_cols = [c for c in df.columns if isinstance(c, str) and c.startswith("SmO2")]
smo_cols = [c for c in smo_cols if "unfiltered" not in c]

if left_is_first_smO2:
    left_col = smo_cols[0]
    right_col = smo_cols[1] if len(smo_cols) > 1 else smo_cols[0]
else:
    right_col = smo_cols[0]
    left_col = smo_cols[1] if len(smo_cols) > 1 else smo_cols[0]

df[left_col] = pd.to_numeric(df[left_col], errors="coerce")
df[right_col] = pd.to_numeric(df[right_col], errors="coerce")

time_s = df[time_col].values
lap = df[lap_col].values

```



```

# warm up start: warm up_duration before first lap 2
mask_lap2 = lap == 2
if not np.any(mask_lap2):
    raise ValueError("No lap 2 found for warm up reference.")
t_lap2_start = time_s[mask_lap2][0]
warmup_start_time = t_lap2_start - warmup_duration_seconds

# trim data to start at warm up start
mask_use = time_s >= warmup_start_time
time_s2 = time_s[mask_use]
lap2 = lap[mask_use]
left2 = df[left_col].values[mask_use]
right2 = df[right_col].values[mask_use]
hr2 = df[hr_col].values[mask_use] if hr_col else None

time_rel = time_s2 - warmup_start_time

# recovery start from lap 8 in this protocol
mask_lap8 = lap2 == 8
if not np.any(mask_lap8):
    raise ValueError("No lap 8 found for recovery reference.")
t_lap8_start = time_rel[mask_lap8][0]
recovery_start = t_lap8_start
recovery_end = recovery_start + recovery_duration_seconds

# stage mapping
stages = []
stage_start = warmup_duration_seconds

for i, speed in enumerate(stage_speeds_mph):
    dur = stage_durations_seconds[i]
    if dur is None:
        stage_end = t_lap8_start
    else:
        stage_end = stage_start + dur
    stages.append(
        {
            "index": i + 1,
            "start": stage_start,
            "end": stage_end,
            "speed_mph": speed,
        }
    )
    stage_start = stage_end

# build smoothed frame

```

```

df2 = pd.DataFrame(
    {
        "time_rel": time_rel,
        "lap": lap2,
        "left": left2,
        "right": right2,
    }
)
if hr2 is not None:
    df2["hr"] = hr2

df2 = df2.sort_values("time_rel").reset_index(drop=True)
df2["left_smooth"] = df2["left"].rolling(window=5, center=True, min_periods=1).mean()
df2["right_smooth"] = df2["right"].rolling(window=5, center=True, min_periods=1).mean()
if hr2 is not None:
    df2["hr_smooth"] = df2["hr"].rolling(window=5, center=True, min_periods=1).mean()

# recovery metrics
warmup_end = warmup_duration_seconds
warmup_last30_start = warmup_end - 30.0
rec_end = min(recovery_end, df2["time_rel"].max())
rec_last30_start = rec_end - 30.0

w_mask = (df2["time_rel"] >= warmup_last30_start) & (df2["time_rel"] <= warmup_end)
r_mask = (df2["time_rel"] >= rec_last30_start) & (df2["time_rel"] <= rec_end)

left_w = df2.loc[w_mask, "left_smooth"].mean()
right_w = df2.loc[w_mask, "right_smooth"].mean()
left_r = df2.loc[r_mask, "left_smooth"].mean()
right_r = df2.loc[r_mask, "right_smooth"].mean()

left_rec_pct = left_r / left_w * 100.0
right_rec_pct = right_r / right_w * 100.0

# plot
fig, ax1 = plt.subplots(figsize=(10, 6))

# warm up shade
ax1.axvspan(0, warmup_end, alpha=0.1)

# stage shades
for i, st in enumerate(stages):
    shade_alpha = 0.15 + 0.05 * (i % 2)
    ax1.axvspan(st["start"], st["end"], alpha=shade_alpha)

# recovery shade

```

```
ax1.axvspan(recovery_start, recovery_end, alpha=0.2, color="lightgray")
```

```
# SmO2 lines
```

```
ax1.plot(
    df2["time_rel"],
    df2["left_smooth"],
    label=f"Left SmO2 (~{left_rec_pct:.0f}% of warm up)",
)
ax1.plot(
    df2["time_rel"],
    df2["right_smooth"],
    label=f"Right SmO2 (~{right_rec_pct:.0f}% of warm up)",
)
ax1.set_xlabel("Time (s)")
ax1.set_ylabel("SmO2 (%)")
```

```
# heart rate
```

```
if "hr_smooth" in df2.columns:
    ax2 = ax1.twinx()
    ax2.plot(df2["time_rel"], df2["hr_smooth"], linestyle="dashed")
    ax2.set_ylabel("Heart rate (bpm)")
```

```
# trend lines per stage
```

```
for st in stages:
    mask_stage = (df2["time_rel"] >= st["start"]) & (df2["time_rel"] <= st["end"])
    if mask_stage.sum() > 5:
        t_seg = df2.loc[mask_stage, "time_rel"].values
        for col in ["left_smooth", "right_smooth"]:
            y_seg = df2.loc[mask_stage, col].values
            coeffs = np.polyfit(t_seg, y_seg, 1)
            y_fit = np.polyval(coeffs, t_seg)
            ax1.plot(t_seg, y_fit, linewidth=1)
```

```
# vertical line at recovery start
```

```
ax1.axvline(recovery_start)
```

```
# speed and pace labels above x axis
```

```
for st in stages:
    speed = st["speed_mph"]
    km_per_hour = speed * 1.60934
    pace_min_per_km = 60.0 / km_per_hour
    pace_min = int(pace_min_per_km)
    pace_sec = int(round((pace_min_per_km - pace_min) * 60))
    if pace_sec == 60:
        pace_min += 1
        pace_sec = 0
```

```

pace_str = f"{pace_min}:{pace_sec:02d}"
x_mid = (st["start"] + st["end"]) / 2.0
ax1.text(
    x_mid,
    -0.08,
    f"{speed:.1f} mph\n{pace_str} min/km",
    ha="center",
    va="top",
    rotation=90,
    transform=ax1.get_xaxis_transform(),
)

ax1.legend(loc="upper left")
title = f"Train.Red SmO2 Ramp – {athlete_name} – {test_date}"
ax1.set_title(title)

plt.tight_layout()
plt.show()

```

example call for Ihab's test

```
build_graph(  
  
    csv_path="/mnt/data/session_20251107_172816.csv",  
  
    athlete_name="Ihab El Attar",  
  
    test_date="November 7, 2025",  
  
    warmup_duration_seconds=180.0,  
  
    warmup_speed_mph=3.0,  
  
    stage_speeds_mph=[3.5 + 0.5 * i for i in range(10)],  
  
    stage_durations_seconds=[60.0] * 9 + [None],  
  
    recovery_duration_seconds=150.0,  
  
    left_is_first_smO2=True,  
  
)
```