# Purpose: This utility infers whether each contact in an amateur radio contest log

#          was made while "Running" (CQing) or "Search & Pounce" (S&P). It adds a

#          'Run' column to the log, classifying each QSO accordingly.

#

# Author: Mark Bailey, KD4D

# Contact: kd4d@kd4d.org

# Date: 2025-07-23

# Version: 0.14.3-Beta

#

# Copyright (c) 2025 Mark Bailey, KD4D

#

# License: Mozilla Public License, v. 2.0

#          (https://www.mozilla.org/MPL/2.0/)

#

# This Source Code Form is subject to the terms of the Mozilla Public

# License, v. 2.0. If a copy of the MPL was not distributed with this

# file, You can obtain one at http://mozilla.org/MPL/2.0/.

# --- Revision History ---

# All notable changes to this project will be documented in this file.

# The format is based on "Keep a Changelog" (https://keepachangelog.com/en/1.0.0/),

# and this project aims to adhere to Semantic Versioning (https://semver.org/).

## [0.14.3-Beta] - 2025-07-23

### Fixed

# - Corrected the "sticky run" logic to only break a run if the consecutive

#   off-frequency QSOs are on the same new frequency, not on multiple

#   different S&P frequencies.

import pandas as pd

from collections import deque

import os

import sys

import traceback

# --- Constants ---

DEFAULT\_RUN\_TIME\_WINDOW\_MINUTES = 10

DEFAULT\_FREQ\_TOLERANCE\_CW = 0.1

DEFAULT\_FREQ\_TOLERANCE\_PH = 0.5

DEFAULT\_MIN\_QSO\_FOR\_RUN = 3

RUN\_BREAK\_QSO\_COUNT = 3

RUN\_BREAK\_TIME\_MINUTES = 2

def \_get\_run\_info\_from\_buffer(base\_freq: float, buffer\_to\_check: deque, min\_qso: int, time\_threshold: pd.Timedelta, tol: float):

    """

    Helper to check if a given `base\_freq` forms a valid run within the `buffer\_to\_check`.

    """

    relevant\_qso\_data = []

    for buffered\_idx, buffered\_time, buffered\_freq in buffer\_to\_check:

        if abs(buffered\_freq - base\_freq) <= tol:

            relevant\_qso\_data.append((buffered\_idx, buffered\_time))

    if len(relevant\_qso\_data) < min\_qso:

        return False, []

    # Check from the most recent QSOs backwards

    for i in range(len(relevant\_qso\_data) - 1, min\_qso - 2, -1):

        latest\_time\_in\_segment = relevant\_qso\_data[i][1]

        oldest\_time\_in\_segment = relevant\_qso\_data[i - (min\_qso - 1)][1]

        if (latest\_time\_in\_segment - oldest\_time\_in\_segment) <= time\_threshold:

            qualifying\_indices = [q[0] for q in relevant\_qso\_data[i - (min\_qso - 1) : i + 1]]

            return True, qualifying\_indices

    return False, []

def \_evaluate\_single\_stream\_run(

    stream\_df\_original: pd.DataFrame,

    datetime\_column: str,

    frequency\_column: str,

    stream\_tolerance: float,

    time\_delta\_threshold: pd.Timedelta,

    min\_qso\_for\_run: int

):

    """

    Evaluates run status for a single operational stream using a "sticky run" state machine.

    """

    inferred\_run\_status = ['S&P'] \* len(stream\_df\_original)

    original\_idx\_to\_list\_pos = {idx: i for i, idx in enumerate(stream\_df\_original.index)}

    active\_run\_freq = None

    last\_qso\_on\_run\_freq\_time = None

    off\_frequency\_qso\_count = 0

    potential\_new\_run\_freq = None # Track the frequency of consecutive S&P QSOs

    qso\_buffer = deque()

    for list\_pos, original\_df\_idx in enumerate(stream\_df\_original.index):

        current\_qso\_time = stream\_df\_original.at[original\_df\_idx, datetime\_column]

        current\_qso\_freq = stream\_df\_original.at[original\_df\_idx, frequency\_column]

        qso\_buffer.append((original\_df\_idx, current\_qso\_time, current\_qso\_freq))

        while qso\_buffer and (current\_qso\_time - qso\_buffer[0][1]) > time\_delta\_threshold:

            qso\_buffer.popleft()

        # --- Main State Machine: Prioritize maintaining an existing run ---

        if active\_run\_freq is not None:

            is\_on\_run\_freq = abs(current\_qso\_freq - active\_run\_freq) <= stream\_tolerance

            timed\_out = (current\_qso\_time - last\_qso\_on\_run\_freq\_time) > pd.Timedelta(minutes=RUN\_BREAK\_TIME\_MINUTES)

            if is\_on\_run\_freq and not timed\_out:

                inferred\_run\_status[list\_pos] = 'Run'

                last\_qso\_on\_run\_freq\_time = current\_qso\_time

                off\_frequency\_qso\_count = 0

                potential\_new\_run\_freq = None

            else:

                inferred\_run\_status[list\_pos] = 'S&P'

                if not is\_on\_run\_freq:

                    # Check if this S&P QSO continues a potential new run

                    if potential\_new\_run\_freq and abs(current\_qso\_freq - potential\_new\_run\_freq) <= stream\_tolerance:

                        off\_frequency\_qso\_count += 1

                    else: # It's a new S&P frequency, reset the counter

                        potential\_new\_run\_freq = current\_qso\_freq

                        off\_frequency\_qso\_count = 1

                if timed\_out or off\_frequency\_qso\_count >= RUN\_BREAK\_QSO\_COUNT:

                    active\_run\_freq = None # Break the run

        # --- If no run is active, check if a new one has formed ---

        if active\_run\_freq is None:

            is\_new\_run, new\_run\_indices = \_get\_run\_info\_from\_buffer(

                current\_qso\_freq, qso\_buffer, min\_qso\_for\_run, time\_delta\_threshold, stream\_tolerance

            )

            if is\_new\_run:

                active\_run\_freq = current\_qso\_freq

                last\_qso\_on\_run\_freq\_time = current\_qso\_time

                off\_frequency\_qso\_count = 0

                potential\_new\_run\_freq = None

                for idx in new\_run\_indices:

                    if idx in original\_idx\_to\_list\_pos:

                        inferred\_run\_status[original\_idx\_to\_list\_pos[idx]] = 'Run'

            else:

                inferred\_run\_status[list\_pos] = 'S&P'

    return pd.Series(inferred\_run\_status, index=stream\_df\_original.index, dtype=str)

def process\_contest\_log\_for\_run\_s\_p(

    df: pd.DataFrame,

    my\_call\_column: str = 'MyCall',

    datetime\_column: str = 'Datetime',

    frequency\_column: str = 'Frequency',

    mode\_column: str = 'Mode',

    band\_column: str = 'Band'

) -> pd.DataFrame:

    """

    Main wrapper function to infer "Run" or "S&P" status for each QSO in a DataFrame.

    """

    processed\_df = df.copy()

    try:

        required\_cols = [my\_call\_column, datetime\_column, frequency\_column, mode\_column, band\_column]

        for col in required\_cols:

            if col not in processed\_df.columns:

                raise KeyError(f"Missing required column for Run/S&P processing: '{col}'")

        processed\_df[datetime\_column] = pd.to\_datetime(processed\_df[datetime\_column], errors='coerce')

        processed\_df[frequency\_column] = pd.to\_numeric(processed\_df[frequency\_column], errors='coerce')

        processed\_df.dropna(subset=[datetime\_column, frequency\_column], inplace=True)

        for col in [my\_call\_column, mode\_column, band\_column]:

            processed\_df[col] = processed\_df[col].astype(str)

        if 'Run' in processed\_df.columns:

            processed\_df.drop(columns=['Run'], inplace=True)

        df\_sorted = processed\_df.sort\_values(by=[my\_call\_column, band\_column, mode\_column, datetime\_column])

        time\_delta\_threshold = pd.Timedelta(minutes=DEFAULT\_RUN\_TIME\_WINDOW\_MINUTES) + pd.Timedelta(seconds=1)

        results = []

        for group\_name, group\_df in df\_sorted.groupby([my\_call\_column, band\_column, mode\_column], group\_keys=False):

            representative\_mode = group\_name[2]

            stream\_tolerance = DEFAULT\_FREQ\_TOLERANCE\_CW if representative\_mode.upper() == 'CW' else DEFAULT\_FREQ\_TOLERANCE\_PH

            group\_results\_series = \_evaluate\_single\_stream\_run(

                group\_df, datetime\_column, frequency\_column, stream\_tolerance,

                time\_delta\_threshold, DEFAULT\_MIN\_QSO\_FOR\_RUN

            )

            results.append(group\_results\_series)

        if results:

            run\_column\_series = pd.concat(results)

            processed\_df['Run'] = run\_column\_series

        else:

            processed\_df['Run'] = 'S&P'

        return processed\_df

    except KeyError as e:

        raise KeyError(f"Error during Run/S&P pre-processing: {e}")

    except Exception as e:

        print(f"An unexpected error occurred during Run/S&P processing: {e}")

        traceback.print\_exc()

        raise

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

    if len(sys.argv) < 2:

        print("Usage: python run\_s\_p.py <input\_csv\_file\_path>")

        sys.exit(1)

    csv\_file\_path = sys.argv[1]

    try:

        initial\_df = pd.read\_csv(csv\_file\_path, sep=',', header=0, dtype=str)

        base\_name = os.path.splitext(csv\_file\_path)[0]

        output\_file\_path = f"{base\_name}\_Run.csv"

        processed\_df = process\_contest\_log\_for\_run\_s\_p(df=initial\_df)

        df\_for\_output = processed\_df.copy()

        if 'Datetime' in df\_for\_output.columns:

             df\_for\_output['Datetime'] = pd.to\_datetime(df\_for\_output['Datetime']).dt.strftime('%Y-%m-%d %H:%M:%S')

        df\_for\_output = df\_for\_output.sort\_values(by='Datetime', na\_position='last').reset\_index(drop=True)

        df\_for\_output.to\_csv(output\_file\_path, index=False)

        print(f"\n--- Processing Complete ---")

        print(f"Processed CSV file created at: {output\_file\_path}")

        if not processed\_df.empty:

            print("\nSummary of Inferred Run counts:")

            print(processed\_df['Run'].value\_counts())

    except Exception as e:

        print(f"Script execution terminated due to an error: {e}")

        sys.exit(1)

The Python program run\_s\_p.py is a utility designed to analyze amateur radio contest logs and infer whether each contact (QSO) was made while "Running" (transmitting on a frequency and calling CQ, waiting for others to respond) or "Search & Pounce" (S&P, actively tuning across the band to find other stations calling CQ). The program adds a new column, 'Run', to the input DataFrame, classifying each QSO as 'Run' or 'S&P'.

The core algorithm is implemented in the \_evaluate\_single\_stream\_run function and leverages a "sticky run" state machine approach, analyzing QSOs within defined time and frequency tolerances.

**Algorithm Description:**

The algorithm processes the contest log by first grouping QSOs based on MyCall, Band, and Mode (e.g., CW, Phone). This allows for independent analysis of each operational "stream" within the contest.

For each stream, the algorithm maintains a state to determine if the operator is currently in a "Run" state or an "S&P" state. It uses a sliding window (a deque buffer) to consider recent QSOs for run detection.

Here's a breakdown of the key components and their interactions:

**1. Preprocessing (process\_contest\_log\_for\_run\_s\_p function):**

* **Input Validation:** Checks for the presence of essential columns (MyCall, Datetime, Frequency, Mode, Band).
* **Data Type Conversion:** Converts Datetime to pandas datetime objects and Frequency to numeric, handling potential errors by coercing to NaT or NaN and then dropping corresponding rows.
* **Sorting:** Sorts the DataFrame by MyCall, Band, Mode, and then Datetime. This is crucial for sequential processing of QSOs within each stream.
* **Grouping:** Iterates through groups defined by the combination of MyCall, Band, and Mode. This ensures that the run/S&P logic is applied independently to distinct operating conditions.
* **Tolerance Setting:** Determines the frequency tolerance (stream\_tolerance) based on the Mode (0.1 kHz for CW, 0.5 kHz for Phone).
* **Time Window:** Sets a time\_delta\_threshold (default 10 minutes + 1 second) for evaluating a run.

**2. Run Detection Helper (\_get\_run\_info\_from\_buffer function):**

* This helper function examines a deque (double-ended queue) of recent QSOs.
* Given a base\_freq, it identifies all QSOs in the buffer that are within the stream\_tolerance of base\_freq.
* It then checks if there are at least min\_qso\_for\_run (default 3) such QSOs within the time\_delta\_threshold.
* If these conditions are met, it indicates a potential "Run" on that base\_freq and returns the indices of the qualifying QSOs.

**3. Single Stream Evaluation (\_evaluate\_single\_stream\_run function):**

* This is the core state machine for determining "Run" or "S&P" status for a given operational stream (a group of QSOs).
* **Initialization:**
  + inferred\_run\_status: A list initialized with 'S&P' for all QSOs in the stream.
  + active\_run\_freq: Stores the frequency of the current active "Run," initialized to None.
  + last\_qso\_on\_run\_freq\_time: Timestamp of the last QSO on the active\_run\_freq.
  + off\_frequency\_qso\_count: Counts consecutive QSOs not on the active\_run\_freq.
  + potential\_new\_run\_freq: Tracks the frequency of consecutive S&P QSOs that might form a new run.
  + qso\_buffer: A deque to store recent QSOs (original index, datetime, frequency) within the time\_delta\_threshold.
* **Iterating through QSOs:** The algorithm processes each QSO in the stream sequentially.
  + **Buffer Management:** Adds the current QSO to qso\_buffer and removes old QSOs that fall outside the time\_delta\_threshold.
  + **State Machine Logic (Prioritizing Existing Run):**
    - **If active\_run\_freq exists (currently in a "Run"):**
      * **On-Frequency & Not Timed Out:** If the current\_qso\_freq is within stream\_tolerance of active\_run\_freq AND the time difference from last\_qso\_on\_run\_freq\_time is within RUN\_BREAK\_TIME\_MINUTES (default 2 minutes), the current QSO is classified as 'Run'. last\_qso\_on\_run\_freq\_time is updated, and counters are reset.
      * **Off-Frequency or Timed Out:** If the current QSO is not on the active\_run\_freq or a significant time has passed since the last run QSO:
        + The current QSO is initially classified as 'S&P'.
        + **"Sticky Run" Break Logic:**

If timed\_out (no QSO on run frequency for RUN\_BREAK\_TIME\_MINUTES), the active\_run\_freq is reset (run is broken).

If current\_qso\_freq is *not* on the active\_run\_freq:

It checks if this S&P QSO is on the *same* potential\_new\_run\_freq as previous S&P QSOs. If so, off\_frequency\_qso\_count increments.

If it's on a *different* S&P frequency, potential\_new\_run\_freq is updated, and off\_frequency\_qso\_count is reset to 1.

If off\_frequency\_qso\_count reaches RUN\_BREAK\_QSO\_COUNT (default 3), the active\_run\_freq is reset (run is broken). This implements the "sticky run" fix, ensuring a run only breaks if consecutive S&P QSOs are on the *same* new frequency.

* + **State Machine Logic (Checking for New Run):**
    - **If active\_run\_freq is None (not currently in a "Run"):**
      * Calls \_get\_run\_info\_from\_buffer using the current\_qso\_freq to see if a new run has formed in the qso\_buffer.
      * If a new run is detected:
        + active\_run\_freq is set to the current\_qso\_freq.
        + The last\_qso\_on\_run\_freq\_time is updated.
        + All QSOs identified as part of this new run (from new\_run\_indices) are reclassified as 'Run' in inferred\_run\_status.
      * Otherwise, the current QSO remains 'S&P'.

**4. Output:**

* The inferred\_run\_status (a pandas Series) is returned for each group.
* These Series are concatenated back into a single Series, which is then added as the 'Run' column to the original DataFrame.
* The modified DataFrame is saved to a new CSV file.

**Summary of the Algorithm:**

The algorithm employs a dynamic, state-based approach to classify QSOs:

1. **Stream Separation:** It first isolates distinct operating "streams" based on MyCall, Band, and Mode. This prevents interference between different operating setups or bands.
2. **Sliding Window Buffer:** For each stream, it maintains a buffer of recent QSOs within a defined time window.
3. **"Sticky Run" State Machine:**
   * **Maintaining a Run:** If a run is active, subsequent QSOs on the same frequency (within tolerance) continue the run. A run is "sticky" in that it persists even through a few off-frequency QSOs, provided they are on the *same* new frequency and don't exceed a defined count or time out.
   * **Breaking a Run:** A run breaks if:
     + No QSO occurs on the active run frequency for a specified time (RUN\_BREAK\_TIME\_MINUTES).
     + A defined number of consecutive off-frequency QSOs (RUN\_BREAK\_QSO\_COUNT) occur on the *same* new frequency.
   * **Initiating a New Run:** If no run is active, the algorithm constantly checks the buffered QSOs to see if a new run pattern (minimum QSOs within time and frequency tolerance) has emerged. If so, it identifies those past QSOs as part of the newly formed run.

This approach effectively distinguishes between a continuous "Running" operation and sporadic "Search & Pounce" activities by considering temporal proximity, frequency consistency, and a "forgiveness" period for brief deviations.