Process Signals Dashboard – Cheatsheet (Python, Bash, Git, CI)

A compact reference for your Streamlit + data processing workflow: structure, debugging, caching, tests, and CI.

1. Project Structure

- **app.py** - Streamlit UI and orchestration (gating with **st.stop()**).
- **src/io_excel.py** - Excel ingest, header discovery, numeric cleaning.
- **src/processing.py** - Moving averages and FFT utilities.
- **src/plotting.py** - Matplotlib (PNG) + Plotly (interactive) figures.
- **tests/** - pytest unit tests; keeps regressions from sneaking in.
- **.github/workflows/tests.yml** - CI pipeline (GitHub Actions).

2. Debugging Workflow (VS Code)

- Breakpoints stop only in Debug mode (**F5**). Running normally ignores breakpoints.
- Core controls: Step Over **F10**, Step Into **F11**, Continue **F5**.
- Use a local harness (e.g., **debug_runner.py**) to debug logic without Streamlit reruns.
- Watch these first when things look wrong: **df.columns**, NaN counts, array lengths, and time monotonicity.

3. Moving Averages (MA)

- **Centered SMA** (offline analysis): uses past+future; no phase shift; not causal.
- **Trailing SMA** (real-time/causal): uses current+past; introduces lag.
- **EMA** (real-time/causal): smoother with less lag than trailing SMA.
- Edge padding avoids artificial drops at the beginning/end of the MA curve.

4. FFT Sanity Checks

- FFT assumes time is monotonic and roughly uniform. Check: **np.all(np.diff(t) > 0)**.
- Sampling rate: **fs = 1 / median(dt)**. Nyquist max frequency approx **fs/2**.
- Large low-frequency spikes usually mean drift/trend (mean removal reduces DC but not slope).

5. Streamlit Mental Model

- Streamlit reruns top-to-bottom on every interaction. Guard expensive steps with gating and caching.
- Cache data I/O and preprocessing; avoid caching plots unless needed.
- Put cached helpers near the top of **app.py** (stable definitions).

6. Python Essentials for Robust Projects

- **pathlib.Path** for paths (portable across Windows/Linux).
- **typing** (e.g., List, Tuple) and type hints for readability and IDE help.
- **dataclasses** for small data containers (cleaner than dicts).
- **logging** instead of print for structured debugging (**logging.info/error**).
- **try/except** + **raise** for predictable failures and good messages.
- Common sanity checks: **len(x)**, **np.isfinite**, **Series.isna().sum()**.

Useful Python snippets

• Safe numeric conversion (pandas):

```
pd.to_numeric(series, errors='coerce')
```

• Time monotonicity check:

```
np.all(np.diff(t) > 0)
```

• File-not-found pattern:

```
if not path.exists(): raise FileNotFoundError(path)
```

7. Bash / CLI Commands You Will Use Constantly

Windows PowerShell equivalents usually work; GitHub Actions runners use Linux-style shells.

- **Create venv**: python -m venv .venv
- **Activate (PowerShell)**: .\.venv\Scripts\Activate.ps1
- **Upgrade pip**: python -m pip install --upgrade pip
- **Install deps**: pip install -r requirements.txt
- **Snapshot deps**: pip freeze > requirements.txt
- **Run tests**: pytest -q
- **Run Streamlit**: streamlit run app.py

8. Git Commands (Team-Ready Workflow)

- **Status**: git status
- **Create branch**: git checkout -b feature/my-change
- **Commit**: git add . then git commit -m "message"
- **Push branch**: git push -u origin feature/my-change
- **Update main**: git checkout main then git pull

9. CI Basics (GitHub Actions)

- Workflows live in **.github/workflows/** and run on push/PR.
- CI should run **pytest** and block merges when red.
- Use branch protection: require CI checks + PR before merging to **main**.

Mindset shift: from 'does it run?' to 'is it correct, testable, and robust?'.