🏔️ Avalanche Scenario Model Chain (2025-11 Update)

⚠️ Handle with care — work in progress

Overview

- The Avalanche Scenario Model Chain is developed with in EUREGIO Project CAIROS
- The Avalanche Scenario Model Chain steps are the preprocessing for the Avalanche Scenario Mapper
- The Avalanche Scenario Model Chain is a orchestrates the full automated avalanche modelling workflow — from raw terrain data to structured delineation of potential release areas (PRAs) and size dependent avalanche simulations.
    - Main entrypoint: cairosModelChain/runCairos.py
    - Orchestrates 15 workflow steps (Step 00–15)
    - Integrates AvaFrame (FlowPy engine) in editable mode via Pixi
    - Manages directory trees, inputs, and processing configurations automatically

---

Cairos/cairosModelChain Repository layout

```
Cairos/
└── cairosModelChain/          # Main Python package (modular CAIROS workflow)
    │
    ├── cairosCfg.ini          # Default configuration (global)
    ├── local_cairosCfg.ini    # Local project override (preferred for runs)
    │
    ├── runCairos.py           # Main driver (Steps 00–15, orchestrates workflow)
    ├── runInitWorkDir.py      # Step 00 – Initialize project directory + logs
    ├── runPlots.py            # Optional plotting entrypoint (not in main workflow)
    │
    ├── com1PRA/               # Step 01–08: Potential Release Area (PRA) workflow
    │   ├── praDelineation.py          # Step 01 – Derive PRA field from DEM + forest
    │   ├── praSelection.py            # Step 02 – Apply thresholds, aspect + region masks
    │   ├── praSubCatchments.py        # Step 03 – Delineate subcatchments (WhiteboxTools)
    │   ├── praProcessing.py           # Step 04 – Clean and polygonize PRA masks → GeoJSON
    │   ├── praSegmentation.py         # Step 05 – Intersect PRAs with subcatchments → GeoJSON
    │   ├── praAssignElevSize.py       # Step 06 – Classify PRAs by elevation + area size
    │   ├── praPrepForFlowPy.py        # Step 07 – Prepare PRAs for FlowPy simulation
    │   ├── praMakeBigDataStructure.py # Step 08 – Build aggregated FlowPy input tree
    │   ├── bottleneckSmoothing.py     # Not used ATM
    │   └── __init__.py
    │
    ├── com2AvaDirectory/      # Step 09–15: FlowPy & Avalanche Directory chain
    │   ├── avaDirBuildFromFlowPy.py   # Step 13 – Convert FlowPy results to AvaDirectory
    │   ├── avaDirType.py              # Step 14 – Build scenario type structure (dry/wet)
    │   ├── avaDirResults.py           # Step 15 – Aggregate final scenario results/maps
    │   └── __pycache__/
```

```
    │
    ├── in1Utils/              #Core utilities (shared across all modules)
    │   ├── cfgUtils.py         # Config handling, GDAL/PROJ setup, manifest writers
    │   ├── dataUtils.py        # Raster/vector I/O, compression, helper functions
    │   ├── plottingUtils.py    # Plotting helpers (matplotlib/geopandas)
    │   ├── workflowUtils.py    # Workflow flag parsing, discovery of FlowPy leaves
    │   └── __pycache__/
    │
    ├── in2Parameter/          # Parameterization + FlowPy integration
    │   ├── compParams.py       # Step 09/11 – Compute size-dependent FlowPy parameters
    │   ├── sizeParameters.py   # Parameter range management for simulation inputs
    │   ├── muxi.py             # Additional parameter computation utilities
    │   └── __pycache__/
    │
    ├── outPlots/              # Optional plotting layer
    │   ├── out1SizeParameter.py # Plot FlowPy parameter outputs (alpha/umax/etc.)
    │   ├── plotFunctions.py    # Common plotting logic
    │   └── __pycache__/
    │
    └── __pycache__/
```

Quick start (Linux)

1. Minimal system prerequisites

- Install only the flexible, OS-level tools system-wide.
- All project dependencies (AvaFrame, CAIROS, GDAL, NumPy, etc.) will live inside Pixi-managed environments.
- System Python (python3.10 or 3.11 on Ubuntu) is kept minimal — used only for tools like VS Code, Pixi bootstrap, etc.
- CAIROS and AvaFrame never touch system Python → they live in isolated .pixi/envs/* environments.
- Optional:
  - Install VS Code for editing, debugging, and integrated terminals: https://code.visualstudio.com/download

```
# System-wide basics
sudo apt update
sudo apt install -y git python3 python3-pip

# Install Pixi (recommended via installer script)
curl -fsSL https://pixi.sh/install.sh | bash
# restart your shell so `pixi` is in PATH
```

2) Install AvaFrame (temporary: dev branch)

```
# choose your workspace directory, e.g. ~/Documents/Applications
cd ~/Documents/Applications

# clone AvaFrame
git clone https://github.com/avaframe/AvaFrame.git
cd AvaFrame
# checkout branch until next release (ATM)
git checkout PS_FP_outputRelInfo
```

Two options from here:

Option A: Use AvaFrame via CAIROS

- Nothing else required — CAIROS links AvaFrame in editable mode automatically through Pixi.
- Skip the manual build step.

Option B: Use AvaFrame standalone

- If you want to run AvaFrame directly (outside CAIROS), you need to compile the Cython parts:

```
# set the avalanche dir in local_avaFrameCfg.py and apply your settungs to
local_com4FlowPyCfg.ini
cd AvaFrame
pixi shell
python setup.py build_ext --inplace
# checkout branch until next release (ATM)
git checkout PS_FP_outputRelInfo
pixi shell --environment dev
cd AvaFrame/avaframe
python runCom4FlowPy.py
```

- Repeat this step whenever Cython code changes or after pulling new updates.

3) Install CAIROS repo

```
# choose your workspace directory next to AvaFrame
cd ~/Documents/Applications
git clone <your-cairos-repo> Cairos    # !!!!!!
cd Cairos
```

4) Setup CAIROS ModelChain env

```
# Clean any old envs if something is corrupted
rm -f pixi.lock
pixi clean -e dev || true
#pixi clean cache || true
rm -rf .pixi

# Install dev env (with local AvaFrame)
pixi install -e dev

# Check that CAIROS uses your local AvaFrame
pixi shell -e dev
python -c "import avaframe, pathlib; print(pathlib.Path(avaframe.__file__).resolve())"
> .../Documents/Applications/AvaFrame/avaframe/__init__.py
```

5) Configure

Copy the defaults and edit the **local** copies:

```
# CAIROS ModelChain config
cd Cairos/cairosModelChain
cp cairosCfg.ini local_cairosCfg.ini

# AvaFrame config
cd ../AvaFrame/avaframe
cp avaframeCfg.ini local_avaframeCfg.ini

# AvaFrame FlowPy config
cd ../AvaFrame/avaframe/com2FlowPy
cp flowPyAvaFrameCfg.ini local_flowPyAvaFrameCfg.ini
```

Running cairos ...

- Fill in `local_cairosCfg.ini` → `[MAIN]` with your project info and input filenames (the files must exist in the run's `00_input/` folder once the project is initialized).
- Adapt your local_*Cfg.ini's
- Details TBA....

```
#
───────────────────────────────────────────────────────────────────────────────────────────────
#   ███████  A V A L A N C H E · S C E N E N A R I O · M O D E L · C H A I N  ████████
#
───────────────────────────────────────────────────────────────────────────────────────────────
#
#   ██████╗  ██╗  ██╗ ██████╗     ████████╗  ██████╗ ███████╗ ███╗   ██╗
#   ██╔══██╗ ██╗  ██║ ██╔══██╗    ╚██╔════╝ ██╔════╝ ██╔════╝ ████╗  ██║
#   ███████║ ██║ ██╔╝ ███████║     ███████╗ ██║      █████╗   ██╔██╗ ██║
#   ██╔══██║ ██║██╔╝  ██╔══██║     ╚════██║ ██║      ██╔══╝   ██║╚██╗██║
#   ██║  ██║ ╚███╔╝   ██║  ███╗██╗████████║ ╚██████╗ ███████╗ ██║ ╚████║ █████╗ ███╗██╗
#   ╚═╝  ╚═╝  ╚══╝    ╚═╝  ╚══╝╚═╝╚═══════╝  ╚═════╝ ╚══════╝ ╚═╝  ╚═══╝ ╚════╝ ╚══╝╚═╝
#
───────────────────────────────────────────────────────────────────────────────────────────────
#   ███████  runCairos.py  ·  runCairos.py  ·  runCairos.py  ·  runCairos.py  ████████
#
───────────────────────────────────────────────────────────────────────────────────────────────
```

```
cd Cairos/cairosModelChain #location of runCairos.py
pixi run -e dev cairos
```

- after first initialzation run you see:

```
INFO:__main__:

       ============================================================================
          ... Start main driver for CAIROS model chain (2025-11-06 13:11:24) ...
       ============================================================================

INFO:__main__: Config file:
/home/christoph/Documents/Applications/Cairos/cairosModelChain/local_cairosCfg.ini
INFO:__main__: Step 00: Initializing project...
INFO:runInitWorkDir: cairosDir:
/media/christoph/Daten/Cairos/ModelChainProcess/cairosTutti/pilotSellaTest/alpha32_3_umax8_18_ma
xS5_
INFO:runInitWorkDir: ...cairosDir: ./.
INFO:runInitWorkDir: ...inputDir: ./00_input
INFO:runInitWorkDir: ...praDelineationDir: ./01_praDelineation
INFO:runInitWorkDir: ...praSelectionDir: ./02_praSelection
INFO:runInitWorkDir: ...praBottleneckSmoothingDir: ./03_praBottleneckSmoothing
INFO:runInitWorkDir: ...praSubcatchmentsDir: ./04_praSubcatchments
INFO:runInitWorkDir: ...praProcessingDir: ./05_praProcessing
INFO:runInitWorkDir: ...praSegmentationDir: ./06_praSegmentation
INFO:runInitWorkDir: ...praAssignElevSizeDir: ./07_praAssignElevSize
INFO:runInitWorkDir: ...praPrepForFlowPyDir: ./08_praPrepForFlowPy
INFO:runInitWorkDir: ...praMakeBigDataStructureDir: ./09_flowPyBigDataStructure
INFO:runInitWorkDir: ...flowPySizeParametersDir: ./09_flowPyBigDataStructure
INFO:runInitWorkDir: ...flowPyRunDir: ./09_flowPyBigDataStructure
INFO:runInitWorkDir: ...flowPyResToSizeDir: ./10_flowPyOutput
INFO:runInitWorkDir: ...flowPyOutputDir: ./10_flowPyOutput
INFO:runInitWorkDir: ...avaDirDir: ./11_avaDirectoryData
INFO:runInitWorkDir: ...avaDirTypeDir: ./12_avaDirectory
INFO:runInitWorkDir: ...avaDirResultsDir: ./12_avaDirectory
INFO:runInitWorkDir: ...avaDirIndexDir: ./12_avaDirectory
INFO:runInitWorkDir: ...avaScenMapsDir: ./13_avaScenMaps
INFO:runInitWorkDir: ...avaScenPreviewDir: ./14_avaScenPreview
```

```
INFO:runInitWorkDir: ...plotsDir: ./91_plots
INFO:runInitWorkDir: ...gisDir: ./92_GIS
INFO:__main__: Step 00: Project initialized in 0.01s
INFO:__main__: Step 00: Log file: runCairos_20251106_131124.log
ERROR:__main__: Step 00: Required input files are missing in ./00_input:
ERROR:__main__:    - DEM=10DTM_pilotSellaTest.tif
ERROR:__main__:    - FOREST=10nDOM_binAgg_100_pilotSellaTest_forestCom.tif
ERROR:__main__:    - BOUNDARY=regionPilotSella.geojson
ERROR:__main__:

          ... Please provide the required input files and run again ...
```

- Copy or prepare these files into your project's `00_input/` directory.
- Their filenames must match the entries defined in your INI's `[MAIN]` section e.g:

```
[MAIN]
DEM      = 10DTM_pilotSellaTest.tif
FOREST   = 10nDOM_binAgg_100_pilotSellaTest_forestCom.tif
BOUNDARY = regionPilotSella.geojson
COMMISSIONREGION = commRegionExtentPilotSella.geojson
COMMISSIONS = commissionsEuregio.geojson
AVAREPORT = avaReportMicroRegions.geojson
```

- run again...

```
cd Cairos/cairosModelChain #location of runCairos.py
pixi run -e dev cairos
```

- when all input is provided and checked you will see:

```
...
INFO:__main__: Step 00: Project initialized in 0.01s
INFO:__main__: Step 00: Log file: runCairos_20251106_113707.log
INFO:__main__: Step 00: Input DEM validated: nodata + CRS check done.
INFO:__main__: Step 00: Input FOREST validated: nodata + CRS check done.
INFO:__main__: Step 00: All raster inputs validated: DEM + FOREST nodata/CRS checked and safe.
INFO:__main__: All inputs complete:
/media/christoph/Daten/Cairos/ModelChainProcess/cairosTutti/pilotSellaTest/alpha32_3_umax8_18_ma
xS5/00_input

       ============================================================================
               ... LET'S KICK IT - AVALANCHE SCENARIOS in 3... 2... 1...
       ============================================================================
...
```

DEM-driven consistency rule

- **NOTE**: All rasters (REL, RELID, ALPHA, UMAX, EXP, Outputs) must inherit:
  - CRS, transform, and nodata from `[MAIN].DEM`
- identical width/height for raster alignment
- **Any deviation triggers a warning during preprocessing or FlowPy parameterization.**

Running a single leaf

- Instead of enumerating the BigData tree, set in `[WORKFLOW]`:

```
#test only
makeSingleTestRun = False
singleTestDir = pra030secE-1800-2000-4
```

- With this Step 09-15 will parameterize **that** leaf (`pra030secE-1800-2000-4`).

What the workflow does (Steps 00–15)

Step 00 — Initialize project folders

- Creates the standardized CAIROS run directory structure based on `[MAIN]` in your `cairosCfg.ini`.
  - Each run lives in its own tree:

```
<workDir>/<project>/<ID>/
├── 00_input/                 ← User-provided inputs (DEM, FOREST, BOUNDARY, etc.)
│
├── 01_praDelineation/        ← Step 01: Derived PRA raster field + terrain layers
(slope/aspect)
├── 02_praSelection/          ← Step 02: Filtered PRA rasters by threshold, elevation, and
aspect
│
├── 03_praBottleneckSmoothing/  ← Not used ATM
├── 04_praSubcatchments/      ← Step 03: Subcatchment rasters + polygons (via WhiteboxTools)
├── 05_praProcessing/         ← Step 04: Cleaned & polygonized PRA masks (GeoJSON)
├── 06_praSegmentation/       ← Step 05: PRAs segmented by subcatchments (GeoJSON)
├── 07_praAssignElevSize/     ← Step 06: PRAs classified by elevation bands and size
├── 08_praPrepForFlowPy/      ← Step 07: Prepared PRA inputs for FlowPy (GeoJSON + metadata)
├── 09_flowPyBigDataStructure/  ← Step 08: FlowPy BigData structure (SizeN/{dry,wet}/Inputs
tree)
│
├── 10_flowPyOutput/          ← Steps 09–12: FlowPy results, size aggregation, compression
│
├── 11_avaDirectoryData/      ← Step 13: Raw AvaDirectory data collected from FlowPy outputs
├── 12_avaDirectory/          ← Steps 14–15: Unified AvaDirectoryType & Results (CSV, GeoJSON,
Parquet)
│
├── 13_avaScenMaps/           ← Step 16 (planned): Automated avalanche scenario map generation
├── 14_avaScenPreview/        ← Optional previews for avalanche scenarios
│
├── 91_plots/                 ← Diagnostic plots, QA visualizations, and size parameter
distributions
└── 92_GIS/                   ← GIS-ready exports (merged shapefiles, GeoPackages, layers)
```

Log file

- Each workflow run automatically creates a timestamped log file:

  ```
  <workDir>/<project>/<ID>/runCairos_YYYYMMDD_HHMMSS.log
  ```

Steps 01–08 — PRA processing (`com1PRA`)

- The PRA chain defines the complete pre-processing stage of CAIROS — from delineating potential release areas to creating structured, FlowPy-ready input datasets.
- Each step builds directly on the previous one, and together they establish the BigData foundation used in later FlowPy and AvaDirectory processing.

Step	Module	Main INI Sections	Description

Step	Module	Main INI Sections	Description
01	`com1PRA/praDelineation.py`	`[praDELINEATION]`, `[MAIN]`	Detects potential release areas (PRA) from DEM and slope; outputs base PRA raster + aspect layer.
02	`com1PRA/praSelection.py`	`[praSELECTION]`	Applies filtering thresholds (e.g. area, elevation, slope) to select relevant PRA regions.
03	`com1PRA/praSubCatchments.py`	`[praSUBCATCHMENTS]`	Generates subcatchment polygons using WhiteboxTools; prepares catchment delineations.
04	`com1PRA/praProcessing.py`	`[praPROCESSING]`	Cleans, dissolves, and vectorizes PRA rasters; outputs unified PRA GeoJSONs.
05	`com1PRA/praSegmentation.py`	`[praSEGMENTATION]`	Intersects PRAs with subcatchments to segment them into manageable units.
06	`com1PRA/praAssignElevSize.py`	`[praASSIGNELEV]`, `[praSEGMENTATION]`	Assigns elevation bands and size classes to each segmented PRA.
07	`com1PRA/praPrepForFlowPy.py`	`[praPREPFORFLOWPY]`, `[WORKFLOW]`	Converts PRAs into FlowPy input-ready GeoJSONs and ensures consistent CRS and naming.
08	`com1PRA/praMakeBigDataStructure.py`	`[praMAKEBIGDATASTRUCTURE]`, `[WORKFLOW]`	Aggregates all PRA data (rasters + GeoJSONs) into the structured BigData tree.

- **NOTE**: The table lists only the primary INI sections.
  - Several steps internally reference additional parameters (e.g. from `[MAIN]`, `[avaPARAMETER]`, or `[praSEGMENTATION]`).

Output of Step 08 — FlowPy BigData Tree

- Each case (PRA × size × elevation band) is written into a **BigData tree** designed to match AvaFrame's expected input structure for FlowPy runs.

```
09_flowPyBigDataStructure/
└── BnCh2_subC500_100_5_sizeF500/         ← Root: parameterized subcatchment/size case
    ├── pra030secS-2000-2200-3/           ← Case: single PRA scenario (aspect/elev/size)
    │   ├── Size2/
    │   │   ├── dry/
    │   │   │   ├── Inputs/
    │   │   │   │   ├── REL/              ← Rasterized release masks (PRA polygons)
    │   │   │   │   │   ├── pra030secS-2000-2200-3-praAreaM.tif
    │   │   │   │   │   └── pra030secS-2000-2200-3-praBound.tif
    │   │   │   │   ├── RELID/            ← PRA IDs encoded as integer rasters
    │   │   │   │   │   └── pra030secS-2000-2200-3-praID.tif
    │   │   │   │   ├── RELJSON/          ← PRA geometry + metadata (GeoJSON)
    │   │   │   │   │   └── pra030secS-2000-2200-3.geojson
    │   │   │   │   ├── ALPHA/            ← Computed FlowPy input (Step 09)
    │   │   │   │   ├── UMAX/
    │   │   │   │   ├── EXP/
    │   │   │   │   └── DEM.tif           ← Optional local DEM reference (if enabled)
    │   │   │   └── Outputs/
    │   │   │       └── com4FlowPy/       ← FlowPy outputs (Step 10)
    │   │   └── wet/
    │   └── Size3/
    │       └── dry/...
    └── pra030secN-2200-2400-5/...
```

Terminology & Naming Conventions

Term	Description
Root	The main parameter-case folder (defined by `[praPROCESSING]`, `[praSUBCATCHMENTS]`, `[praSEGMENTATION]`). Example: `BnCh2_subC500_100_5_sizeF500` (constructed from default PRA parameters).
Case	A single PRA release scenario, combining PRA ID, elevation range, and size. Formed from `[praDELINEATION]`, `[praSELECTION]`, `[praASSIGNELEV]`, `[avaPARAMETER]`. Example: `pra030secS-2000-2200-3`.
SizeN	Size class folder derived from the case's maximum potential size (`[avaPARAMETER]`.sizeRange). Example: `pra...-4` → `Size2`, `Size3`, `Size4`.
Scenario	Flow regime folder: either `dry/` or `wet/`.
Leaf	The lowest-level folder — `SizeN/scenario/` — containing `Inputs/` and `Outputs/` subdirectories for FlowPy processing.

- **NOTE**: No `Size5` for `wet/` Avalanches!!!

Summary:

- Steps 01–08 create the foundation of the CAIROS workflow.
- They transform raw terrain and PRA data into a fully structured **BigData input tree**, ready for parameterization (Step 09) and FlowPy simulations (Step 10).

Steps 09–15 — FlowPy and AvaDirectory Chain

Step 09 — Parameterization (per leaf)

- Code: `in2Parameter/compParams.py`
- Inputs: DEM + PRA release (`Inputs/REL/pra*.tif`)
- Uses `[avaPARAMETER]` and `[avaSIZE]` to compute `ALPHA`, `UMAX`, and `EXP` once per leaf.
- **Folder rule:** if a leaf path contains `.../SizeN/...`, the computed size is **clamped to** `N` before mapping to ALPHA/UMAX/EXP.
- DEM selection logic (handled via `workflowUtils.demForLeaf(...)`):
  - For BigData leaves (default): use `00_input/<DEM>` from `[MAIN].DEM`
  - For single or manual runs: fallback to `Inputs/DEM.tif` if present

Step 10 — Run FlowPy (per leaf)

- **NOTE**: FlowPy is now executed directly through AvaFrame — there is no `runCairosFlowPy.py` anymore.
- Driver: `cairos/runCairos.py`
- FlowPy INI: `AvaFrame/avaframe/com4FlowPy/com4FlowPyCfg.ini`
  - Copy to `local_com4FlowPyCfg.ini` before editing

Example FlowPy configuration used for CAIROS runs:

```
[GENERAL]
infra = False
previewMode = False
variableUmaxLim = True              # important for CAIROS
varUmaxParameter = uMax             # important for CAIROS
variableAlpha = True                # important for CAIROS
variableExponent = True             # important for CAIROS
forest = False                      # important for CAIROS
...
```

```
# computational defaults
tileSize = 12000
tileOverlap = 4000
procPerCPUCore = 1
chunkSize = 50
maxChunks = 500

[PATHS]
outputFileFormat = .tif
outputNoDataValue = -9999
outputFiles = zDelta|travelLengthMax|fpTravelAngleMax|cellCounts|relIdPolygon
useCustomPaths = False
deleteTempFolder = True
useCustomPathDEM = True             # important for CAIROS
workDir =
demPath = ...00_input/10DTM_pilotSellaTest.tif  # important for CAIROS
...

[FLAGS]
plotPath = False
plotProfile = False
saveProfile = True
writeRes = True
fullOut = False
```

Step 11 — Back-map FlowPy outputs to size (optional)

- Description:
  - Writes new size-based results into:
    - `<leaf>/Outputs/com4FlowPy/sizeFiles/res_<uid>/...`
  - where `<uid>` is the FlowPy run identifier created by AvaFrame.
  - Each size file corresponds to a resampled or aggregated result from the original FlowPy output, grouped per PRA and per size class.
- Code:
  - `in2Parameter/compParams.py::computeAndSaveSize`
- Controlled by:
  - `[WORKFLOW].flowPyOutputToSize`
- Writes new size-based results into:
  - `<leaf>/Outputs/com4FlowPy/sizeFiles/res_<uid>/...`
  - where `<uid>` is the FlowPy run identifier created by AvaFrame.

Step 12 — Output management and cleanup (optional)

- TBA

Step 13 — Build AvaDirectory from FlowPy

- Description:
  - Collects all `com4FlowPy` outputs for each scenario and merges them into a structured **AvaDirectoryData** tree.
  - Handles optional `RELJSON` merges, per-PRA splitting, and raster clipping for both **dry** and **wet** flow scenarios.
- Code:
  - `com2AvaDirectory/avaDirBuildFromFlowPy.py`
- Controlled by:
  - `[WORKFLOW].avaDirBuildFromFlowPy`
- Inputs:
  - `09_flowPyBigDataStructure/<caseFolder>/pra*/Size*/dry|wet/Outputs/com4FlowPy/`
- Outputs:
  - `11_avaDirectoryData/<caseFolder>/com4_/praID.geojson` + rasters
  - `11_avaDirectoryData/<caseFolder>/avaDirectory.csv`

Step 14 — Build AvaDirectory Type

- Description:
  - Merges all PRA-level GeoJSONs into a unified avaDirectoryType dataset.
  - Cleans, normalizes, and deduplicates attributes across all dry/wet and rel/res combinations.
  - Provides the master dataset for raster path enrichment in Step 15.
- Code:
  - com2AvaDirectory/avaDirType.py
- Controlled by:
  - [WORKFLOW].avaDirType
- Inputs:
  - 11_avaDirectoryData/<caseFolder>/com4_*/praID*.geojson
- Outputs:
  - 12_avaDirectory/<caseFolder>/avaDirectoryType.csv
  - 12_avaDirectory/<caseFolder>/avaDirectoryType.geojson
  - 12_avaDirectory/<caseFolder>/avaDirectoryType.parquet

Step 15 — Build AvaDirectory Results

- Description:
  - Builds the enriched avaDirectoryResults dataset by attaching relative raster paths to each (praID, resultID) combination
  - The .pkl index maps:
  - (praID, resultID) → { rasterType: path, ... } for all available simulation outputs.
  - These results form the foundation for Avalanche Scenario Mapper (scenario mapping, under development).
- Code:
  - com2AvaDirectory/avaDirResults.py
- Controlled by:
  - [WORKFLOW].avaDirResults
- Inputs:
  - 12_avaDirectory/<caseFolder>/avaDirectoryType.parquet
  - `11_avaDirectoryData//com4_*/.tif
- Outputs:
  - 12_avaDirectory/<caseFolder>/avaDirectoryResults.csv
  - 12_avaDirectory/<caseFolder>/avaDirectoryResults.geojson
  - 12_avaDirectory/<caseFolder>/avaDirectoryResults.parquet
  - 12_avaDirectory/<caseFolder>/indexAvaFiles.pkl

Summary:

- Steps 09–15 form the complete FlowPy + AvaDirectory pipeline.
- They parameterize, simulate, post-process, and structure all avalanche scenarios into reusable, indexed datasets — ready for mapping, visualization, and scenario-based analysis.

INI

- tbc...

---

thats it for now - tbc...