

OSF — Cadre Méthodique (A)

DD-Coherence - exécution de l'opérateur DD sur CSV

0) Identité / périmètre

- **Nom recommandé OSF (Title) :** *DD-Coherence (Loki) — exécution de l'opérateur DD sur séries CSV*
- **Dépôt source (URL) :** github.com/dalozedidier-dot/DD-Coherence (alias)
; lecture/README redirige vers [dalozedidier-dot/Loki](https://github.com/dalozedidier-dot/Loki).
- **Type de ressource (Resource Type) :** *Software*
- **Statut observé :** dépôt public, 5 commits, Python 100%.
- **Version à figer (Registration snapshot) :**
 - Branch : [main](#)
 - Tag : [\[...\]](#)
 - Commit hash : [\[...\]](#)
 - Date de figement : [\[...\]](#)

1) Définition (Description / Abstract)

DD-Coherence (Loki) est un outil d'exécution de l'opérateur **DD** appliqué à des **données tabulaires (CSV)**. Il charge une ou plusieurs séries, applique un traitement DD selon un **paramétrage externalisé (JSON)**, puis produit des **artefacts auditables** : rapport structuré et séries dérivées. Le système est conçu pour être exécuté localement et/ou via **GitHub Actions** (smoke test et exécution batch).

2) Matériels / composants à archiver (Materials)

Inclure dans l'OSF (idéalement en **copie**/archive, pas seulement lien) :

- Répertoires :
 - [dd_coherence_tool/](#) (moteur + scripts + tests + CSV d'exemple)

- `.github/workflows/` (orchestration CI)
- Fichiers de paramètres : `dd_params.example.json`, `dd_params.small.json`
- Documentation : `README.md`, `HOWTO_GIT_FILES.md`
- **Archive figée** (recommandé) : ZIP du dépôt au commit/tag enregistré (OSF “Files”).

3) Données (Data / Inputs)

- **Format** : CSV.
- **Exigences structurelles** (à déclarer explicitement dans OSF) :
 - Colonnes cibles : ex. `'Failure rate'`, `'Avg job run time'` (exemples dans la commande).
 - Longueur minimale : un CSV “1 ligne” est marqué **skipped** (règle déclarée).
 - Ordre : DD nécessite un **ordre** (temps ou séquence) ; si absent, l'ordre retenu doit être déclaré comme hypothèse de protocole.

4) Procédure opératoire (Methods)

Installation (local)

```
pip install -r dd_coherence_tool/requirements.txt
```

Exécution unitaire (exemple)

```
python dd_coherence_tool/scripts/run_dd.py \
  --input dd_coherence_tool/1.csv \
  --outdir out_dd \
  --cols "'Failure rate" "'Avg job run time"
```

Exécution batch (exemple)

```
python dd_coherence_tool/scripts/run_dd_batch.py \
  --outdir out_dd \
  --config dd_params.small.json \
  dd_coherence_tool/1.csv dd_coherence_tool/2.csv
dd_coherence_tool/3.csv
```

Orchestration CI (GitHub Actions)

- `dd_smoke.yml` : smoke test + production possible de `dd_components.csv.gz`
- `dd_on_committed_csvs.yml` : exécution DD sur CSV commités (`1.csv`, `2.csv`, `3.csv`)

5) Sorties / produits analytiques (Outputs)

Chaque run écrit (sorties déclarées) :

- `dd_report.json`
- `dd_series.csv`
- `dd_components.csv.gz` (si activé et si série assez longue)

6) Validation / critères de vérifiabilité (QA / Validation)

- Critères “succès” minimaux :
 - exécution sans erreur (exit code 0)
 - présence des fichiers de sortie attendus (liste ci-dessus)
 - conformité structurelle (à définir) : schéma JSON minimal pour `dd_report.json` + format CSV attendu
- Critère “skipped” :
 - série trop courte ⇒ run marqué “skipped” (à tester et documenter).

7) Licence / réutilisation (License)

- **Champ OSF License** : à renseigner explicitement.
- Si aucune licence n’est présente/affichée côté dépôt : indiquer “non spécifiée” (constat) et, si souhaité, ajouter un fichier LICENSE dans la version figée.

8) Limites (Limitations)

- Sans **tag/hash** + archive figée, l'objet "software" reste non-fixe.
- Sans schéma d'entrée/sortie, l'audit de conformité se réduit à la présence de fichiers.
- Le dispositif ne produit pas d'explication causale : sorties = constats computationnels (mesures/ruptures/flags) selon paramètres.