Ça marche sur ma machine, et c'est le rapport de bogue

Le même script, les mêmes données et une autre version de pandas produisent d'autres nombres. Un fichier de verrouillage est ce qui transforme « ça marche sur ma machine » d'une défense en une affirmation vérifiable, et cela coûte une commande.

Pierre Robentz Cassion

Leçon 3 sur 8

Ce que couvre cette leçon

- Ce que l'ordinateur d'un collègue fait différemment
- Python : `uv`
- R : `renv`
- Pourquoi un fichier de verrouillage et non `requirements.txt`
- Ce que cette plateforme fige, et pourquoi un verrou est inhabituel
- Les conteneurs, et quand ils en valent la peine
- Le test qui le prouve
- Rapportez-le en entier
- La suite

Ce que l'ordinateur d'un collègue fait différemment

Ce qui diffère	Ce que cela change
Version de paquet	Un argument par défaut, une règle d'arrondi, un ordre de tri
Version du langage	Ordre des dictionnaires, division entière, traitement des chaînes
Système d'exploitation	Fins de ligne, ordre des fichiers, séparateurs de chemin, locale
Locale	`1,234` interprété comme 1234 ou comme 1,234
Ce qui est déjà installé	Une bibliothèque que votre script importe sans jamais la déclarer

Ce que l'ordinateur d'un collègue fait différemment

- Rien de tout cela n'est dans votre dépôt — raison pour laquelle un code identique ne suffit pas

Python : uv — Shell

```
uv init                       # creates pyproject.toml
uv add pandas matplotlib      # records the dependency and resolves it
uv run python run.py          # runs inside the pinned environment
```

Python : `uv`

- Deux fichiers, et les deux sont commités — `pyproject.toml` dit ce que vous avez demandé — `pandas>=2.0` — et `uv.lock`...
- Un collègue lance `uv sync` et dispose de votre environnement — sur son système, sans que vous sachiez ce qu'il avait...
- Figez aussi la version du langage

Python : uv — Exemple

```
# pyproject.toml
requires-python = ">=3.12,<3.13"
```

- Une plage de versions, non une version unique — Figer à `3.12.4` exactement signifie qu'un collègue en `3.12.7` ne peut...

R : renv — En R

```
renv::init()                # snapshots the project library
renv::snapshot()            # after adding a package
renv::restore()             # on the colleague's machine
```

- `renv.lock` est l'artefact équivalent et il est commité — Il enregistre chaque paquet, sa version et le dépôt d'où il...
- Enregistrez la version de R, ce que `renv` fait automatiquement — et vérifiez-la dans le script là où une différence...

R : renv — En R

```r
if (getRversion() < "4.3.0") stop("This analysis requires R >= 4.3.0")
```

R : renv — En Python

```python
# The Python equivalent, at the top of the entry point.
import sys
assert sys.version_info >= (3, 12), "This analysis requires Python 3.12+"
```

Pourquoi un fichier de verrouillage et non `requirements.txt` — Exemple

```
pandas>=2.0
matplotlib
```

Pourquoi un fichier de verrouillage et non `requirements.txt`

- Ce fichier ne dit presque rien — Il se résout en versions différentes selon les jours et ne mentionne pas les cinquante...

Pourquoi un fichier de verrouillage et non `requirements.txt`

	Déclare	**Reproduit**
`requirements.txt`	L'intention	Non
`requirements.txt` avec des `==`	Une couche	En partie — les dépendances transitives flottent
`uv.lock` / `renv.lock`	L'arbre entier	**Oui**

Pourquoi un fichier de verrouillage et non `requirements.txt`

- Commitez l'intention et le verrou — Le fichier d'intention est ce qu'un humain édite ; le verrou est ce depuis quoi une...

Ce que cette plateforme fige, et pourquoi un verrou est inhabituel — Exemple

```
{
  "packageManager": "pnpm@11.9.0",
  "engines": { "node": ">=22" },
  "devDependencies": { "typescript": "^6.0.3" }
}
```

Ce que cette plateforme fige, et pourquoi un verrou est inhabituel

- Le gestionnaire de paquets lui-même est figé — parce qu'un pnpm différent résout le fichier de verrouillage…
- TypeScript est maintenu en 6.x délibérément — TypeScript 7 — le compilateur natif — n'expose pas encore l'API…
- C'est la forme que devrait avoir un verrou — Une contrainte de version sans commentaire est une contrainte que personne…

Ce que cette plateforme fige, et pourquoi un verrou est inhabituel — Exemple

```
# pandas is pinned below 3.0 because the copy-on-write default changes the
# behaviour of the recode in src/clean.py:88. Revisit when that is rewritten.
pandas = ">=2.1,<3.0"
```

Les conteneurs, et quand ils en valent la peine

- Ils en valent la peine quand — l'analyse a des dépendances hors Python ou hors R — GDAL, une distribution TeX, un…
- Ils n'en valent pas la peine quand — un fichier de verrouillage reproduit déjà le résultat et que le public est deux…

Les conteneurs, et quand ils en valent la peine — Exemple

```
FROM python:3.12-slim
COPY pyproject.toml uv.lock ./
RUN pip install uv && uv sync --frozen
COPY . .
CMD ["uv", "run", "python", "run.py"]
```

Les conteneurs, et quand ils en valent la peine

- Commencez par le fichier de verrouillage — Passez au conteneur quand vous pouvez nommer la dépendance qu'il fige et que…

Le test qui le prouve

- Reproduisez votre propre résultat sur une machine qui n'a jamais vu le projet — Un clone neuf dans un répertoire...

Le test qui le prouve — Shell

```
git clone <repo> /tmp/check && cd /tmp/check
uv sync
uv run python run.py
diff -r outputs/ ~/project/outputs/
```

Le test qui le prouve — En R

```
# renv::restore() then source("run.R"), and compare.
```

Le test qui le prouve

- Si vous ne pouvez pas le faire, « c'est reproductible » n'est pas testé — L'ordinateur d'un collègue vaut mieux et un...

Rapportez-le en entier — Exemple

```
Computational environment

  Python 3.12, dependencies pinned in uv.lock (committed). Reproduce with:

      uv sync && uv run python run.py

  pandas is held below 3.0 because the copy-on-write default changes the
  recode in src/clean.py; this is revisited when that function is rewritten.

  The pipeline runs on every push in CI on a clean machine, so the claim
  that a fresh checkout reproduces these outputs is tested rather than
  asserted.

  Analysis run on 2026-03-14 with commit a3f9c21.
```

Rapportez-le en entier

- La dernière ligne est celle qui rend le reste utilisable un an plus tard — Un résultat sans le commit qui l'a produit...

La suite

- Un environnement figé produit encore une réponse différente à la seconde exécution si le code lit l'horloge, tire un nombre aléatoire, ou se fie à l'ordre dans lequel les fichiers reviennent.

Lire la leçon complète, avec le code exécutable

https://data-analysis.cassion.dev/fr/cours/reproducible-workflows/repro-03-pin-the-environment