Principles for Reproducible Code

Anyone can run your code, obtain same results, understand why, and verify correctness.

1. DRY (Don't Repeat Yourself)

Single implementation, parameterize differences.

- Write functions for repeated operations
- Use configuration files
- Create reusable modules
- Avoid copy-paste

2. Config = Data

Save configurations with models, load automatically, no hardcoding.

- Store parameters in JSON, YAML, or code configs
- Save config alongside outputs
- Never hardcode magic numbers or paths

3. Stable Paths

`output/eq/` not `output/eq_20250927/`

`file.py` not `file_v2.py` or `file_old.py`

Overwrite on rerun; use git for history.

4. Named Arguments

`f(value=x)` not `f(x)`

Self-documenting, prevents argument order mistakes.

5. Version Control Everything

Track: source code, configs, documentation, tests, build scripts.

Do not track: large data, generated outputs, secrets, temp files.

6. Explicit Dependencies

Declare all dependencies with versions.

- Python: `pyproject.toml` with uv
- R: `renv.lock` with renv

7. Deterministic Execution

Set random seeds explicitly. Avoid time-dependent behavior.

```
np.random.seed(config["random_seed"])
```

8. Specification as Single Source of Truth

Specifications in external docs. Code implements naturally.

- Descriptive names reflecting spec
- Structure makes functionality obvious
- No comments for intent (in spec) or functionality (obvious)

9. Testable Code

- Modular, testable functions
- Test properties (monotonicity, bounds), not exact values
- Unit and integration tests
- Test on dummy data before expensive compute

10. Programmatic Generation

Generate all outputs programmatically. Never manually edit generated files.

Project Structure

```
project/
├── input/      # Raw data (never modify)
├── cleaned/    # 3NF data
├── output/     # Results (stable paths)
├── src/        # Source code
├── scripts/    # Execution scripts
├── test/       # Tests
├── config/     # Configuration
└── docs/       # Documentation
```

Workflow

1. Plan
2. Implement (follow principles)
3. Test (small data, verify properties)
4. Configure
5. Execute
6. Validate
7. Document
8. Commit

Anti-Patterns to Avoid

- Hardcoded values
- Timestamped outputs
- Versioned files
- Manual edits to generated files
- Missing dependencies
- Non-deterministic code
- Relying on comments for intent

Checklist

- [ ] Parameters in config files
- [ ] Stable paths
- [ ] Explicit random seeds
- [ ] Dependencies declared
- [ ] Code tested
- [ ] Spec exists and is current
- [ ] Named arguments
- [ ] Version control