Streamline DCF Project - Coding Standards

This document contains the coding principles that shall be followed for the DCF project. These standards should not be altered in order to ensure efficiency, readability and simplicity of the code.

1. General Principles

- Prioritize readability and simplicity. Code should be self-explanatory and well-structured.
- Follow the "Convention over Configuration" mindset: rely on established defaults (PEP 8, pytest, etc.) rather than custom styles.
- Write small, single-purpose functions and classes. Each unit should do one thing and do it well.

2. Language & Style

- **PEP 8 compliance**: adhere to line length of 88 characters (Black default) or 100 characters if preferred.
- **Indentation**: use 4 spaces per indent level. No tabs.
- **Type annotations**: required for public functions, methods, and data models (PEP 484).
- **Imports**:
    - Group and order imports as: standard library, third-party, local packages.
    - Use absolute imports unless there is a strong reason for relative.

3. Naming Conventions

- **Modules & packages**: lowercase with underscores (e.g., av_ingestor.py).
- **Classes**: CapWords (e.g., FinancialStatement).
- **Functions & methods**: lowercase_with_underscores (e.g., fetch_financials).
- **Constants**: ALL_CAPS with underscores (e.g., API_TIMEOUT_SECONDS).
- **Private members**: prefix with a single underscore (e.g., _parse_raw_json).

4. Project Structure

```
├── dcf_empower_backend/
│   ├── __init__.py
│   ├── ingestors/
│   │   ├── alpha_vantage.py
│   │   └── edgar.py
│   ├── models/
│   │   └── financial_statement.py
│   ├── transformers/
│   │   └── normalize.py
│   ├── utils/
│   │   └── helpers.py
│   └── cli.py
├── tests/
│   ├── unit/
```

```
│   └── fixtures/
├── .pre-commit-config.yaml
├── CODE_STYLE.md
├── README.md
└── pyproject.toml
```

5. Docstrings & Documentation

- Use Google style docstrings for all public modules, classes, and functions.
- Include a brief (Args), (Returns), and (Raises) section as needed.
- Maintain a docs/ folder for higher-level architecture diagrams or decision logs.

6. Testing

- Use pytest for all tests.
- Test files mirror production modules, prefixed with test_ (e.g., test_alpha_vantage.py).
- Aim for > 80% coverage on core modules. Critical logic must be covered.
- Use fixtures to share test data; avoid hard-coded values.

7. Version Control & Commit Messages

- Follow **Conventional Commits**: e.g., feat: add alpha vantage ingestor, fix: correct date parsing.
- Create feature branches prefixed with feat/, fix/, docs/, test/ as appropriate.
- Pull requests must include a concise description of changes and link to any relevant issues.

8. Pre-commit Hooks

- **Black**: auto-formatting.
- **isort**: import sorting.
- **flake8**: linting for style, complexity, unused imports.
- **mypy**: static type checking (configured for strictness where feasible).
- Install hooks via pre-commit install.

9. Continuous Integration

- On each PR, run:
    - flake8 linting
    - mypy --strict
    - pytest --maxfail=1 --disable-warnings -q
- Fail the build on any error.

10. Dependency Management

- Define dependencies in pyproject.toml (if using Poetry) or requirements.txt.
- Pin versions for reproducibility; use semantic versioning ranges cautiously.

- Update dependencies via automated tooling (e.g., Dependabot).

11. Issue Tracking & Branching

- Use GitHub issues to track tasks, bugs, and enhancements.
- Label issues by category (enhancement, bug, documentation).
- Merge to main only via reviewed PRs; consider a develop branch if needed.