Document

File Structure

```
.
├── FunctionalJStrategies
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-310.pyc
│   │   └── discrete_j_1_strategy.cpython-310.pyc
│   └── discrete_j_1_strategy.py
├── Implementing_Gap_Filling_Algorithms_in_Chaotic_Time_Ser
ies_Analysis.pdf
├── Interfaces
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-310.pyc
│   │   ├── functional_J_strategy.cpython-310.pyc
│   │   ├── min_distance_strategy.cpython-310.pyc
│   │   └── vector_field_f_stategy.cpython-310.pyc
│   ├── functional_J_strategy.py
│   ├── min_distance_strategy.py
│   └── vector_field_f_stategy.py
├── MinDistanceStrategies
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-310.pyc
│   │   └── min_distance_bruteforce_store_all.cpython-310.p
yc
│   ├── min_distance_bruteforce_store_all.py
│   └── test
│       └── test_mdbs.py
├── README.md
├── VectorFieldFStrategies
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-310.pyc
```

```
│   │   └── vector_field_f_discrete_mid_point_strategy.cpyt
hon-310.pyc
│   └── vector_field_f_discrete_mid_point_strategy.py
├── __init__.py
├── data_generator
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-310.pyc
│   │   └── data_generator.cpython-310.pyc
│   └── data_generator.py
├── filling_gap_implement.ipynb
├── filling_gap_implement_demo.html
├── filling_gap_implement_demo_files
│   ├── MathJax.js
│   └── require.min.js
├── filling_gaps.pdf
├── gap_filling_pipline
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-310.pyc
│   │   └── gap_filler.cpython-310.pyc
│   ├── gap_filler.py
│   └── test
│       ├── __pycache__
│       │   └── test_gap_filler.cpython-310.pyc
│       ├── interactive_test.ipynb
│       └── test_gap_filler.py
├── main.py
└── tools
    ├── __init__.py
    ├── __pycache__
    │   ├── __init__.cpython-310.pyc
    │   ├── lorenz.cpython-310.pyc
    │   └── utils.cpython-310.pyc
    ├── lorenz.py
    ├── test
    │   ├── __pycache__
    │   │   └── test_tree_to_layer.cpython-310.pyc
```

```
    │   └── test_tree_to_layer.py
    └── utils.py
```

- The `filling_gap_implement.ipynb` file serves as a comprehensive demonstration, detailing every function explicitly and implementing each part step-by-step. This includes information on each step such as the type, length, shape, and content of the variables used, complemented by interactive graphs created with Plotly for visualization. The corresponding `filling_gap_implement_demo.html`, along with the `filling_gap_implement_demo_files` folder, contains a static HTML file that preserves all outputs and interactive graphs, maintaining interactivity from the original Jupyter notebook.
- The `data_generator` folder utilizes the Runge-Kutta method with the Lorenz system (parameters `s`=10, `r`=28, `b`=2.67) to generate test or demonstration data.
- The `Interfaces` folder contains all the defined interfaces, aligning with a strategy pattern approach (the exact term momentarily escapes me). It includes `FunctionalJStrategy`, `MinDistanceStrategy`, and `VectorFieldFStrategy`, allowing for easy incorporation of any specific class that implements these interfaces into the overall pipeline. This design strikes a balance between flexibility, decoupling, readability, and implementability—carefully considering the trade-offs between overdesigning and adaptability to ensure the software remains stable yet improvable. Future enhancements might involve separating the gap-connection, branching, and minimizing components into distinct strategies, but the current configuration is well-suited for implementation and functional requirements.
- The `J` and `F` functions within the code correspond to the same functions discussed in the referenced academic paper.
- The `FunctionalJStrategies`, `MinDistanceStrategies`, and `VectorFieldFStrategies` folders contain the actual implementations of the aforementioned interfaces. Any class inheriting from an interface can be seamlessly integrated into the pipeline.
- The `tools` folder includes implementations of the Lorenz function, an `iterate_solver` for numerical integration of the Lorenz equations, and other utility functions.
- The `gap_filling_pipeline` is the core component where all elements of the project converge.

- Finally, `main.py` acts as the entry point of the program. Users can input their time series data and run the program directly from this file.