SSIS to PySpark Conversion

Mentor- Gaurav Kumar verma

**1. Requirement**

The goal is to convert SSIS .dtsx packages into equivalent, runnable PySpark scripts that replicate the original ETL logic. The solution must extract source/target configurations, transformation logic, and execution flow from SSIS to support accurate migration.

**2. Approach**

**Phase 1: .dtsx Parsing & Metadata Extraction**

Parse the ODS file to extract :

* **Source Configuration**: Extracted using **parse\_ssis\_connection\_info()**
* **Data Flow & Target Logic**: Extracted using  **process\_dtsx\_to\_dataframe()**
* **Metadata & Parameters**: Extracted using **extract\_metadata()** and **extract\_params().**
* **Execution Control**: Extracted using **extract\_execution\_flow() .**

Output:

Meta data extracted to **Outputs/<filename>.xlsx**(with separate sheets for )

**Phase 2: Visualization Generation**

This phase generates flowcharts that visualize the data flow logic:

* The summary from data flow is converted into JSON.
* A generative AI model (gemini-2.0-flash) produces Graphviz Python code.
* The code is executed to generate a .png diagram using **visualization\_script()** and **visual\_representation()**

Output:

All the generated images(.png) in the **Visualization/<filename>.xlsx**

**Phase 3: PySpark Code Generation (In Progress)**

In this phase based on all the extraction we generate all the

* All five extracted DataFrames containing the metadata are compiled into a prompt for the AI model (gemini-2.5-pro).
* The model generates runnable PySpark code.
* Status: Code generation initiated, but validation is pending due to unavailable SQL Server environment.

Output:

The generated code will be pasted in the next cell directly for execution

**3. Our Findings**

GitHub Source ([File link](https://github.com/escobarana/SSIS_DWH)):

The project revealed a common three-layer ETL architecture used in the SSIS solution:

* **ODS (Operational Data Store):** Responsible for extracting and loading raw data from various source systems.
* **STA (Staging Area):** Performs transformations and intermediate processing on ODS data.
* **DWH (Data Warehouse):** Loads the final, transformed data into fact and dimension tables for analytics.
* Scheduler.dtsx: package was found that acts as the **master orchestrator**, executing the ODS, STA, and DWH packages in the correct sequence.

Entire Project File Structure:

SSIS\_Project/

├── SSIS/

│ └── SSIS\_DWH/

│ └── SSIS\_DWH/

│ ├── \*.dtsx #Contains all .dtsx files from ODS, STA, DWH, and Scheduler

│

├── Code/

│ ├── Outputs/ # Contains multi-sheet Excel files for each .dtsx file

│ │ ├── <filename>.xlsx # Sheets: Source, Data Flow, Meta-Data, Params

│ ├── Visualization/ # Contains Graphviz-generated PNG flowcharts

│ │ ├── <filename>\_flowchart.png

│ │

│ ├── requirement.txt # Dependency listing

│ ├── testing.ipynb # Initial experimentation and prototyping

│ └── testing1.ipynb # Final implementation notebook including full logic

**4. Output**

Output File and Visualization Mapping:

|  |  |
| --- | --- |
| **File Path/Name** | **Component** |
| **Code/Outputs/<dtsx\_filename>.xlsx** | Excel Output |
| **Code/Visualization/<dtsx\_filename>\_flowchart.png** | Visual Flowchart |

Excel Sheet Descriptions:

|  |  |
| --- | --- |
| **Sheet Name** | **Description** |
| **Source** | Contains extracted connection and configuration details from SSIS sources. |
| **Data-Flow** | Lists the SSIS components, their types, and the data movement pipeline. |
| **Meta-Data** | Describes column-level metadata including name, data type, and length. |
| **Params** | Includes parameters and variable mappings used in SSIS transformations. |
| **Execution-Control** | Captures control flow between tasks using precedence constraints. |

**5. Progress**

|  |  |
| --- | --- |
| **Phase** | **Status** |
| **Phase 1** | ✅ Done – Successfully parsed .dtsx files and extracted all metadata. |
| **Phase 2** | ✅ Done – Generated flowchart visualizations using AI and Graphviz. |
| **Phase 3** | ⚠️ Partial – PySpark code generation completed, but validation is pending. |

**Next Step:**

* Validate PySpark Code with a test SQL Server environment.
* Expand to STA & DWH layers.
* Replicate Scheduler.dtsx orchestration in tools like Databricks Workflow.
* Improve AI prompts for better PySpark output.
* Train/ Fine Tune the AI model for better and more consistent response

## **6. Dependencies**

### Libraries Used

* **Core Libraries:** json, os, re, xml.etree.ElementTree
* **Data Handling & Excel Output:** pandas, xlsxwriter
* **Visualization:** graphviz
* **Generative AI Integration:** google.generativeai
* **Configuration & Environment:** python-dotenv.

.env

* Configure a .env file in the working directory with a Gemini Api key

Graphviz

* Need to Install Graphviz.zip file and the provide the path of it in this function visual\_representation()

function in this part   
 os.environ["PATH"] += os.pathsep + <your\_path\_to\_Graphviz>

**7. Limitations**

* **Genrative A**I can provide inconsistent output and might hallucinate.
* No live validation of PySpark output
* Requires **Graphviz** installation for visualization.

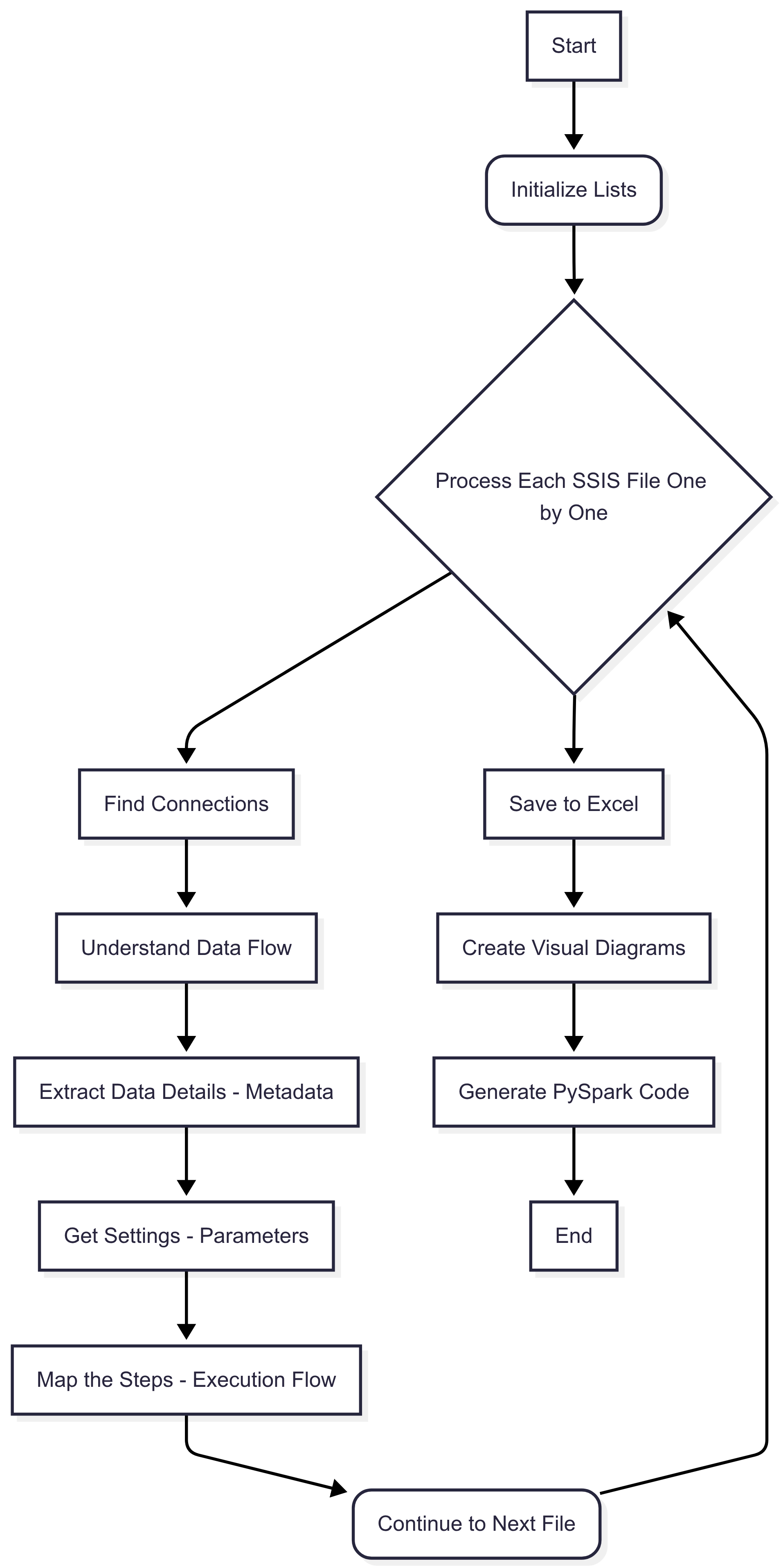
### **8 . Execution**

* Entry Point

The script begins execution from the main loop that processes each SSIS .dtsx file:

Python

for file in file\_name:  
 # ... (function calls inside the loop)

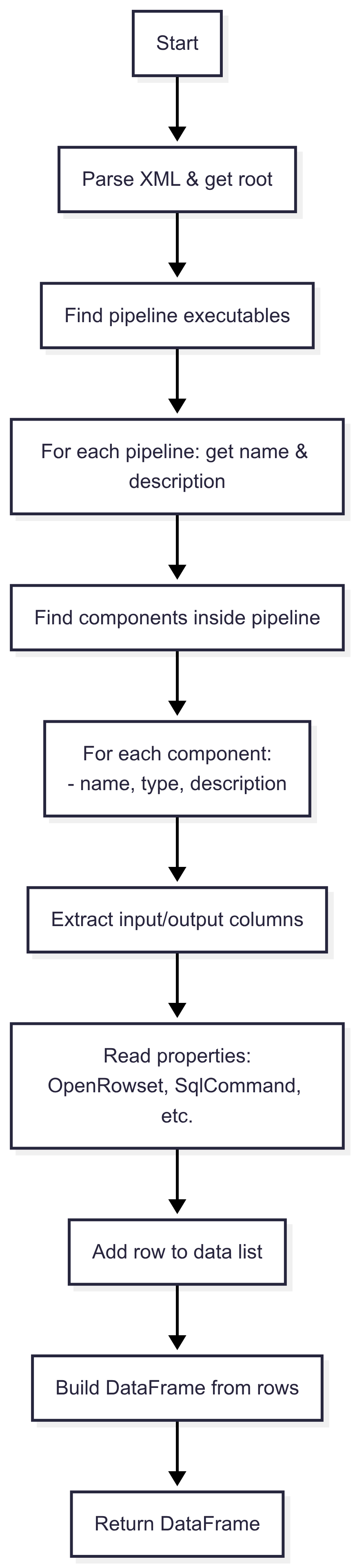


* Execution Flow

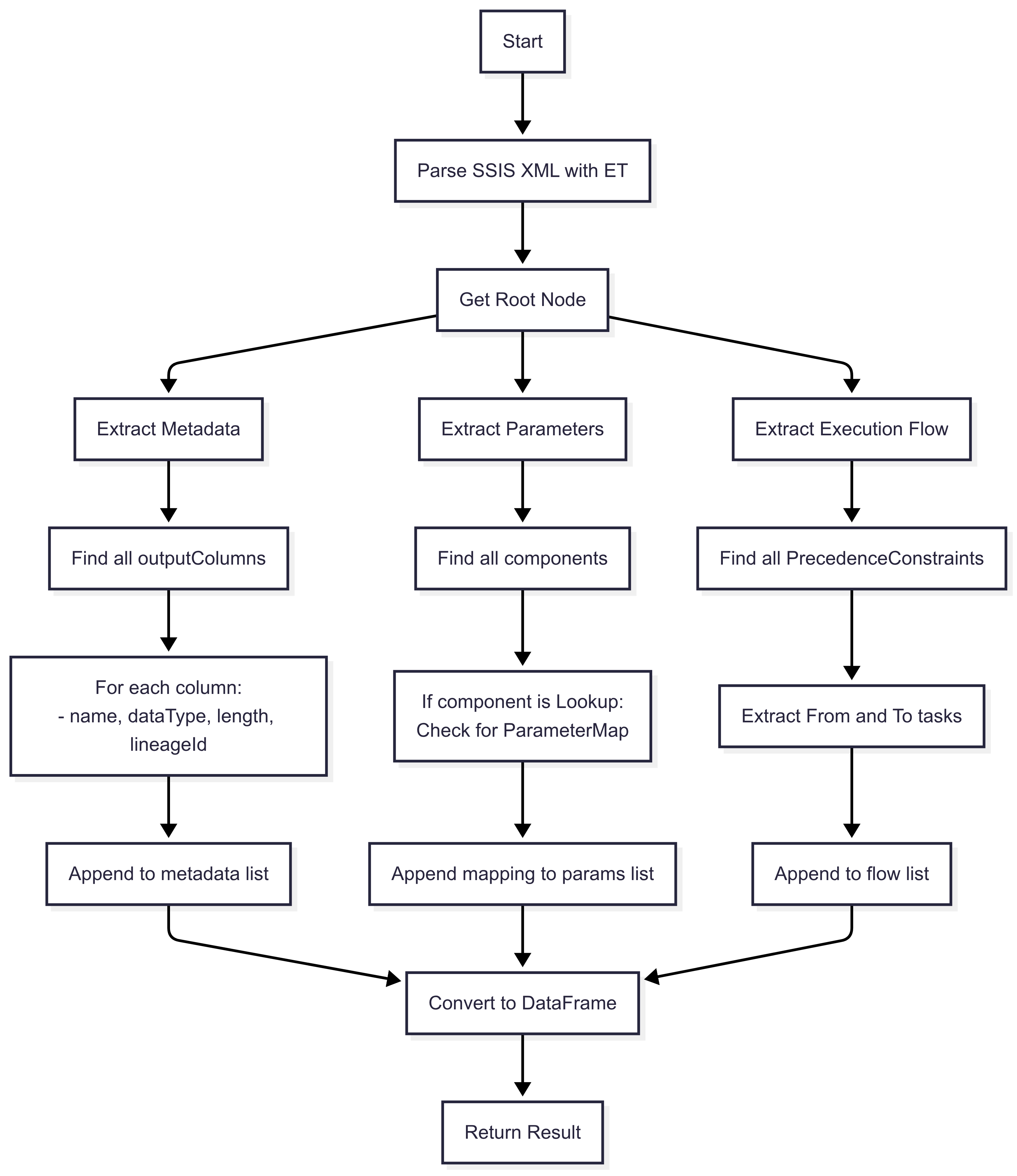
1. **Initialize DataFrames:** Empty lists (df\_source, df\_summery, df\_metadata, df\_params, df\_execution\_flow) are initialized to store data from processed files.
2. **Process SSIS Files:** For each .dtsx file in file\_name:
   1. parse\_ssis\_connection\_info(): Extracts source connection information.



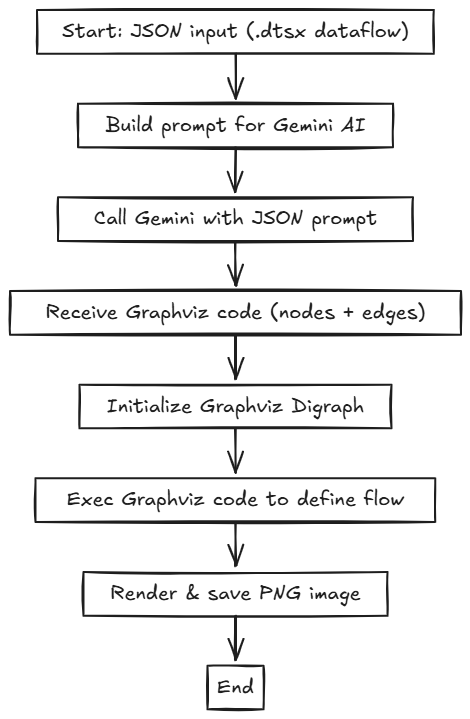
* 1. process\_dtsx\_to\_dataframe(): Processes the DTSX file to extract data flow logic.



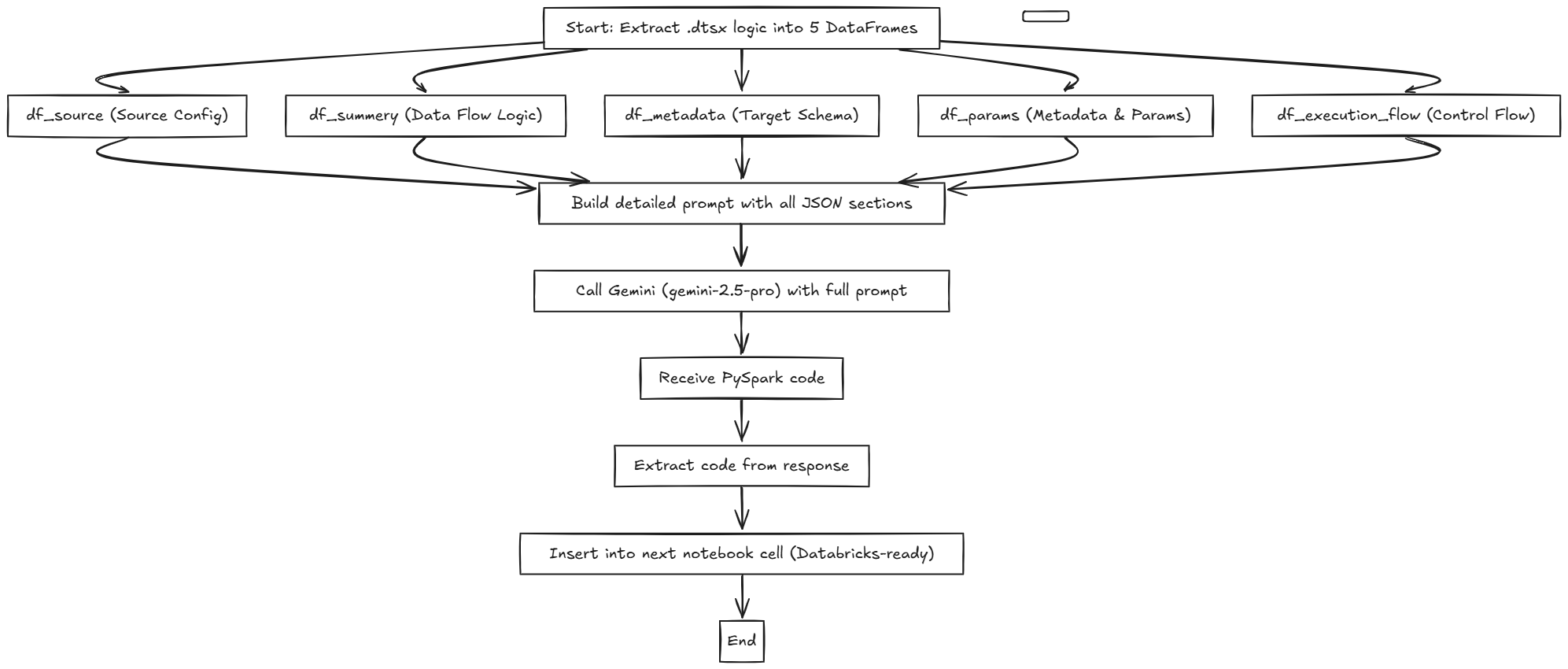
* 1. extract\_metadata(): Extracts metadata such as column names, data types, and lengths.
  2. extract\_params(): Extracts parameter mappings.
  3. extract\_execution\_flow(): Extracts the control flow and precedence constraints.



1. **Write to Excel Sheets:** The extracted data for each file is written into separate sheets within an Excel file (e.g., 'Source', 'Data-Flow', 'Meta-Data', 'Params', 'Execution-Controle') using write\_to\_sheet().
2. **Generate Visualizations:** For each data flow summary:
   1. The data flow summary DataFrame is converted to JSON.
   2. visualization\_script() is called to generate Graphviz Python code based on the JSON data, leveraging the gemini-2.0-flash model.
   3. extract\_python\_code() extracts the pure Python code from the model's response.
   4. visual\_representation() executes the generated Graphviz code to create and save a visual representation (PNG image) of the data flow.



1. **Generate PySpark Code (Future Step / User Interaction):** A prompt is constructed using the first processed file's extracted data to request PySpark code generation from the gemini-2.5-pro model. The generated PySpark code is then extracted and inserted as a new code cell in the notebook.



**9 . Conclusion**

This project successfully establishes a foundational pipeline for automating the discovery and migration of SSIS .dtsx packages into PySpark workflows. Through structured metadata extraction, AI-assisted visualization, and generative code transformation, we have created a modular and extensible system that simplifies SSIS-to-Spark migration. While the current implementation focuses on ODS packages, the framework is well-positioned to scale across STA, DWH, and scheduler layers. Future efforts will focus on validation, orchestration, and expanding coverage to support more complex transformation logic and enterprise-grade data flows.

Prepared By

Satwik Biswas, Arinjoy Pramanik