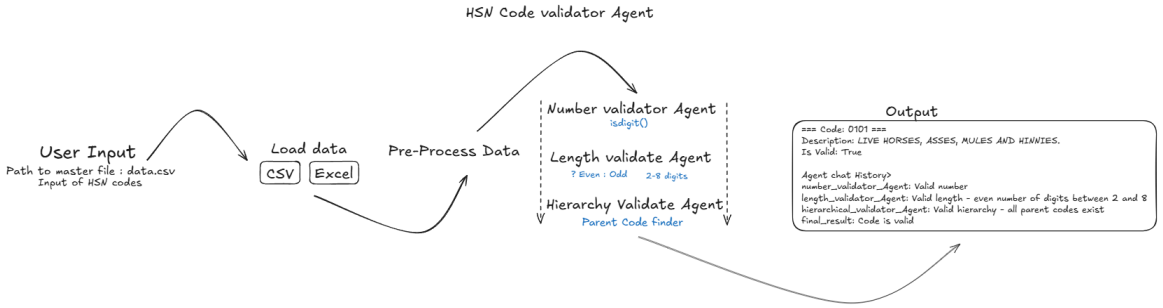
HSN-Code-validation-agent

Agent Design:



Key Components:

- Agent class with core validation methods
- Data storage using pandas DataFrame
- Caching system using lru_cache
- Conversation history tracking
- Memory usage monitoring

Input Handling:

- Single code: validate_agent("010110")
- Multiple codes: validate_agent(["010110", "020110"])
- Automatically detects input type (string or list)

Output Format:

- Returns a tuple: (is_valid, conversation_history, (code, description))
- is_valid: Boolean indicating validation status
- conversation_history: List of validation steps and messages
- (code, description): Tuple with code and its description

Data Handling:

- **Data Loading**: python

  Apply to agent.py

```
agent.load_data("HSN_Master_Data.xlsx")
```

Preprocessing:

- Converts HSN codes to strings
- Creates lookup dictionary for faster access
- Precomputes parent codes for hierarchy validation
- Uses tqdm for progress tracking during preprocessing

Validation Logic:

- **Format Validation**

```
def number_validate(self, code: str) → Tuple[bool, str]:
    # Checks if code contains only digits
    if code.isdigit():
        return True, "Valid number"
    return False, "Invalid number"
```

- **Length Validation**

```
def length_validate(self, code: str) → Tuple[bool, str]:
    # Checks if length is even and between 2-8 digits
    length = len(code)
    if length < 2 or length > 8 or length % 2 != 0:
        return False, "Invalid length"
    return True, "Valid length"
```

- **Hierarchical Validation**

```
def hierarchical_validate(self, code: str, hsn_dict: Dict) → Tuple[bool, str]:
    # Checks if code and its parent codes exist
    if code not in hsn_dict:
        return False, "Code not found"
```

```
    # Checks parent codes (e.g., 01011010 → 010110, 0101, 01)
    for i in range(2, len(code), 2):
        parent = code[:i]
        if parent not in hsn_dict:
            return False, f"Parent code {parent} not found"
    return True, "Valid hierarchy"
```

Agent Response:

- **Valid Code Response**

```
# Example for code "010110"
 {
    "is_valid": True,
    "conversation": [
        "number_validator: Valid number",
        "length_validator: Valid length",
        "hierarchical_validator: Valid hierarchy",
        "final_result: Code is valid"
    ],
    "code_info": ("010110", "Pure-bred breeding animals")
 }
```

- **Invalid Code Response**:

```
 # Example for code "12345"
 {
    "is_valid": False,
    "conversation": [
        "number_validator: Valid number",
        "length_validator: Invalid length - odd number of digits",
        "hierarchical_validator: Code not found",
        "final_result: Code is invalid"
    ],
    "code_info": ("12345", "Description not found")
 }
```

Performance Optimizations:

- Caching of validation results
- Precomputed parent codes
- Efficient data structures (dictionaries for lookups)
- Memory usage monitoring
- Progress bars for long operations

⇒ Furthur improvements:
This is an implementation of the Agent using simple Python Code, I have used Langchain and Crew AI for building AI agents but specifically in this project there is no need of having an LLM so a simple implementation is enough I guess. Rather than over engineering it.

⇒ Furthure improvements could be like, An LLM marked up to give an analysis and explanation for the result of each input,
⇒ Implementing RAG to fetch data about related HSN code values using Cosine similarity (Vector Search) using euclidean distance. In case if we need to provide context to the llm about the question we are asking
⇒ Using precise prompt templates and persistance withing subagents in order to provide more clarity.
⇒ Having the data stored to a Database and using a Webhook to add up the changes instead of deploying it from scratch again.
⇒ Note : The current implementation of the workflow is because the scope of the problem statements lies below using LLM, Chains, tools and Much more Agentic properties.

Thank you.