Functional Specification for Refactored Python Program for Data Monitoring and Drift Detection Using WhyLogs

1. Overview

This functional specification describes a Python program designed for monitoring data quality, detecting data drift, and validating feature constraints using WhyLogs. The program processes multiple batches of the Iris dataset, logs their profiles, performs drift analysis, and validates the data against predefined constraints. The code is designed for scalability, efficiency, and ease of maintenance.

2. Environment Setup

- WhyLabs Environment Variables:

* `WHYLABS\_DEFAULT\_ORG\_ID`: Organization ID for WhyLabs.
* `WHYLABS\_API\_KEY`: API key to authenticate with WhyLabs.
* `WHYLABS\_DEFAULT\_DATASET\_ID`: Dataset ID to specify the project within WhyLabs.

3. Data Loading

- Source Data:

* The program loads seven batches of the Iris dataset from predefined URLs into Pandas DataFrames.
* The URLs are stored in a list, and the data is dynamically loaded into a list of DataFrames (`batch\_data`).

- Feature Columns:

- `sepal length (cm)`

- `sepal width (cm)`

- `petal length (cm)`

- `petal width (cm)`

- `state`

- Operations:

* The program separates features and target labels for each batch using list comprehensions:
* `X\_batches`: List of DataFrames containing the feature columns.
* `y\_batches`: List of Series containing the target labels.

4. WhyLogs Profile Generation

- Profile Creation:

* The program logs the profiles of the feature DataFrames using WhyLogs, storing the resulting profile views in a list (`profile\_views`).
* Each profile view captures statistical summaries and distributions of the data.

5. Data Drift Detection

- Visualization Setup:

* A `NotebookProfileVisualizer` is instantiated to facilitate the comparison of data profiles.
* The program compares a reference profile (`profile\_views[5]`) with a target profile (`profile\_views[0]`) to generate a summary drift report and visualize the distribution of specific features (e.g., "petal length (cm)" and "petal width (cm)").

- Drift Score Calculation:

* Drift scores between the target and reference profiles are calculated using the `calculate\_drift\_scores()` function.
* The scores are printed in a structured, readable format using a custom `print\_formatted\_scores()` function that dynamically handles the unknown structure of the score dictionary.

6. Data Quality Validation

- Constraints Definition:

* The program defines data quality constraints for specific features in a dictionary (`features\_constraints`), specifying acceptable value ranges for each feature.

- Validation Function:

- The `validate\_features` function is designed to:

* Accept a profile view as input.
* Apply the predefined constraints to the profile using a loop, ensuring the feature values fall within the specified ranges.
* Optionally print a constraints report if `verbose=True`.

- Validation Execution:

* The program validates and visualizes constraints for multiple profiles using the `validate\_features` function.
* It checks if all constraints pass and prints the results.

7. Output and Visualization

* Drift Summary and Histograms:
* The program generates a summary drift report and visualizes histograms for specific features using `NotebookProfileVisualizer`.

- Constraints Reports:

* Constraints reports are generated for the profiles and visualized in the notebook environment.

- Validation Results:

* The program checks and prints whether all constraints are valid for each profile.

- Exporting Profiles:

* The profile for the last batch is converted to a Pandas DataFrame and printed for further analysis.

8. Error Handling

* Type Safety in Formatting Function:
* The `print\_formatted\_scores` function ensures that the `indent` parameter is always an integer, avoiding type errors when printing nested structures.

9. Dependencies

- Python Libraries:

* `whylogs`: For logging data profiles, calculating drift scores, and validating constraints.
* `pandas`: For data manipulation.
* `os`: For setting environment variables.

- External Services:

- WhyLabs API for logging, monitoring, and generating reports.

10. Limitations and Assumptions

- Assumptions:

* The Iris dataset batches are correctly formatted and accessible via the provided URLs.
* The WhyLabs API key, organization ID, and dataset ID are valid and correctly configured.
* The script is executed in an environment that supports WhyLogs and the necessary Python libraries.

- Limitations:

* The program assumes a consistent dataset structure across all batches.
* External dependencies such as internet access for fetching datasets and WhyLabs API access are required for the program to function correctly.

11. Execution Environment

* Notebook or Script Execution:
* The program is designed to be executed within a Python environment that supports notebooks (e.g., Jupyter) to take advantage of the visualizations provided by `NotebookProfileVisualizer`.

Conclusion

This refactored Python program provides a comprehensive and scalable solution for monitoring data quality, detecting data drift, and validating data against predefined constraints using WhyLogs. By leveraging arrays and loops, the program minimizes code duplication, making it easy to maintain and extend. The dynamic handling of data structures ensures robustness when dealing with unknown or complex data formats.