**Documentation - Decision Tree Converter**

The Decision Tree Converter is a program consisting of five code files that build a decision tree model from a dataset and export the resulting model algorithm in a Structured Text file. Below is a detailed description of each code file:

1. *conversion.py*

This code generates a decision tree and converts it into a structured text syntax. The following information is requested from the user:

* Dataset file name;
* Name of the .txt file to store the converted algorithm;
* Name for the output variable.

After gathering this information, the script:

* Checks if the .csv file exists, reporting an error otherwise;
* Calls the *generate\_decision\_tree* function from the *dt\_generator.py* to train and build the Decision Tree based on the provided dataset;
* Uses the *convert\_tree\_to\_text* function from *dt\_conversion.py* to adjust the decision tree to a Structured Text syntax suitable for application in the PLC.

1. *dt\_generator.py*

This script is responsible for generating a decision tree from a dataset file. Using the pandas and scikit-learn libraries, it processes the data and builds the decision tree model. The script has a predefined structure that users can modify freely, adjusting the tree parameters as needed. The process mainly includes:

* Building and training a decision tree model (adaptable for regression or classification);

1. *dt\_model.py*

This script inserts the trained model’s structure into a .txt file, simplifying visualization and interpretation of the model for the user. The generated file is used for converting the algorithm to the desired Structured Text syntax.

1. *dt\_conversion.py*

This script generates a .txt file by converting the output file produced by *dt\_generator.py*. It contains the main function *convert\_tree\_to\_text*, which performs the necessary conversion and additional operations with the support of functions organized in *dt\_functions.py*. The process includes:

* Identifying the tree type (regression or classification);
* Translating the algorithm syntax;
* Generating a .txt file containing the decision tree algorithm in the Structured Text format;
* Generating a .txt file containing a table with information about the input and output parameters.

1. *dt\_functions.py*

This function receives a vector representing the tree’s nesting levels and the file path where adjustments will be made. It inserts 'If' and 'Elsif' terms in the appropriate positions, ensuring that the logical structure is correctly organized.

* *def If\_Elsif\_Insertion(vector, filepath):*

This function receives a vector representing the tree’s nesting levels and the file path where adjustments will be made. It inserts 'If' and 'Elsif' terms in the appropriate positions in the decision tree file, ensuring that the logical structure is correctly organized.

* *def Decreased\_indentation(filepath):*

This function analyzes line indentations in the specified file to identify blocks where the indentation decreases, signaling points for inserting ‘end\_if’. The output is a list of line indices where indentation decreases relative to the previous line, indicating shifts in the conditional structure, such as the end of a nested condition.

* *def Insertion\_terms(filepath, index):*

This function identifies positions in the decision tree file where 'end\_if' terms should be added based on the analysis of line indentations. It takes as input the file path to be read and a list of line indices where indentation decreases, as identified by the *Decreased\_indentation* function. The output includes three lists: the first contains line indices where 'end\_if' should be inserted; the second stores indentation levels for lines with a subsequent line of lower level; and the last stores indentation levels for lines without a subsequent line of lower level.