Lab1: Pima Indians Diabetes Database

This lab uses the classic Pima Indians Diabetes dataset, which contains medical measurements collected from female patients of Pima Indian heritage.

Dataset description

The Pima Indians Diabetes dataset contains 768 records, each representing one female patient of Pima Indian heritage aged 21 years or older. For each patient, the following variables are recorded:

- Pregnancies

  Number of times the patient has been pregnant (integer count).
- Glucose

  2-hour plasma glucose concentration (mg/dL) from an oral glucose tolerance test.
- BloodPressure

  Diastolic blood pressure (mm Hg).
- SkinThickness

  Triceps skinfold thickness (mm), a proxy for body fat.
- Insulin

  2-hour serum insulin (μU/mL).
- BMI

  Body Mass Index, defined as weight in kg divided by the square of height in meters (kg/m^2).
- DiabetesPedigreeFunction

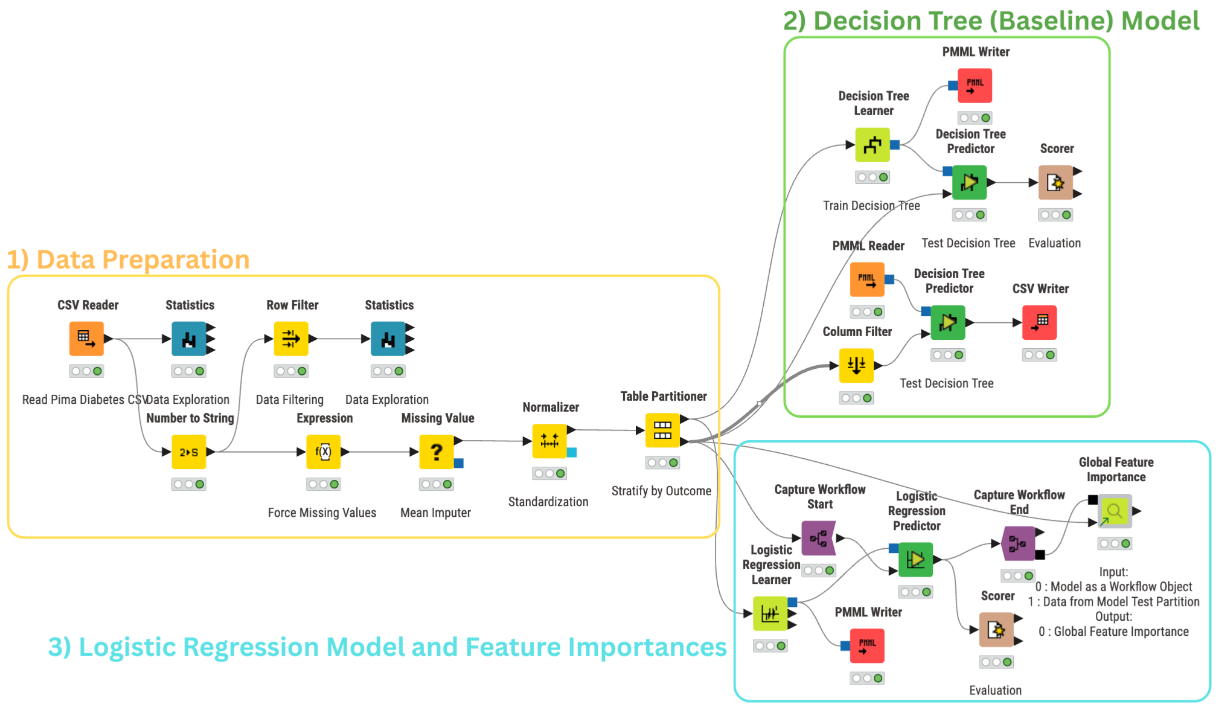
  A derived score that estimates the patient's hereditary risk of diabetes, based on family history.
- Age

  Age of the patient in years.
- Outcome

  Binary target variable:
  - 1 = patient is diagnosed with diabetes
  - 0 = patient is not diagnosed with diabetes

Your task is to build and analyse binary classification models that predict **Outcome** from these clinical features. You will first clean and prepare the data in KNIME, then train a baseline decision tree and a logistic regression model, evaluate their performance, and explore global feature importance to understand which variables contribute most to the diabetes prediction.

KNIME Instructions



1) Data Preparation block

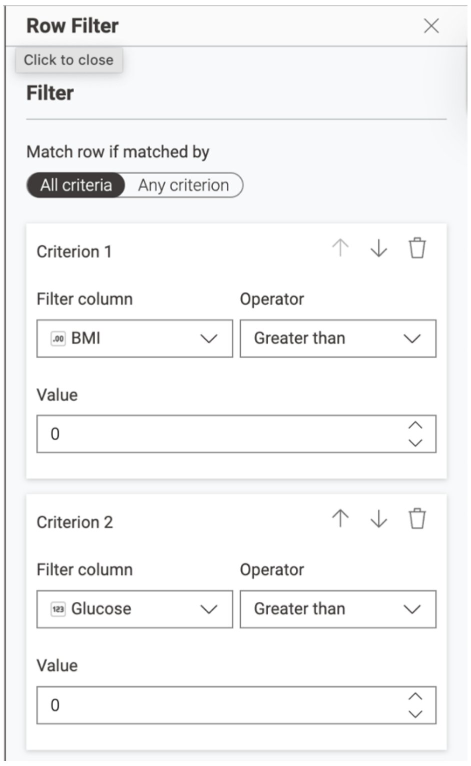
1.1 **CSV Reader** – Drag CSV Reader, connect nothing to it, double-click, and set the path to **diabetes.csv** so KNIME can read the Pima Diabetes data.

1.2 **Statistics (raw)** – Drag Statistics, connect it to the CSV Reader output; execute and open the view to quickly inspect distributions and spot weird values (0's, outliers, class balance).

1.3 **Number to String (Outcome)** – Drag Number to String, connect from CSV Reader, configure it so **Outcome** is converted from integer to string; this gives KNIME a proper *nominal* class column for classification nodes.

1.4 Row Filter (drop-0s demo)

Drag a **Row Filter** node, connect it after **CSV Reader**, and set criteria such as BMI > 0, Glucose > 0, BloodPressure > 0, SkinThickness > 0, and Insulin > 0 with "Match row if matched by: All criteria" so KNIME keeps only records with non-zero medical values, demonstrating a naïve "drop rows with 0s" cleaning approach.


Row Filter

Click to close

Filter

Match row if matched by: All criteria [✓] / Any criterion

Criterion 1

Filter column: BMI — Operator: Greater than

Value: 0

Criterion 2

Filter column: Glucose — Operator: Greater than

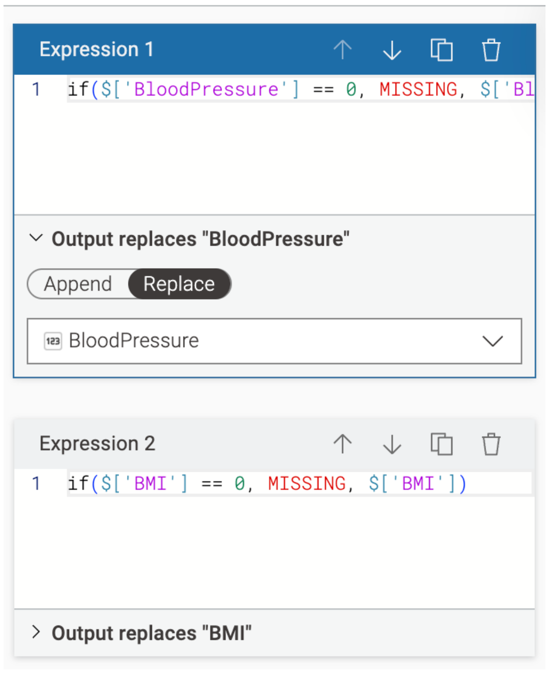
Value: 0

1.5 Statistics (show data loss from drop-0s)

Drag a **Statistics** node, connect it after this Row Filter, execute, and open the view to see that the row count falls from 768 to only ~400 rows, illustrating that this hard filtering discards many patients and motivating the other branch, where we convert 0s to missing values and impute them instead.

1.6 Expression (force 0 values to missing)

Drag an **Expression** node, connect it after **Number to String**, and for each clinical column (Glucose, BloodPressure, SkinThickness, Insulin, BMI) add an expression like **if($Glucose$ == 0, MISSING, $Glucose$) (set "Output replaces 'Glucose'")** so that any impossible 0 is turned into a missing value that can later be filled by the **Missing Value** (mean imputation) node instead of dropping whole rows.



Expression 1

```
if($['BloodPressure'] == 0, MISSING, $['Bl
```

Output replaces "BloodPressure"

Append [✓ Replace]

BloodPressure

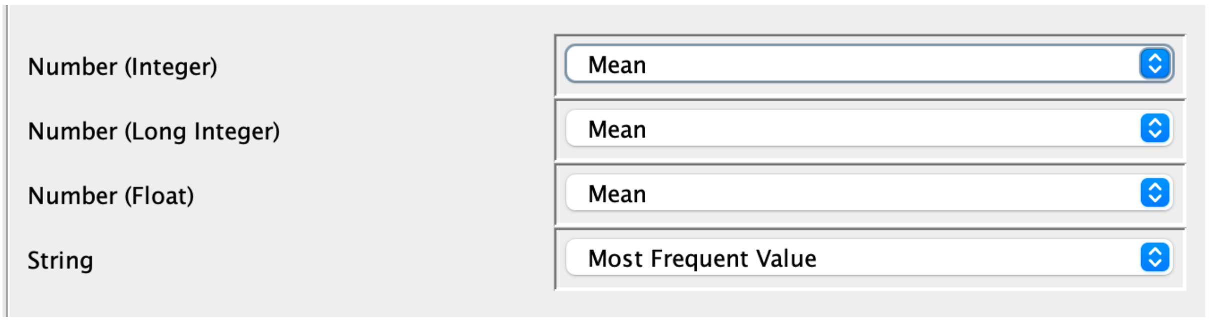
Expression 2

```
if($['BMI'] == 0, MISSING, $['BMI'])
```

Output replaces "BMI"

1.7 Missing Value (mean imputation for numeric columns)

Drag a **Missing Value** node, connect it after **Expression**, then double-click to open its configuration and on the **Default** tab set `Number (Integer)`, `Number (Long Integer)` and `Number (Float)` to **Mean** (leave `String` as "Most Frequent Value"); this fills all the new missing values in the medical numeric columns with their column means so the next nodes receive a complete table with no missing entries.



Number (Integer)	Mean
Number (Long Integer)	Mean
Number (Float)	Mean
String	Most Frequent Value

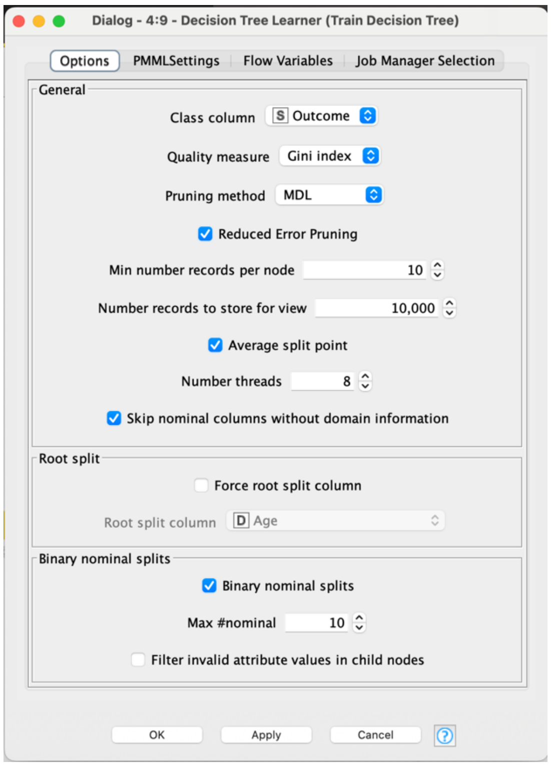
1.8 **Normalizer (Standardization)** – Drag Normalizer, connect from Missing Value, set method to *z-score/standardization* for all feature columns; this gives zero-mean, unit-variance inputs for Logistic Regression.

1.9 **Table Partitioner** – Drag Table Partitioner, connect from Normalizer, choose a train/test split (e.g. 70/30) and tick "Stratified" by **Outcome**; this creates a fair split with similar class balance in both partitions.

---

2) Decision Tree (Baseline Model) block

2.1 **Decision Tree Learner** – Drag Decision Tree Learner, connect its data input to the *first* output port of Table Partitioner (training data), configure **Outcome** as class column, and other settings (Gini, pruning, etc.), and execute to train the baseline tree.


Dialog - 4:9 - Decision Tree Learner (Train Decision Tree)

Options | PMMLSettings | Flow Variables | Job Manager Selection

General

Class column: S Outcome

Quality measure: Gini index

Pruning method: MDL

☑ Reduced Error Pruning

Min number records per node: 10

Number records to store for view: 10,000

☑ Average split point

Number threads: 8

☑ Skip nominal columns without domain information

Root split

☐ Force root split column

Root split column: D Age

Binary nominal splits

☑ Binary nominal splits

Max #nominal: 10

☐ Filter invalid attribute values in child nodes

OK | Apply | Cancel

2.2 **Decision Tree Predictor (direct KNIME evaluation)** – Drag Decision Tree Predictor, connect its model input to the Decision Tree Learner and its data input to the *second* output port of Table Partitioner (test data); execute to get predictions.

2.3 **Scorer** – Drag Scorer, connect from Decision Tree Predictor output and configure Outcome as true class; execute to view accuracy, confusion matrix, ROC etc. – this is the KNIME-side evaluation of the baseline.

2.4 **PMML Writer (export tree to Python)** – Drag PMML Writer, connect the model port from Decision Tree Learner, and configure it to write d-tree.pmml into the destination folder; this file will later be loaded and evaluated in Python.

2.5 Column Filter (simulate unlabeled deployment data)
Drag a **Column Filter** node, connect it from the **second output of Table Partitioner**, and *deselect* the **Outcome** column so only the predictor features remain; this simulates a real deployment situation where you want to score new patients but do not know their diagnosis yet.

2.6 PMML Reader + Decision Tree Predictor (deployment branch)
Use **PMML Reader** to load the saved **d-tree.pmml**, then drag a second **Decision Tree Predictor** and connect its *model* input to the PMML Reader and its *data* input to the Column Filter output; this shows how a trained tree model, stored as PMML, can be reloaded and applied to feature-only data in a separate workflow or at deployment time.

2.7 CSV Writer (prediction.csv for external use)
Drag a **CSV Writer**, connect it to the second Decision Tree Predictor, and configure it to write prediction.csv into the destination folder; this exports the predicted class and probabilities for each patient, which can be used as the final deployment output.

---

3) Logistic Regression Model & Global Feature Importances block

3.1 **Logistic Regression Learner** – Drag Logistic Regression Learner, connect its data input to the *first* output of Table Partitioner (standardized training data), set Outcome as the target and run it to train the LR model.

3.2 **Logistic Regression Predictor** – Drag Logistic Regression Predictor, connect the model input from the LR Learner and the data input from the *second* output of Table Partitioner (test data); execute to obtain probability scores and predicted class.

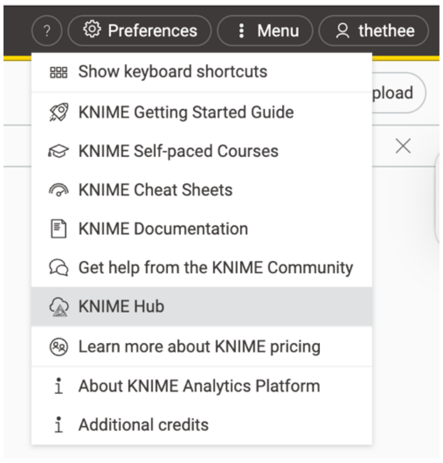
3.3 **Scorer (LR evaluation)** – Drag another Scorer, connect from LR Predictor, configure Outcome as the true class, and execute to compare LR performance with the Decision Tree baseline.

3.4 **PMML Writer (export LR to Python)** – Drag PMML Writer, connect from Logistic Regression Learner's model port, configure the file name log-reg.pmml in the destination folder; this allows the Python notebook to load and evaluate the KNIME LR model via PyPMML.

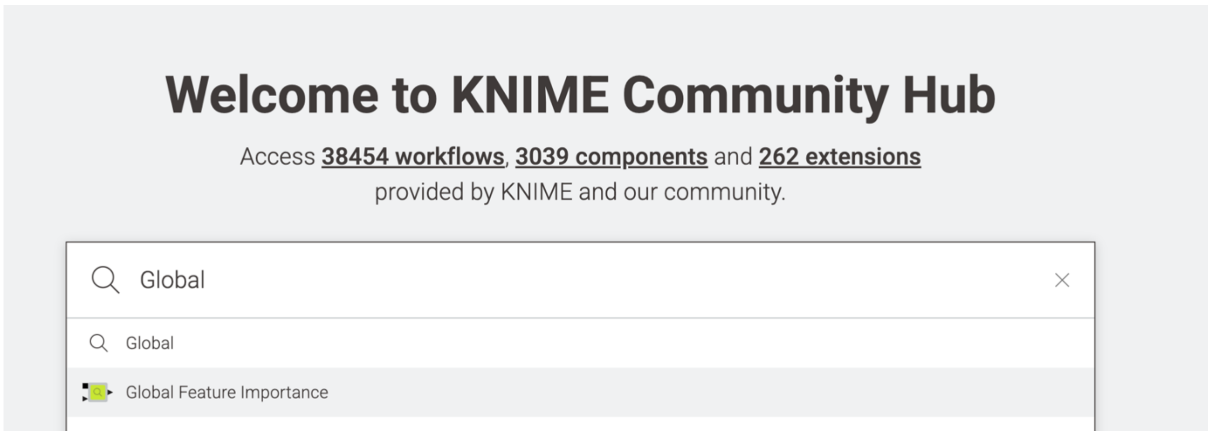
3.5 **Capture Workflow Start / End (wrap LR workflow as object)** – Drag Capture Workflow Start before the LR Predictor/Scorer branch and Capture Workflow End after them; this turns the whole LR scoring workflow into a **workflow object** that the Global Feature Importance component can call internally.

3.6 Global Feature Importance (download component from KNIME Hub and connect it)

At the top-right of KNIME, click '**(?) button'⟶ KNIME Hub,**



then in the Hub search box type "**Global Feature Importance**" and open the component page;



Welcome to KNIME Community Hub

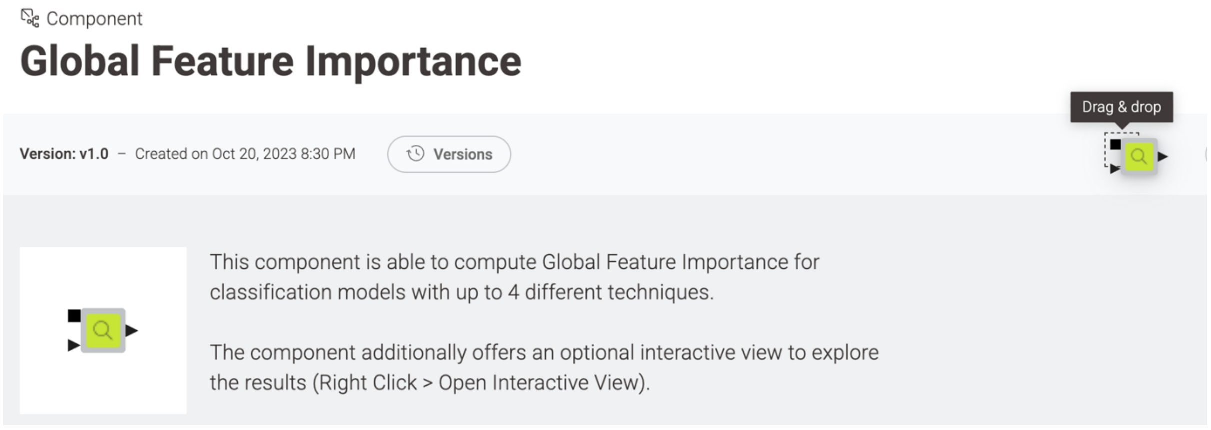
Access 38454 workflows, 3039 components and 262 extensions provided by KNIME and our community.

Global

Global

Global Feature Importance

on that page, grab the small green **Drag & drop** icon and drag it straight into your workflow editor,



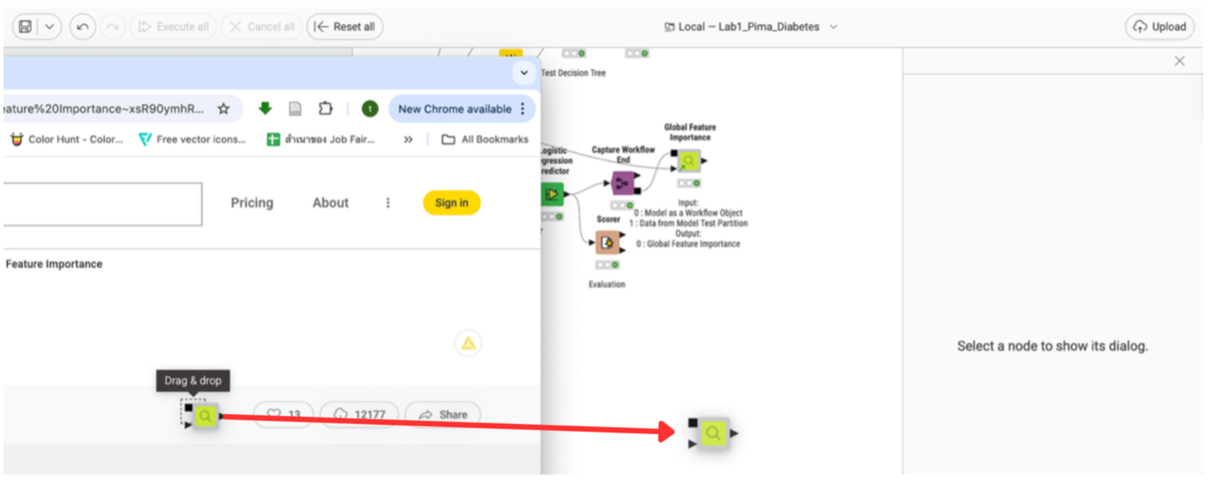
Component

Global Feature Importance

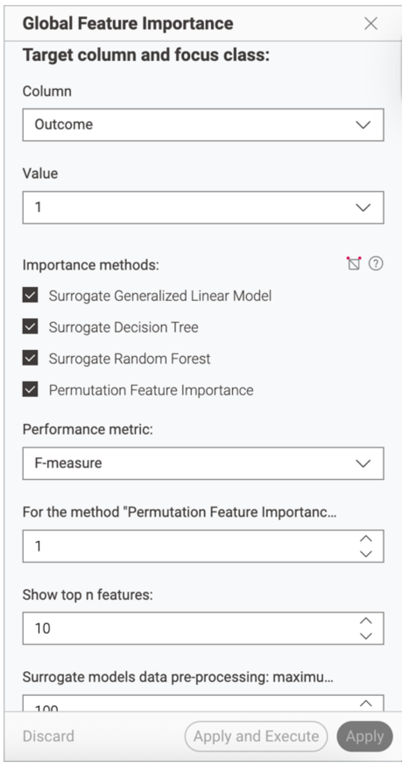
Version: v1.0 – Created on Oct 20, 2023 8:30 PM

This component is able to compute Global Feature Importance for classification models with up to 4 different techniques.

The component additionally offers an optional interactive view to explore the results (Right Click > Open Interactive View).


Execute all
Cancel all
Reset all
Local – Lab1_Pima_Diabetes
Upload
New Chrome available
Pricing
About
Sign in
Feature Importance
Drag & drop
Share
Test Decision Tree
Global Feature Importance
Capture Workflow End
Scorer
Evaluation
Input:
0 : Model as a Workflow Object
1 : Data from Model Test Partition
Output:
0 : Global Feature Importance
Select a node to show its dialog.

then connect its **model workflow** input port `0: Model as a Workflow Object` to **Capture Workflow End**, its **data** input port `1: Data from Model Test Partition` to the **second output of Table Partitioner** (test data), execute the node, and finally right-click ⟶ **Open Interactive View** to explore how each feature contributes to the Logistic Regression model.



Global Feature Importance

Target column and focus class:

Column: Outcome

Value: 1

Importance methods:

- [✓] Surrogate Generalized Linear Model
- [✓] Surrogate Decision Tree
- [✓] Surrogate Random Forest
- [✓] Permutation Feature Importance

Performance metric: F-measure

For the method "Permutation Feature Importanc…: 1

Show top n features: 10

Surrogate models data pre-processing: maximu…: 100