

Diabetes prediction with Gradient Boosted Decision Trees

[#2 Place in the ML challenge](#) jointly organized by the University of Milan-Bicocca & KNIME

Co-authors: [Daniel Montalbano](#) and [Giuseppe Sabino](#)

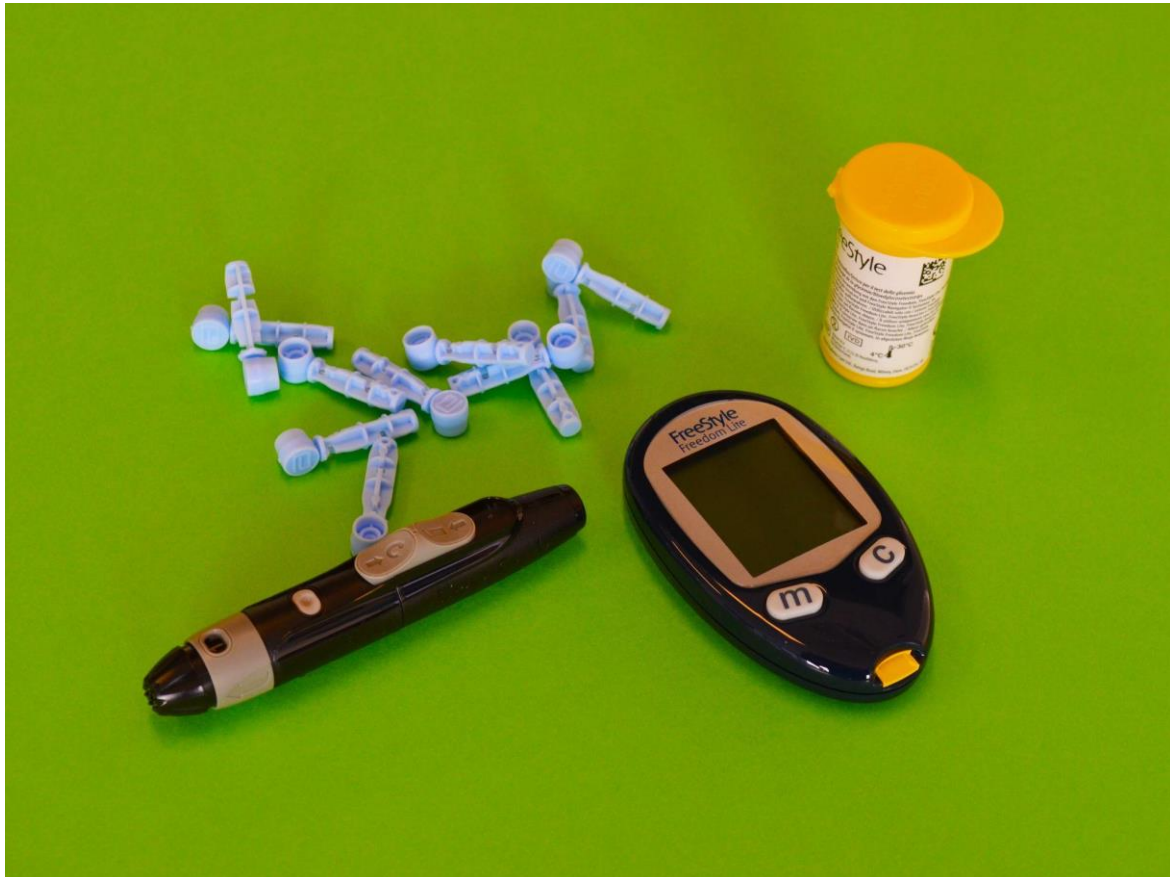


Photo by [Diabetesmagazijn.nl](#) on [Unsplash](#).

Diabetes is a chronic medical condition characterized by high-blood sugar levels. It occurs when the body cannot produce enough insulin, which is a hormone that regulates blood sugar levels, or cannot effectively use the insulin it produces. This can lead to a variety of complications over time, including damage to the heart, blood vessels, eyes, kidneys, and nerves.

Diabetes affects millions of people worldwide and is a major cause of disability and death. Accurate forecasting of diabetes can help healthcare providers to better manage the disease and reduce the risk of complications. Machine learning and predictive analytics have emerged as powerful tools for diabetes forecasting, allowing researchers to analyze large datasets and identify complex patterns that may be difficult to detect using traditional statistical methods.

In this data story, we will explain the phases of a machine learning pipeline using a Gradient Boosted Decision Tree (GBDT) model to predict the probability that a person has diabetes or not given a set of data.

The [KNIME workflows](#) described hereafter can be downloaded for free from the KNIME Community Hub.

Data Access and Preprocessing

We started off by accessing and preprocessing the dataset at hand in order to *improve its quality* for better modeling down the pipeline.

The Dataset

The dataset, available on [Kaggle](#), contains about 40.000 rows and the columns corresponds to lifestyle factors, general information and also clinical factors about the person:

- *Age*: 3-level age category 1 = 18-24, 9 = 60-64, 13 = 80 or older
- *Sex*: 0 = female, 1 = male
- *HighChol*: 0 = no high cholesterol, 1 = high cholesterol
- *CholCheck*: 0 = no cholesterol check in 5 years, 1 = yes cholesterol check in 5 years
- *BMI*: Body Mass Index
- *Smoker*: Have you smoked at least 100 cigarettes in your entire life? 0 = no, 1 = yes
- *HeartDiseaseorAttack*: Coronary heart disease (CHD) or myocardial infarction (MI) 0 = no, 1 = yes
- *PhysActivity*: Physical activity in past 30 days - not including job 0 = no, 1 = yes
- *Fruits*: Consume Fruit one or more times per day 0 = no, 1 = yes
- *Veggies*: Consume Vegetables 1 or more times per day 0 = no, 1 = yes
- *HvyAlcoholConsump*: Adult male: more than 14 drinks per week. Adult female: more than 7 drinks per week. 0 = no, 1 = yes
- *GenHlth*: Would you say that in general your health is: (scale 1-5) 1 = excellent, 2 = very good, 3 = good, 4 = fair, 5 = poor
- *MentHlth*: Days of poor mental health scale 1-30 days
- *PhysHlth*: Physical illness or injury days in past 30 days scale 1-30
- *DiffWalk*: Do you have serious difficulty walking or climbing stairs? 0 = no, 1 = yes
- *Hypertension*: 0 = no hypertension, 1 = hypertension
- *Stroke*: 0 = no, 1 = yes
- *Diabetes*: 0 = no diabetes, 1 = diabetes (target attribute)

Duplicate rows

First of all, using the Duplicate Row Filter node, duplicate data points have been eliminated to remove noise from training and testing.

Number to string

While inspecting every attribute in the dataset, we noticed that the target column type (Diabetes) was Integer but we needed it to be String in order to process it into the model. Hence, we converted its type using the Number to String node.

Partitioning

A partitioning operation with *stratified sampling* on the target column was then performed, dividing the total dataset into training set (80%) and test set (the remaining 20%). The smaller partition will be used to test the model. Furthermore, any subsequent

preprocessing operations will be implemented only after partitioning and through the apply nodes, in order to avoid incurring the phenomenon of data leakage (e.g., the presence of test set data within the training set).

Missing values

We checked whether the dataset at hand contained missing values. With this dataset, this was not the case so no further operations were necessary.

However, looking forward to the deployment phase where the users have the possibility to upload their own file, the workflows were designed to verify this circumstance and handle missing values by replacing them with the most frequent value in the same column. To do that, in KNIME we can use the Missing Value e Missing Value (Apply) nodes.

Outliers

We also analyzed the presence of outliers in every attribute in order to exclude rows that could bring *noise* into our model. The output of the Box Plot node in Figure 1 shows the presence of some outliers in the following attributes:

- MentHlth
- PhysHlth
- BMI

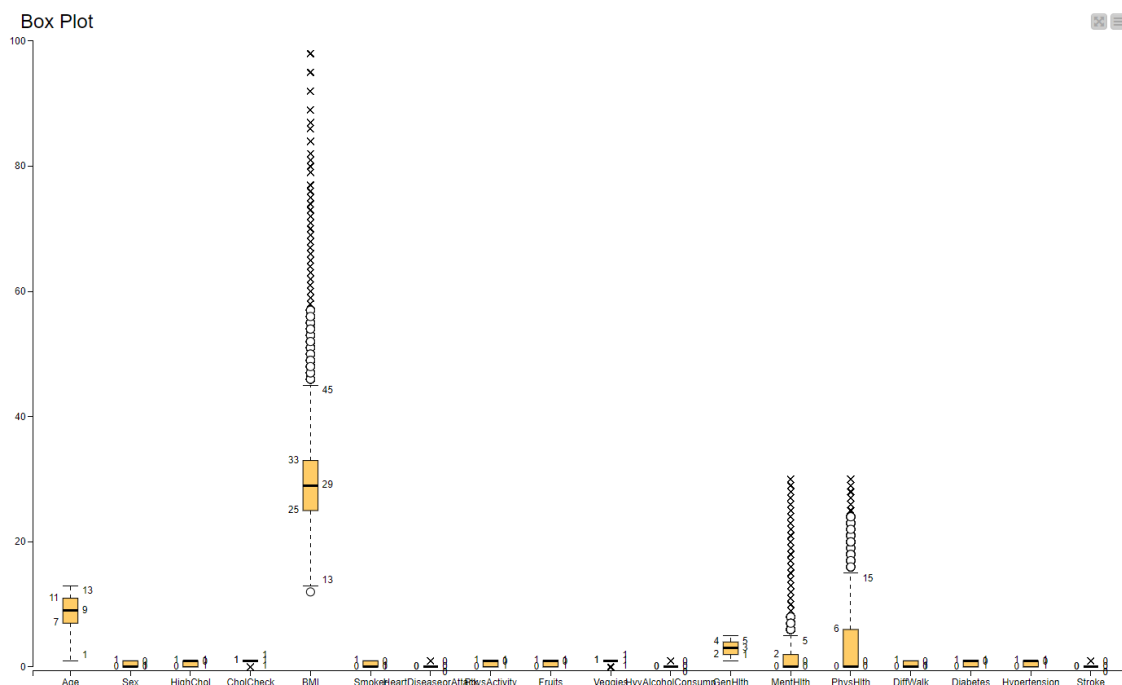


Figure 1: Outliers displayed by the Box Plot node.

In MentHlth and PhysHlth, the observations were within the scale provided by the specification, so we decided not to remove the values because we considered them relevant to analyze all the people, regardless of their health issues. However, outliers in the BMI column were removed using the Numeric Outlier node because, according to the definition of the Body Mass Index, values can range only between 12 and 45 [1]. Therefore, values less than 12 and greater than 45 were excluded, as we considered them possible input errors. In this way, we tried to obtain a cleaner dataset to feed into the model.

Column merger

In order to decrease the dimensionality of the dataset, we decided to merge the binary attributes Fruits and Veggies to create a new column called *healthy_food*, which is the OR combination of Fruits and Veggies. The latter columns were then removed.

Normalization

Different numeric scales in the features could cause issues during the modeling phase, as the model may misleadingly attribute more importance to features with a greater magnitude. *Normalization* solves these problems by mapping original values to new ones using a scale common to all numeric columns. Additionally, normalization maintains the overall distribution and proportions in the source data. We normalized using a min-max scale in the Normalizer node.

Class imbalance

When dealing with a classification task of potentially rare phenomena (e.g., fraud prediction, churn, etc), it is good practice to verify whether the target column contains balanced classes, a necessary condition to train a generalizable classifier. In the dataset at hand, class imbalance is not a relevant issue. Yet, we decided to apply oversampling via the SMOTE node to generate synthetic data rows and obtain a perfectly balanced target column.

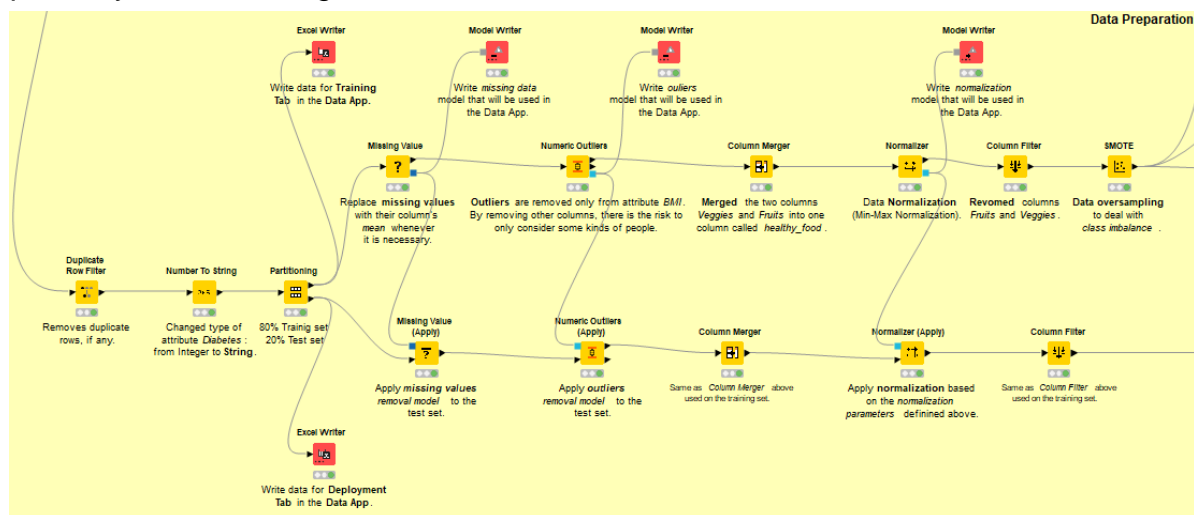


Figure 2: Preprocessing steps in KNIME Analytics Platform.

Model Training

In the modeling phase, we experimented with different machine learning and deep learning models in order to achieve top-notch performance. In this article, we'll only report on the best model, which was included in our solution.

Gradient Boosted Decision Trees

A decision tree is a highly reliable algorithm [2] that poses several questions, each of which narrows down the possible values until you are confident enough to make a prediction. For each answer, there are separate branches. Regardless of the answers to the questions, at the end a prediction is reached (leaf node).

Here there are the steps involved [3]:

1. *Initial Prediction*: begin with an initial prediction for each individual, represented as the $\log(\text{odds})$ of having diabetes. Convert the $\log(\text{odds})$ to a probability using the logistic function.
2. *Calculate Residuals*: compute the residuals for each observation by subtracting the predicted value from the actual value (1 for diabetes, 0 for no diabetes).
3. *Predict Residuals*: build a decision tree to predict the residuals using the provided features. The predicted residuals in each leaf are derived by transforming the initial $\log(\text{odds})$ prediction to $\log(\text{odds})$ residuals.
4. *Obtain New Probability*: pass each sample through the decision tree to obtain predicted residuals. Multiply the predicted residuals by the learning rate and add them to the previous prediction. Convert the new $\log(\text{odds})$ prediction into a probability using the logistic function.
5. *Obtain New Residuals*: calculate new residuals by subtracting the new predicted values from the actual values.
6. *Repeat Steps 3 to 5*: repeat the process of building decision trees, predicting residuals, obtaining new probabilities, and new residuals until the residuals converge close to zero or the specified number of iterations is reached.
7. *Final Computation*: after calculating the output values for all the trees, the final $\log(\text{odds})$ prediction for an individual having diabetes is obtained by summing the predictions from all the trees. Convert the final $\log(\text{odds})$ prediction to a probability using the logistic function. Using a probability threshold of 0.5, classify individuals as "Yes" if the predicted probability of diabetes is greater than 0.5, and "No" otherwise.

For this work, the output of the model corresponds to the *probability* P that the patient has diabetes.

In order to obtain predictions that were as close as possible to the target value, we evaluated each model we experimented with using the *Log-Loss*. The model that returned the lowest Log-Loss value was chosen. In our case, the Gradient Boosted Decision Trees outperformed all other models.

Feature Selection

Feature Selection is the process of detecting relevant features and removing irrelevant, redundant, or noisy ones. This process speeds up data mining algorithms, improves predictive accuracy, and increases comprehensibility. Irrelevant features are those that provide no useful information, and redundant features provide no more information than the currently selected features [4].

In our work, we implemented a forward feature selection, that is an iterative method that starts with no feature and adds the feature(s) that improves the model the most at each iteration. Inside the *feature selection loop* (using KNIME's Feature Selection nodes), there are the learner and the predictor of the model. The data for the training of the model is fed into the learner, and the data for testing is fed into the predictor. These two data patients are not fixed. Indeed, *cross validation* with 3 folds is used in order to generalize the learning process of the model. This method uses a subdivision of the total dataset into k parts of equal size and at each step the k -th part of the dataset becomes a validation set, while the others constitute the training set. In this way, the model is trained on k different parts to

avoid overfitting problems. In KNIME, this is done using the X-Partitioner and X-Aggregator nodes.

Log-Loss was used for the evaluation. This metric tells us how close the prediction probability is to the corresponding actual/true value. The more the predicted probability diverges from the actual value, the higher is the Log-Loss value [5].

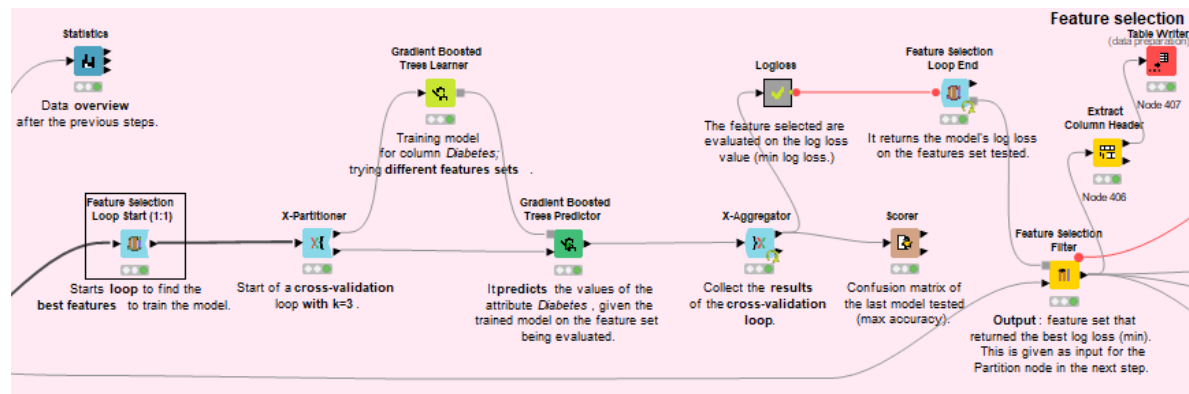


Figure 3: Feature Selection.

Model optimization

Before proceeding with the testing phase, it is useful to optimize the model by performing a validation on its hyperparameters in order to select those that minimize loss.

As far as the GBDT is concerned, some of the hyperparameters we decided to optimize are:

- *Limit number of levels (tree depth)*: number of tree levels to be learned. For a GBDT algorithm, usually a depth less than 10 is sufficient. Deeper trees will quickly lead to overfitting [6].
- *Number of models*: the number of decision trees to learn. A "reasonable" value can range from very few (say 10) to many thousands for small data sets with few target category values [6].

Similarly to what was shown above, there is a *parameter optimization loop* via the Parameters Optimization Loop nodes, which updates the parameters applied to the model at each iteration. In our case, the depth parameter (depth) can vary from 1 to 10 and the number of models (n_models) from 1 to 500. A search strategy called *Hillclimbing* is used: a random start combination is created and direct neighbors are evaluated. The best combination of neighbors is the starting point for the next iteration. If no neighbor improves the objective function, the cycle ends [7]. As before, *cross validation* was also used to avoid overfitting, and the parameters that returned a lower Log-Loss were selected.

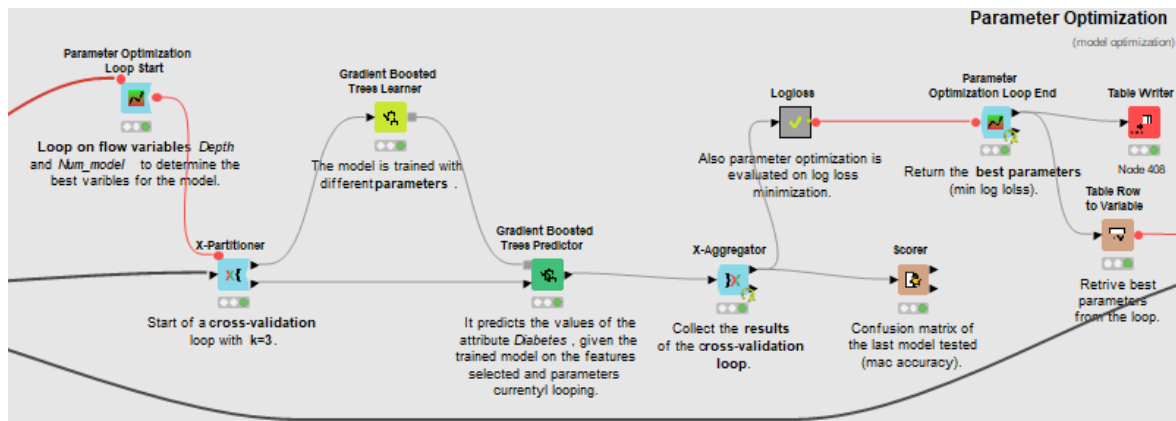


Figure 4: Parameter Optimization.

Model Testing

Once the suitable parameters and features were selected, we tested the model before deployment.

The data used for validation is fed into the model learner, whereas test data is fed into the predictor. These partitions were created at the beginning of the workflow, in such a manner that we had "*unseen*" data (test data) that could be used to simulate predictions.

Also in this case, model evaluation was done primarily via Log-Loss. However, for a more comprehensive testing procedure, other methods were included, such as a ROC curve and class and overall accuracy statistics.

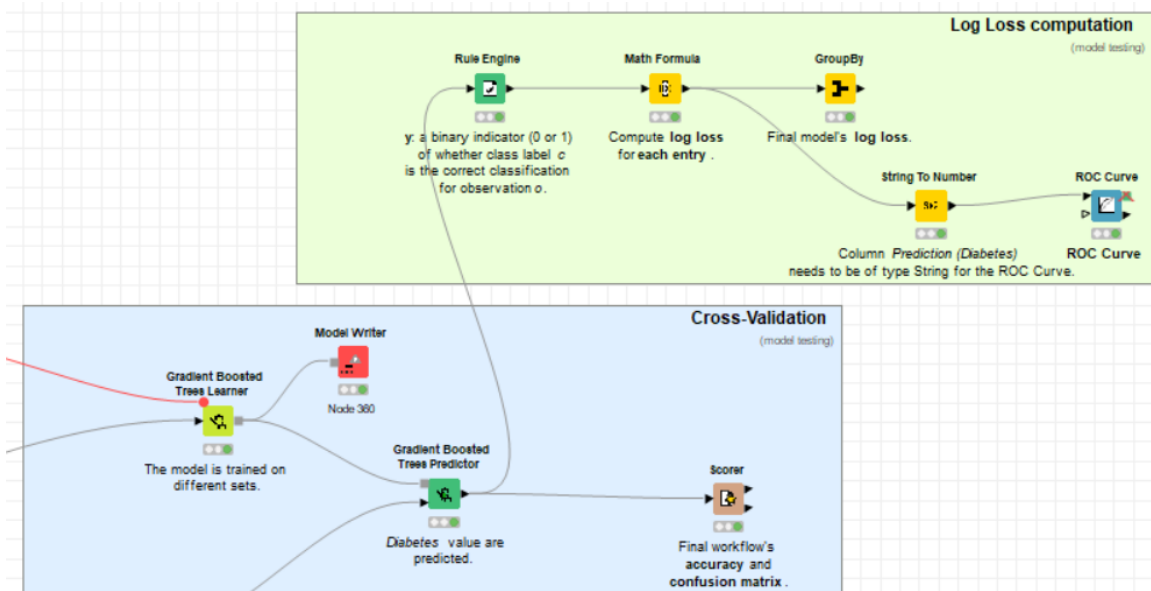


Figure 5: Applying model to test data.

The final result led to a Log-Loss of 0.531 and to an overall accuracy of 73.6%.

Accuracy statistics - 3:335 - Scorer (Final workflow's)

File
Edit
HiLite
Navigation
View

Table "default" - Rows: 3

Spec - Columns: 11

Properties

Flow Variables

Row ID	I TruePo...	I FalsePo...	I TrueNe...	I FalseN...	D Recall	D Precision	D Sensitivity	D Specifity	D F-meas...	D Accuracy	D Cohen'...
0	2418	827	2943	1085	0.69	0.745	0.69	0.781	0.717	?	?
1	2943	1085	2418	827	0.781	0.731	0.781	0.69	0.755	?	?
Overall	?	?	?	?	?	?	?	?	?	0.737	0.472

Figure 6: Class and overall accuracy statistics.

Finally, the *ROC curve* was plotted. The Receiver Operating Characteristic Curve (ROC) is a curve showing the performance of a classification model at different probability thresholds. The ROC plot is obtained by plotting False Positive Rate (x-axis) and True Positive Rate (y-axis) on the Cartesian frame of reference for different probability threshold values between 0.0 and 1.0.

In general, the model is more accurate the closer its ROC curve is to the upper-left corner of the graph. This goodness value is called Area Under the Curve (AUC), which takes values between [0,1] and represents a measure of accuracy: the higher the value, the higher the accuracy [8].

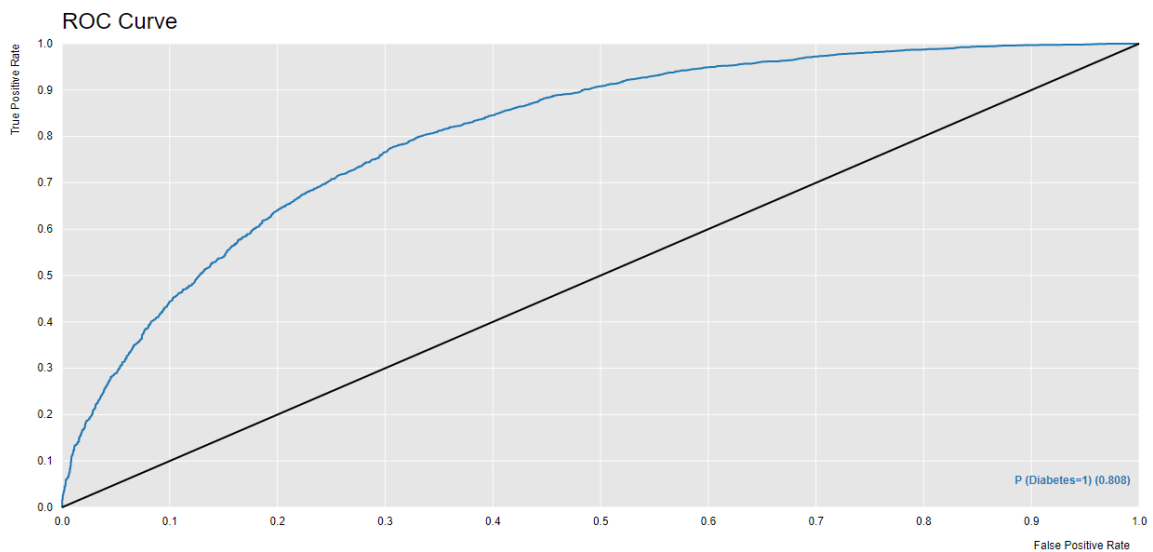


Figure 7: ROC Curve.

Deployment

After reaching a satisfactory level of model performance, the last stages of our project dealt with model deployment and the building of an interactive data app. Our goal was to use the trained model to output predictions within a user-friendly data app.

As you can see in the Figure 2 (pre-processing) we used a few Model Writer nodes to save the model used to pre-process the training data and in the Figure 5 (testing) we used another Model Writer node to save our *best trained model*.

We created a deployment workflow, we used several Model Reader nodes:

1. To read our best trained model;
2. To read the models used to preprocess the training data (in the “Preprocessing” metanode), in such a way that we could reproduce the same steps also on new unseen data.

The predictor node is fed with the best trained model in order to output predictions.

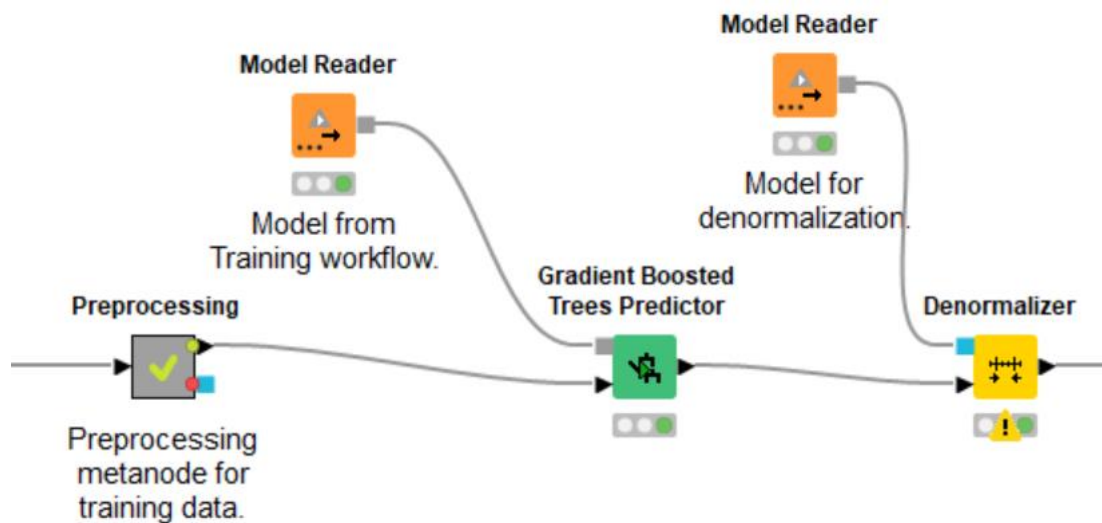


Figure 8: Deployment branch inside the data app.

Data App

A *Data App* is a powerful tool to consume data analyses and visualizations, while keeping hidden the complexity of the underlying workflow. It allows end-users to monitor, get insights, and interact with the data in a variety of ways, including charts, graphs, and tables.

We developed our data app with the goal of creating a tool that could be used by physicians and health care personnel to conduct both mass screening and individual inspections of their patients. Our tool is intended to support diagnosis and early diabetes detection, and not as a substitute for the work of medical professionals.

Figure 9 provides an overview of the data app workflow:

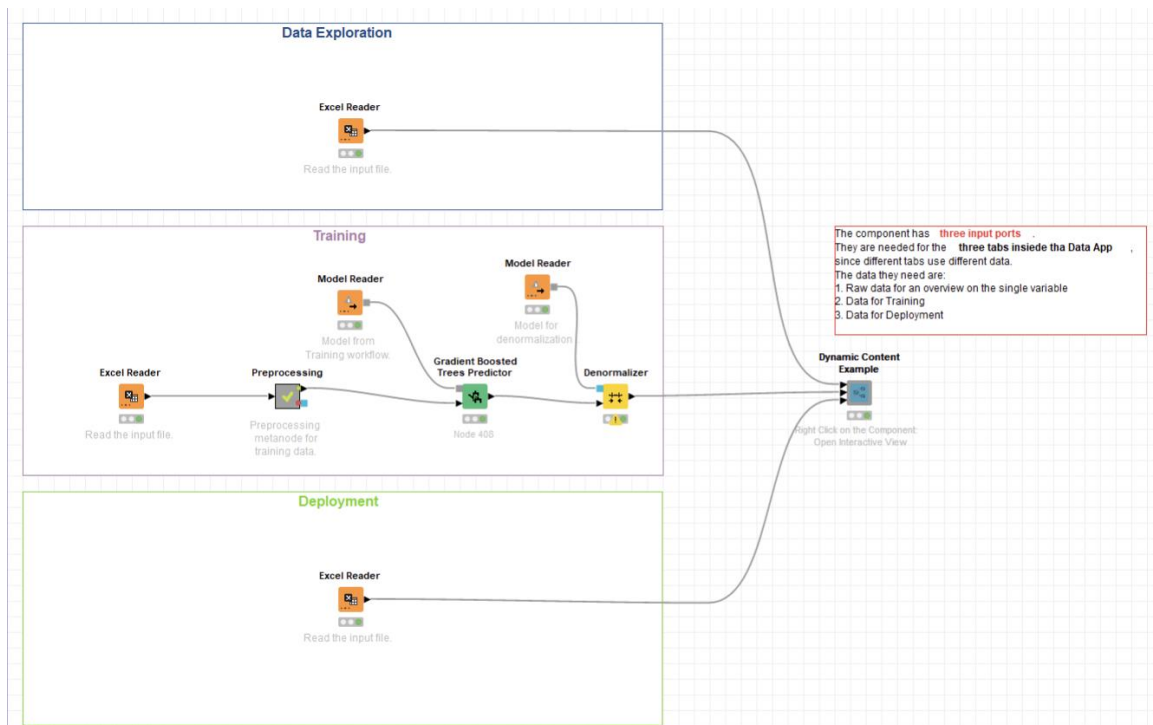


Figure 9: Model deployment and data app.

All the contents of the app and the visualizations are inside the “Dynamic Content Example” component, which has *three inputs*:

1. Raw data to allow data exploration
2. The results of the training process
3. Test data to use for deployment

Our dashboard is composed of *five pages*:

1. Welcome page
2. Descriptive statistics
3. Training results
4. Deployment (upload and explore data)
5. Single patient inspection

Figure 10 gives an overview of what’s inside the “Dynamic Content Example” component. We created a dynamic menu orchestrated by the Single Selection Widget and Case Switch Start node to enable the user to switch from one page to another and interact with the corresponding visualizations. It is worth noticing that each page is wrapped inside a separate component.

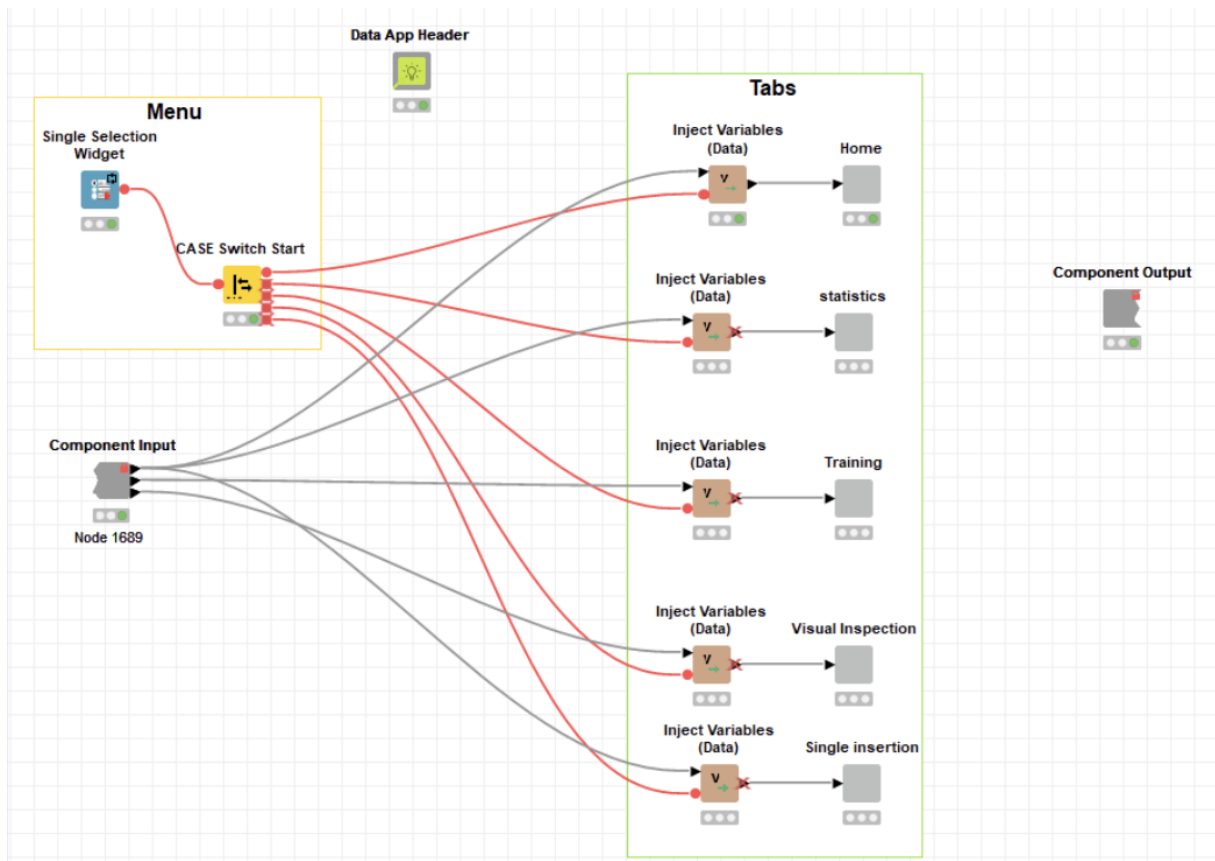



Figure 10: Page orchestration in the data app.

We will now provide a brief description of each page in the data app, showing both the final look & feel and the underlying workflow.

Welcome page

The first page of the Data App is a welcome page which provides some relevant information to the user. In particular, this first part contains:

- *Instructions*: a short overview about all the dashboards inside the data app, describing what each of them contains and how to interact with it;
- *Data description*: an explanation of every column and the value they contain.



Diabetes Prediction with GB Trees Model

Screening and prediction of the Diabetes disease

Choose what to visualize:

☒ Welcome to the Data App!
 ☐ Descriptive statistics
 ☐ Training results
 ☐ Deployment exploration
 ☐ Single insertion

Welcome to the Diabetes Data App!

Instructions

The dashboard has four interactive tabs:

- Descriptive Statistics section:** here it is possible to have an introductory overview of each variable given as input to the model. By selecting the variable of interest, the main statistics and introductory graphs relating to it will be displayed;
- Training result section:** the efficiency of the model on the training set is shown, with the appropriate accuracy measures. Also, the features and the parameters chosen for the model are displayed.
- Deployment exploration section:** here it is given the possibility to select the individual patient in such a way as to have as output the prediction of the model (diabetes/non-diabetes) with the relative probability of confidence. The prediction results can be explored by choosing one variable and compare the positive prediction with the negative ones. It is also possible to download the entire csv with all the predictions or the predictions regarding the single patient.
- Single insertion:** in this section you can enter your data through the screen and have a personal prediction on the fly. It is necessary to be aware of the meaning of the columns, so please check the Variable description provided in the page!

Below there is the explanations of all the columns in the dataset we used for the predictions:

Target Variable

- **Diabetes:** 0 = no diabetes, 1 = diabetes (Target variable)

Integer Binary Variables

- **Sex:** 0 = female, 1 = male
- **HighChol:** 0 = no high cholesterol, 1 = high cholesterol
- **CholCheck:** 0 = no cholesterol check in 5 years, 1 = yes cholesterol check in 5 years
- **Smoker:** Have you smoked at least 100 cigarettes in your entire life? [Note: 5 packs = 100 cigarettes] 0 = no, 1 = yes
- **HeartDiseaseorAttack:** Coronary heart disease (CHD) or myocardial infarction (MI) 0 = no, 1 = yes
- **PhysActivity:** Physical activity in past 30 days - not including job 0 = no, 1 = yes
- **Fruits:** Consume Fruit one or more times per day 0 = no, 1 = yes
- **Veggies:** Consume Vegetables 1 or more times per day 0 = no, 1 = yes
- **HvyAlcoholConsump:** Adult male: more than 14 drinks per week. Adult female: more than 7 drinks per week. 0 = no, 1 = yes
- **DiffWalk:** Do you have serious difficulty walking or climbing stairs? 0 = no, 1 = yes
- **Hypertension:** 0 = no hypertension, 1 = hypertension
- **Stroke:** 0 = no, 1 = yes

Integer Variables

- **Age:** 3-level age category (_AGEG5YR see codebook) 1 = 18-24, 9 = 60-64, 13 = 80 or older
- **BMI:** Body Mass Index
- **GenHlth:** Would you say that in general your health is: (scale 1-5) 1 = excellent, 2 = very good, 3 = good, 4 = fair, 5 = poor
- **MentHlth:** Days of poor mental health scale 1-30 days
- **PhysHlth:** Physical illness or injury days in past 30 days scale 1-30

These variables were sometimes modified or new ones were inserted to make the dashboard clearer to be interpreted.

Figure 11: Welcome page screenshots.

The workflow for this page is very simple (Figure 12). Texts were included in the dashboard via the Text Output Widget nodes that allow us to write HTML chunks. In this way, it was possible to customize the page appearance. This was achieved through the use of a CSS Editor node.

To deliver a homogeneous UX, each dashboard contains a CSS Editor node that brings consistency between all the pages by applying the same color palette and visual elements.

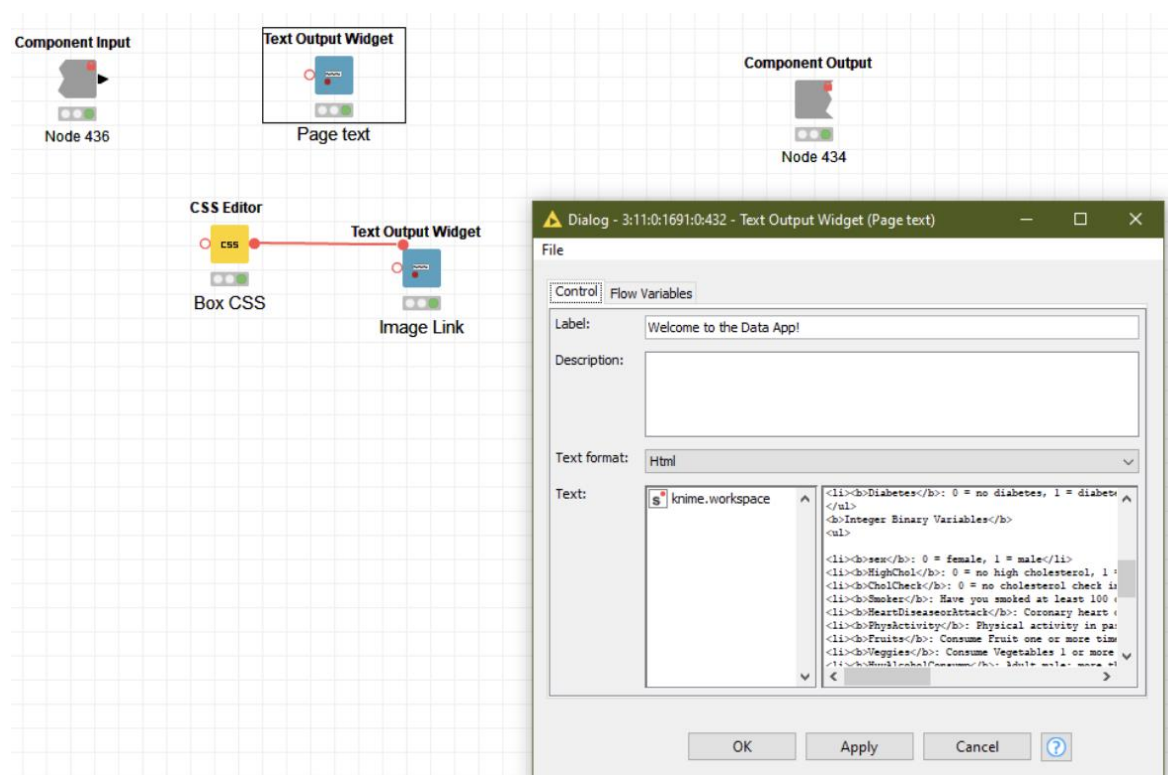


Figure 12: Welcome page workflow.

Descriptive statistics

This is the real first page of the Data App. It gives the user the possibility to select a variable and visualize its descriptive statistics results using tables and charts. The Column Section

Widget node allows the user to choose a column, and the “Variable description” field updates on this choice in order to provide key info on the column.

According to the column’s data type, the plots change. If the attribute is binary, then the user will see a Pie Chart, whereas for non-binary attributes a Bar Chart and a Box Plot will appear.

Figure 13 and Figure 14 show what the dashboard displays when the user selects a continuous attribute and a binary attribute, respectively.

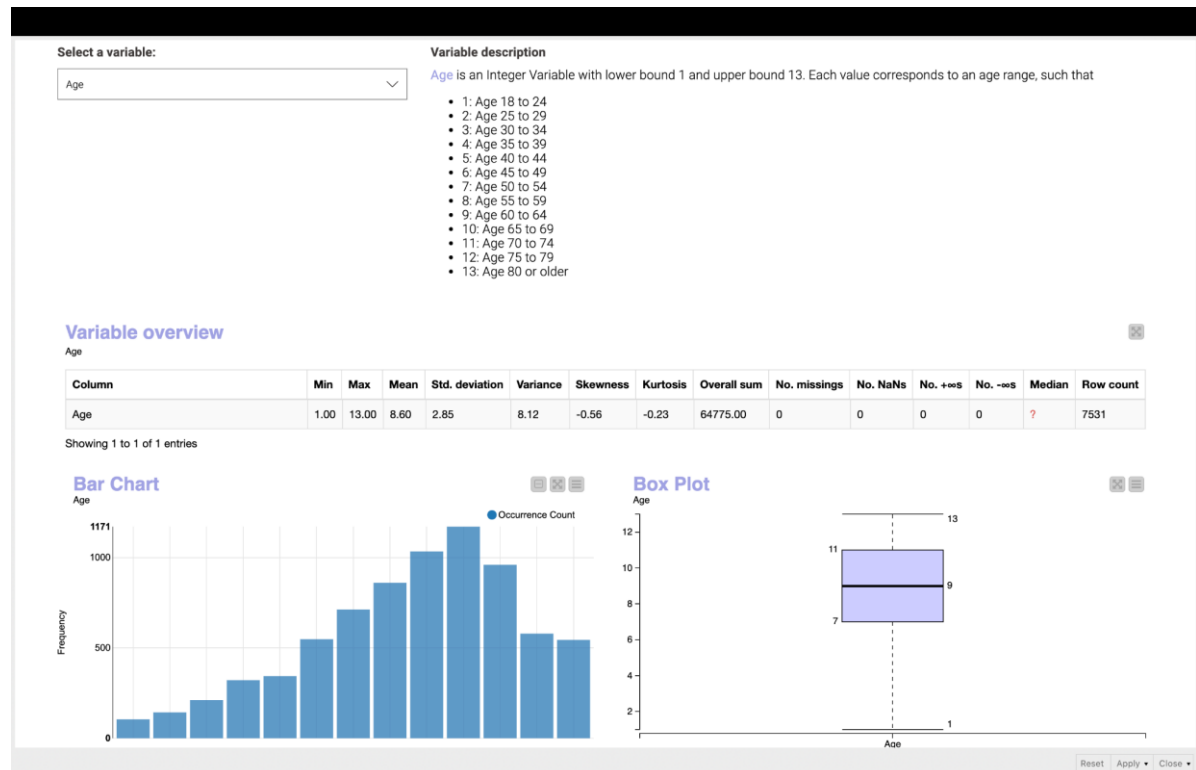


Figure 13.a: Descriptive Statistics for an attribute of type Integer.

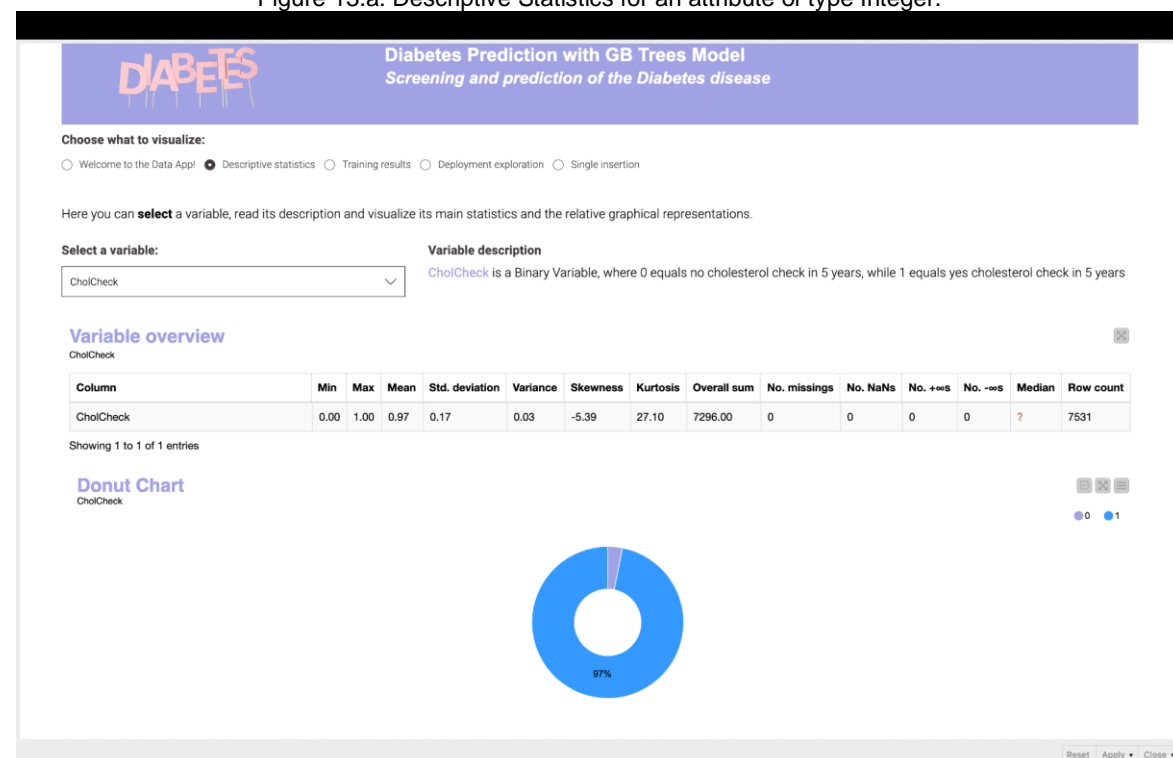


Figure 13.b: Descriptive Statistics for a binary attribute.

The underlying workflow is displayed in Figure 14. Here we can clearly distinguish all the sections that compose the dashboard:

1. *Variable selection*: the user selects a column via the Column Selection Widget node. The available columns are converted into a list of String objects before passing them onto the widget.
2. *Variable Description*: here there is a table containing all the descriptions of the columns. Once a variable is selected, the Row Filter node picks the right description based on the value it receives through the Flow Variable originating from the widget. As a result, the relevant description is visualized in the Text Output Widget node.
3. *Variable's synthesis table*: it contains summary statistics for the attributes, such as mean, median and standard deviation.
4. *Rule Engine and Case Switch Start*: these nodes are meant to check whether an attribute is binary or not and activate the proper branch accordingly.
5. *Plots for integer variables*: a Bar Chart and a Box Plot are displayed.
6. *Plots for binary variables*: a Pie Chart is created.

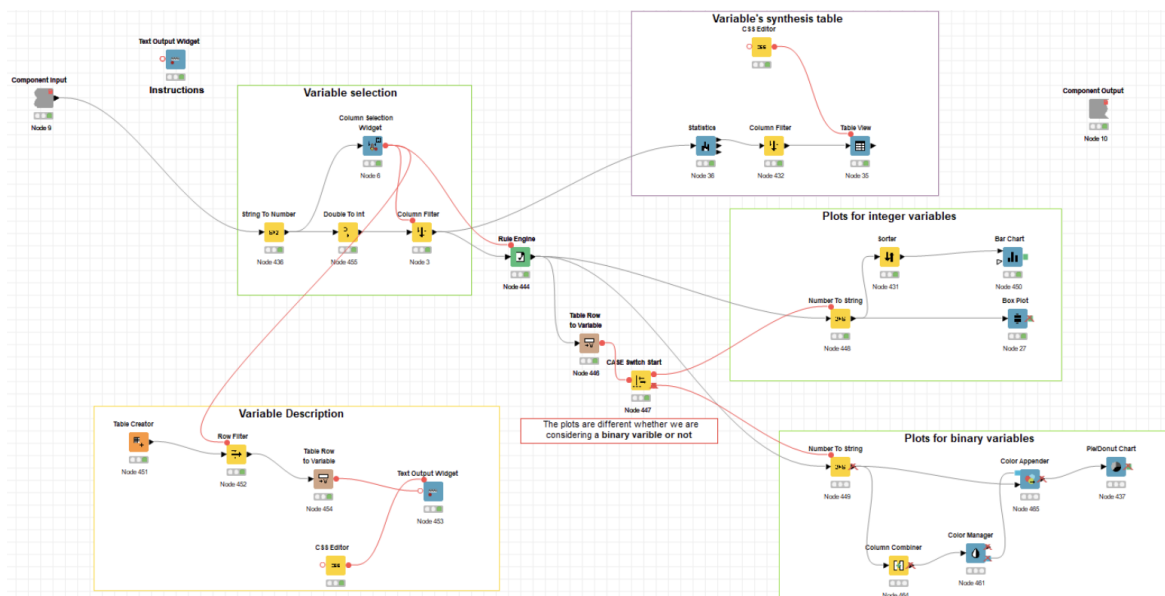


Figure 14: Descriptive Statistics workflow.

Training results

This page of the Data App shows the *results of the model training phase*. This information was saved in the Training workflow by using the Table Writer nodes and later retrieved from the current workflow via the Table Reader nodes. The results are presented via tables and plots. Here we can find:

- *Feature Selection*: it reports the columns selected by the feature selection loop.
- *Scorer View*: it includes the confusion matrix and the overall accuracy statistics, both output by the Scorer node.
- *Log-Loss*: the text displays the final Log-Loss metric value, captured on the training data.
- *ROC Curve*: the plot shows the Roc Curve.
- *Best Parameters*: the best model parameters, resulting from the Parameters Optimization Loop.

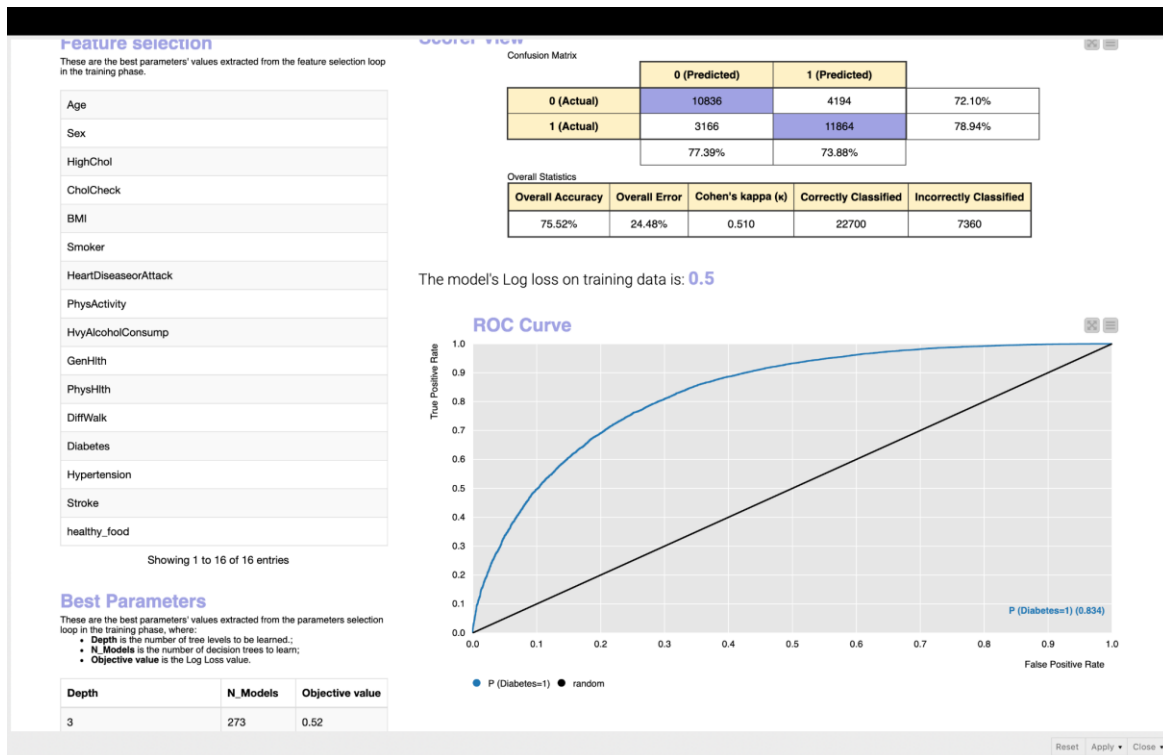


Figure 15: Training dashboard.

The underlying workflow is illustrated in Figure 16. We can observe that the various elements were created in a straightforward and simple manner.

The majority of the dashboard visuals were graphically customized using the CSS Editor node for the sake of consistency.

The Feature and Best Parameters tables were displayed by the Table View nodes and the data they contain was captured by the Table Reader nodes.

As for the ROC Curve, this was created with the ROC Curve node, which also takes as input the output of a Color Manager node to assign colors in the chart.

The Scorer View table is created with a Scorer (Javascript) node, which automatically creates the two tables described above.

Finally, the Log-Loss output is inserted in the dashboard, after being calculated, through the Text Output Widget node.

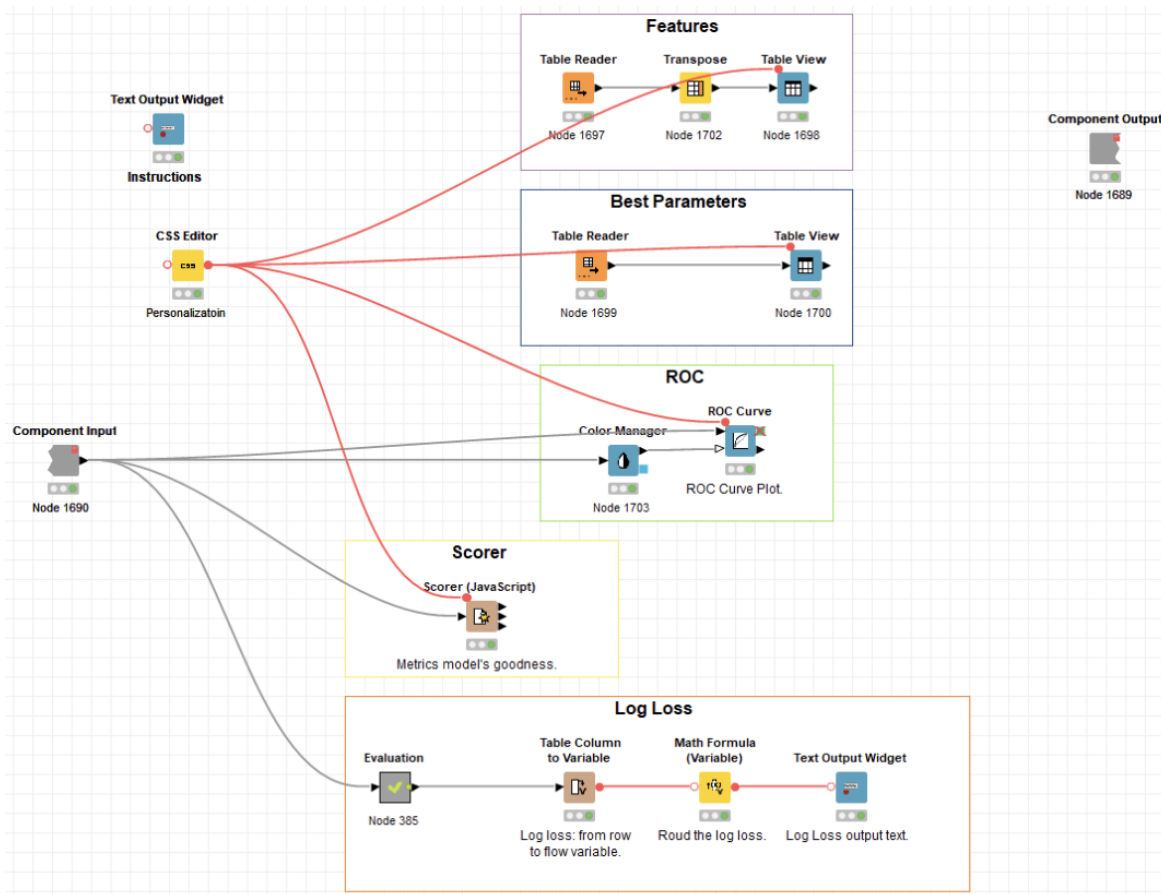


Figure 16: Training results workflow.

Deployment (upload and explore data)

In this page, the user can *upload their own file* and *explore the predictions* created by the trained model. The file can have any number of rows but it must be a .xlsx file and contain the exact same columns described in the Welcome page. If the data does not respect these constraints, the page will return an error message.

Figure 17 shows how the page appears before file upload. The user can enter the data by clicking on the button *Select file* and, after clicking on *Refresh*, the page will reload and show the results.

Here you can upload **your own file** and see the predictions for the diabetes. Make sure to upload a file .xlsx that contains the same columns specified in the Welcome Page. Once you selected the file, click on the **Refresh** button to load the page.!

Upload your file Click here to refresh the page with new data!
Select file Test_set.xlsx (414.34 KB) Refresh

Figure 17: Deployment dashboard screenshot *before* uploading the data.

After the data is processed, the user can start exploring the outcomes. Figure 18 shows the *first part* of the page. Here it is possible to select a patient through the *Choose a Patient* menu and visualize:

- **Patient Overview:** a table containing each patient's information. This will be input into the model to generate predictions. The results are also displayed in this table. Indeed, the last column is called *Final Prediction* and contains "No Diabetes"/"Diabetes" labels color-coded in green and red, respectively.

- **Diabetes Probability Comparison:** a Bar Chart that graphically compares the predicted probabilities of having or not diabetes.

In order to obtain permanent results for further inspection, the user can *download* the results. The file will contain all the columns in the dataset, to which predictions are appended. Moreover, the user can choose to download the entire dataset or just the selected row in the *Choose a Patient* menu.



Figure 18: Deployment dashboard *after* loading the data (1/3).

The *second part* of the page displays attribute distribution for each predicted class, giving a few insights into what attributes may influence the model's decision-making. After selecting an attribute, the user can view a brief description of it (as seen in the previous tabs) and a graphical representation appears. The plots change depending on the attribute data type.

In Figure 19, after selecting HighChol, two Pie Charts are displayed. On the left-hand side, the first Pie Chart visualizes data distributions for patients that were predicted to have diabetes, whereas on the right-hand side, the cart refers to patients that were not predicted to have diabetes. In this example, we can see that the majority of patients considered diabetic by the model have a high cholesterol level (1). Therefore, we could infer that this is a condition that drives the model to predict such patients prone to develop diabetes.

After you selected a variable of choice, you can explore the comparison between the patients that were predicted as diabetic and the ones there were not predicted as such. Keep in mind that the plot are done on the **results predicted by the model** and not on the original dataset.

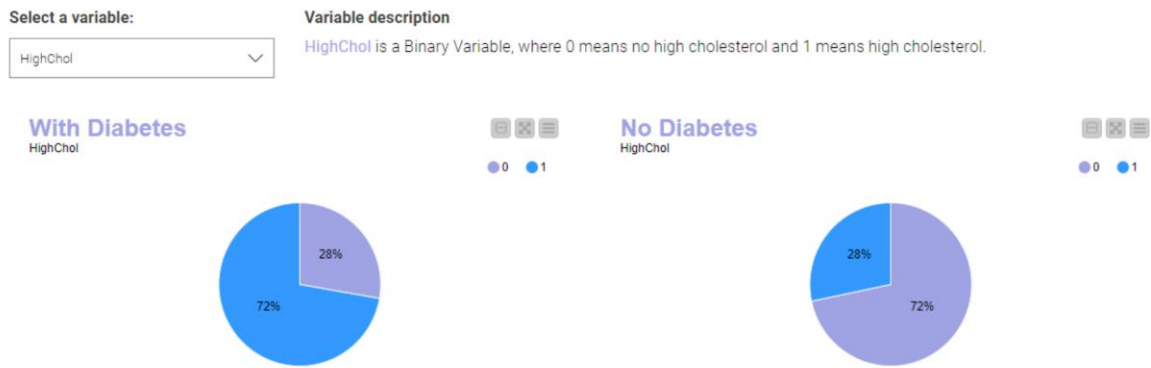


Figure 19: Deployment dashboard *after* loading the data (2/3): Binary attributes.

For what concerns continuous attributes, the user can extract the same information from a Line Plot, where the orange line represents patients forecasted as diabetic (1) and the blue one as non-diabetic (0).

After you selected a variable of choice, you can explore the comparison between the patients that were predicted as diabetic and the ones there were not predicted as such. Keep in mind that the plot are done on the **results predicted by the model** and not on the original dataset.

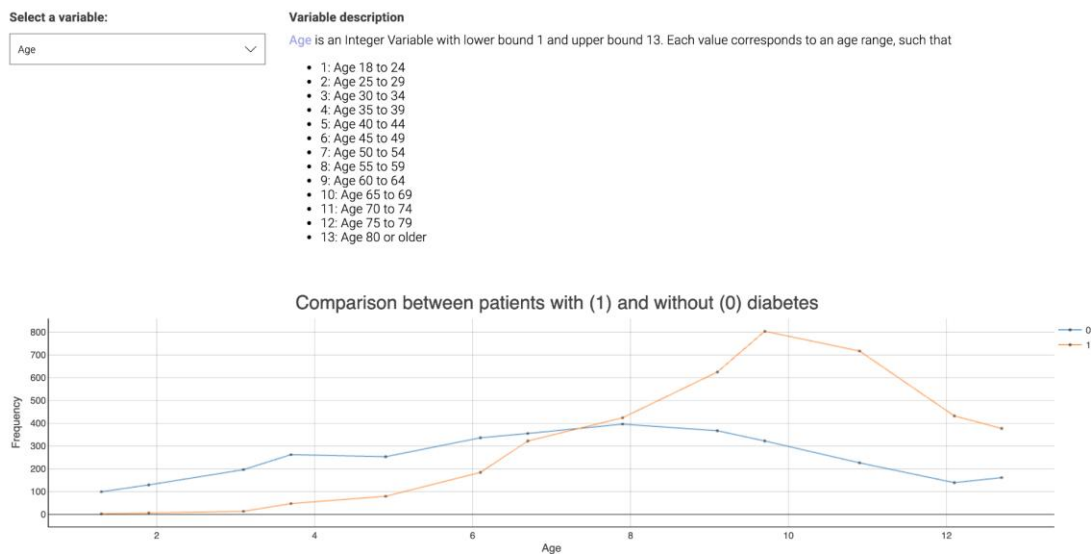


Figure 20: Deployment dashboard *after* loading the data (3/3): Continuous attributes.

The underlying workflow for this page is shown in Figure 21. Even if it may look complicated, it is just the combination of the various elements described in the previous workflows.

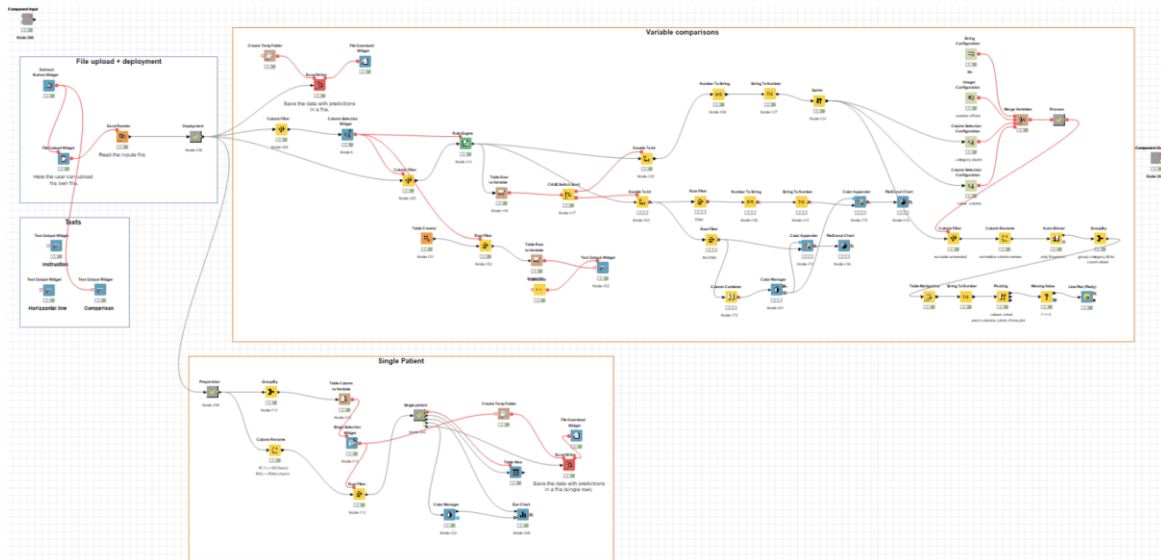


Figure 21: Deployment Dashboard workflow.

Single patient inspection

This is the last page of the Data App and the most *interactive* one. The goal is to provide GPs and healthcare personnel with a tool that helps them screen patients and assess their individual risk of developing diabetes. To this end, the user can obtain a diabetes prediction based on the provided (individual) input data whenever they need it. Therefore, this page is designed to deliver quick and intuitive results that facilitate action-taking.

Here you can enter the patient's features and see the **diabetes predictions**. Once you are done, click the **Refresh** button to start the computation.

Need help with the columns? Check them here!

Age

Variable description

Age is an Integer Variable with lower bound 1 and upper bound 13. Each value corresponds to an age range, such that

- 1: Age 18 to 24
- 2: Age 25 to 29
- 3: Age 30 to 34
- 4: Age 35 to 39
- 5: Age 40 to 44
- 6: Age 45 to 49
- 7: Age 50 to 54
- 8: Age 55 to 59
- 9: Age 60 to 64
- 10: Age 65 to 69
- 11: Age 70 to 74
- 12: Age 75 to 79
- 13: Age 80 or older

Figure 22: Single Patience Insertion screenshot (1/2).

Patient's name:

Age

1

1 13

Weight (kg)

80

30 160

Height (cm)

160

100 210

GenHlth

1

1 5

MentHlth

13

1 30

PhysHlth

18

1 30

Sex: 1

HighChol: 0

CholCheck: 0

Smoker: 0

HeartDiseaseorAttack: 0

PhysActivity: 0

Fruits: 0

DiffWalk: 0

HvyAlcoholConsump: 0

Stroke: 0

Hypertension: 0

Veggies: 0

Click to go

Refresh

Figure 23: Single Patience Insertion screenshot (2/2).

Figure 22 and Figure 23 illustrate how the user can interact with the dashboard. First of all, there is the usual menu where it is possible to choose an attribute and retrieve a short description. Next, the actual user-driven data collection begins. The user can specify the patient's name, general information as well as lifestyle and clinical factors. Once all info/parameters have been introduced, clicking on the *Refresh button* returns two outputs (Figure 24):

- *Patient overview*: a table containing all the information inserted by the user, including patient's name and final diabetes prediction;
- *Diabetes Probability Comparison*: a Bar Chart that clearly shows whether a patient was predicted to have or not.

The user can download the data in the Patient overview table and store it locally.



Figure 24: Single Patient Insertion results.

Figure 25 the underlying workflow in this page. All the inputs are controlled by a single Refresh Button Widget node that triggers the re-execution of all downstream nodes as well as the data preprocessing operations upon clicking on it. Next, the data inserted is collected and a single row is created. This information is given as input to the trained model that is able to make predictions. Finally, the results are made available to the user in the dashboard.

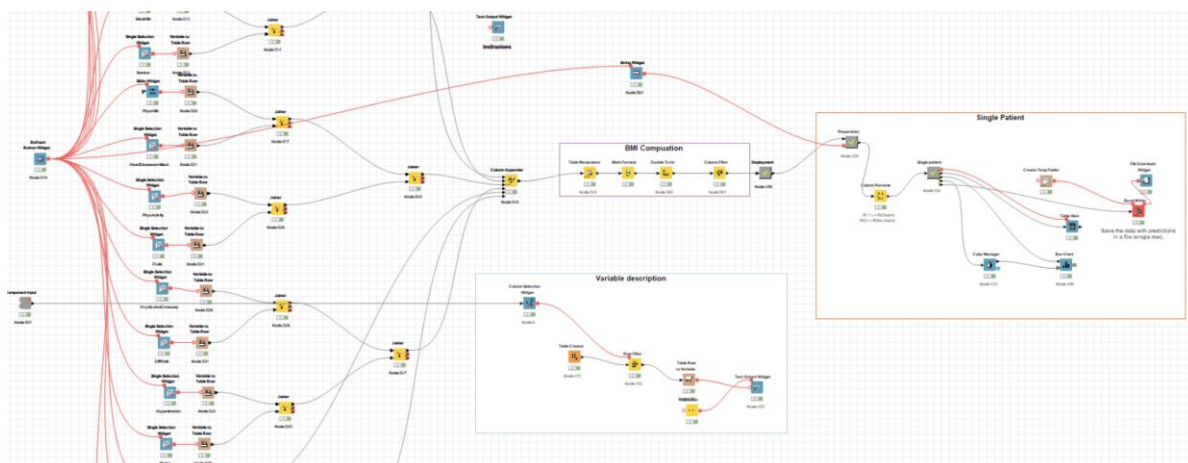


Figure 25: Single Patient Insertion workflow.

Conclusions

In terms of accuracy, the model correctly predicted the onset of diabetes in about 3 out of 4 people. Another important factor is related to the confusion matrix. The number of FN

was smaller than FP. This means that fewer errors were made in predicting an unidentified diabetic patient, compared to identifying a non-sick patient as diabetic.

A Log-Loss value of 0.531 indicated that the model's predictions were reasonably close to the true values, meaning that the model output accurate predictions for the majority of cases. This implies that our predictive model behaves fairly well at identifying patients with diabetes. It is worth noting that the threshold for what is considered a "good" Log-Loss value may depend on the specific requirements and also on the costs associated with different types of errors (false negatives may be more costly than false positives in this context).

Finally, the creation of a dynamic and user-friendly Data App could hopefully help healthcare professionals raise awareness on the risks of diabetes and be used as a tool for early diagnosis.

References

- [1] [BMI Calculator](#)
- [2] S. Ravikumar and P. Saraf, "Prediction of Stock Prices using Machine Learning (Regression, Classification) Algorithms", 2020
- [3] [Gradient Boosting Trees for Classification: A Beginner's Guide](#)
- [4] [Vipin Kumar and Sonajharia Minz, Feature Selection: A literature Review, 2014](#)
- [5] [Intuition behind Log-loss score](#)
- [6] [Gradient Boosted Trees Learner](#)
- [7] [Parameter Optimization Loop Start](#)
- [8] [CURVA ROC: APPLICAZIONE AI TEST DIAGNOSTICI](#)