**KNIME PROJECT REPORT**

**ON**

**CREDIT CARD FRAUD DETECTION**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **TOPIC** | **PAGE No.** |
| **1** | **Abstract** | **5** |
| **2** | **Introduction** | **6** |
| **3** | **ABOUT DATA SET** | **7** |
| **4** | **PROJECT ARCHITECTURE** | **8** |
| **5** | **Nodes Used** | **9-13** |
| **6** | **Workflow** | **14** |
| **7** | **Magnified Workflow** | **15** |
| **8** | **WORKFLOW EXPLANATION** | **16-17** |
| **9** | **ISOLATION FOREST technique output table** | **18** |
| **10** | **LOGISTIC REGRESSION technique output table** | **19** |
| **11** | **QUANTILE BASED technique output table** | **20** |
| **12** | **DISTRIBUTION BASED technique output table** | **21** |
| **13** | **RANDOM FOREST technique output table** | **22** |
| **14** | **EXECUTION TIME COMPARISON** | **23** |
| **15** | **COMPARISON OF FRAUD DETECTION TECHNIQUE** | **24** |
| **16** | **Conclusion** | **25** |
| **17** | **BIBLIOGRAPHY** | **26** |

**Abstract**

* It is vital that credit card companies are able to identify fraudulent credit card transactions so that customers are not charged for items that they did not purchase. Such problems can be tackled with Data Science and its importance, along with Machine Learning, cannot be overstated.
* This project intends to illustrate the modelling of a data set using machine learning with Credit Card Fraud Detection. The Credit Card Fraud Detection Problem includes modelling past credit card transactions with the data of the ones that turned out to be fraud. This model is then used to recognize whether a new transaction is fraudulent or not.
* In this process, we have focused on analyzing and pre-processing data sets as well as the deployment of multiple anomaly detection techniques such as Logistic and Naïve Bayes algorithm on the PCA transformed Credit Card Transaction data.

**Introduction**

* 'Fraud' in credit card transactions is unauthorized and unwanted usage of an account by someone other than the owner of that account. Fraud detection involves monitoring the activities of populations of users in order to estimate, perceive or avoid objectionable behavior, which consist of fraud and intrusion.
* Machine learning algorithms are employed to analyze all the authorized transactions and report the suspicious ones.
* Our objective here is to detect 100% of the fraudulent transactions while minimizing the incorrect fraud classifications. Credit Card Fraud Detection is a typical sample of classification.

**ABOUT DATA SET**

* First of all, we obtained our dataset from Kaggle, a data analysis website which provides datasets. Inside this dataset, there are 31 columns out of which 28 are named as v1-v28 to protect sensitive data. It contains only numerical input variables which are the result of a PCA transformation.
* The other columns represent Time, Amount and Class. Time shows the time gap between the first transaction and the following one. Amount is the amount of money transacted. Class 0 represents a valid transaction and 1 represents a fraudulent one.

**PROJECT ARCHITECTURE**

READ

Portioning

Missing values

Rule Engine

Joiner

TRANSFORM

Logistic regression

Isolation forest

Quantile Based

Random forest

Distribution based

ANALYZE

Timer info

Bar chart

EXPLORE

DEPLOY

**Nodes Used**

* **File Reader**

Reads the most common text files. To auto-guess the structure of the file click the Autodetect format button. If you encounter problems with incorrect guessed data types disable the Limit data rows scanned option in the Advanced Settings tab. If the input file structure changes between different invocations, enable the Support changing file schemas option in the Advanced Settings tab.

* **Rule Engine**

This node takes a list of user-defined rules and tries to match them to each row in the input table. If a rule matches, its outcome value is added into a new column. The first matching rule in order of definition determines the outcome.

Each rule is represented by a line. To add comments, start a line with // (comments can not be placed in the same line as a rule). Anything after // will not be interpreted as a rule. Rules consist of a condition part (antecedent), which must evaluate to *true* or *false* , and an outcome (consequent, after the => symbol) which is put into the new column if the rule matches.

* **Partitioning**

The input table is split into two partitions (i.e. row-wise), e.g. train and test data. The two partitions are available at the two output ports. The following options are available in the dialog:

* **Merge Variables**

Merges flow variables into one stream. This node aggregates variables defined in different input connections into one variable connection. It does not modify variables but only merges them. Note, if there are the same variables defined in different inputs, the standard conflict handling is applied: Top most inputs take priority and define the value of a variable. This node can also be used as a common barrier point to control the execution order of nodes, i.e. nodes connected to the output port will not start executing until all upstream nodes have been executed.

* **Timer info**

This node reports individual and aggregate timing/execution information for all nodes of the workflow at this level and for (nested) meta nodes up until the specified depth.

The output table lists all nodes in the workflow that were executed since the last reset. This also includes nodes in meta nodes up to the specified nesting depth.

* **Joiner**

This node combines two tables similar to a join in a database. It combines each row from the top input port with each row from the bottom input port that has identical values in selected columns. Rows that remain unmatched can also be output.

* **Random forest**

Learns a random forest\*, which consists of a chosen number of decision trees. Each of the decision tree models is built with a different set of rows (records) and for each split within a tree a randomly chosen set of columns (describing attributes) is used. The row sets for each decision tree are created by bootstrapping and have the same size as the original input table. The attribute set for an individual split in a decision tree is determined by randomly selecting sqrt(m) attributes from the available attributes where m is the total number of learning columns. The attributes can also be provided as bit (fingerprint), byte, or double vector. The output model describes a random forest and is applied in the corresponding predictor nod

* **Logistic regression**

Performs a multinomial logistic regression. Select in the dialog a target column (combo box on top), i.e., the response. The solver combo box allows you to select which solver should be used for the problem (see below for details on the different solvers). The two lists in the center of the dialog allow you to include only certain columns which represent the (independent) variables.

* **Isolation forest**

Implements the isolation forest method for anomaly detection

The data is expected to have two class values for the class attribute, which is ignored at training time. The distribution For Instance () method returns the

anomaly score as the first element in the distribution, the second element is one minus this score.

To evaluate performance of this method for a dataset where anomalies are known, simply code the anomalies using the class attribute: normal cases should correspond to the second value of the class attribute, anomalies to the first one.

* **Scorer**

Compares two columns by their attribute value pairs and shows the confusion matrix, i.e., how many rows of which attribute and their classification match. Additionally, it is possible to highlight cells of this matrix to determine the underlying rows. The dialog allows you to select two columns for comparison; the values from the first selected column are represented in the confusion matrix's rows and the values from the second column by the confusion matrix's columns. The output of the node is the confusion matrix with the number of matches in each cell

* **Table view**

Displays data in an HTML table view. The view offers several interactive features, as well as the possibility to select rows.

The node supports custom CSS styling. You can simply put CSS rules into a single string and set it as a flow variable 'custom CSS' in the node configuration dialog.

* **Bar chart**

A bar chart based on the NVD3 library.

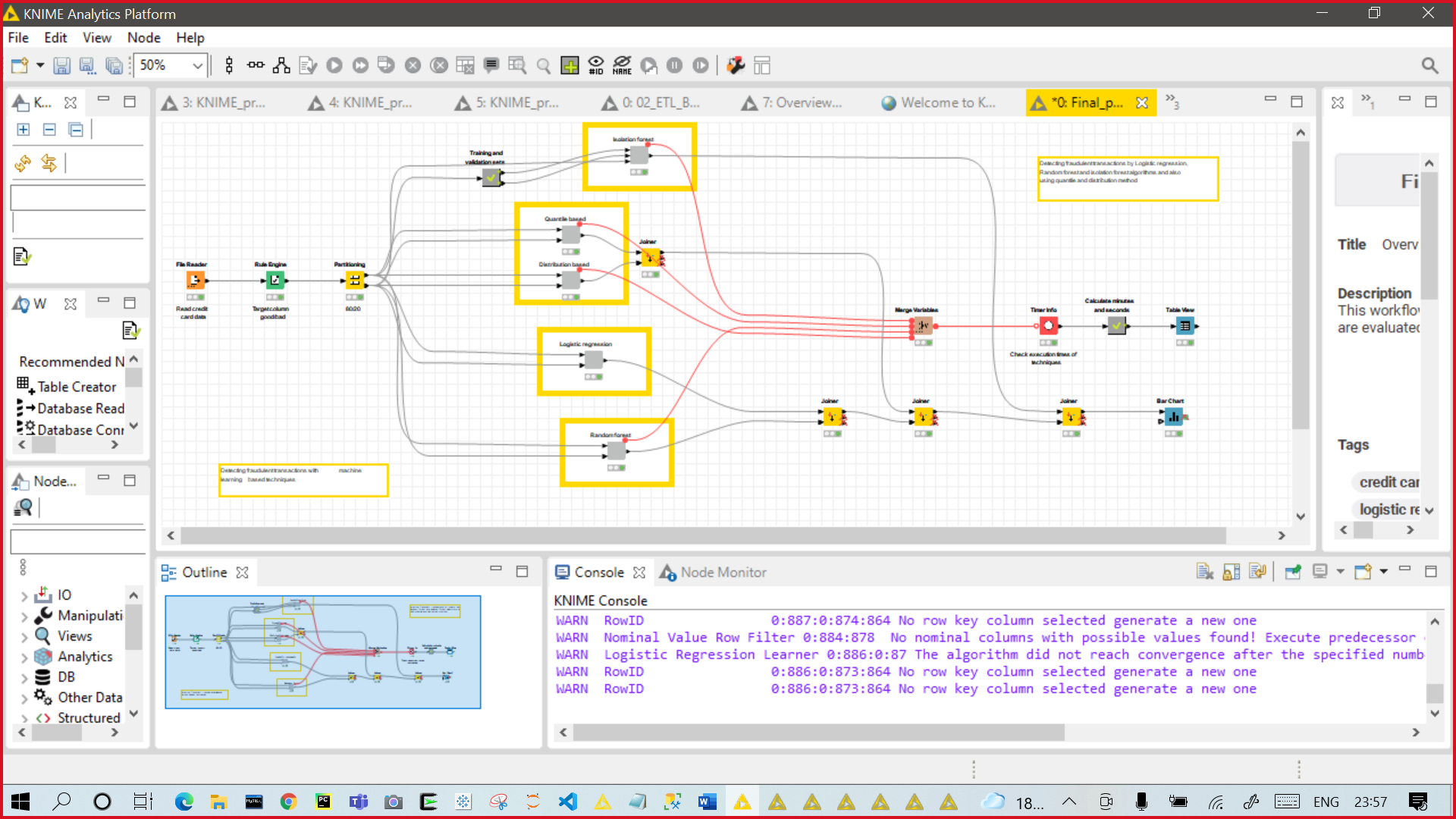
The node supports custom CSS styling. You can simply put CSS rules into a single string and set it as a flow variable 'custom CSS' in the node configuration dialog.

* **Timer Info**

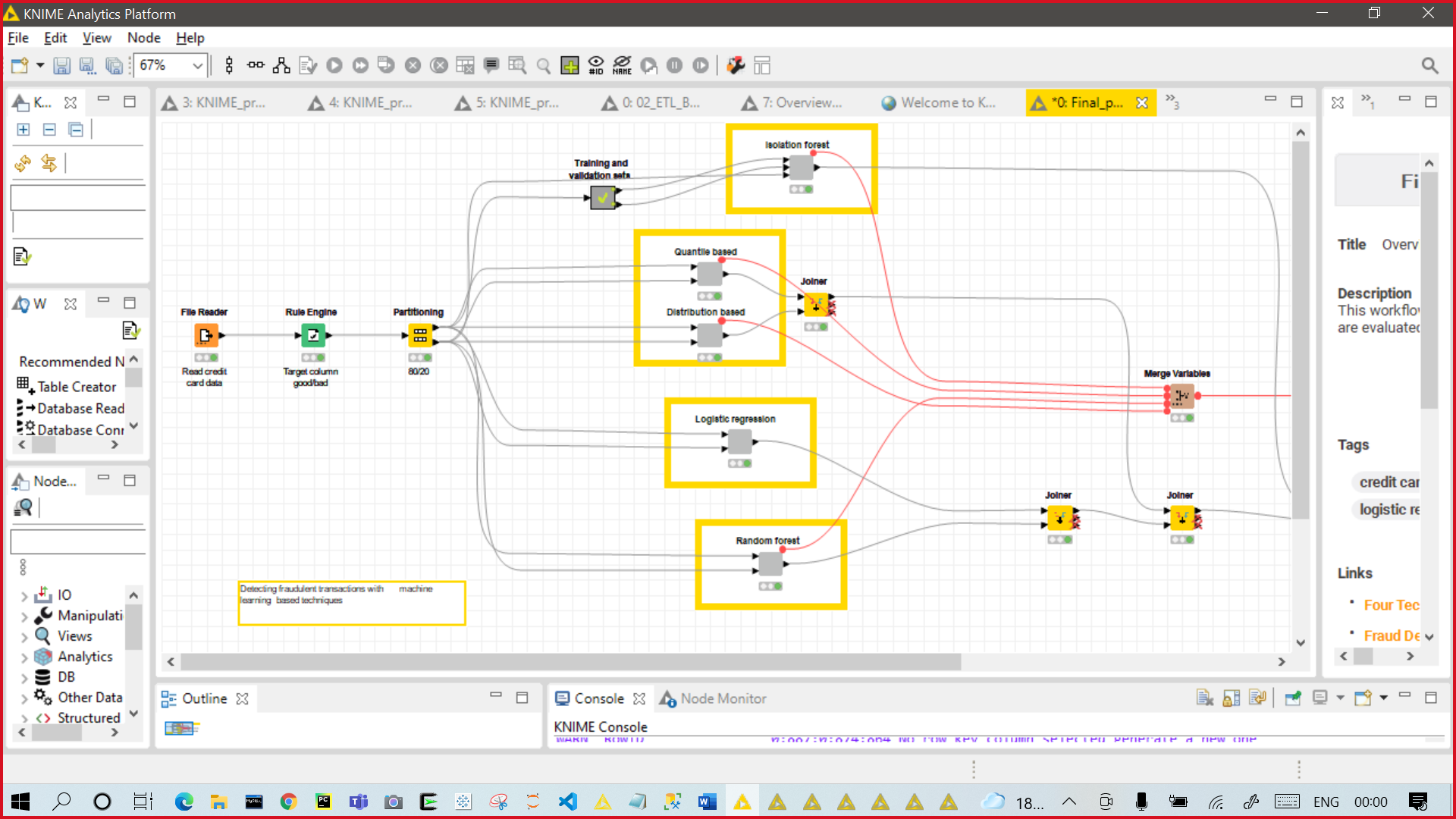
This node reports individual and aggregate timing/execution information for all nodes of the workflow at this level and for (nested) meta nodes up until the specified depth.

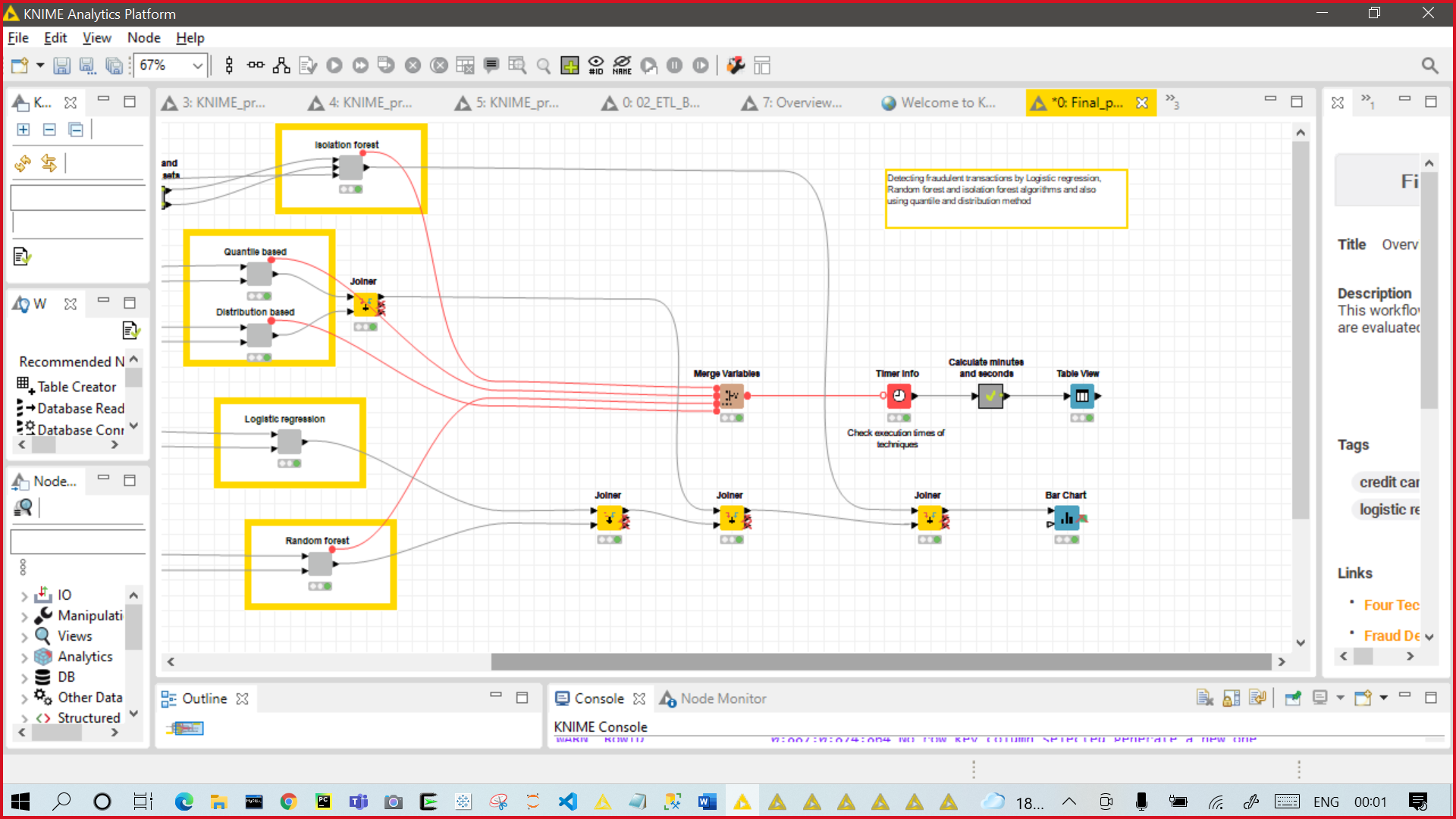
The output table lists all nodes in the workflow that were executed since the last reset. This also includes nodes in meta nodes up to the specified nesting depth.

**Workflow**



**Magnified Workflow**





**WORKFLOW EXPLANATION**

* Using File Reader node first we have read our data files.



* Then we have used Rule Engine node to filter out the Data based on specified conditions.



* After that we have used Partitioning node to split our data into two parts that is test and train data.



* Then we have used 5 nodes for 5 techniques to analyze our data which are:
* Isolation forest
* Quantile based
* Distribution based
* Logistic regression
* Random forest



* Then we have used Joiner node to club all the outputted results from the five data analyzing techniques.



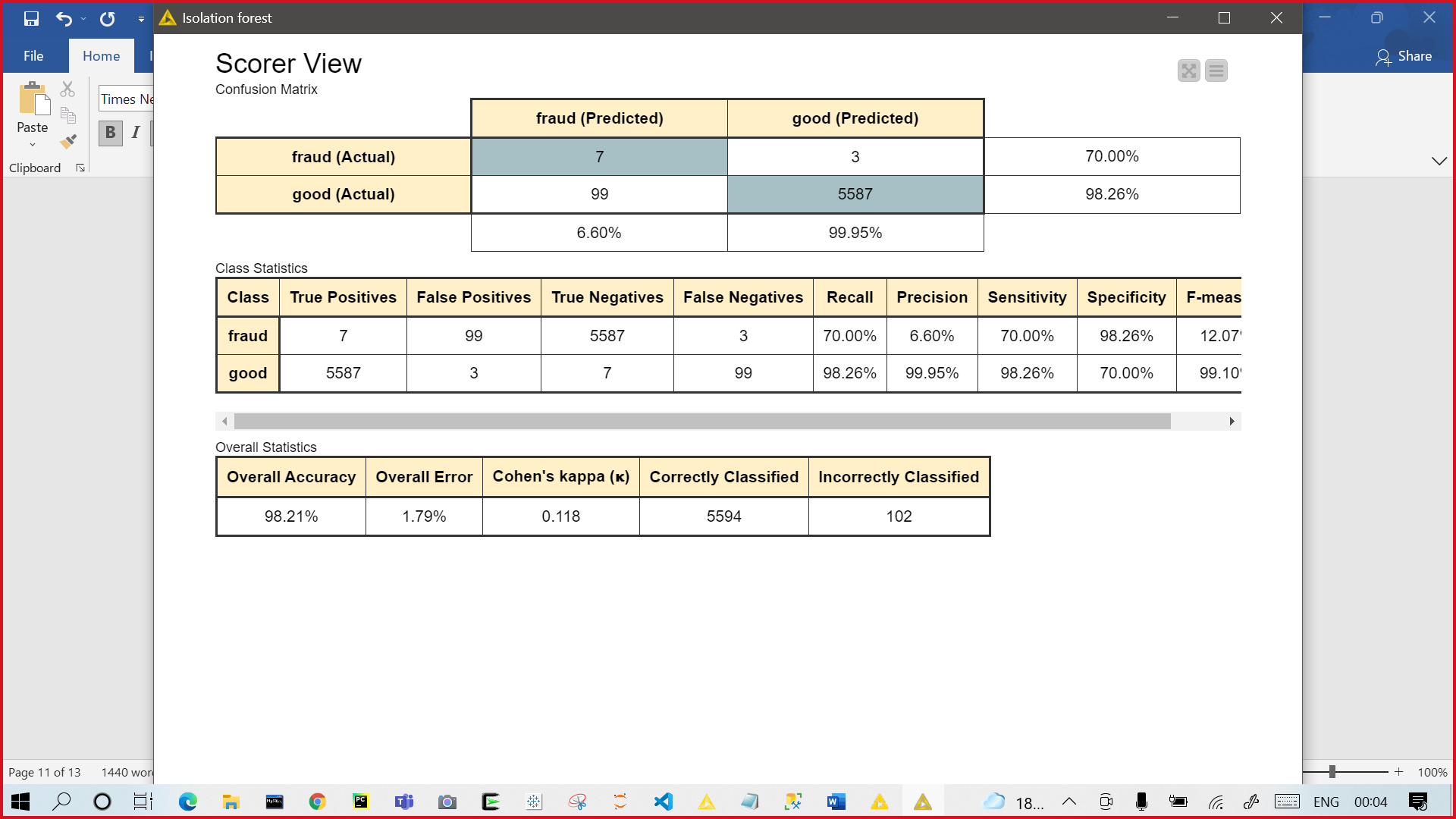
* Then we plotted a Bar graph using Bar chart node to compare our all-Fraud detection techniques.



* We also used Timer info node to calculate the time taken by these algorithmic techniques and then displayed the results using table view node.

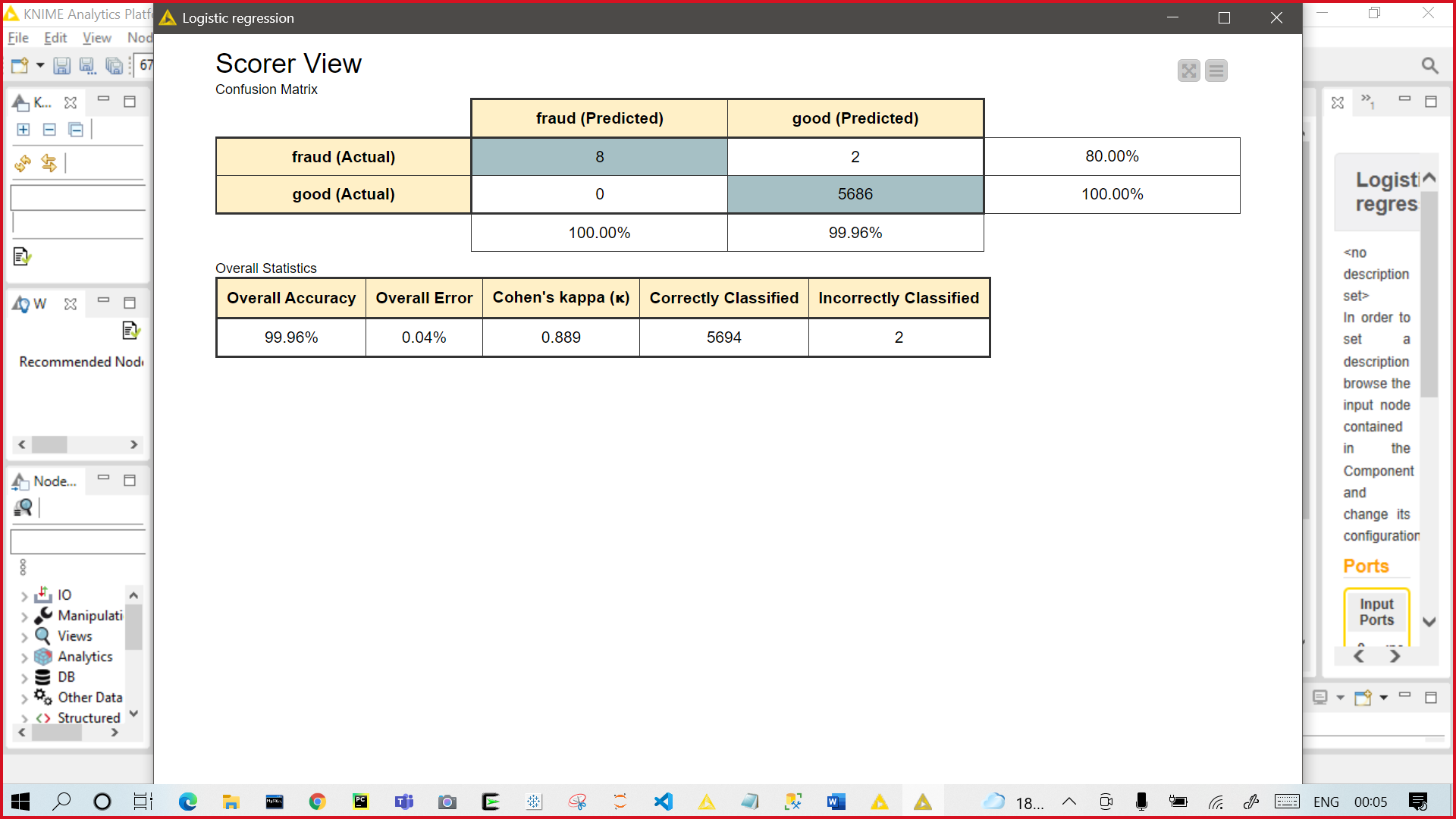
**ISOLATION FOREST technique output table**

* The scorer view mentioned below contains the confusion matrix table, class statistics table and overall statistics table.



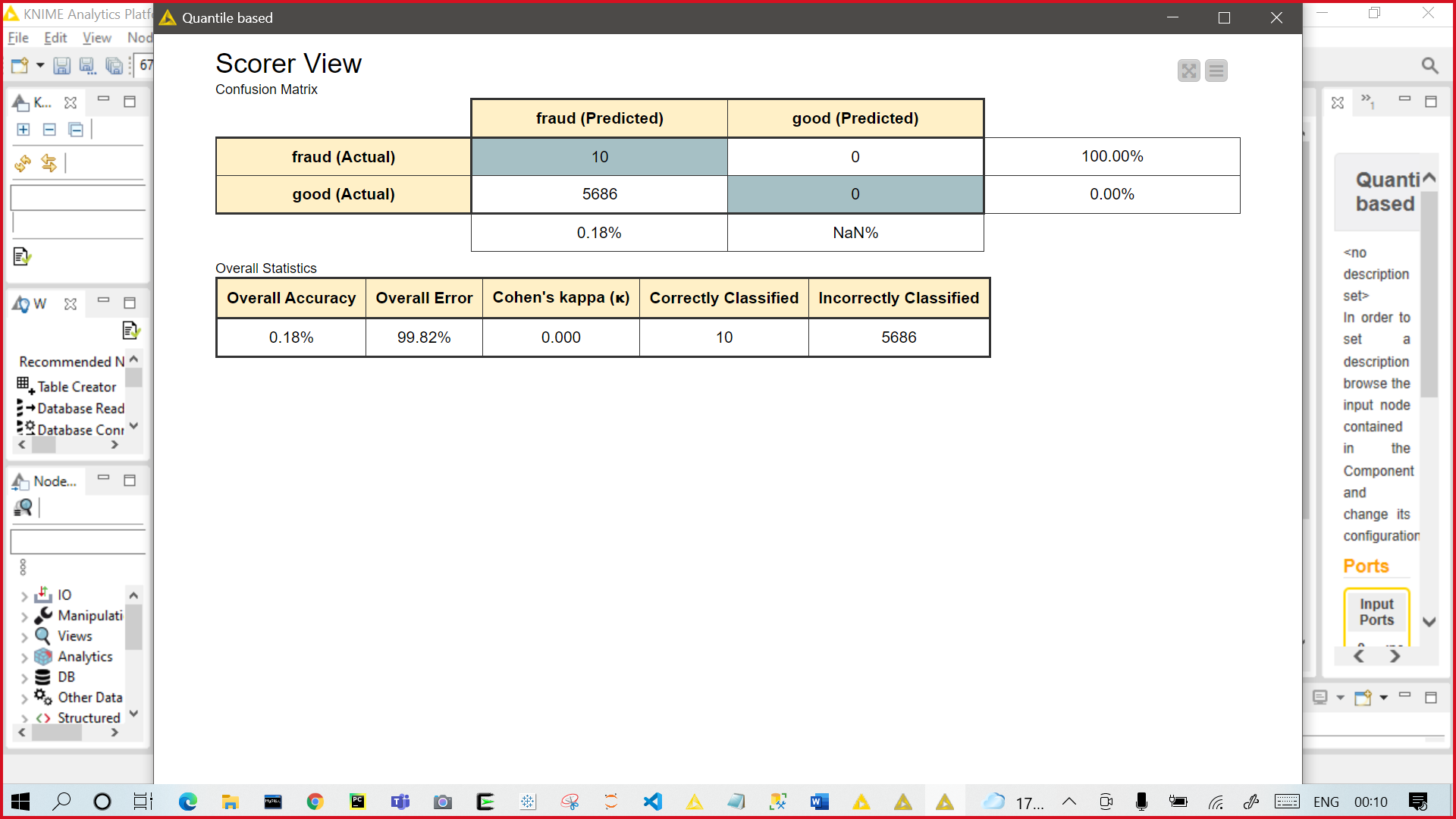
**LOGISTIC REGRESSION technique output table**

* The scorer view mentioned below contains the confusion matrix table and overall statistics table.



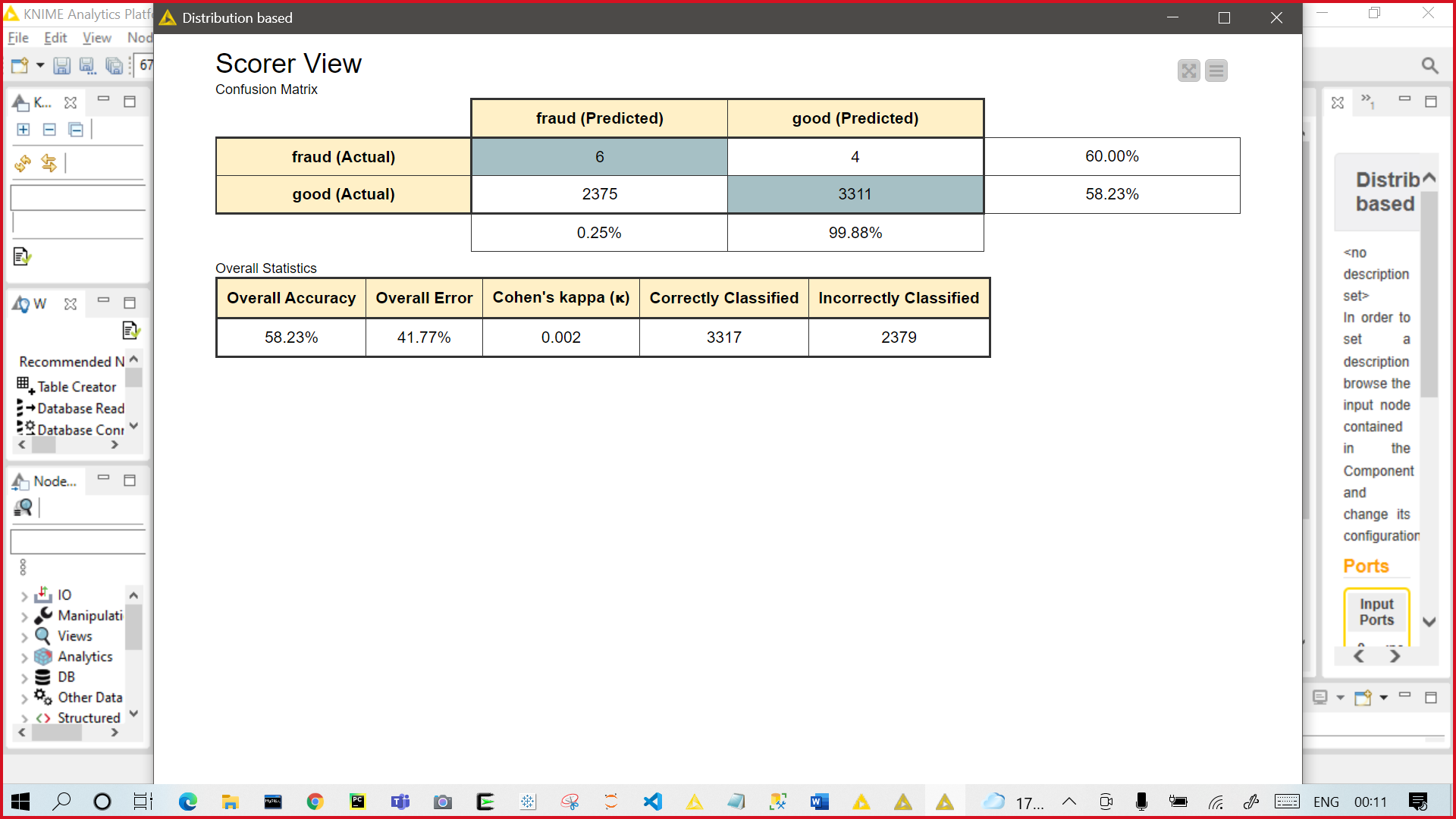
**QUANTILE BASED technique output table**

* The scorer view mentioned below contains the confusion matrix table and overall statistics table.



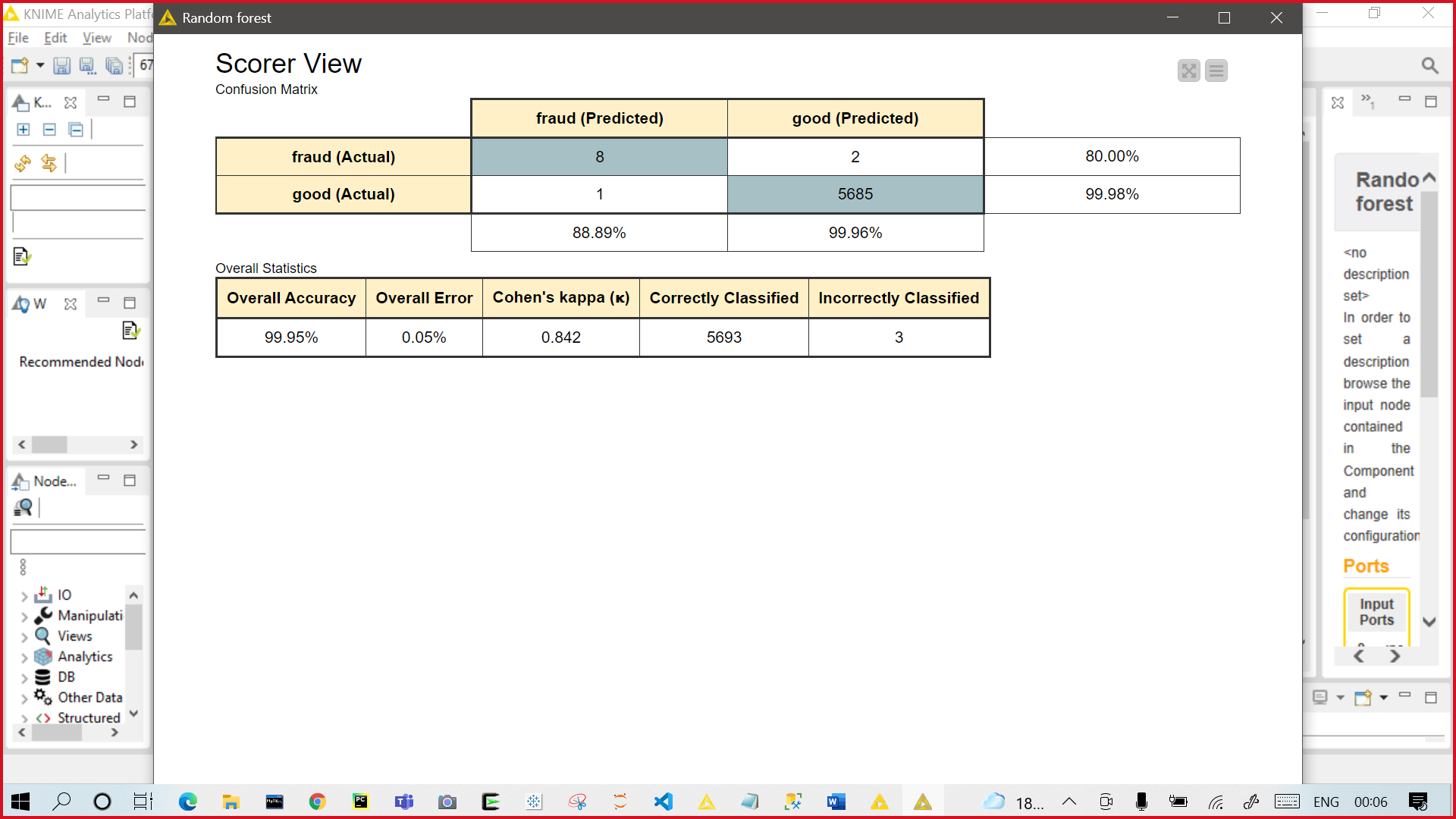
**DISTRIBUTION BASED technique output table**

* The scorer view mentioned below contains the confusion matrix table and overall statistics table.



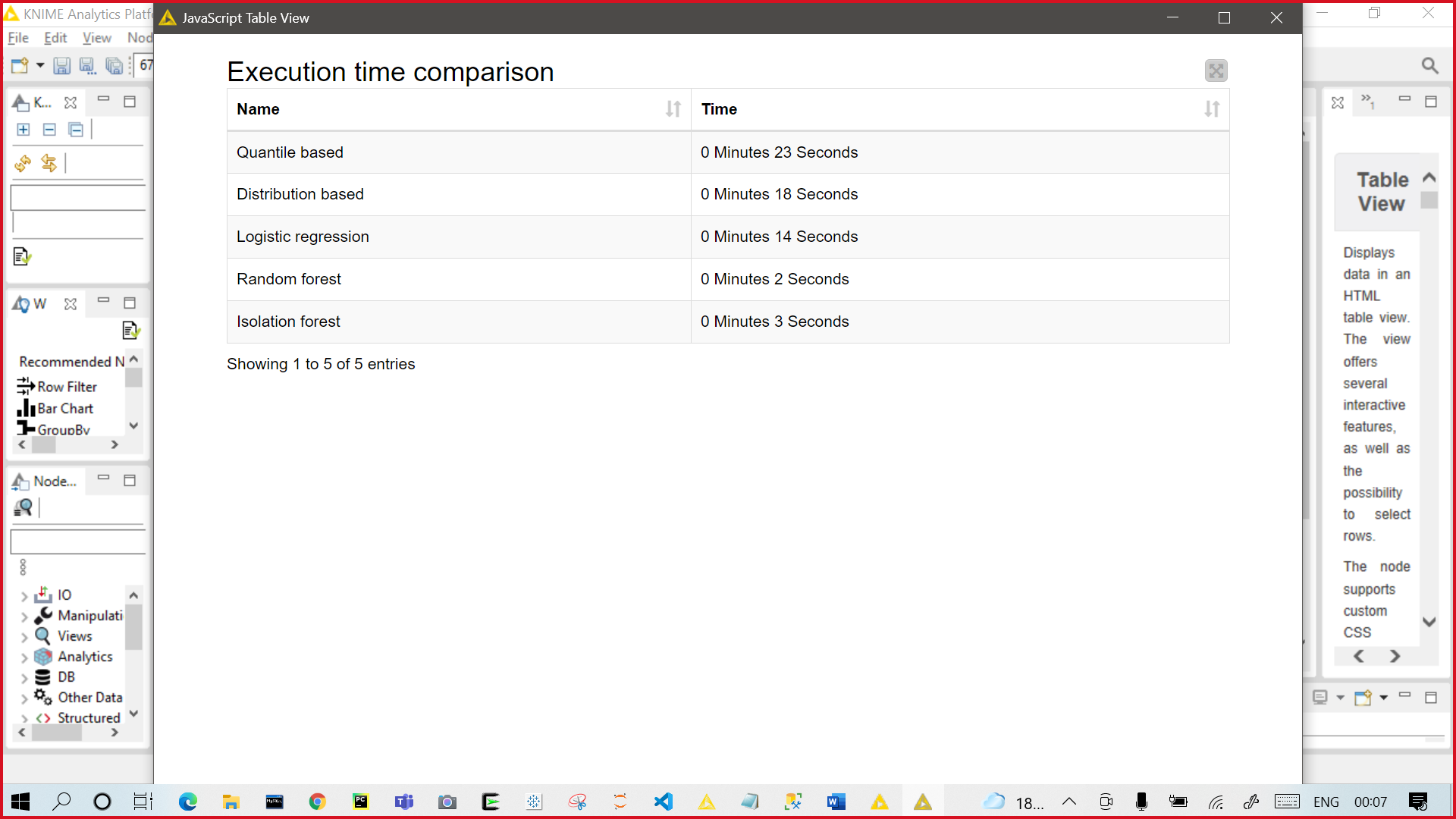
**RANDOM FOREST technique output table**

* The scorer view mentioned below contains the confusion matrix table and overall statistics table.



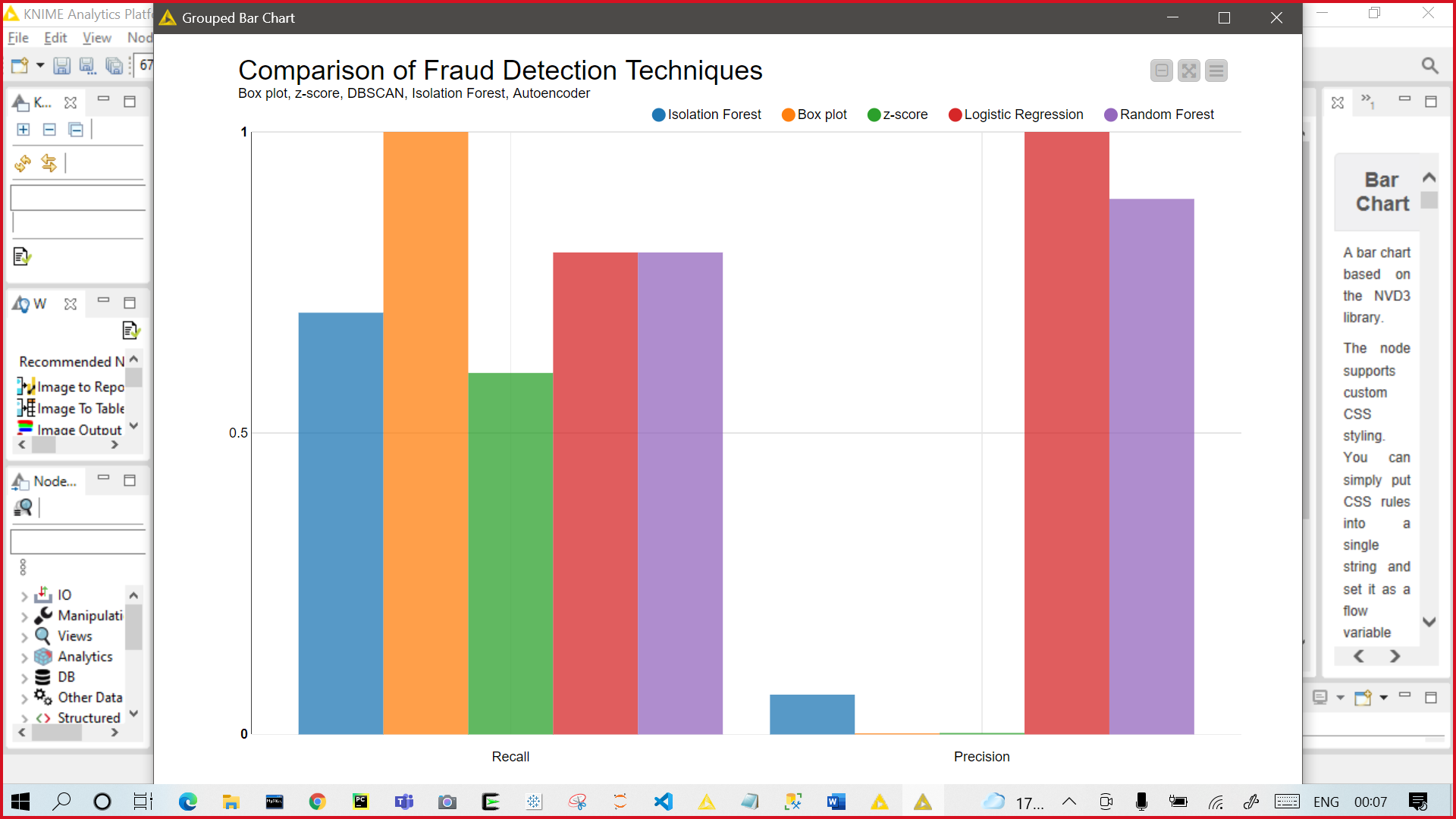
**EXECUTION TIME COMPARISON**

* The table shown below represents the execution time for all the Fraud detection techniques.



**COMPARISON OF FRAUD DETECTION TECHNIQUE**

* Using a bar graph, we have compared all the fraud detection technique as shown below:



**Conclusion**

* Credit card fraud is most common problem resulting in loss of lot money for peoples and loss for some banks and credit card company. We analyzed 5 techniques in our workflow which can efficiently separate the fraud and fraud less transaction.

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Accuracy (%)** | **Precision** | **Recall** |
| Logistic Regression | 99.89 | 0.8330 | 0.5 |
| Random Forest | 99.95 | 0.8800 | 0.8 |
| Isolation forest | 98.21 | 0.0660 | 0.7 |
| Quantile based | 99.82 | 0.0017 | 1.0 |
| Distribution based | 58.23 | 0.0025 | 0.6 |

* After analyzing the accuracy, precision and recall of all the 5 techniques mentioned above, we concluded that **Random Forest is the best technique** to detect credit card frauds with an accuracy of **99.95%.**