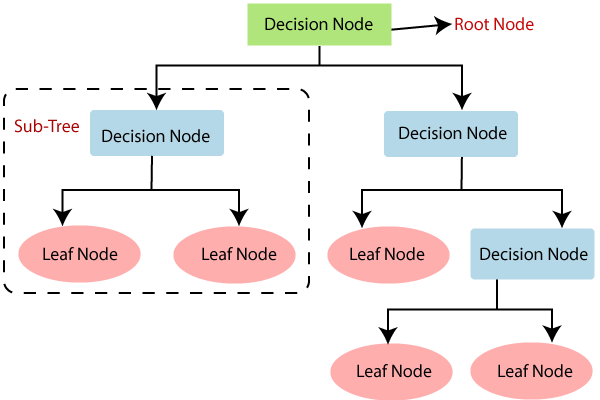
**EXPERIMENT 6**

**AIM:** To implement any one of the classification algorithms (Decision tree/Naive Bayes) /Technique using Rapid Miner, Python Library, and self-defined function.

**THEORY:**

* A Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.
* In a Decision tree, there are two nodes: the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
* The decisions or the test are performed on the basis of features of the given dataset.
* It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.
* It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
* In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.
* A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.



Decision Tree Terminologies

* Root Node: The root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
* Leaf Node: Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
* Splitting: Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
* Branch/Sub Tree: A tree formed by splitting the tree.
* Pruning: Pruning is the process of removing unwanted branches from the tree.
* Parent/Child node: The root node of the tree is called the parent node, and other nodes are called the child nodes.

Working of Decision tree algorithm:

Step-1: Begin the tree with the root node, says S, which contains the complete dataset.

Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).

Step-3: Divide the S into subsets that contain possible values for the best attributes.

Step-4: Generate the decision tree node, which contains the best attribute.

Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Advantages of the Decision Tree

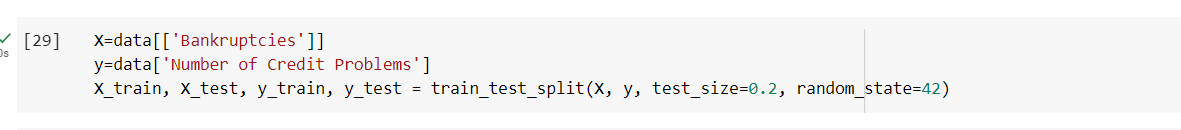
* It is simple to understand as it follows the same process which a human follows while making any decision in real life.
* It can be very useful for solving decision-related problems.
* It helps to think about all the possible outcomes of a problem.
* There is less requirement for data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

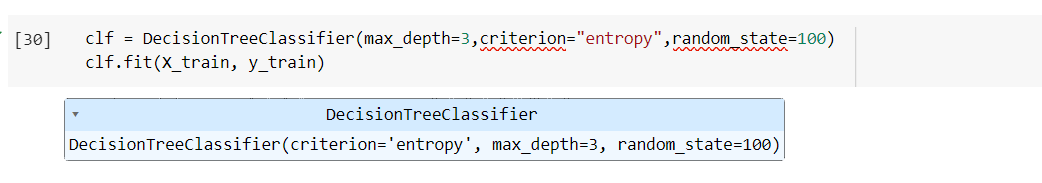
* The decision tree contains lots of layers, which makes it complex.
* It may have an overfitting issue, which can be resolved using the Random Forest algorithm.
* For more class labels, the computational complexity of the decision tree may increase.

**IMPLEMENTATION:**

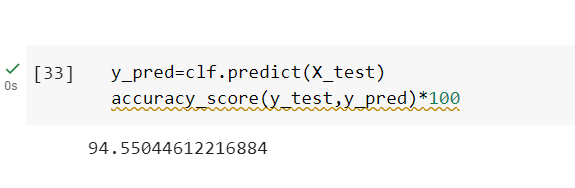
1. **USING AN IN-BUILT PYTHON FUNCTION**
2. Split data into train and test set



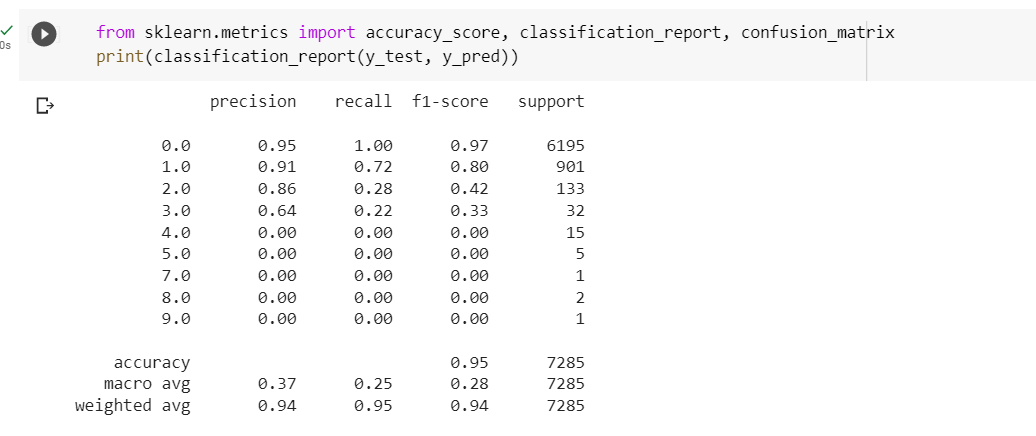
1. Building a Classification Model



1. Check accuracy



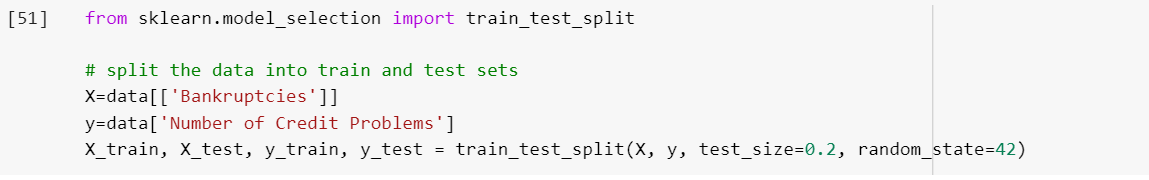
1. Calculating metrics based on test data



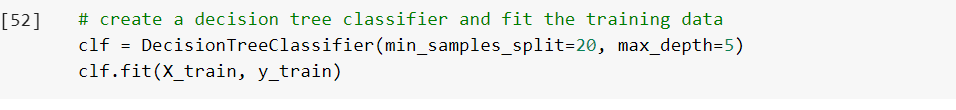
1. **USING USER-DEFINED FUNCTION**
2. User-defined function for decision tree classifier

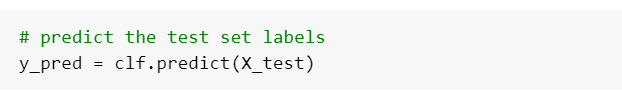
| import pandas as pd  import numpy as np  class DecisionTreeClassifier:  """  Decision Tree Classifier implementation using the Node class.    """  def \_\_init\_\_(self, min\_samples\_split=20, max\_depth=5):  self.min\_samples\_split = min\_samples\_split  self.max\_depth = max\_depth  self.tree = None  def fit(self, X, y):  """  Train the decision tree model on the input data and targets.  """  self.tree = self.\_build\_tree(X, y)    def \_build\_tree(self, X, y, depth=0):  """  Build the decision tree recursively.  """  # Create a new node and save the input data and targets  node = Node(Y=y, X=X, min\_samples\_split=self.min\_samples\_split, max\_depth=self.max\_depth, depth=depth)    # If the current node has a depth greater than the maximum depth or the number of samples in the node  # is less than the minimum number of samples for a split, we return the node as it is, as a leaf node  if node.depth >= self.max\_depth or node.n < self.min\_samples\_split:  return node  # Find the best feature and value to split the data on  best\_feature, best\_value = node.best\_split()    # If no best feature was found, we return the current node as it is, as a leaf node  if best\_feature is None:  return node    # Split the data into left and right nodes based on the best feature and value  left\_data = X[X[best\_feature] < best\_value]  left\_targets = y[X[best\_feature] < best\_value]  right\_data = X[X[best\_feature] >= best\_value]  right\_targets = y[X[best\_feature] >= best\_value]    # Recursively build the left and right subtrees  node.left = self.\_build\_tree(left\_data, left\_targets, depth=depth+1)  node.right = self.\_build\_tree(right\_data, right\_targets, depth=depth+1)    # Save the best feature and value for the current node  node.best\_feature = best\_feature  node.best\_value = best\_value    return node    def predict(self, X):  """  Predict the class labels for new input data.  """  predictions = []  for i, row in X.iterrows():  # Traverse the tree recursively to predict the class label for each row  node = self.tree  while node.left:  if row[node.best\_feature] < node.best\_value:  node = node.left  else:  node = node.right  predictions.append(node.yhat)  return np.array(predictions) |
| --- |

1. Split data into train and test set

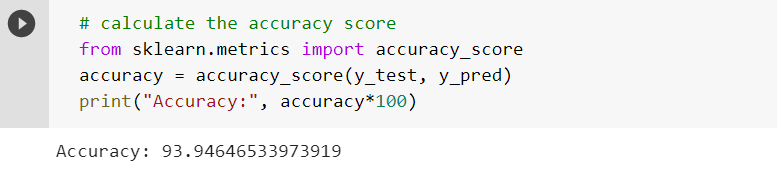


1. Building a Classification model

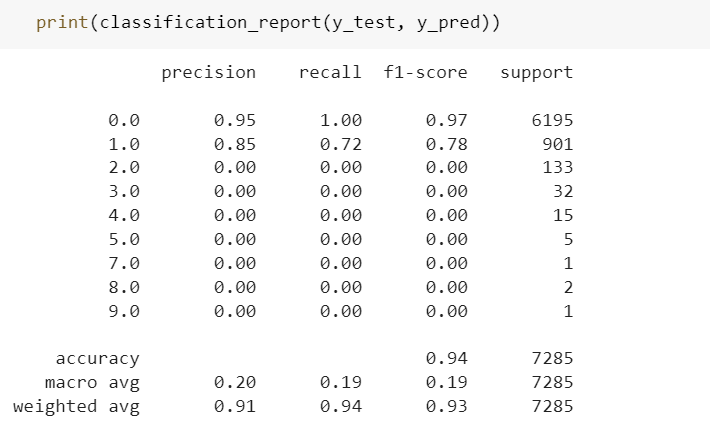




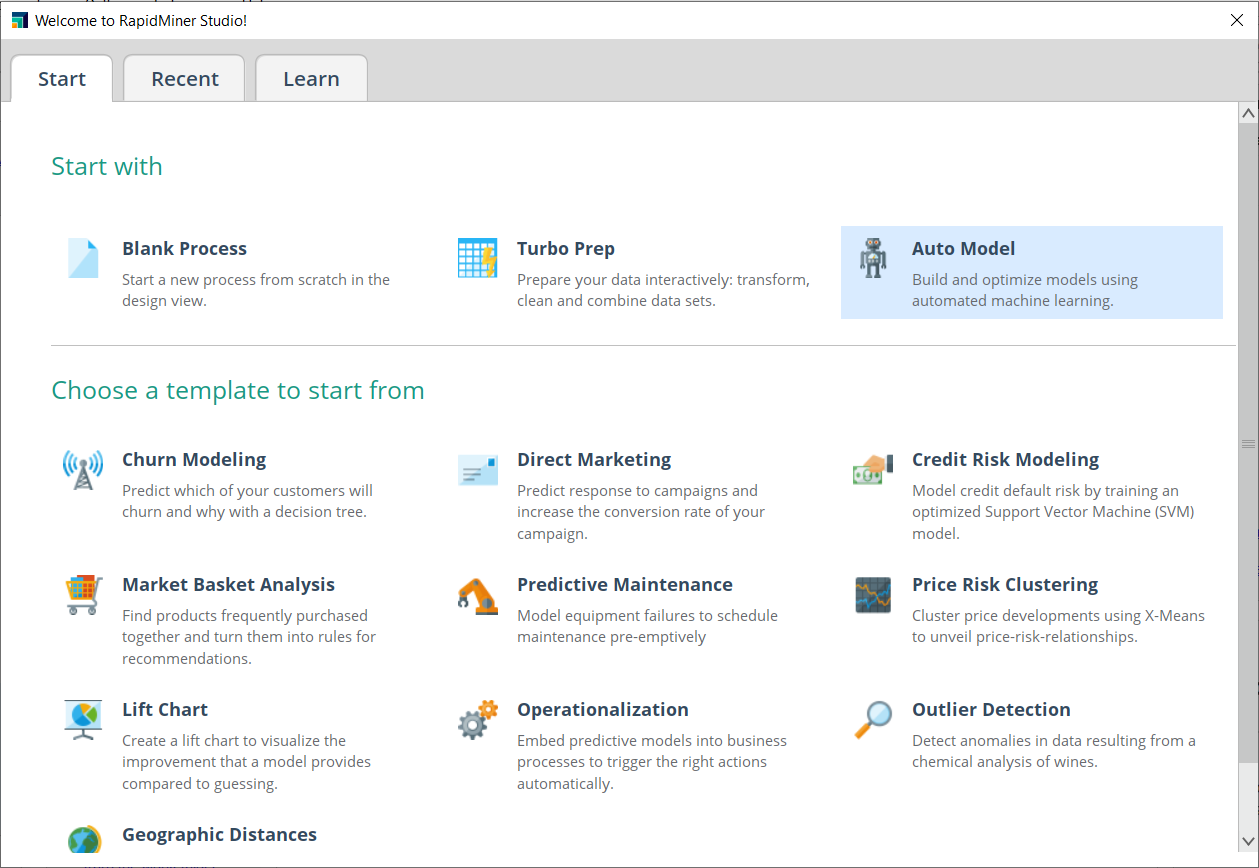
1. Check accuracy



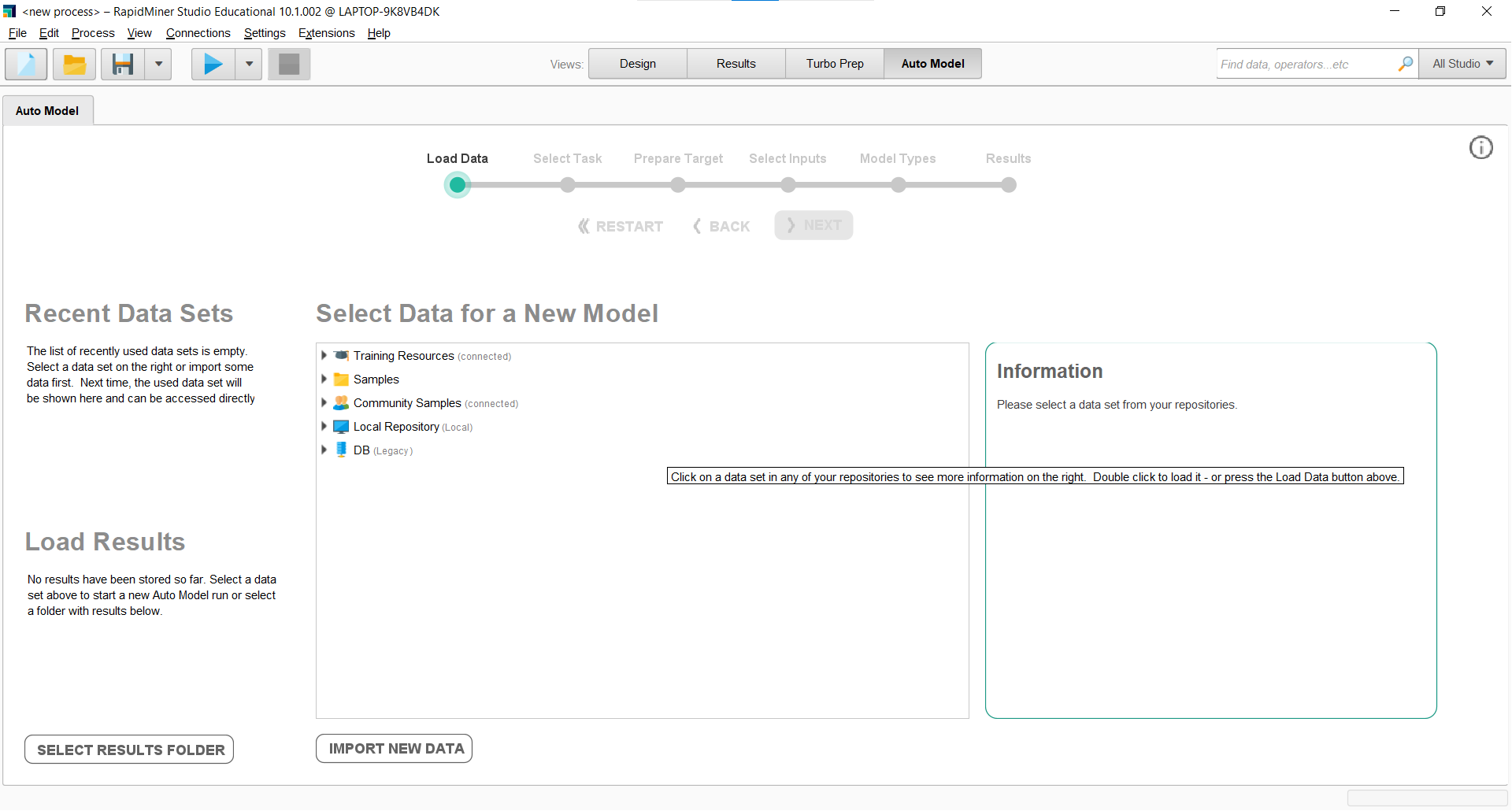
1. Calculating metrics based on test data

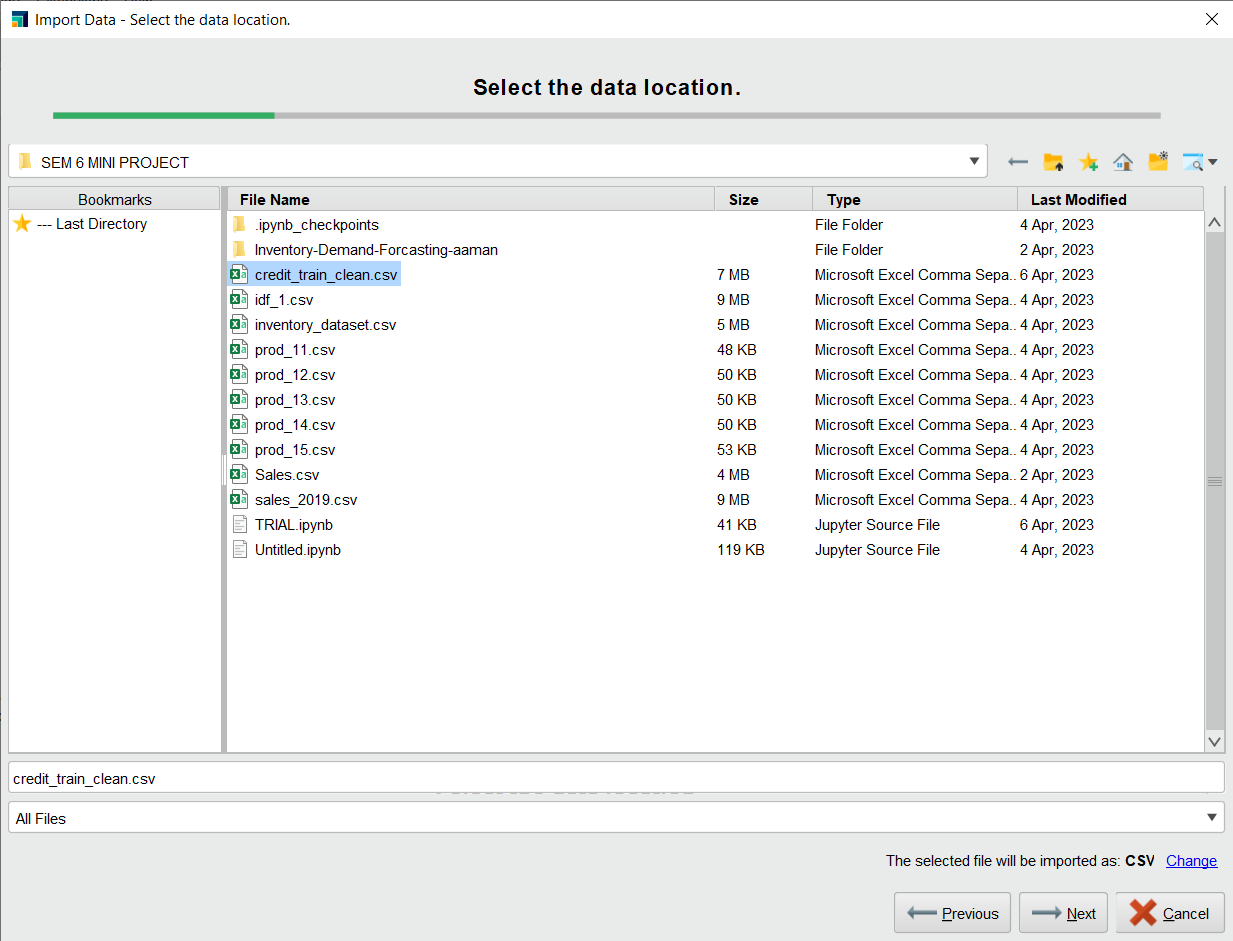


1. **USING RAPID MINER**
2. Create a new process on Rapidminer and select the ‘auto model’ option.

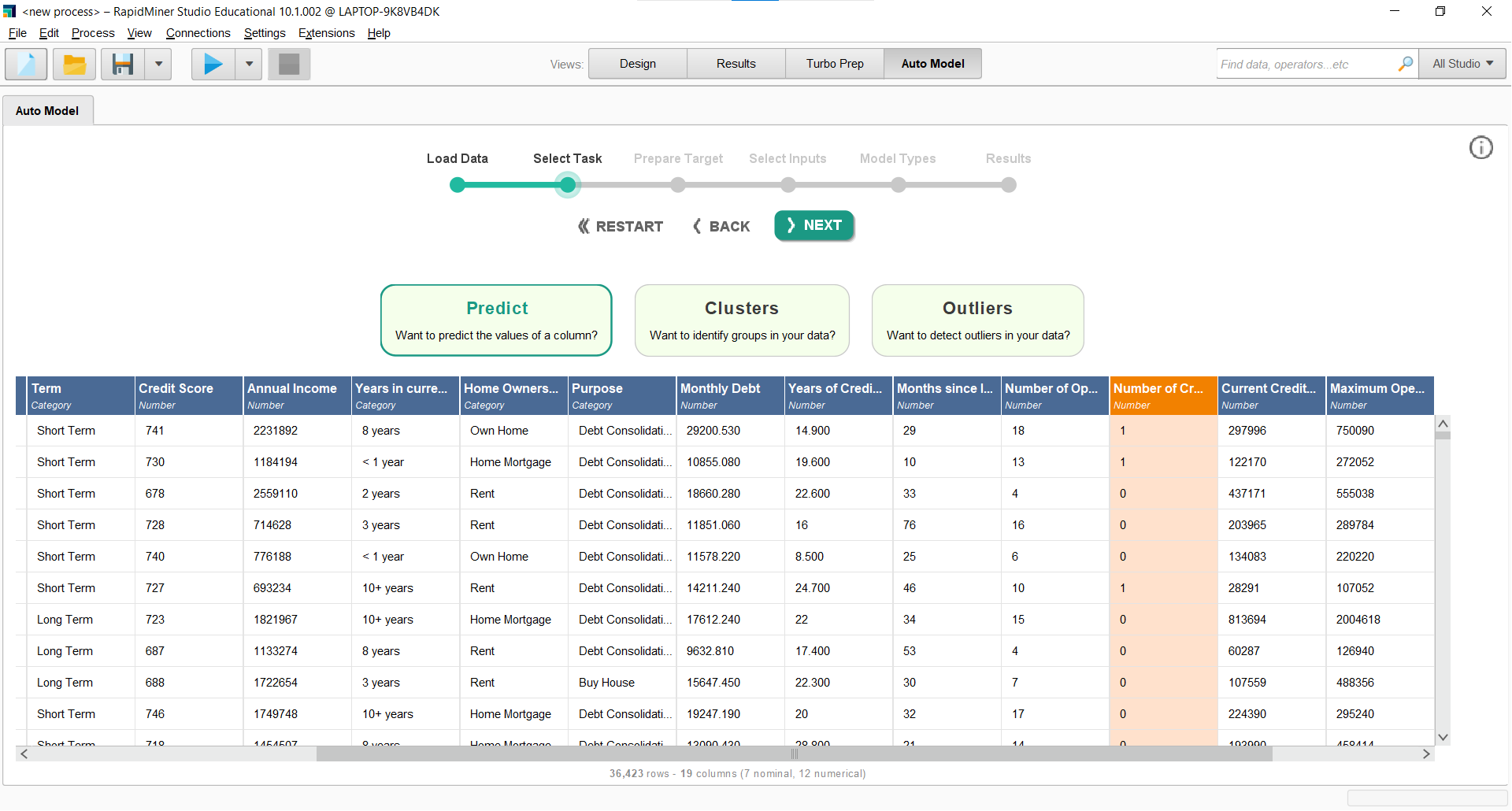


1. Import dataset

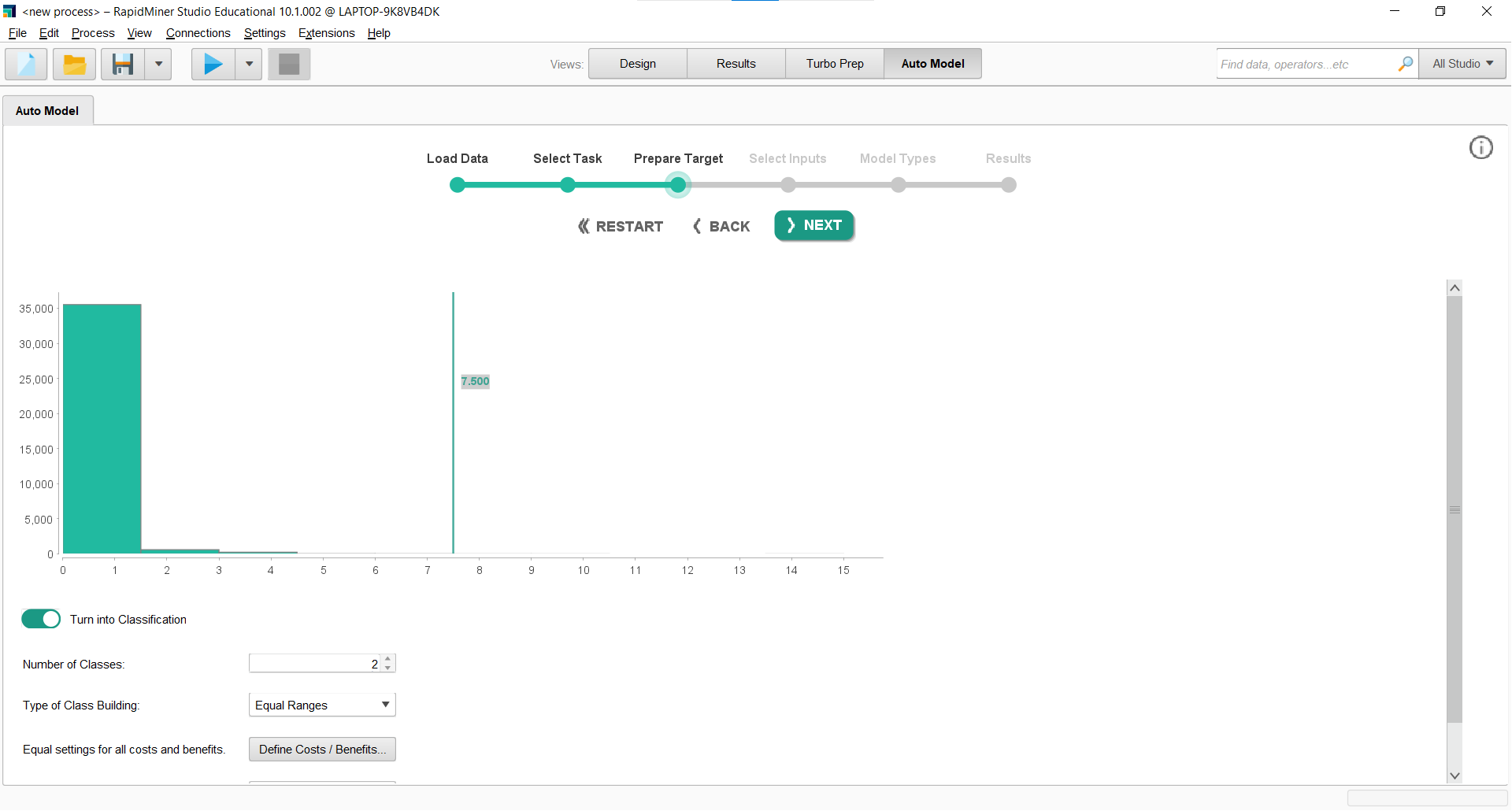




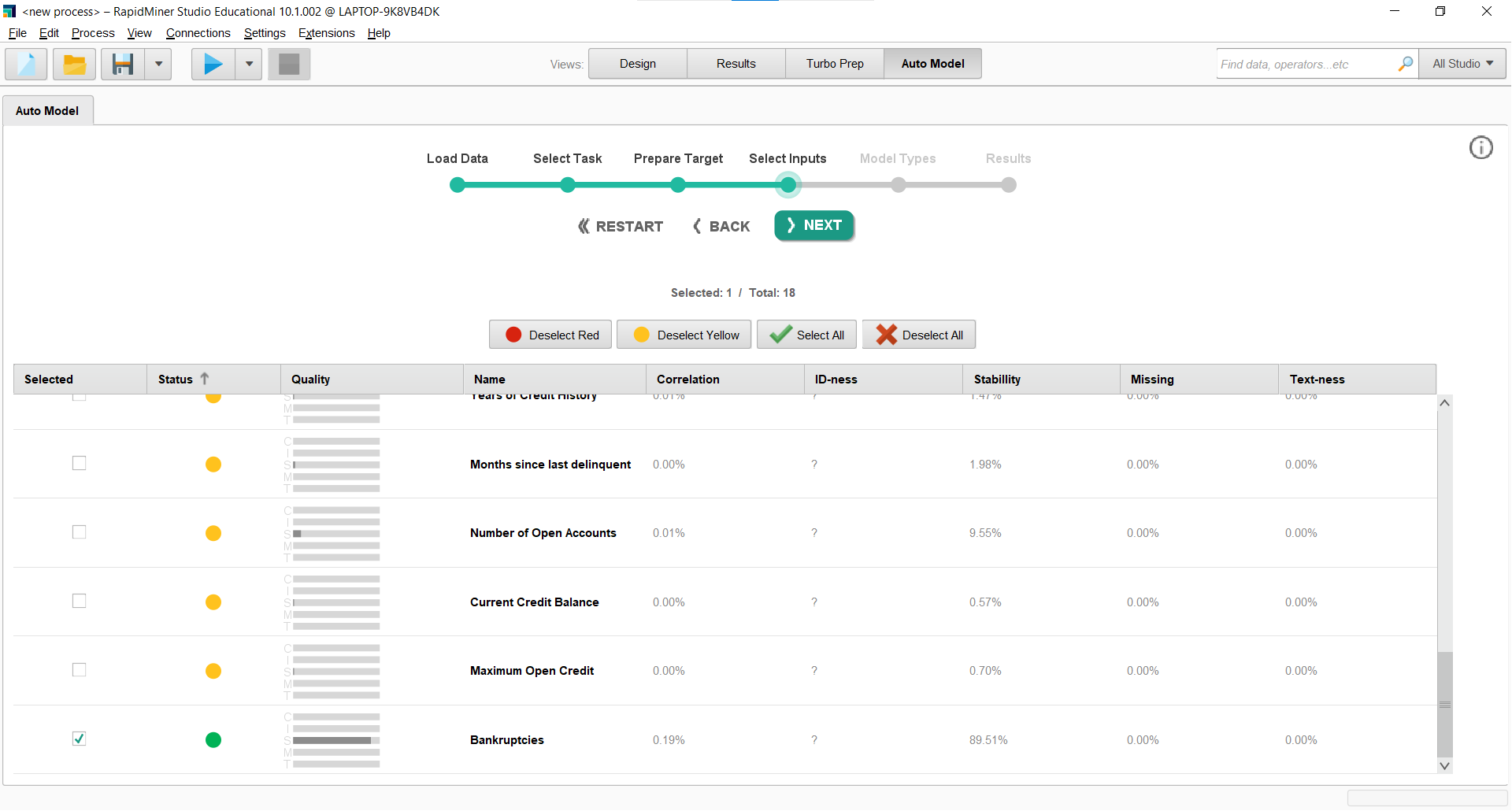
1. Select the task and target variable column



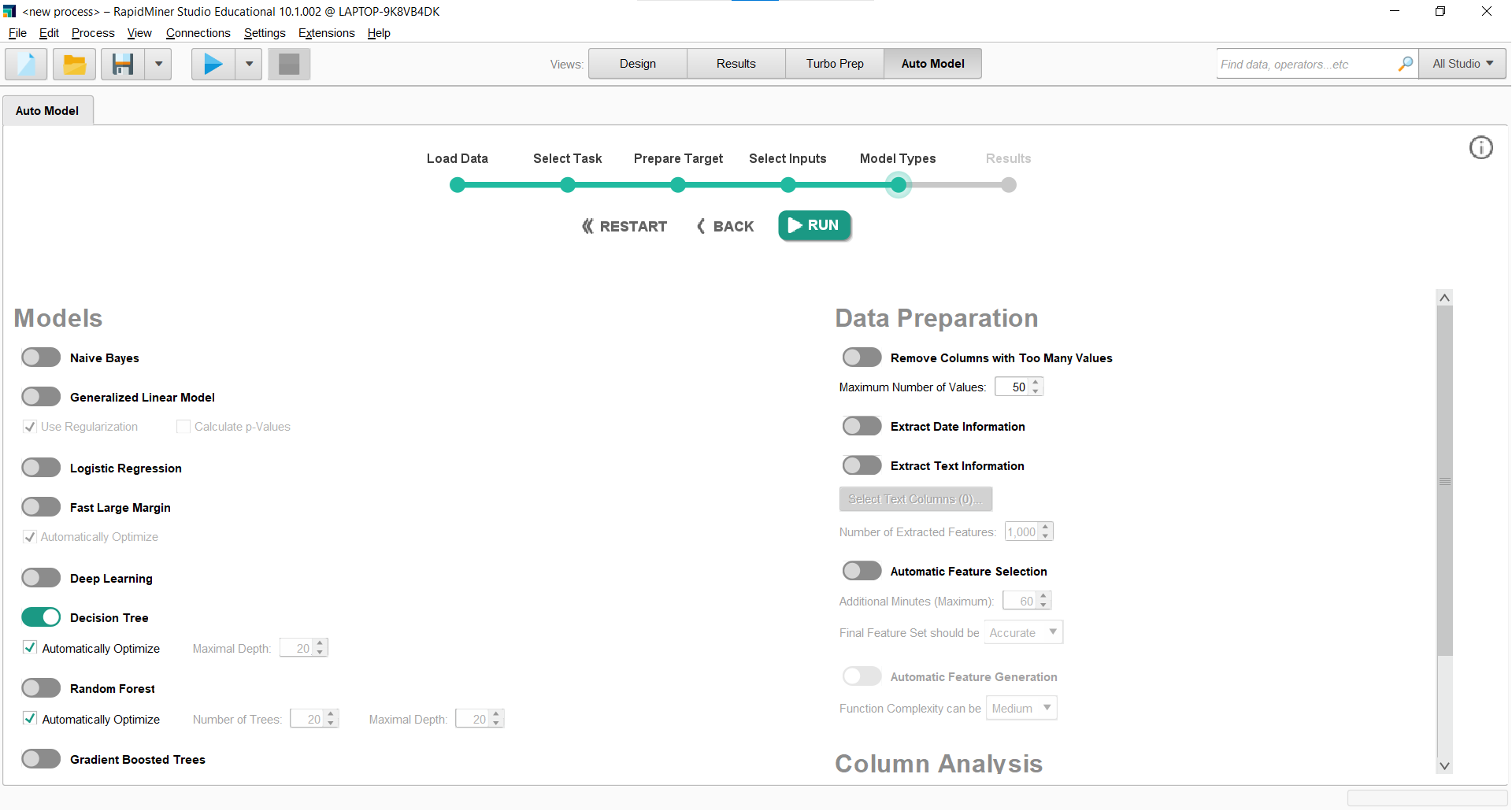
1. Turn on Classification and the select number of classes



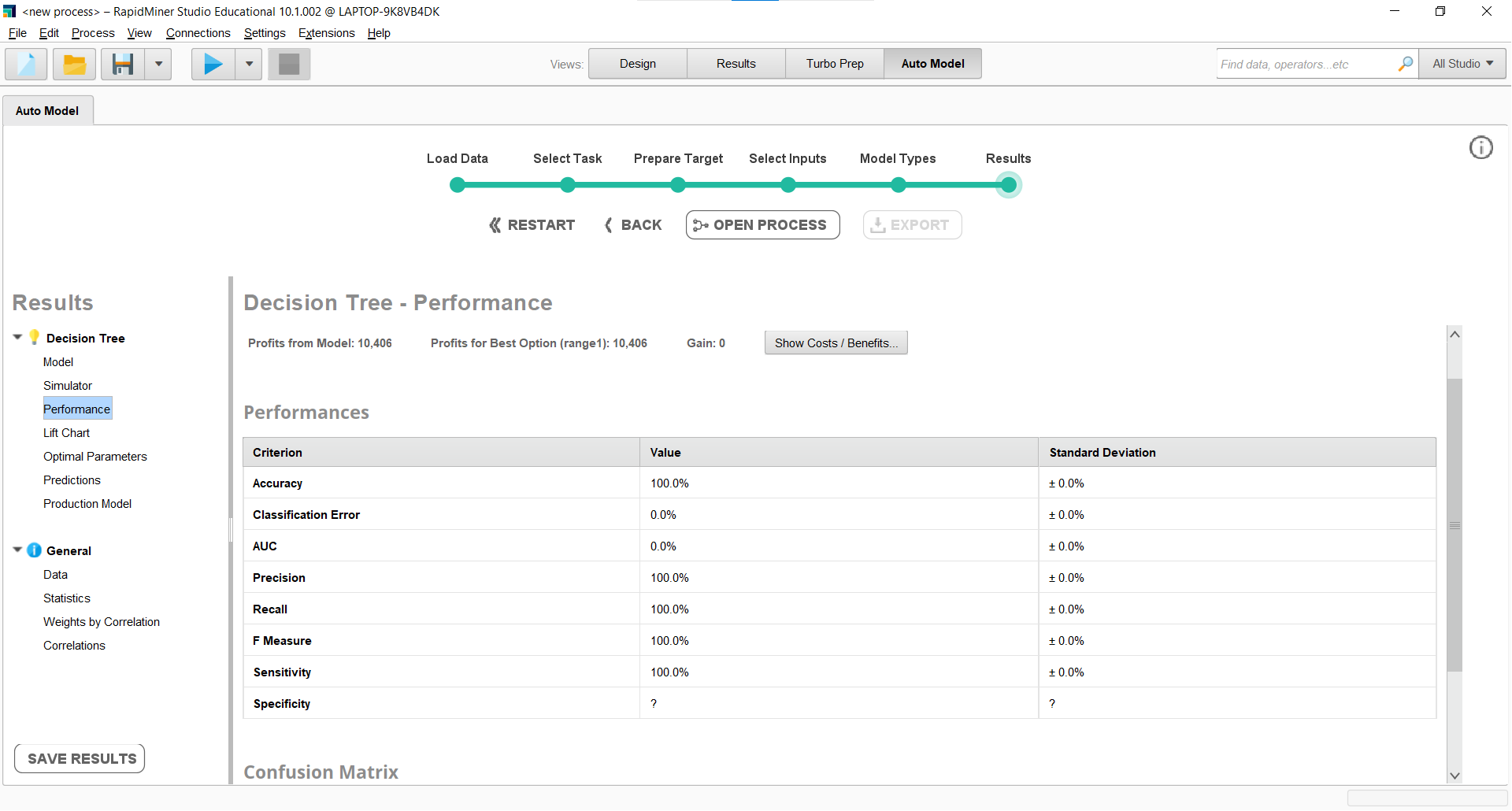
1. Select the input variable



1. Select the model



1. Check the performance of the model



**COMPARISON OF ACCURACY OBTAINED BY ALL 3 METHODS**

|  | Inbuilt python function | User-defined function | Rapid Miner |
| --- | --- | --- | --- |
| Accuracy(%) | 94.55044612216884 | 93.94646533973919 | 100 |

**CONCLUSION:**

In this experiment, we have implemented a Decision tree classifier using Rapid Miner, Python Library, and a self-defined function, and the best accuracy was obtained by the Rapid Miner, followed by an inbuilt Python function and then a user-defined function