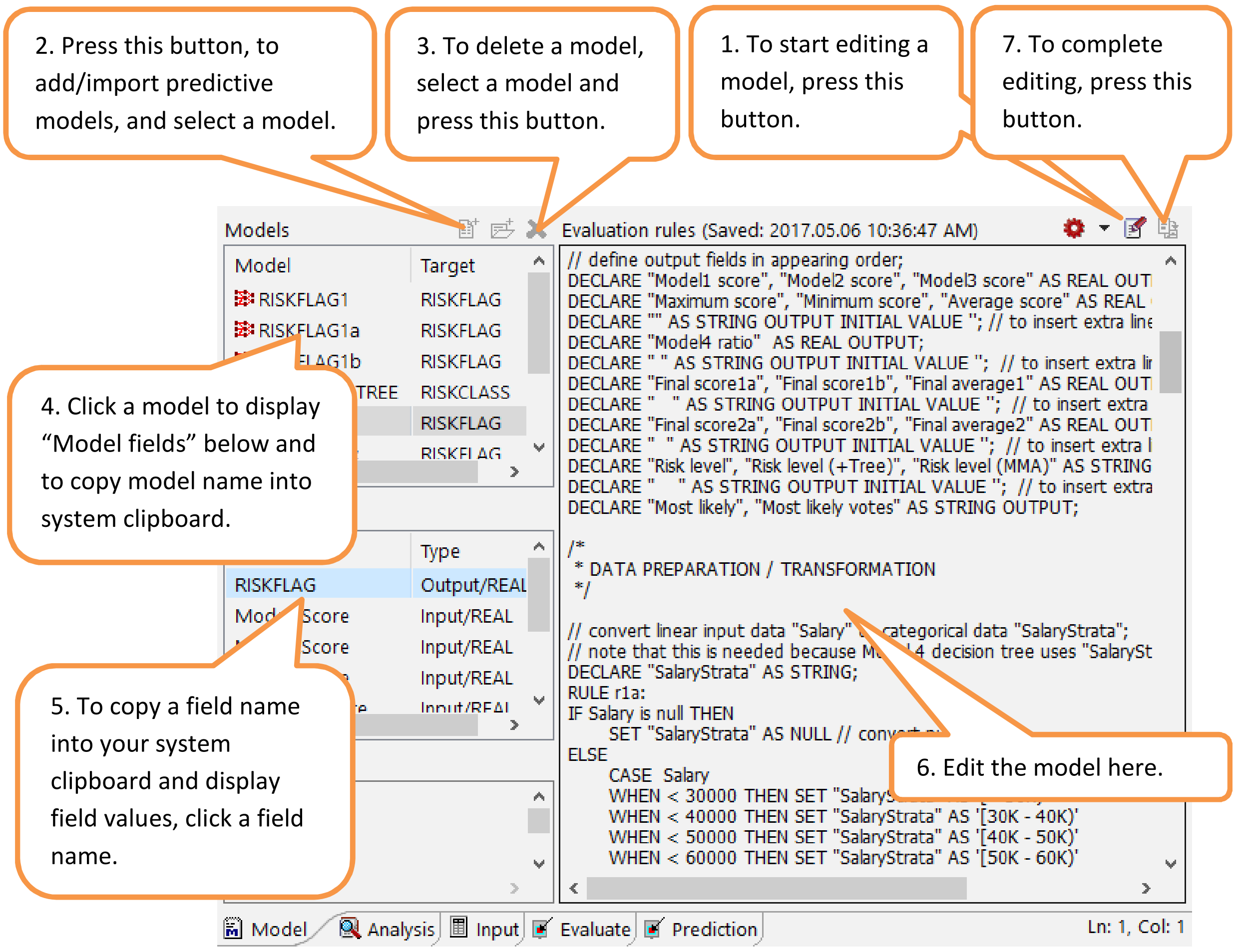
RME-EP Modeling Guide


2. Press this button, to add/import predictive models, and select a model.
3. To delete a model, select a model and press this button.
1. To start editing a model, press this button.
7. To complete editing, press this button.
4. Click a model to display "Model fields" below and to copy model name into system clipboard.
5. To copy a field name into your system clipboard and display field values, click a field name.
6. Edit the model here.
Models
Evaluation rules (Saved: 2017.05.06 10:36:47 AM)
Model
Analysis
Input
Evaluate
Prediction
Ln: 1, Col: 1

Model	Target
RISKFLAG1	RISKFLAG
RISKFLAG1a	RISKFLAG
LAG1b	RISKFLAG
TREE	RISKCLASS
	RISKFLAG
	RISKFLAG

	Type
RISKFLAG	Output/REAL
Mod Score	Input/REAL
Score	Input/REAL
	Input/REAL
	Input/REAL

```
// define output fields in appearing order;
DECLARE "Model1 score", "Model2 score", "Model3 score" AS REAL OUT
DECLARE "Maximum score", "Minimum score", "Average score" AS REAL
DECLARE "" AS STRING OUTPUT INITIAL VALUE "; // to insert extra line
DECLARE "Model4 ratio"  AS REAL OUTPUT;
DECLARE " " AS STRING OUTPUT INITIAL VALUE ";  // to insert extra li
DECLARE "Final score1a", "Final score1b", "Final average1" AS REAL OUT
DECLARE "   " AS STRING OUTPUT INITIAL VALUE ";  // to insert extra
DECLARE "Final score2a", "Final score2b", "Final average2" AS REAL OUT
DECLARE "  " AS STRING OUTPUT INITIAL VALUE ";  // to insert extra l
DECLARE "Risk level", "Risk level (+Tree)", "Risk level (MMA)" AS STRING
DECLARE "    " AS STRING OUTPUT INITIAL VALUE ";  // to insert extra
DECLARE "Most likely", "Most likely votes" AS STRING OUTPUT;

/*
 * DATA PREPARATION / TRANSFORMATION
 */

// convert linear input data "Salary"   categorical data "SalaryStrata";
// note that this is needed because M   4 decision tree uses "SalarySt
DECLARE "SalaryStrata" AS STRING;
RULE r1a:
IF Salary is null THEN
      SET "SalaryStrata" AS NULL // conv
ELSE
      CASE  Salary
      WHEN < 30000 THEN SET "SalaryS
      WHEN < 40000 THEN SET "SalaryStrata" AS '[30K - 40K)'
      WHEN < 50000 THEN SET "SalaryStrata" AS '[40K - 50K)'
      WHEN < 60000 THEN SET "SalaryStrata" AS '[50K - 60K)'
```

To create a new RME-EP model, select "New RME-EP model" of CMSR "Modeling" menu. To open an existing model, double click the model from the "Models" tab.

To edit a model, perform the sequence of procedures describe in the above figure.

After finishing editing as described in step 7, make sure to save the model!

1. Declaring Variables

Variables are defined using the following syntax;

> **DECLARE** *<fact-name>*[, *<fact-name>* …] **AS** *<data-type>*
> [**GLOBAL** | **LOCAL**]
> [**INPUT** | **OUTPUT** | **INOUT**]
> [**INITIAL** [**VALUE**] *<initial-value>*] ;
> **[OPTIONAL]**
> [**MANDATORY**]
> [**VALUES IN** <*model-field-name*> **OF** <*model-name*> |
> **VALUES** (<*value*> [,<*value*>, ...])]
> [**TEMPLATE** [**VALUE**] *<template-value>*] ;
> [**CHECK BETWEEN** *<lowest-value>* **AND** *<highest-value>*]

Fact (=variable=field names) can be any non-reserved word alpha-numeric names or any double-quote enclosed expressions. Alpha-numeric names may include '.', '_', etc. For example, A10, A10.abc, a10_abc, "COUNT", "My Day", etc.

Data type can be STRING, INTEGER, REAL, BOOLEAN, DATE, TIMESTAMP and TIME. INPUT, OUTPUT, INOUT are specified for user interfaces. INPUT can be from user input or from database tables. OUTPUT is to print result values. INOUT is for both input and output. Note that input and output variables in user interface will appear following the order of definition. INITIAL VALUE specifies initial values.

OPTIONAL is used for output variables. When used, variables will not appear in simple output report.

Other options are for input variables. They are used by applications that accept input values, such as from MyDataSay Android app and web server. MANDATORY is specified to disallow empty null value input. VALUES expression specifies a list of possible input values. "**VALUES IN** <*model-field-name*> **OF** <*model-name*>" derives from a predictive model input field automatically. "**VALUES** (<*value*> [,<*value*>, ...])" enumerates ALL possible values. TEMPLATE value is to provide initial value for input interface. CHECK is used to limit input value ranges for numerical variables.

For example, fields can be defined as follows;

```
DECLARE gender AS STRING INPUT MANDATORY VALUES ('Male', 'Female');
DECLARE gender AS STRING INPUT VALUES IN "Gender" OF "Model1";
DECLARE flag1, flag2, flag3 AS REAL INPUT VALUES (0, 0.5, 1);
DECLARE Count1 AS INTEGER INITIAL VALUE 0;
DECLARE Score1, Score2, Score3 AS REAL OUTPUT;
DECLARE Date1 AS DATE INPUT TEMPLATE VALUE 'y-m-d';
DECLARE "Year" AS INTEGER INPUT CHECK BETWEEN 1900 AND 2020;
```

2. Literals

Literal values are as follows;

1. Integers: 0, 1, 2, -1, -2, …
2. Reals: 0.0, 1.24, -3.15, 1.2536E-7, -1.5367E7, …
3. Strings: 'Male', 'Female', 'F', 'M', …

3. Comments

There are two types of comments. Line comments start with // letters. After // all letters to the end of the line are considered as comments. Block comments are expressed with /* and */. It can spread multiple lines. Followings show examples of comments;

```
IF TRUE THEN // this is a line comment.
/*
  This is a block comment.
 */
```

4. Declaring Rules with IF-THEN-ELSE-END

Simple rules are defined using the following syntax;

```
RULE <rule-id>:
     IF <boolean-expression>
         THEN <consequence-expression>
       [ ELSE <consequence-expression> ]
     END;
```

<*rule-id*> can be an integer number or alpha-numeric identifier or quoted identifiers. For example, 1, 2, 3, R10, “Rule 10”, etc.

<*boolean-expression*> specified conditions. For example, TRUE, Age > 10, “Age” < 20 AND “Weight” > 100, Age < 20 OR Age > 60, etc.

<*consequence-expression*> can be variable assignment operation as follows;

SET [NOFIRE] <*variable-name*> **AS** <*value-expression*>

For example;

```
SET Gender AS 'Male'
SET Score AS MAX(v1, v2, v3, v4)
SET NOFIRE message AS message + 'Append this.'
```

<*consequence-expression*> can contain multiple operations using “{ …; …; …;}” expression. The followings are example rule definitions containing variable assignments. Notice the “RULE 2” having “TRUE” as Boolean expression. It is often used to specify rules that have to be fired without conditions.

```
RULE 1: // single assignment
  IF Age < 20 THEN
      SET Status AS 'Ok'
 END;

RULE 2: // multiple assignments
  IF TRUE THEN
    {
      SET Score1 AS MAX(v1, v2, v3, v4);
      SET Score2 AS MIN(v1, v2, v3, v4);
    }
 END;
```

5. Declaring Rules with CASE-WHEN-END

There are two types of rules with CASE-END blocks. The first takes the following format;

```
RULE <rule-id>:
    CASE
        WHEN <boolean-expression> THEN <consequence-expression>
        WHEN <boolean-expression> THEN <consequence-expression>
          …
      [ ELSE <consequence-expression> ]
    END;
```

For example, the following is an example of this format;

```
RULE 3:
    CASE
        WHEN Age < 20 THEN  SET Status AS 'Under 20'
        WHEN Age < 30 THEN  SET Status AS '20s'
        WHEN Age < 40 THEN  SET Status AS '30s'
        ELSE  SET Status AS '40 or over'
    END;
```

The second format takes the following form;

```
RULE <rule-id>:
    CASE <value-expression-1>
        WHEN <op> <value-expression-2> THEN <consequence-expression>
        WHEN <op> <value-expression-2> THEN <consequence-expression>
          …
      [ ELSE <consequence-expression> ]
    END;
```

<*op*> is comparison operators such as “=”, “<”, “>”, “<=”, “>=”, etc. <*value-expression-1*> is compared to <*value-expression-2*> for testing conditions. With this format, “RULE 3” example can be rewritten as follows;

```
RULE 4:
    CASE  Age
        WHEN < 20 THEN  SET Status AS 'Under 20'
        WHEN < 30 THEN  SET Status AS '20s'
        WHEN < 40 THEN  SET Status AS '30s'
        ELSE  SET Status AS '40 or over'
    END;
```

6. Nesting Conditions

IF-THEN-ELSE-END and CASE-WHEN-ELSE-END blocks can be nested to a number of times. The following rule is an example that uses nesting of IF-THEN-ELSE-END and CASE-WHEN-ELSE-END blocks;

```
RULE 5:
  IF Gender = 'Male' THEN
      CASE
        WHEN Age < 20 THEN  SET Status AS 'Under 20 male'
        WHEN Age < 30 THEN  SET Status AS '20s male'
        WHEN Age < 40 THEN  SET Status AS '30s male'
        ELSE  SET Status AS '40 or over male'
      END

  ELSE
      CASE  Age
        WHEN < 20 THEN  SET Status AS 'Under 20 female'
        WHEN < 30 THEN  SET Status AS '20s female'
        WHEN < 40 THEN  SET Status AS '30s female'
        ELSE  SET Status AS '40 or over female'
      END

  END;
```

The following rules are more examples for nested conditions. They transform numerical data "Salary" into categorical items "SalaryStrata";

```
RULE 6:
  IF Salary is null THEN
        SET SalaryStrata AS NULL  // if null, assign null;
  ELSE
      CASE
        WHEN Salary < 30000 THEN  SET SalaryStrata AS '[-30K]'
        WHEN Salary < 40000 THEN  SET SalaryStrata AS '[30K-40K)'
        WHEN Salary < 50000 THEN  SET SalaryStrata AS '[40K-50K)'
        ELSE     SET SalaryStrata AS '[50k-]'
      END

  END;

RULE r6a:   // This does the same as the above rule 6.
  IF Salary is null THEN
        SET SalaryStrata AS NULL  // if null, assign null;
  ELSE
      SET SalaryStrata AS
          CASE
              WHEN Salary < 30000 THEN  '[-30K]'
              WHEN Salary < 40000 THEN  '[30K-40K)'
              WHEN Salary < 50000 THEN  '[40K-50K)'
              ELSE    '[50k-]'
          END

  END;
```

7. Predictive Models

<*value-expression*> can contain predictive model outputs using the following syntax;

> **PREDICT** (<*model-name*>)
>
> **USING** (<*comma-separated-argument-list*>)

"MODEL" may be used synonymously in place of "PREDICT". The comma-separated argument list of USING blocks consists of the following expression. The first part is the name of a predictive model input field. The second part is any valid SQL value expressions that match the data type of the input field.

> <*model-field-name*> **AS** <*value-expression*>

For example, the following defines three input fields. The value of the fact named as "cust.age" is assigned to the model input field "Age". The value of the fact named as "cust.gender" is assigned to the model input field "Gender". The result of the computation "cust.weight / 100.0" is assigned to the model input field "Pound";

```
RULE 7: // compute Model 1 prediction;
IF TRUE THEN
   SET "Model1 score" AS PREDICT(Model1) USING (
           "Age" AS cust.age,
           "Gender" AS cust.gender,
           "Pound" AS cust.weight / 100.0
   )
END;
```

For more information, please read the “RME-EP-Manual.pdf” file. The lengthy first chapter explains Forward Chaining Reasoning. The manual is available from the “CMSR_HOME/MANUALS” directory. The “Demo” project of CMSR Data Miner contains many example RME-EP models. Especially, the “Deep learning” and “Deep learning (+Tree)” and “Deep learning (ALL)” models can be useful for risk modeling.