Run a Platform without Knowing the Number or Names of Columns

Ingredients:

- Expression handling

Sample Data Tables: Cars 1993
Difficulty – Hard
Video Length – 4:25

A common practice is to run a regular analysis using data where the number or names of the inputs differ. This recipe provides a general method for getting columns into an analysis platform without knowing how many there are or their names. This is accomplished through the power of expression handling.

Steps:

1. Start with the following three lines

```
Names Default to Here(1);

tblRef = Current Data Table();
colList = tblRef << Get Column Names();
```

   This code scopes the variables created within the script to this script window, creates a reference to the current table, and gets a list of column references, respectively.
2. Open a data table with at least one continuous, one ordinal, and one nominal column. The Cars 1993 sample data table can be used. Run Distribution with one of each of these types of columns. Because the report output differs depending on column type, the platform may treat them differently. Save the script to a script window.
3. The output should look similar to this

```
Distribution(
  Nominal Distribution( Column( :Vehicle Category ) ),
  Continuous Distribution( Column( :Fuel Tank Capacity ) )
);
```

   Distribution treats all categorical (Ordinal and Nominal) columns identically. This is the case for most platforms.
4. Create an expression for the empty platform message

```
distributionExpr = Expr(Distribution());
```

5. Loop over each column and insert into the expression created in the step above, an expression corresponding to the column analysis. The column analysis expression can be taken from the code saved after Distribution was run interactively. `Substitute` will be used to build the column analysis expressions. The value returned from `Substitute` is then inserted into the main expression.

```
For(i=1,i<=N Items(colList),i++,
  If(Column(tblRef,colList[i]) << Get Modeling Type == "Continuous",
    nextColExpr = Substitute(
      Expr(Continuous Distribution(Column(col))),
      Expr(col), colList[i]
    );
  ,//ELSE
    nextColExpr = Substitute(
      Expr(Nominal Distribution(Column(col))),
```

```
        Expr(col), colList[i]
      );
    );
    Insert Into(distributionExpr,Name Expr(nextColExpr));
  );
```

When `Substitute` is used for expression handling it takes an odd number of arguments. The first argument corresponds to the expression into which substitutions are made. `Substitute` evaluates its first argument; we'll need the `Expr` function to treat it as an expression. The remaining pairs of arguments give the expression within the first argument to be replaced and the value to use. `Name Expr` is used to keep `nextColExpr` from evaluating when inserted into the main expression.

6. At the end of the loop, insert the argument into the `Distribution` expression. Use `Name Expr`, so the contents of the variable `nextColExpr` are inserted as an expression and not fully evaluated.
7. When the `Distribution` expression is complete, create an expression explicitly messaging the data table then evaluate it.

```
Eval(
  Substitute(
    Expr(tblRef << anExpr),
    Expr(anExpr),Name Expr(distributionExpr)
  )
);
```

`Name Expr` is needed for the value of `anExpr` because we want its contents as an expression not its evaluated value. Leaving it off would cause the expression in `distributionExpr` to evaluate prematurely, before being explicitly sent to the data table. This would not cause a problem in the current script because the current data table has not changed. To see how this could go wrong and what would happen, remove `Name Expr` from the third argument and insert

```
Open("$SAMPLE_DATA/BIG CLASS.JMP");
```

after creating the variable for the current data table. Run the code. The Big Class data table will open but the report window will not be created. An error message will be written to the log indicating that the data table does not recognize the message.

The code below shows an alternative more succinct way of looping through the columns.

```
For(i=1,i<=N Items(colList),i++,
  If(Column(tblRef,colList[i]) << Modeling Type == "Continuous",
    nextExpr = Expr(Continuous Distribution(Column(col))),
    nextExpr = Expr(Nominal Distribution(Column(col)))
  );
  nextColExpr = Substitute(Name Expr(nextExpr),Expr(col), colList[i]);
  Insert Into(distributionExpr,Name Expr(nextColExpr));
);
```

`Name Expr` is used with the first argument to `Substitute` because we want to use the value stored in `nextExpr`, not (literally) `nextExpr`, which is what `Expr(nextExpr)` would return.

Hints for Success:

- Expression handling is a difficult topic. The more example you see, the more you should understand.

- `Substitute` can be used to build expressions. The first argument holds the expression and the remaining argument pairs hold the expression to replace and the value to use.
- `Expr` and `Name Expr` can be used to control evaluation. The contents of `Expr` are taken literally, as is the once evaluated contents of `Name Expr`.
- `Eval` can be used to evaluate an expression.
-