Intégrer un nouveau composant dans TANAGRA

Ce document décrit la démarche à suivre pour programmer et insérer un nouveau composant dans TANAGRA. Nous nous contenterons d'une procédure statistique simple, elle consiste à :

- Vérifier qu'il n'y a pas de variables TARGET.
- Vérifier qu'il y a au moins une variable INPUT, toutes continues.
- Effectue le calcul de la moyenne pour chaque variable continue, sur les individus sélectionnés.
- Afficher les résultats sous forme de tableau dans la fenêtre de visualisation.

Le nouveau composant doit apparaître dans l'onglet STATISTICS.

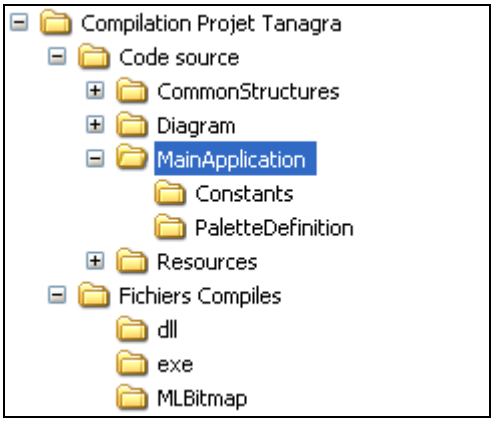
Note : Il est entendu que le lecteur connaît bien la programmation DELPHI. Si ce n'est pas votre cas, la référence http://delphi.developpez.com/ devrait vous aider pour une meilleure compréhension des concepts évoqués dans ce didacticiel.

Préalable : Compiler TANAGRA avec DELPHI

Nous considérons que le compilateur et les bibliothèques associées ont été correctement installés. Le code source a été téléchargé sur le site de TANAGRA, il a été décompressé dans un répertoire dédié, en respectant les chemins de destination. Le projet a été ouvert avec DELPHI, une première compilation a été possible.

Pour plus d'informations sur cette première phase, nous nous référons au didacticiel y afférent http://tutoriels-data-mining.blogspot.com/2008/04/compiler-le-projet-tanagra.html

Nous obtenons l'organisation des dossiers suivante après décompression du code source.



Le fichier principal du projet **TANAGRA.DPR** est accessible dans le sous répertoire MAINAPPLICATION.

Remarque (1) : Qu'importe la dénomination du répertoire racine. Le plus important est de respecter les chemins des sous répertoire. DELPHI utilise des chemins relatifs.

Remarque (2) : En ce qui concerne le répertoire de destination des fichiers compilés, « ***Fichiers Compiles*** » dans notre exemple, comme nous l'indiquons dans le didacticiel cité en référence plus haut, le mieux est de le construire en s'appuyant sur le setup du logiciel.

Ajouter un composant à TANAGRA

Créer un nouveau module

Plusieurs classes doivent être surchargées pour ajouter un composant fonctionnel dans TANAGRA. Nous pouvons les regrouper dans un module (unité dans la terminologie DELPHI). Nous le nommons « **UCompDemonstration.pas** ».

Rappelons qu'une unité comporte 2 grandes parties : INTERFACE contient le descriptif des classes ; IMPLEMENTATION comporte l'implémentation des méthodes.

Dans la partie haute de l'unité, avant le mot clé UNIT, nous insérons quelques commentaires. Ils permettent de mieux situer le rôle du nouveau composant, la méthode statistique programmée, les références utilisées, la version en cours de développement, le suivi des mises à jour, etc.

```
(*************************************************************)
(* UCompDemonstration.pas - Copyright (c) 2008 Ricco RAKOTOMALALA *)
(*************************************************************)

{
@abstract(Module de démonstration : insertion d'un nouveau composant)
@author(Ricco Rakotomalala)
@created(21/04/2008)

Cette unité montre comment insérer un nouveau composant dans TANAGRA.
Nous nous contentons de calculer la moyenne des variables INPUT. Nous
créons des classes héritières (4 principalement) des classes pré-existantes
pour cela. Quelques méthodes doivent impérativement être surchargées.
}

unit UCompDemonstration;
```

La clause USES

La clause USES permet de faire appel à d'autres modules. Trois types de modules sont nécessaires dans la grande majorité des cas : ceux de DELPHI, les outils de gestion des composants, les classes de calcul, et les modules d'accès aux données.

```
USES
  //unités std delphi
  Classes, IniFiles, Forms, Sysutils,
  //gestion des composants
  UCompDefinition, UOperatorDefinition,
  //gestion de l'accès aux données
  UCompManageDataset, UDatasetDefinition, UDatasetImplementation,
  UDatasetExamples;
```

La classe génératrice de composants

Cette classe a pour vocation de gérer la méthode statistique dans la palette de composants. Elle génère l'instance de calcul lorsqu'on insère par « glisser déposer » l'icône dans le diagramme de traitements.

```
//************************************************************************
//** générateur de composant, son rôle est de créer une instance de composant
//** qui est inséré dans le diagramme de traitements
//************************************************************************
TMLGenCompDemo = class(TMLGenComp)
protected
//spécifier dans quel onglet de la palette doit apparaître l'icône
procedure   genCompInitializations(); override;
public
//spécifier quelle est la classe du composant à générer sur le diagramme
function    getClassMLComponent: TClassMLComponent; override;
end;
```

La première méthode permet de spécifier l'onglet dans lequel le générateur doit être placé.

```
procedure TMLGenCompDemo.genCompInitializations;
begin
 //onglet STATISTICS
 FMLComp:= mlcDescriptiveStat;
end;
```

La seconde méthode spécifie la classe qui sera instanciée lorsque l'icône est placée dans le diagramme.

```
function TMLGenCompDemo.getClassMLComponent: TClassMLComponent;
begin
 result:= TMLCompDemo;
end;
```

Il y a peu de programmation à ce stade. Beaucoup de choses ont déjà été implémentées dans la classe ancêtre.

Note : Il est impératif d'enregistrer cette classe pour qu'il soit retrouvé dans le fichier de configuration au démarrage du logiciel. Nous rajoutons une commande spécifique dans la clause INITIALIZATION à la fin de l'unité.

```
initialization
 RegisterClass(TMLGenCompDemo);
end.
```

La classe de gestion du composant

Cette classe définit le comportement du composant dans le diagramme de traitement. Ici également, il y a peu de choses à implémenter, beaucoup a été fait dans la classe ancêtre. Nous pouvons exploiter cette classe si nous voulons définir un comportement particulier, par exemple autoriser le placement du composant uniquement à la suite de certains traitements statistiques.

```
//*************************************************************************
//** classe du composant graphique inséré dans le diagramme
//*************************************************************************
TMLCompDemo = class(TMLCompLocalData)
protected
//spécifier l'opérateur de calcul
function    getClassOperator: TClassOperator; override;
end;
```

La seule méthode à surcharger est la désignation de la classe de calcul.

```
function TMLCompDemo.getClassOperator: TClassOperator;
begin
 result:= TOpDemo;
end;
```

La classe de gestion des calculs

Cette classe prend en charge le calcul et l'affichage des résultats.

```
//************************************************************************
//** classe opérateur de calcul : prend en charge le calcul
//** et l'affichage des résultats
//************************************************************************
TOpDemo = class(TOpLocalData)
private
//tableau des moyennes INPUT
FTabAvg: array of double;
protected
//spécifier la classe de paramétrage -- surcharge obligatoire
function    getClassParameter: TClassOperatorParameter; override;
//contrôle de cohérence des attributs avant les calculs
function    checkAttributes(): boolean; override;
//lancer effectivement les calculs -- surcharge obligatoire
function    coreExecute(): boolean; override;
public
//libération de la mémoire allouée
destructor  destroy(); override;
//affichage des résultats au format HTML -- surcharge obligatoire
function    getHTMLResultsSummary(): string; override;
end;
```

Le champ privé « FTABAVG » sert à conserver les résultats calculés. Elle doit être libérée à la destruction de l'instance, c'est ce que nous réalisons dans la méthode DESTROY. La fonction GETCLASSPARAMETER spécifie la classe de gestion des paramètres de calcul. Il s'agit simplement d'indiquer la bonne référence TOPRMDEMO.

La méthode CHECKATTRIBUTES est primordiale. Elle définit les conditions de sélection de variables qui autorisent le lancement des calculs. Dans notre cas, il faut : (1) qu'il n'y ait pas de variables TARGET ; (2) qu'il y ait au moins 1 variable INPUT, lesquelles doivent être toutes continues. La fonction renvoie FALSE si toutes ces conditions ne sont pas réunies. Le calcul est interrompu et un message d'erreur standard est affiché par TANAGRA.

```
function TOpDemo.checkAttributes: boolean;
var ok: boolean;
begin
 //premier test, pas de TARGET
 ok:= (self.WorkData.LstAtts[asTarget].Count = 0);
 //second test, au moins 1 INPUT et tous continus
 ok:= ok and
      ((self.WorkData.LstAtts[asInput].Count > 0) and
        self.WorkData.LstAtts[asInput].isAllCategory(caContinue));
 //renvoyer la réponse
 result:= ok;
end;
```

La méthode COREEXECUTE est le cœur de l'affaire. Il réalise les calculs. On notera le mode d'accès aux variables et aux observations. Le calcul est réalisé uniquement sur les variables INPUT et sur les individus sélectionnés.

La fonction renvoie la valeur FALSE si une erreur est survenue lors des calculs, TRUE dans le cas contraire. Je me contente d'un gestionnaire d'exceptions dans la grande majorité des cas. Il faut simplement faire attention aux fuites de mémoire si les opérations ne vont pas jusqu'à leur terme.

Enfin, dernier point important, la sélection de l'utilisateur peut être modifié entre 2 exécutions successives. Il est donc important que les tableaux ou objets intermédiaires soient préparés correctement. C'est pour cette raison que le tableau dynamique FTABAVG est initialisé dans cette fonction. Nous aurions pu le faire également dans la fonction CHECKATTRIBUTES. En tous les cas, nous ne pouvons pas nous contenter de l'initialiser une seule fois dans le constructeur de la classe.

```
function TOpDemo.coreExecute: boolean;
var j,i: integer;
    nbAtt,nbExamples: integer;
    avg: double;
    att: TAttribute;
begin
 TRY
 //préparer le tableau de résultats
 nbAtt:= self.WorkData.LstAtts[asInput].Count;
 setLength(FTabAvg,nbAtt);
 //nb. d'individus
 nbExamples:= self.WorkData.Examples.Size;
 //pour chaque variable, calculer la moyenne
 for j:= 0 to nbAtt-1 do
  begin
   //se brancher sur la variable
   att:= self.WorkData.LstAtts[asInput].Attribute[j];
   //calcul de la somme
   avg:= 0.0;
   for i:= 1 to nbExamples do
    avg:= avg + att.cValue[self.WorkData.Examples.Number[i]];
   //moyenne
   FTabAvg[j]:= avg / (1.0 * nbExamples);
  end;
 //pas de soucis
 result:= true;
 EXCEPT
 //il y a eu des soucis
 result:= false;
 END;
end;
```

Dernière méthode importante dans cette classe, la production des résultats avec GETHTMLRESULTSSUMMARY. Nous créons une chaîne de caractères qui est envoyée à la méthode appelante. Elle doit respecter le standard HTML. Dans notre exemple, nous produisons un tableau. Les chaînes de caractères en DELPHI sont limitées à 2 Go, nous pouvons produire des sorties assez sophistiquées.

```
function TOpDemo.getHTMLResultsSummary: string;
var s: string;
    j: integer;
begin
 s:= '<table border = 1><tr><th>Attribute</th><th>Average</th></tr>';
 for j:= 0 to self.WorkData.LstAtts[asInput].Count-1 do
  begin
   s:= s + '<tr>';
   s:= s + format('<td align=left>%s</td><td align=right>%.4f</td>',
            [self.WorkData.LstAtts[asInput].Attribute[j].Name,FTabAvg[j]]);
   s:= s + '</tr>';
  end;
 s:= s + '</table>';
 //
 result:= s;
end;
```

Il est possible d'élaborer une fenêtre d'affichage spécifique. La programmation n'est plus standardisée. Mais cela peut être nécessaire pour certaines méthodes (ex. dessin du dendrogramme pour la classification ascendante hiérarchique). Nous devons surcharger la méthode GETNEWFORM dans ce cas (cf. UCompClusteringHAC.pas pas exemple).

La classe de gestion des paramètres du composant

Cette classe gère les paramètres de la méthode implémentée. Elle s'occupe de son affichage dans une éventuelle boîte de dialogue, la sauvegarde des valeurs dans les flux d'entrées-sorties.

```
//************************************************************************
//** classe paramétrage de l'opérateur de calcul
//** gère la boîte de paramétrage et l'E/S des paramètres
//************************************************************************
TOpPrmDemo = class(TOperatorParameter)
protected
//création de la boîte de dialogue de paramétrage -- surcharge obligatoire
//(doit renvoyer "nil" si pas de boîte à générer)
function    CreateDlgParameters(): TForm; override;
//spécifier les paramètres par défaut de la méthode -- surcharge obligatoire
procedure   SetDefaultParameters(); override;
end;
```

Dans notre exemple, nous n'avons pas de paramètres. Nous nous contentons d'implémenter les méthodes (c'est obligatoire) en les laissant vides.

```
function TOpPrmDemo.CreateDlgParameters: TForm;
begin
 //pas de boîte de dialogue de paramétrage
 result:= nil;
end;

procedure TOpPrmDemo.SetDefaultParameters;
begin
 //>> pas de paramètres
end;
```

Pour plus d'informations sur cette classe, voir l'unité **UCompPlsBase.pas.**

Conclusion : À ce stade, la programmation dans TANAGRA est finalisée. En compilant le projet, la nouvelle méthode est intégrée dans l'exécutable. Il ne nous reste plus qu'à configurer l'environnement de l'application pour que le nouveau composant soit pris en compte lors du démarrage de l'application.

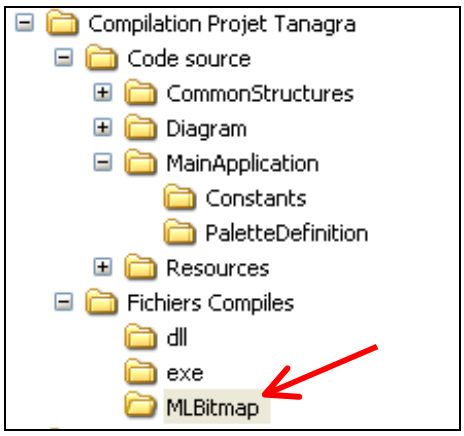
Créer l'icône représentant le composant

Anecdotique certainement, j'y passe un temps insensé pour ma part, mes talents artistiques étant proches du néant intégral, il s'agit de créer une image censée représenter la méthode. Ce n'est pas si facile.

N'importe quel outil de création de fichier BMP convient. J'utilise l'éditeur d'image de DELPHI (OUTILS / EDITEUR D'IMAGE). Le fichier doit être au format BMP, de taille 16 x 16 en 16 couleurs.

Pour ce didacticiel, l'icône MLDemonstration.BMP a été créé. Il ne ressemble à rien  .

Note : Ce qui importe surtout, c'est qu'il soit copié dans le sous répertoire MLBITMAP du dossier de compilation.



Mettre à jour le fichier de configuration XML

Dernière étape, nous référençons le nouveau composant dans le fichier de configuration TANAGRA_COMPONENTS .XML situé dans la racine du répertoire de compilation. Pour nous, il serait dans le répertoire FICHIERS COMPILES.

Plusieurs informations doivent être fournies. Nous noterons que le nom de la classe génératrice du composant est fondamental pour que la nouvelle méthode de Data Mining soit reconnue dans TANAGRA.

```
<component class_name="TMLGenCompDemo">
        <name>
        Demonstration
        </name>

        <bitmap>
        MLDemonstration.bmp
        </bitmap>

        <description>
        Demonstration : how to add a new component into TANAGRA.
        </description>

        <precondition>
        A dataset must be available i.e. one or more descriptors, and one or more examples.
        </precondition>

        <target-attributes>
        None.
        </target-attributes>

        <input-attributes>
        At least one variable. All variables must be continuous.
        </input-attributes>

        <postcondition>
        None.
        </postcondition>

</component>
```

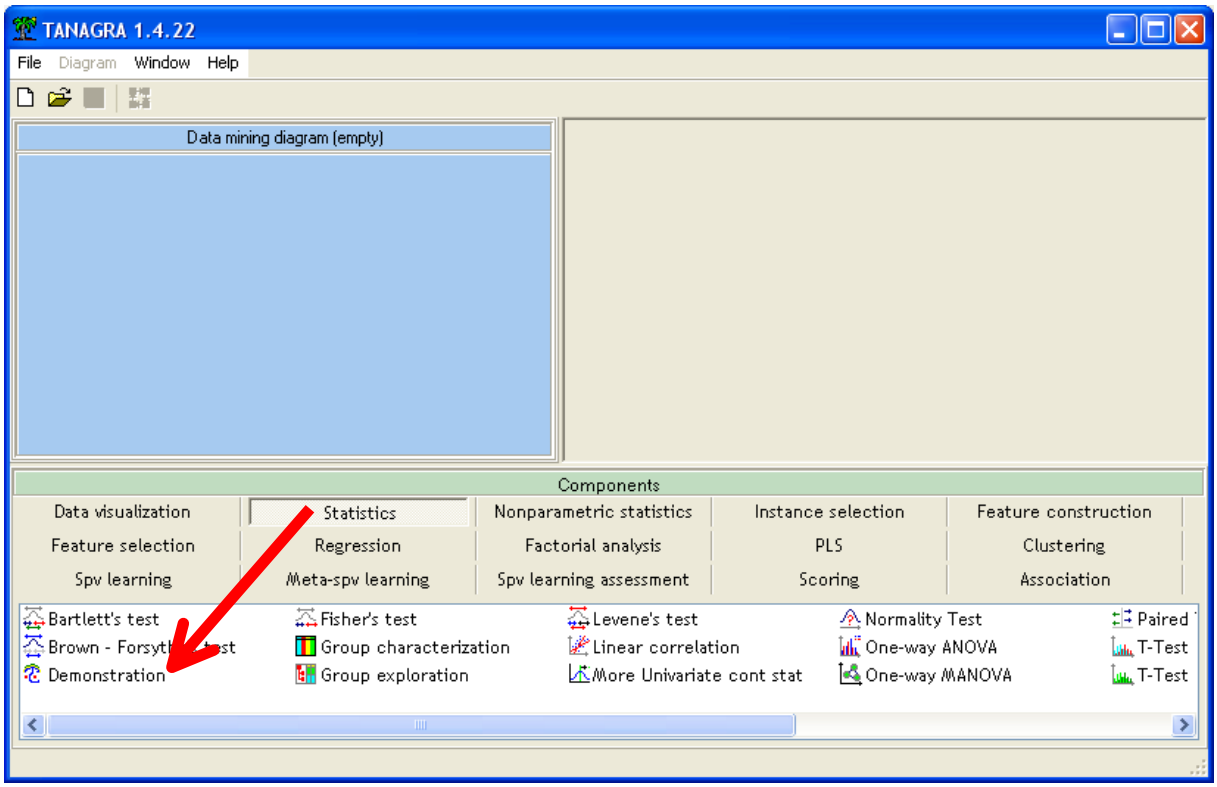
Conclusion : Tout est prêt maintenant. Au prochain de démarrage de TANAGRA, le composant doit apparaître dans la palette, précisément dans l'onglet STATISTICS. On doit pouvoir l'insérer dans le diagramme et réaliser des calculs. Vérifions cela.

Tester le composant

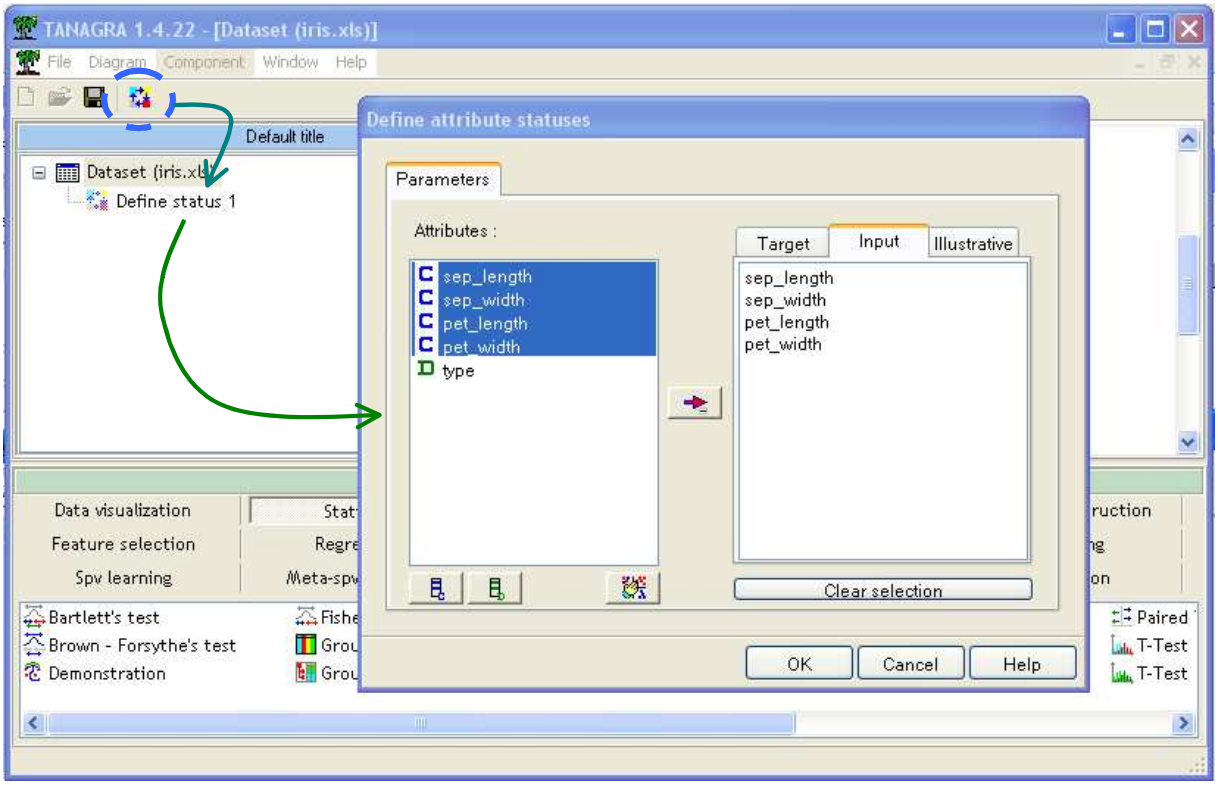
Pour tester le composant, nous utilisons le fichier IRIS.XLS. Nous considérons que le lecteur sait manipuler TANAGRA : importer des données, définir des calculs, etc[1]. Nous voulons calculer la moyenne sur les 4 variables continues de la base.

Première étape très importante, au lancement de l'exécutable, vérifions que l'icône apparaît bien dans l'onglet STATISTICS. C'est déjà une victoire.

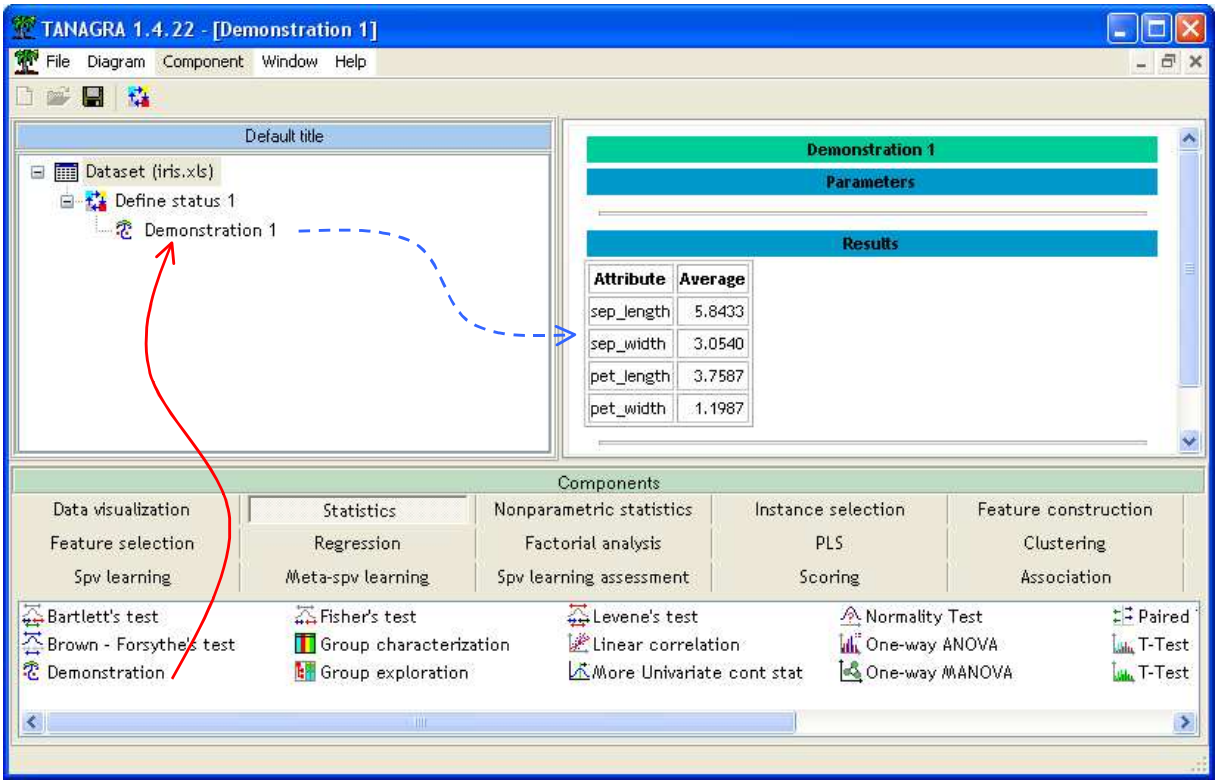
[1] En cas de doute, voir http://tutoriels-data-mining.blogspot.com/



Nous importons les données en créant un nouveau diagramme (FILE / NEW, Importation fichier EXCEL). Puis, à l'aide du composant DEFINE STATUS accessible dans la barre d'outils, nous plaçons les 4 variables continues en INPUT.



Nous insérons le composant dans le diagramme et nous activons le menu contextuel VIEW.



Nous obtenons le résultat voulu. Ouf !

Conclusion :

Ce didacticiel présente de manière assez succincte la programmation d'un nouveau composant dans TANAGRA. La méthode implémentée est très simple, mais nous avons là tous les éléments pour définir rapidement un objet de calcul qui tient la route.

Plusieurs points doivent être vérifiés pour valider la programmation. Entre autres, que se passe-t-il si nous insérons une variable en TARGET dans le composant DEFINE STATUS ? Aucune variable en INPUT ? Des variables catégorielles parmi les INPUT. Autre aspect important, si nous effectuons un échantillonnage, est-ce les calculs sont réellement réalisés sur la fraction des individus sélectionnés ? Etc.

La liste des vérifications peut s'allonger rapidement… surtout si le composant est destiné à être utilisé par d'autres personnes. Il faut véritablement cadenasser les conditions d'utilisation pour prévenir les erreurs intempestives.

Les fichiers qui accompagnent ce didacticiel sont accessibles en ligne http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/exemple_ajouter_composant_dans_tanagra.zip