OOSTATS – A New Approach to Statistics via APL

Introduction

The advent of Dyalog APL's interpreter with object-oriented features (Versions ≥11) suggested an interesting task of investigating how the new facilities could be exploited to provide APL software for users to perform statistical advantages. The following pages report my part-time efforts during the last year.

APL rightly has a reputation for being a good language for performing statistical calculations. Since my first publication (with Jake Ansell) in Quote Quad (Sig APL Conference in Seattle 1985) I have reported some of my efforts, throughout which I have generally assumed that the data was nicely sanitised with no missing values. When it comes to dealing with a genuine application, a good deal of cleaning up has to be done before using APL to produce a statistical report. This can be a laborious job – moreover, since you don't want to remove cases unnecessarily, cleaning needs to be done within the immediate context. (If, for example, you are using regression to predict the variable y using x_1, then you need to remove any cases where there are missing values in y and x_1 – change the model to include a second or different variable x_2 and you need to clean from scratch again.) It follows that a database structure which will handle missing values is essential.

It was clear from the start that an object-oriented approach would be really useful in Statistics -statistical results need to be self-explanatory to some degree, so a set of variables in a database requires names, variable descriptors and, where appropriate, labels associated with specific values. Thinking of these as public fields of a database object was a natural step to take that would also facilitate dealing with missing values. Gradually a structure materialised:–

- A database object (rectangular database) would be created from a set of user variables.
- This object has methods and fields to expedite database activity
- It also contains 'statistical methods'
- A statistical method filters out missing values and creates an appropriate statistical object for the analysis required.

A further design criterion became clear after a little work – whilst being able to incorporate the statistical tools into a user's program was clearly paramount, for the routine task of investigating a data set and trying different models, deleting outliers where appropriate etc, a gui environment is ideal, and so, from the outset, simple gui tools were developed, e.g. for declaring variable formats – it was then a small conceptual and structural step to include all the facilities in a gui framework.

The Database Object

The database object is the coding `s_db` in the workspace `oostats`. The command:-

```
      db←⎕NEW s_db(('alan' 'peter')((1.1 2.1 3.1 4.1)(4 3 2 1)))
```

creates a database of two variables known by the names `alan` and `peter`. (An error message results if variables are of unequal length.)
The database as a formatted matrix is given by:-

```
      db.Mat
1 1.1 4 1 1
2 2.1 3 1 1
3 3.1 2 1 1
4 4.1 1 1 1
```

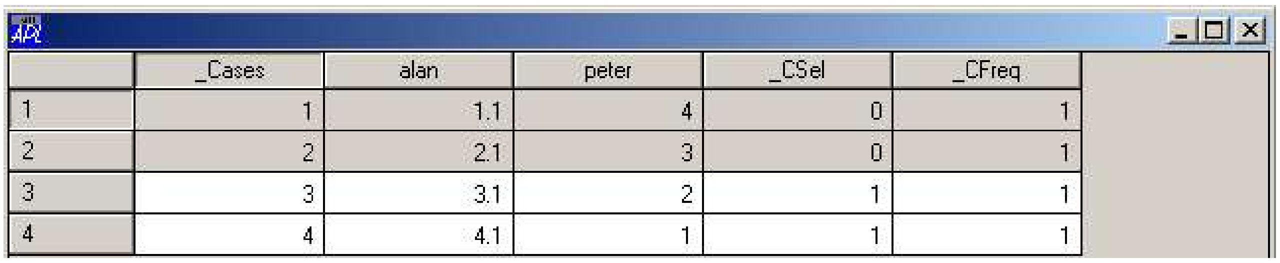
where the first column lists **case numbers** (useful if a different ordering is used) and the last two columns indicate the **cases selected** and the **case frequency** (see later for a use of the latter).

To select cases:-

```
     db.SelectCases 'alan>2.2'
Cases selected by alan>2.2
2 cases not selected
     db.Mat
1 1.1 4 0 1
2 2.1 3 0 1
3 3.1 2 1 1
4 4.1 1 1 1
```

If you want to browse the database as a grid object:-

```
     db.View
```

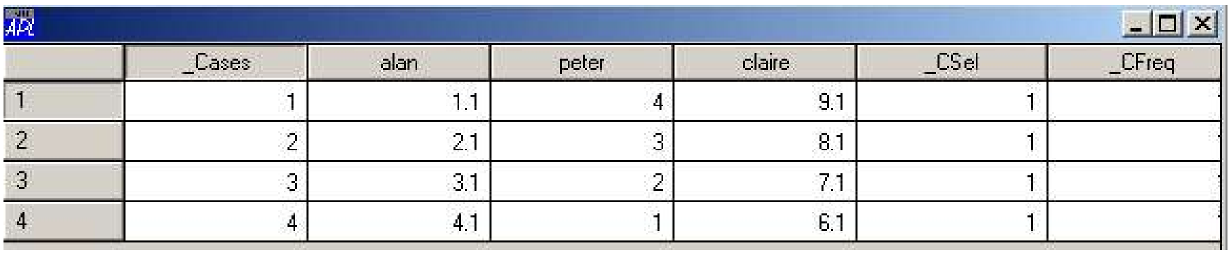


	_Cases	alan	peter	_CSel	_CFreq
1	1	1.1	4	0	1
2	2	2.1	3	0	1
3	3	3.1	2	1	1
4	4	4.1	1	1	1

Note that non-selected cases appear in grey.

New variables may be calculated:-

```
     db.SelectCases 1
Cases selected by numerical expression
0 cases not selected
     db.AddVariable 'claire←alan+2×peter'
Variable claire created by expression 'alan+2×peter' added to database
db.View
```



	_Cases	alan	peter	claire	_CSel	_CFreq
1	1	1.1	4	9.1	1	
2	2	2.1	3	8.1	1	
3	3	3.1	2	7.1	1	
4	4	4.1	1	6.1	1	

Note here that the formula for the new variable would be applied only on those cases where all variables involved are not missing.

Other database facilities are available, e.g. declaring formats which apply to viewing the data through the field `Mat`, ordering the database, deleting a variable and so on. Some of these will be used later in the exposition, but the facility for dealing with missing values needs special mention.

Missing Values

Missing values fall into two categories – *system missing* and *user-declared missing*. Obviously a `⎕null` value in a vector is a system missing value – other candidates tend to appear, such as `⊂''` and `⊂⍳0`, particularly when importing files from Excel. These are contained in the database field `_SM`.

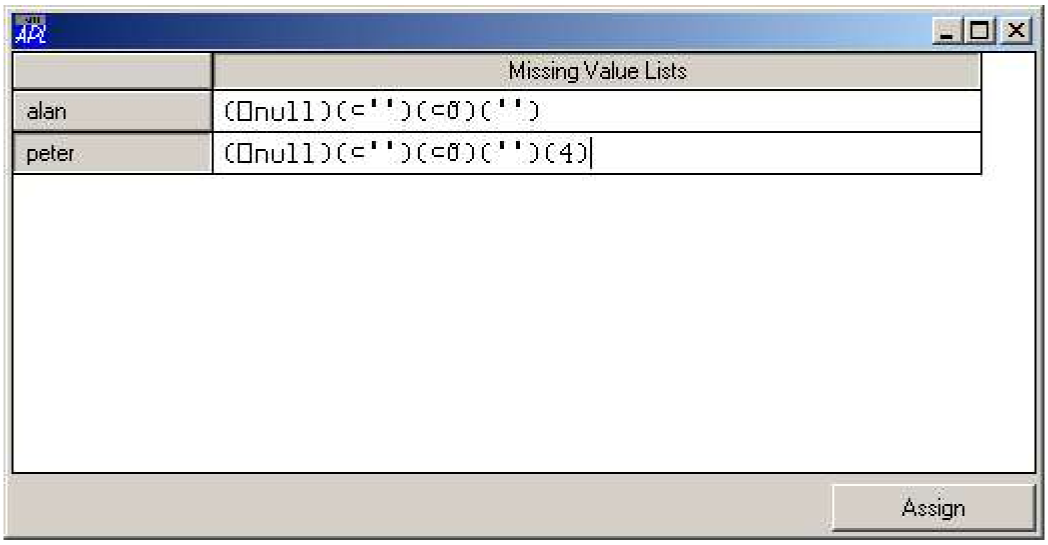
The need for user-declared missing values is particularly clear within the context of survey databases, where **all** fields should have a numeric value recorded even when a respondent fails to enter anything. This is because the analyst may wish to use that field as a predictor – if so, the numeric value recorded is declared to be a missing value; alternatively if the analyst wished to explore whether respondents of a certain type are more likely not to have filled in that field, then cases corresponding to these ‘missing values’ need to be included in the analysis.

Missing values are collected in a nested text vector, so that individual elements of the vector, when executed, yield a vector of values. This facilitates knowing just exactly what values are there and adding to them through a grid object:-

```
      db←⎕new s_db (('alan' 'peter')((1.1 2.1 3.1 4.1)(4 3 2 1)))
       db.MissVals
 (⎕null)(⊂'')(⊂⍬)('')  (⎕null)(⊂'')(⊂⍬)('')
```

To alter or add to these values:-

```
      db.GetMissVals
```



	Missing Value Lists
alan	(⎕null)(⊂'')(⊂⍬)('')
peter	(⎕null)(⊂'')(⊂⍬)('')(4)

Adding `(4)` to the entry for variable `peter` and assigning the values would ensure that in any statistical work done using the variable `peter` the first case would not be used.

Statistical Methods

The database object code `s_db` contains a number of statistical methods which can be listed using:-

```
   db.StatsMethods
UniqueFrequency
Unistats
Regress
TwoSampleMeans
CrossTabs
MatchedPairs
OneWayAnova
Scatterplot
```

These represent my ‘base camp’ beyond which I have resisted temptation to move on before presenting this work and spending time consolidating the basic database code.

Each method creates a sub-database object which is declared **public** so that its fields and methods are available to the user. In all but the first method, missing values are filtered out as necessary before the creation of the sub-object, so that the coding of the sub-objects is free of this obligation! All such methods have object code beginning with `s_`, e.g. `s_unistats`. In terms of functionality, they represent that part of Statistics which would be included in an introductory course on the subject, except that `Regress` includes the functionality of the Generalized Linear Models facility that I developed at Swansea and which is still used there by students in their final year.

As an introduction to some of these methods, we use the database of house prices that will be familiar to many an APLer as it is often used by Adrian Smith in demonstrating *Rainpro*.

```
   Open 'oostats/housedata.adb'
Object 'db' has been created using s_db from file oostats/housedata.adb
```

(A database can be saved in an APL component file using the command `db.Save filename`.)
To list the variables in the database:-

```
   db.VarNames
 price  area  beds  type  age  _CSel  _CFreq
```

To find out more about these variables:-

```
↑  db.VarLabels
Price of house in £1000
Floor area in square feet
Number of bedrooms
Type of house
Age in years
```

In this case, the variable `type` has associated value labels:-

```
   db.ValueLabels[4]
 1  terraced house
 2  semi-detached
 3  detached
 4  bungalow
```

indicating that the value 1 is for a ‘terraced house’ and so on.

To obtain values for a variable:-

```
   db.GetVar 'price'
156 55 65 46.95 46.95 47.5 110 65 54 85 70 82.5 78.5 49.95 77 37.5 65 75
96.5 53.95 75 60.5 148 79.95 120 102 145 68 98
```

Now we create a univariate statistics object for the first variable.

```
   db.Unistats 'price'
Created object #.price.?
Currently, # cases excluded = 0
```

Note that this object is created only for those cases selected. (It seemed sensible here to create the object at the top level – other objects remain within the database object.)

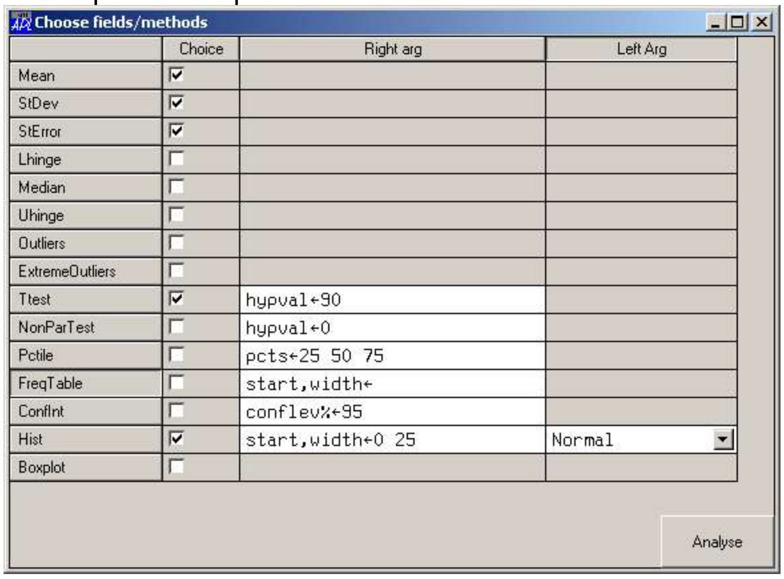
```
   price.Mean
79.7845
   price.StDev
31.3536
   price.Pctile 25 50 75
25 55
50 75
75 96.5
```

To see what options are available to us (they may be fields or methods):-

```
   price.Options
Mean  StDev  StError  Lhinge  Median  Uhinge  Outliers  ExtremeOutliers
Ttest  NonParTest  Pctile  FreqTable  ConfInt  Hist  Boxplot
```

Some of these require a right argument and may also have a potential left-argument – to facilitate this, all statistical objects have an 'Explore' method which helps you to choose the results you want:-

```
     price.Explore
```



Choose fields/methods

	Choice	Right arg	Left Arg
Mean	✓		
StDev	✓		
StError	✓		
Lhinge	☐		
Median	☐		
Uhinge	☐		
Outliers	☐		
ExtremeOutliers	☐		
Ttest	✓	hypval←90	
NonParTest	☐	hypval←0	
Pctile	☐	pcts←25 50 75	
FreqTable	☐	start,width←	
ConfInt	☐	conflev%←95	
Hist	✓	start,width←0 25	Normal
Boxplot	☐		

Analyse

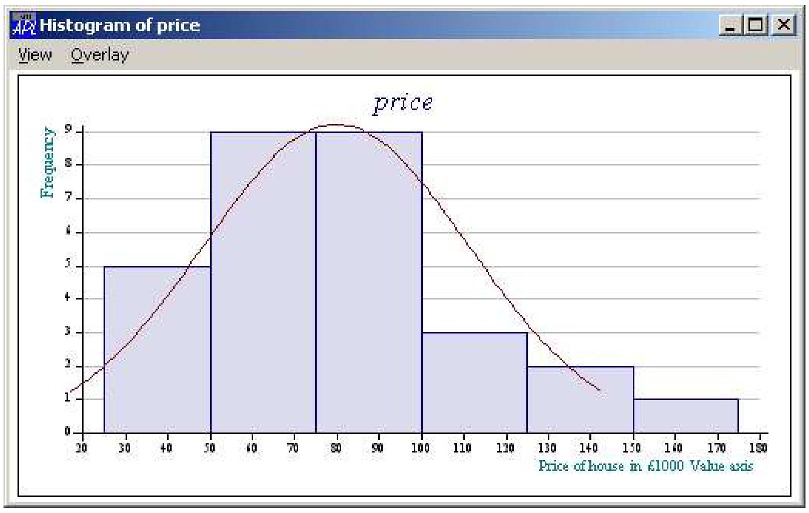
Various options have been ticked, a right-argument for the `Ttest` of 90 has been entered (overwriting the default value of 0) and a histogram is required with class intervals starting at 0 of width 25 with a Normal distribution overlay. Text output results:-

```
Univariate Statistics for price
-------------------------------

 Statistic   Value
 # cases        29
 Mean       79.78
 StDev      31.35
 StError     5.822

Testing the hypothesis that the true mean is 90
-----------------------------------------------

 Statistic          Value
 #               29
 Mean            79.78
 Std Error        5.822
 t-Stat          ¯1.755
 p-val            0.09027
 lower 95% conf  67.86
 upper 95% conf  91.71
```



On the histogram form, menu items allow you to view the histogram in full screen mode or to try an alternative distribution overlay (at present, the choice is between Exponential and Normal distribution). Note that the title of the variable is automatically included in the value axis label.

The Crosstabs Object

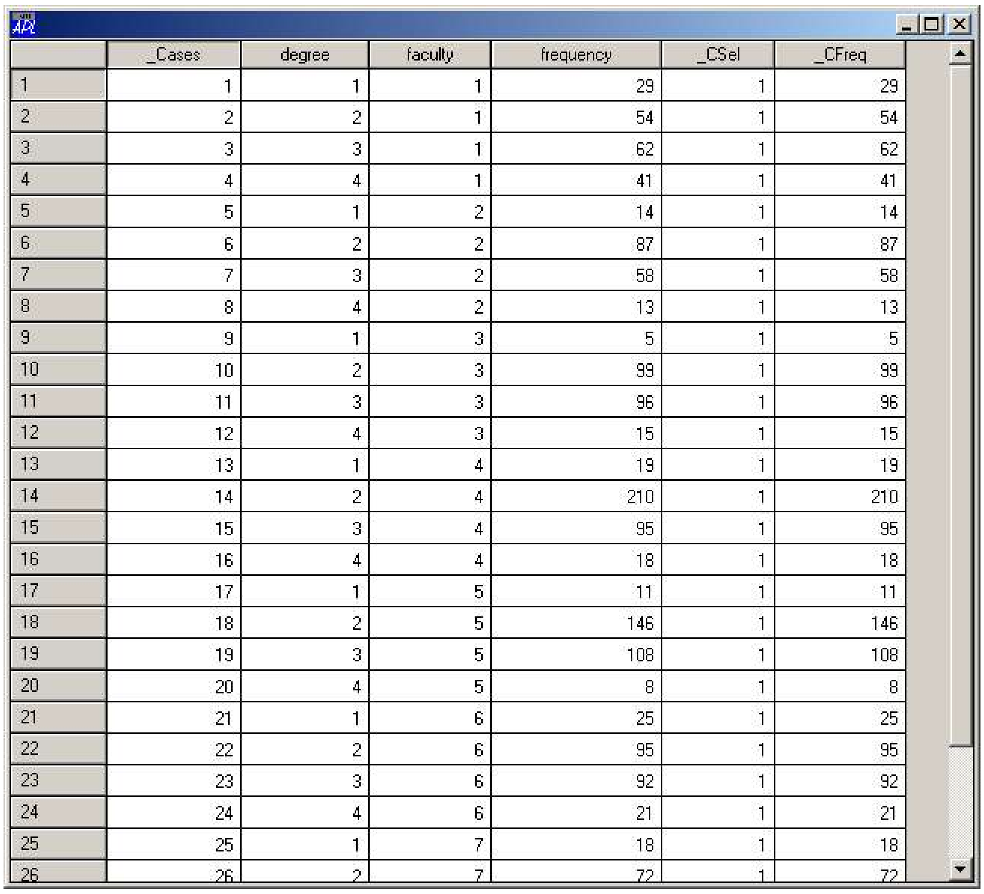
A very useful tool is that of cross-tabulation – a bivariate frequency table is calculated giving the frequencies associated with each pair of values of two variables. Staying with the housedata database, we can examine the frequencies associated with the two variables *beds* and *type*.

```
   db.CrossTabs'type' 'beds'
Sub-Object ct created using s_crosstabs
      db.ct.Table
                 1 2 3 4 5
 terraced house  1 2 2 2 2
 semi-detached   1 0 9 1 0
 detached        0 0 2 3 0
 bungalow        0 1 3 0 0
```

Note how the values 1,2,3,4 for the type of house are replaced by their value labels.

Often, the motivation for a cross-tabulation analysis is to investigate whether there is any relationship between the two variables. If there is no relationship, then the observed proportions in each row (row frequencies divided by row sums) should be much the same for each row. This hypothesis is tested by computing the chi-square test statistic for observed frequencies and expected frequencies calculated assuming the hypothesis of independence. A good example of this at work is to see whether the degree classifications at a well-known university are independent of the faculty.

```
  Open 'oostats/degclass.adb'
Object 'db' has been created using s_db from file oostats/degclass.adb
      db.VarNames
 degree  faculty  frequency  _CSel  _CFreq
      db.View
```



	_Cases	degree	faculty	frequency	_CSel	_CFreq
1	1	1	1	29	1	29
2	2	2	1	54	1	54
3	3	3	1	62	1	62
4	4	4	1	41	1	41
5	5	1	2	14	1	14
6	6	2	2	87	1	87
7	7	3	2	58	1	58
8	8	4	2	13	1	13
9	9	1	3	5	1	5
10	10	2	3	99	1	99
11	11	3	3	96	1	96
12	12	4	3	15	1	15
13	13	1	4	19	1	19
14	14	2	4	210	1	210
15	15	3	4	95	1	95
16	16	4	4	18	1	18
17	17	1	5	11	1	11
18	18	2	5	146	1	146
19	19	3	5	108	1	108
20	20	4	5	8	1	8
21	21	1	6	25	1	25
22	22	2	6	95	1	95
23	23	3	6	92	1	92
24	24	4	6	21	1	21
25	25	1	7	18	1	18
26	26	2	7	72	1	72

The degree classifications have been recorded as 1,2,3,4 with value labels, and similarly for the seven faculties. The frequency column gives the frequency of each combination of degree and faculty, and so the command

```
   db.CaseFrequency 'frequency'
Case Frequency is user defined by expression frequency
```

was used prior to the database being saved. (If this declaration was not made, then the cross-tabulation frequency matrix would have all entries equal to 1.)

```
   db.CrossTabs 'faculty' 'degree'
Sub-Object ct created using s_crosstabs
      db.ct.Options
  Observed   Expected   StandResid   Table   ChiSquareTest   FullTable
ViewFullTable  TowerChart
```

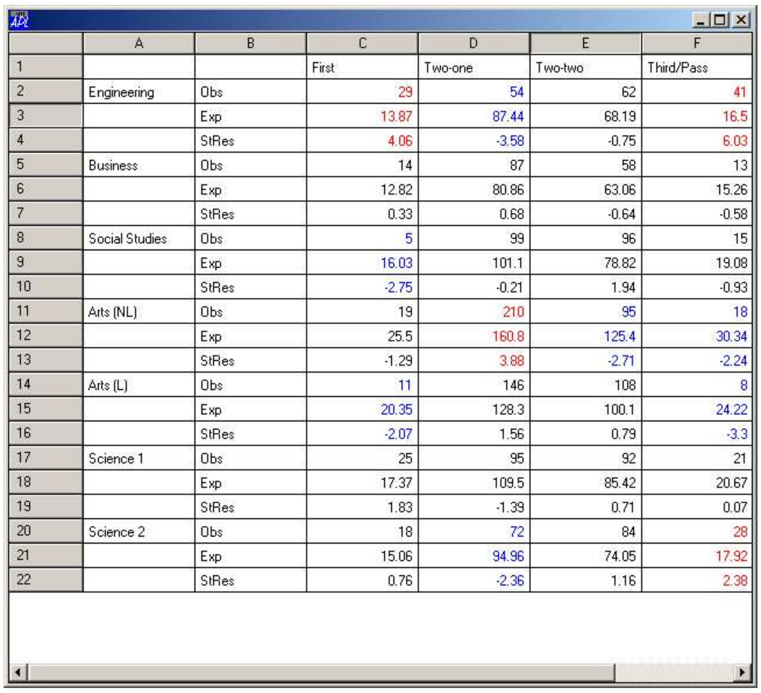
To test the hypothesis of independence between the two category variables we use the Chi-square test:-

```
   Tab db.ct.ChiSquareTest
 Chi-square  Degrees of Freedom  p-value
    146.037                  18        0
```

(N.B. the Function `Tab` merely puts column labels on top of the matrix of results – for reporting such output later, it is advantageous to have column labels separate from the rest of the table.)

The *p*-value rounded to zero implies that there is much evidence to suggest that the distribution of degree classifications depends very much on the faculty in question. To investigate this further:-

```
db.ct.ViewFullTable
```



	A	B	C	D	E	F
1			First	Two-one	Two-two	Third/Pass
2	Engineering	Obs	29	54	62	41
3		Exp	13.87	87.44	68.19	16.5
4		StRes	4.06	-3.58	-0.75	6.03
5	Business	Obs	14	87	58	13
6		Exp	12.82	80.86	63.06	15.26
7		StRes	0.33	0.68	-0.64	-0.58
8	Social Studies	Obs	5	99	96	15
9		Exp	16.03	101.1	78.82	19.08
10		StRes	-2.75	-0.21	1.94	-0.93
11	Arts (NL)	Obs	19	210	95	18
12		Exp	25.5	160.8	125.4	30.34
13		StRes	-1.29	3.88	-2.71	-2.24
14	Arts (L)	Obs	11	146	108	8
15		Exp	20.35	128.3	100.1	24.22
16		StRes	-2.07	1.56	0.79	-3.3
17	Science 1	Obs	25	95	92	21
18		Exp	17.37	109.5	85.42	20.67
19		StRes	1.83	-1.39	0.71	0.07
20	Science 2	Obs	18	72	84	28
21		Exp	15.06	94.96	74.05	17.92
22		StRes	0.76	-2.36	1.16	2.38

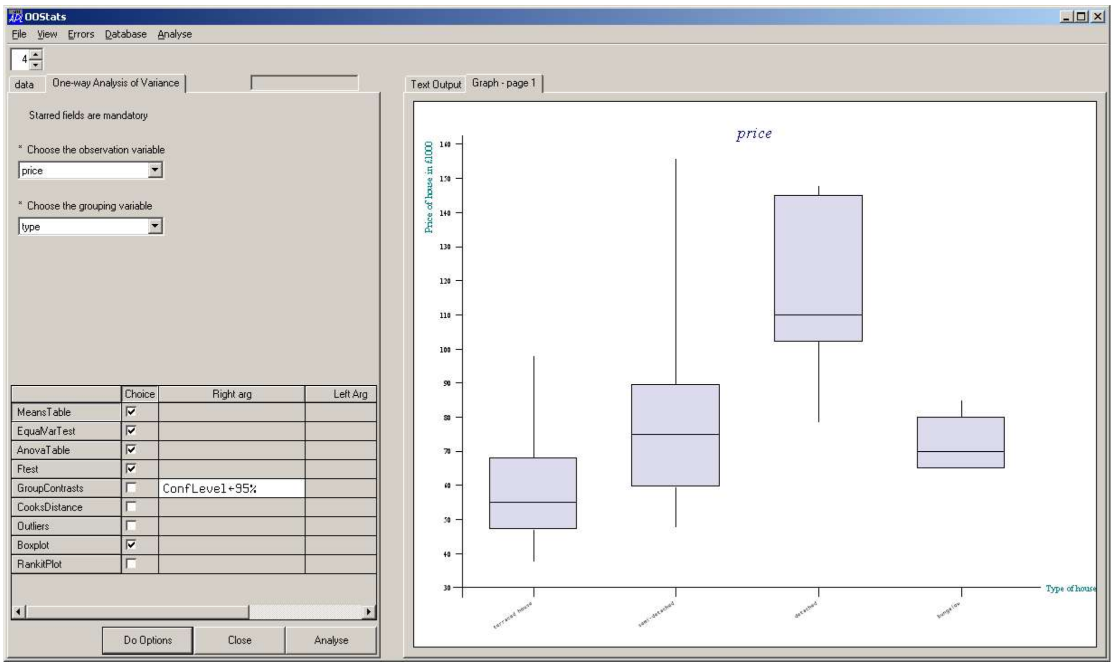
In this full table for each cell in the frequency table, the observed frequency is listed with the Expected frequency and a standardised residual. Values of the latter greater than 2 are shown in red, and those less than -2 are shown in blue. It then becomes clear that the Engineering and Science faculties tend to produce more Firsts and Third/Pass results than the more discursive faculties where degree results tend to bunch into the Two-one and Two-two categories. (In my experience, this part of the analysis is almost always excluded from statistical textbooks – a sad state of affairs as this really is the interesting and important part!)

A Gui Front End

With the structure adopted, it turned out to be relatively painless to wrap the objects with a gui front end. Technically, a new object `s_guidb` was constructed inheriting all the methods and properties of `s_db`. The design of the front end makes use of tabbed sub-forms to maximise the use of the screen. A tabbed sub-form is created for any choice of statistical method in `s_db` that allows the user to select appropriate variables to define the statistical object. An [Analyse] button creates the statistical object and then the statistical object's `Explore` method provides the grid object to allow the user to specify the output required. The example below shows the results of first opening the housedata in gui mode using

```
   1 Open 'oostats/housedata.adb'
```
and then:-

- selecting a one-way analysis of variance from the [Analyse] menu
- specifying the price of house as the observation value and the type of house as the grouping variable
- pressing [Analyse]
- pressing [Do Options] for the default options ticked.



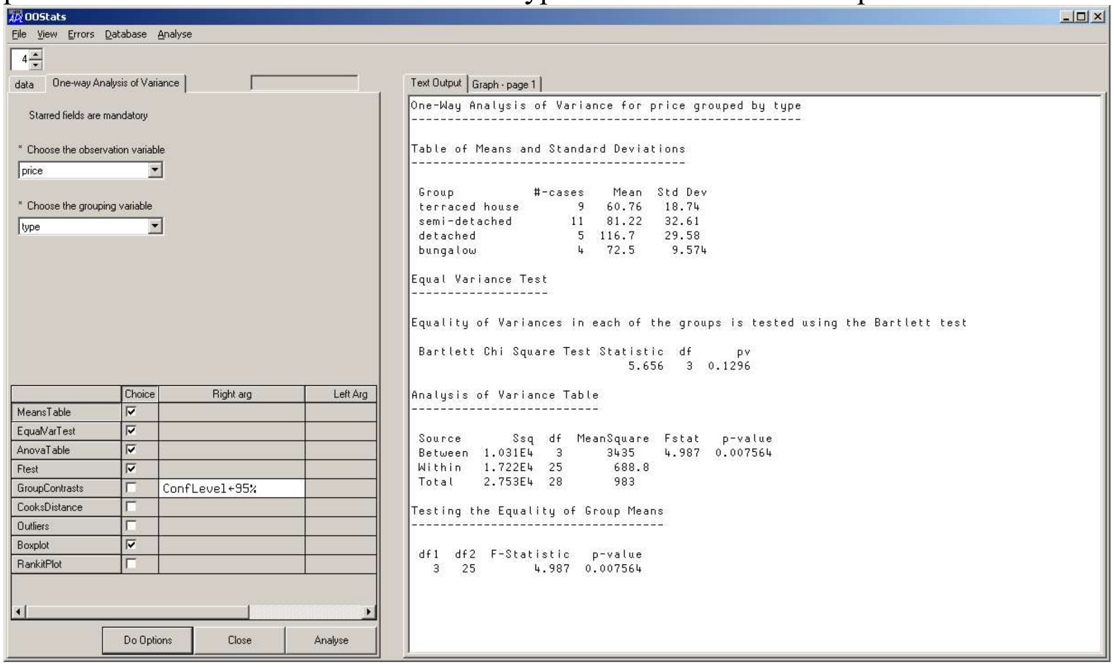
Starred fields are mandatory

* Choose the observation variable: price

* Choose the grouping variable: type

	Choice	Right arg	Left Arg
MeansTable	✓		
EqualVarTest	✓		
AnovaTable	✓		
Ftest	✓		
GroupContrasts		ConfLevel=95%	
CooksDistance			
Outliers			
Boxplot	✓		
RankitPlot			

The boxplot gives you some idea what to expect when you see the text output – that the average price of houses is not the same for each type of house. The text output confirms this:-


OOStats
File View Errors Database Analyse
4
data
One-way Analysis of Variance
Text Output
Graph - page 1
Do Options
Close
Analyse

Starred fields are mandatory

* Choose the observation variable: price

* Choose the grouping variable: type

	Choice	Right arg	Left Arg
MeansTable	✓		
EqualVarTest	✓		
AnovaTable	✓		
Ftest	✓		
GroupContrasts		ConfLevel=95%	
CooksDistance			
Outliers			
Boxplot	✓		
RankitPlot			

```
One-Way Analysis of Variance for price grouped by type
------------------------------------------------------


Table of Means and Standard Deviations
--------------------------------------


 Group            #-cases    Mean  Std Dev
 terraced house         9   60.76   18.74
 semi-detached         11   81.22   32.61
 detached               5  116.7    29.58
 bungalow               4   72.5     9.574

Equal Variance Test
-------------------


Equality of Variances in each of the groups is tested using the Bartlett test

 Bartlett Chi Square Test Statistic  df      pv
                              5.656   3  0.1296

Analysis of Variance Table
--------------------------


 Source       Ssq  df  MeanSquare  Fstat   p-value
 Between  1.031E4   3        3435  4.987  0.007564
 Within   1.722E4  25       688.8
 Total    2.753E4  28         983

Testing the Equality of Group Means
-----------------------------------


 df1  df2  F-Statistic   p-value
   3   25        4.987  0.007564
```

Other Facilities

Clearly it is impossible to demonstrate all the functionality of OOStats to date in this report. A help-file is in construction and should be available shortly. It will give examples of the other statistical analyses that are currently included in addition to those shown here (e.g. regression, two-sample means analysis, matched pairs, scatterplot modelling, generalized linear models).

Other facilities not mentioned are:-

- The output from the gui-wrap is saved in an APL component file and the results of a session may be printed out as a *Newleaf* report, complete with any pictures generated;
- The ability to edit the data (although I would stress that the editing facilities are basic);
- Formatting of variables allow you to display value labels rather than the values, or both;
- A variable may be coded (unique values are replaced by 1,2,3,… and associated original values stored as value labels – this is particularly useful for string variables;
- A vector of dates may be declared as a date variable, in which case, the values in the form dd/mm/yyyy are converted to Julian dates, but you can still see the actual dates if you require;
- An Excel spreadsheet may be converted using `{1} OpenExcel fn`

Acknowledgements

I would like to acknowledge the use of Adrian Smith's software *Rainpro* and *Newleaf* (now supported by ***Dyalog***) which is used for graphical and text output respectively. I would also like to acknowledge the encouragement of Morten, Gitte and Dan Baronet throughout the work done and also Vince for answering my gui queries so quickly.

Alan Sykes
September 2008