This manual is for a suite of programs called xtcc which is a tool to help in Market Research Analysis

XTCC - Cross Tabulations Compiler, User and D

Edition [No value for "e"] for Release [No value for "cde"]

Neil Xavier D'Souza

This manual is for a suite of programs called xtcc which is a tool to help in Market Research Analysis

Copyright © 2003,2004,2005,2006,2007 Neil Xavier D'Souza, 502, Premier Park, Orlem, Bombay, India 400064.

Permission is granted to ...

Table of Contents

1 Overview of XtCC

XTCC - Cross Tabulation compiler, is a set of tools for Market Research MR, companies to assist in Data Processing part of their day to day work. DP -> Data Processing and a people working and this department tend to be called DP(s). For an very brief overview of MR read mainly from a DP point read the chapter "Overview of MR". The main program is a compiler that takes user a user input program and applies the "edit" to repeatedly to every line of data, after which it tabulates the data based on the given tabulation spec.

The input program consists of 3 parts - an Edit section - containing transformations you would like to apply to the data, the Tabulation section marking the tables which need to be tabulated and the Axes section listing each individual table in the program.

2 Overview of MR

Basic workflow:

1. A client approaches an MR Company to research a particular topic
2. A researcher in MR company interacts with the client - together they work out a questionnaire which will help answer clients needs
3. The client commissions the study after which the questionnaire goes into field. The MR company looks at the target group, checks sample quotas that would make the information representable for the target group and does interviews - with (mostly) common people like you and me
4. The questionnaires are collected and deposited in the data entry department of the company
5. The Data processing department then gets the data punched by data entry programmers. There may be - pre or post cleaning of the data based on the choice of the MR company
6. While data entry is on a DP person starts preparing the analysis specs for the program. If necessary she may write a cleaning edit to ensure that the data meets valid logical and quota checks that were in the initial field plan
7. If there are open ended questions the DP or researcher - depending on whose responsibilty it is for that particular company will brief a Coder - person who looks at verbatim responses and classifies them for a statistical point of view
8. Once all the data is entered, the DP will generate tables on the data, run her edits on the data, resolve and logical errors with the researcher, generate tables, check them against raw data using hole counts and finally send the output of the tables to the researcher.
9. The researcher then studies the tables - looking for a specific pattern, and presents the findings to the client.

Although this might not be the way things happen in a particular company, it captures the essence of what goes on.

The main role of a DP is converting raw data into information for a researcher.

For a DP the main work revolves around

1. The questionnaire
2. The questionnaires from field
3. Giving a final set of 100% checked correct tables to the researcher

By Checking of tables against data most companies imply checking of tables against hole counts. In most MR companies, data is punched in an ASCII file. Every respodent has a unique identified the "Serial number". Depending the format in which the file is written, a respodent may have one or more "cards" per serial number. Each question asked by the Field is given a unique column number in the data file - the card number helps indentify a column number over a span of multiple lines for a particular respondent. A holecount is a frequency distribution, showing the data punched at each column number accumulated over all the respondents. Given this information it is possible to check tables against raw data.

For Example: if we know that Q1 is punched at column 59 in the data file - by looking at the absolute counts at column 59 and seeing the counts tabulated in Q1 - we can assert if Q1 is tabulated correctly by a DP.

So suppose Q1 happens to be Respondents Gender where code 1 = Male and Code 2=Female

The table output looks like: Q1 Gender Male 103 Female 110

The hole counts looks like code 1 code 2 Col 59 | 103 110

By looking at the questionnaire - we can see that Male - is represented by code 1 and Female by code 2, and looking at the table and raw data - we can say that the information tabulated is correct.

3 Scope for Computer assistance in DP

1. Computer assistance in Managing data files and formats, including data files created on different machine architectures
2. Powerful editing language for data manipulation
3. Conversion of ouput tables into various different formats
4. Assistance in table checking
5. Assistance in creation of initial edits
6. Assistance in creation of axes, and tabulation setup

4 Design goals

First a little background. Before I joined the MR industry as DP I was a C /C++ programmer. The main package used in all three organizations I worked in was quantum, now an SPSS-MR product. The reason I speak about quantum below is that I am keeping a similar overall structure, athough not being compatible.

Here are some positives for me about quantum

1. It compiles code to C - which then gets compiled to a final program - so the programs are really fast whent they execute.
2. Good basic concepts for generating tables - the logical unit of axes , and applying cross products of 2 or more such axes to get various combination of tables is really very ingenious and powerful.

On the Other hand I felt there are some key design issues at both macro and micro level

1. Restrictive syntax in the edit section - i.e. data manipulation language
2. Usage of labels in loops and conditional branching -> in fact the edit language encourages use of goto
3. The edit language does not seem to be a formal grammar parser, the if else if else does not seem to work like in a normal computer language. Continuation lines have the start with a "+" sign at the beginning of line.
4. Restriction to 199 characters per line of edit/ axis
5. Weak support for subroutines - you cannot call one subroutine from another. No type checking done for subroutine arguments
6. Lack of precedence for normal operators in expressions
7. Restrictions on number of lines an edit section can have
8. Bad error reporting - sometimes it does not catch the correct place an error has occurred.
9. In consistent usage of small case and large case tokens
10. Lack of block comments - there are tricks to get this but basically hte internal language does not support it
11. Bad usage of native data type - most computers that run quantum are 8 bit =1 byte architecture - yet by design they have kept 12 punches - so that multicoded data has to be stored in 2 bytes of which 4 bits can get wasted. Not only that - this requires a level of indirection while reading data slowing down a potentially fast program. No doubt this was done so that a data file could be visually viewed in a editor for most of the data, but this does not work for multi-punch data. Data is stored ASCII chars, floating point and integer data is stored in actual visible decimal characters, severely limiting the amount of data that can be stored for a particular number of characters reserved for a question. For example if Q10 was the annual income of a Company and you reserved 8 characters for this information, the max value can be '99999999'.
12. Bad keywords - for example n01 -> represents count , n10 represents total for a basing computation n12 stands for mean etc. Difficult learning curve
13. Forced syntax for programmer -> all axes statements have to be anchored at the start of a line.

When I decided to design my own package I had to decide about compatibility. But, some of the flaws above are unacceptable, so I have decided to not maintain compatibility - at least not for now. Here are the design goals laid down.

1. Speed, Performance, Powerful edit language, Logically named keywords, strong error reporting:

   Fast, desgined for performance compiler, with native data types as close to the underlying architecture as possible, C like language with no restrictions on program size, syntax, good error reporting and normal language looping and branching statements, with functions and functions prototypes and strong type checking. Also automatic conversion of raw data to the correct type by looking at data type on LHS of assignment operator.

   Basic work on this is already relatively - I have a ready compiler which does meet most of the above requirements, and generates a file for tables which can be parsed further for different types of output formats

2. Data file portability:

   Advisory, self contained binary data file, again as close to underlying machine architecture. If the file is sent to another computer for processing, the file itself contains enough information to be able to determine if conversion is required or data is compatible. The system should be extensible so that people can easily add converters to the suite of programs to convert from one architecture to another.

   To achieve this, I am going to spec out a data format which will have all the above capabilities

3. Automatic Creation of Basic axes:

   The data file should also contain an embedded logical map. The logical map should contain logical question names for the data in the questionnaire, along with stub-level information if any. This will allow for another program to create initial data edits and axes setup for data present in the map. This data is not compulsory, but if present will allow for other tools to work on it, we should not have errors being thrown as a result of lack of this information. In particular the data file has information embedded in it so that the program can determine where the actual data starts so that it can process the data independent of the metadata information.

4. Layout independence:

   The program should hide from the DP programmer data column nos as far as possible, yet if the DP so desires can have raw access to the raw data, overriding system generated defaults. The program should have assistive tools which automatically create logical variables for each question. These variables are autmatically assigned the correct column nos from the data file. The DP programmer uses the logical variables in the axis setup, but can modify variables as and when she wants, because there are normal native variables. In explaining this, what Im trying to do is introduce a logical layer between the raw data and the edit a DP writes. This allows for data independence, in a way that even if a layout changes, only the column number assignments would have to change to maintain the program. This means layout independence

5. Different output formats for generated data:

   This can be achieved through a program which reads the intermediate ouput data and then applies converters which the right parameters as necessary.

6. Another important design goal is for a person to be easy to join in the development effort. This means for allowing for example - people who dont know grammars to be able to contribute by demarcating long term design decisions, anticipating well in advance, where slightly advanced programming is required, and if possible laying out framework / algorigthms to serve as guide. Not only this, but the entire package should comprise of a series of co-operating programs, aware of each others capabilites and achieving the common goal through co-operation. This way someone else can develop a black-box application, just knowing what her inputs, outputs should be; programs remain relatively smaller, and therefore relatively easier to manage (and hopefully more bug free). It will also allow programming happen in parallel.

5 Requirements

6 Installing and building the compiler and associated tools

7 Testing suite

8 Overview of XtCC Compiler

9 Detailed user manual for each individual program

10 Tutorial and Reference documentation of a person who wants to get in and work in developing this software