

# ExaStat Census Demo

Interactive Visualization of Huge Data Sets: A Demo Using the 2000 U.S. Census

Lee E. Edlefsen

ExaMetrix, Inc.

Updated: January 30, 2007

Introduction.....	2
System Requirements.....	5
Installation.....	5
Starting R.....	5
Starting and Using ExaStat's Interactive Mode.....	6
How to Use the Code in this Document.....	7
Specifying the DataFile to Use.....	7
Specifying Servers to Use.....	8
The U.S. Age/Sex Distribution in 2000.....	9
The Variables "age" and "sex".....	9
Descriptive Statistics Using the Interactive Analysis Functions.....	10
Computing the Age/Sex Distribution.....	11
Graphing the Age/Sex Distribution in R.....	12
Wage Income by Age and Sex in 2000.....	15
Income, Age, and Sex.....	15
Weeks Worked.....	18
Average Wages per Week.....	19
Income Quantiles by Age and Sex.....	22
Income, Age, Sex, and Education.....	23
Adding Race.....	26
Adding Marital Status, Number of Children, and Disability.....	28
Marital Status.....	28
Number of Children at Home.....	29
Work Disability Status.....	29
Adding Race and Marital Status to the DataFile.....	29
Regression Models of the Joint Effects.....	30
Appendices.....	32
Using Multiple Computers.....	32
About the IPUMS Census Data.....	34
Creating New Data Files from the Census Data.....	35
ExaStat's Formula Notation.....	37
Cubes and Cube Regressions.....	38
References.....	40

## Introduction

The objective of this demo is to show how it is possible to use ExaStat, along with R, to do exploratory data analysis on huge data sets. The data set used as an example is the 2000 U.S. Census 5% IPUMS sample [1], which has 14.5 million rows, 265 variables, and is about 17 GB in size (this data set is discussed in more detail in the appendix [About the IPUMS Census Data](#) at the end of this document).

ExaStat is used to process the data, to estimate models, and to put the results into a form suitable for graphing. R is used to display the results as Trellis plots. The results are transferred from ExaStat to R in the form of tab delimited text files.

Trellis plots were developed by William Cleveland [2] as an approach to visualizing multivariate data, and are ideally suited to visualizing relationships in a huge data set such as the Census. Most obviously, they allow visualization of large amounts of information of the type that has historically been produced by the Census bureau in the form of printed tables. ExaStat makes it possible to rapidly compute much more detailed tables than are provided by the Census bureau and it also allows the use of regression modeling techniques to be used to “control” the elements of the tables for the effects of covariates. Much more is possible as well, such as the visualization of income quantiles by age and sex that is shown below. I am not an expert on Trellis plots, and the plots shown in this document are relatively rudimentary. It is a tribute to the power of Trellis plots, and of their implementation in R, that just a few lines of R code can produce very illuminating graphs summarizing huge amounts of information.

The example topics in this demo are the age/sex distribution and the relationship between female and male wage income. These topics have been chosen because they are interesting, because the Census is a good source of information for examining them, and because they are useful for illustrating techniques for using ExaStat and R to interactively visualize and explore a huge data set.

What makes it possible to do these analyses “interactively”?

1. Speed – the ability to do the required computations rapidly.
2. Capacity – the ability to handle huge models, with thousands or even tens of thousands of parameters and with numerous singularities.
3. Convenience – the ability to specify models and to obtain the results in a form suitable for visualization, relatively easily.

By “interactive” I mean roughly that it is possible to sit in front of a computer, issue commands, and get the results without getting bored. The objective for this demo has been to keep the computation time for each of the steps of the analysis under 3 minutes on a 3GHz single processor laptop with 512MB of RAM and a slow hard drive. A couple of the steps which create or add to data files take about 5 minutes on that computer, but most of the steps take less than a minute. On a modern dual processor workstation with a

fast hard drive, these times are reduced by a factor of 2 to 4. Distributing the computations across computers reduces the time roughly in proportion to the number of computers.

Interactivity also means that it is possible to issue a small number of commands to achieve the desired result. This document contains the code for doing computations and for creating data extracts in ExaStat and for creating the graphs in R. It is expected that you will copy and paste this code into the command lines of ExaStat and R, but it would not be too burdensome to have to type it.

It should be pointed out that all of the ExaStat computations in this demo can be done in one pass through the data, and that doing so takes only about 4 minutes on the 3GHz laptop. This is because ExaStat allows multiple models to be computed simultaneously, and because it pre-analyzes the models jointly to figure out how to minimize the computations. Showing this is beyond the scope of this demo, which focuses on a limited type of interactive computations, and which illustrates only a small fraction of the capabilities of ExaStat.

The general approach described in this document is to:

1. Work with the raw data, at least initially; select observations and create and transform variables as needed. ExaStat can be used to create new data sets with a subset of observations and with new variables, and can do this rapidly, but doing so inhibits making changes to the selection criteria and to the way variables are defined and transformed.
2. Compute summary statistics within categories, breaking continuous variables into intervals. Doing this allows the shapes of relationships to be visualized.
3. Fully interact categories as much as possible so that the nature of interactions can be visualized.
4. Use regression techniques to account for the effects of covariates when cell counts get too small.
5. Graph the results using R's Trellis plots, which are ideally suited to visualizing multivariate relationships.

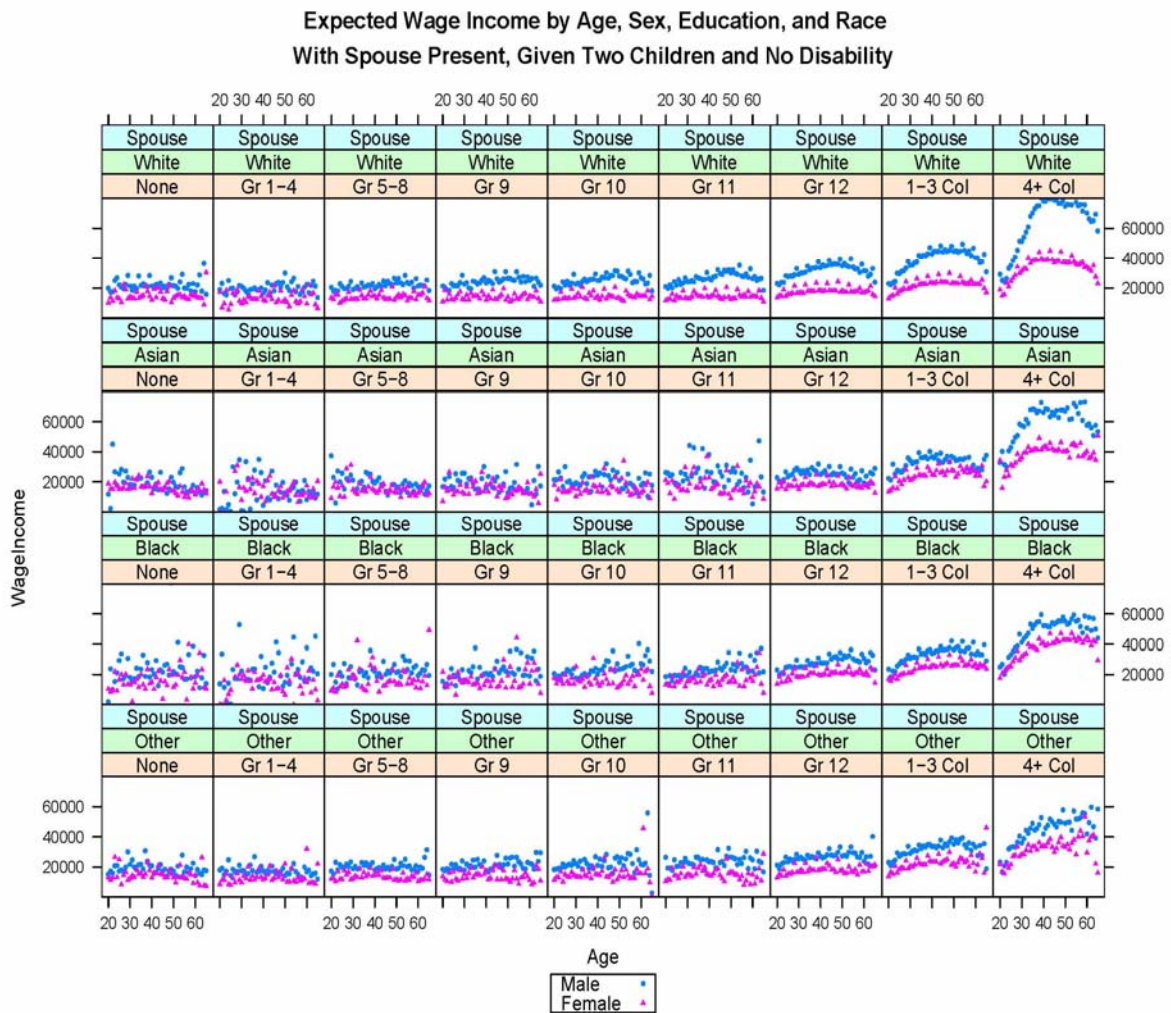
The demo is relatively long. I have tried to give enough detail about what is going on to make it possible for you to be able to explore the Census data, or other huge data sets, on your own.

To give you a feeling for the types of plots that are created in this demo, I have included one below. This is derived from a large regression model (with 7,912 parameters) that is estimated and evaluated in the last part of the demo. The plot shown below is the second page of the graph created by the following code (I recommend waiting until you have everything set up before trying to execute this code):

```

IASetDataSource("CensusWageExtract");
Analysis ao = plotwagemodel("incwage ~
Age:sex:educrec:Race:MarSpPres + nchild:Age:sex +
disabwrk:Age:sex", "WageASERM_ND21", "nchild = 2, disabwrk
= 'No disability that affects work'", "Expected Wage Income
by Age, Sex, Education, and Race\nWith Spouse Present,
Given Two Children and No Disability");

```



This is a set of scatter plots of expected annual wage income by age and sex, and it contains a large amount of information. Each row of panels represents a racial grouping, and each column of panels represents an educational attainment level. Thus, the top row corresponds to “Whites” and the column on the right corresponds to people with 4 or more years of college. Within each panel, expected wage income is graphed against age (for ages 20 to 65) for both males and females. The blue symbols represent males and the red symbols represent females. This page of the plot represents people with a spouse present, 2 children, and no work disabilities. The reason the plots look “fuzzier” for lower levels of education is that there are relatively few people who fall into those categories.

I would appreciate feedback of any type about this demo, at [censusedemo@exametrix.com](mailto:censusedemo@exametrix.com).

## **System Requirements**

- At least one Windows PC. The demo has only been tested on Windows XP, but should work on NT.
- Multiple networked PCs may be used for distributed computations. Multiple processors on any of the computers will be used if available.
- The demo can probably be run on a 256 MB computer as long as not many other programs are running.
- Enough disk space to hold at least CensusDemoData.xdf (627 megabytes), plus R (54 MB), plus ExaStat (60 MB). (It is possible to split CensusDemoData.xdf into smaller pieces, each of which is placed on different computers; this is discussed in the appendix [Creating New Data Files from the Census Data](#))

## **Installation**

1. Install ExaStat on the client and on any servers (see the appendix [Using Multiple Computers](#)). The ExaStat “binaries” are available at <http://www.exametrix.com/downloads/>.
2. Install the CensusDemo and download and unzip one or more of the Census data files (CensusDemoData.xdf, CensusIp20001.xdf, or CensusWageExtract.xdf) into the CensusDemo directory on the client and on any servers. They are available at <http://www.exametrix.com/downloads/>.
3. Install R on the client. R is available at <http://www.r-project.org/>.

Some of the graphs are displayed in PDF files, so you must have Adobe Reader or the equivalent on the client computer.

## **Starting R**

Before you start the demo, you will want to start R. To do so, you can click on the “Start R” shortcut in the Program grouping “ExaStat Census Demo”, or double click on the shortcut “R Census Demo” that is installed in the CensusDemo directory (you can also just double click on the .RData file in this directory). This will start R with the working directory set to be the CensusDemo directory, and a few functions will be loaded that are used in the demo.

You should see R’s command window, with the prompt:

>

When you copy and paste R code from this document, you will paste it on this command line.

## ***Starting and Using ExaStat's Interactive Mode***

There are three general ways to use ExaStat's functionality:

- **Interactive Mode:** ExaStat is integrated with an open source C++ interpreter (Cint), and ExaStat code can be executed using the Cint command line.
- **Text Editor,** such as the free Crimson Editor or emacs: ExaStat code can be written using any editor and then executed using ExaStat's integrated Cint interpreter. The Crimson Editor and emacs can easily be customized to "run" ExaStat code in this way.
- **Visual Studio:** Visual Studio can be used as both an editor and a compiler for ExaStat code.

This demo illustrates using ExaStat's interactive mode to explore the Census data. To start ExaStat in interactive mode, you can click on the "Start ExaStat" shortcut in the Program grouping "ExaStat Census Demo," or double click on the "ExaStat Census Demo" shortcut in the CensusDemo directory.

Starting ExaStat in this way loads some code (in the file CensusDemoFns.cpp) into the interpreter for use during the demo and also sets the appropriate command window properties.

A word of warning: The "QuickEdit Mode" property of the Windows command window is turned on, which makes it possible to paste text into the window simply by right clicking, and also makes it possible to select an area of the window for copying simply by clicking. Unfortunately, if you accidentally click in the window, you will create a selection, which freezes execution of code until you press enter. If you want to turn QuickEdit Mode off, right click at the top border of the window and choose "Properties" at the bottom of the menu. On the "Options" page, deselect QuickEdit Mode. At the same time, you may want to permanently change the size of the window to something appropriate for your screen. You can do so on the Layout page (at the same time, make sure that the Screen Buffer Size Width and Height are at least 220 and 5000, respectively). When you click OK on the dialog, select "Save properties for future windows with same title."

When the interactive mode starts, you should see the copyright notices for ExaStat and for the Cint interpreter, followed by the prompt '>' .

You can enter and execute ExaStat (C++) code at the prompt. There are also special interactive mode "dot" commands (preceded by a '.'). To see help on these commands you can use `.help` (however, almost all of these commands are CINT-specific, and are not needed for this demo).

Also, refer to the document *Introduction to Programming in ExaStat*, which is installed in the ExaStat\Doc directory.

To exit the interactive mode at any time, use `.q`. You can use Ctrl-C to stop the current function execution.

## ***How to Use the Code in this Document***

In the following demo, it is expected that you will copy code from this document and paste it in into the interactive command lines of ExaStat and R.

There are 3 types of code shown in this document. In order to distinguish them, different fonts and colors are used for them:

- ExaStat code that you are expected to copy, paste and execute:

```
IAShowInfo( );
```

- ExaStat code that you are not necessarily expected to execute (and may not be executable because it is only partial code or it references variables that are not defined at that point):

```
FactorVar Age = CreateFactor( age-20, seq(20,65,1) );
```

- R code that you are expected to copy, paste and execute:

```
AgeSex <- read.table('AgeSex.txt', header=TRUE)
```

## ***Specifying the DataFile to Use***

The default is that the file `CensusDemoData.xdf` will be used for the first part of the demo. This file has all of the rows of the full file (`CensusIp20001.xdf`), but contains only those variables used in the demo. It is about 630 MB in size.

You can also use the full IPUMS census **DataFile**, `CensusIp20001.xdf`, which is about 17 GB. Computation time is not significantly affected by which of these two files you use, because ExaStat only reads the data it needs.

Another file that can be used is `CensusWageExtract.xdf` file. This is about half the size of `CensusDemoData.xdf`. This file is restricted to people between the ages of 20 and 65 who worked at least 20 weeks in the year prior to the Census, and it contains some new variables would otherwise be created during the course of the demo. Computation is faster when this file is used, because the data selection and the new variable creation have already been done. To specify that you want to use this file (if you have downloaded it and unzipped it into the `CensusDemo` directory), use the following command:

```
IASetDataSource("CensusWageExtract");
```

If you use this file, the results discussed below that apply to all observations in the Census will of course be different. Also, the code that selects observations and creates new variables should be omitted.

To see a summary of the default data source, copy and paste the following code to the ExaStat command line, and press enter:

```
IAShowInfo();
```

The semi-colon at the end of the line is optional in interactive mode. The output from this function is explained below.

The “IA” prefix on the functions `IASetDataSource()` and `IAShowInfo()` stands for “Interactive Analysis.” There are several such functions that are used in this demo. As the prefix indicates, these functions are designed for doing interactive data analysis. These functions allow the user to specify the name of a data source and some other settings, and then to use ExaStat’s high level analysis capabilities interactively. They are thin wrappers around underlying object oriented `Analysis` class methods. These functions are declared in the header file `InteractiveAnalysis.h` (which is installed in the ExaStat directory) and are defined in `InteractiveAnalysis.cpp` (which you will have only if you have installed the ExaStat code tree). You can look in `InteractiveAnalysis.h` for documentation of these functions.

All of the analyses are weighted by the variable `perwt` (see the appendix [About the IPUMS Census Data](#)). Use of this weight has been specified at startup by the command (which you do not need to execute):

```
IASetWeightName("perwt", "pweight");
```

You may be able to speed up the computations substantially by changing the amount of data ExaStat processes at one time. ExaStat reads data in blocks of rows and columns, and the default is to read one block at a time, which means that for all of the computations discussed below it is processing less than about 2 megabytes at a time. If you have enough memory, and especially if you have multiple processors, you can benefit by reading more blocks at once. For instance:

```
IASetBlocksPerRead(50);
```

## ***Specifying Servers to Use***

ExaStat can distribute the computations in this demo across multiple computers. If you are interested in doing this, read the appendix [Using Multiple Computers](#) in this document. Once you have followed the instructions there, you can specify which computers to use with a statement such as the following:

```
IAUseServers("server1, server2", "10, 20");
```

# The U.S. Age/Sex Distribution in 2000

## *The Variables “age” and “sex”*

We will start by looking at the IPUMS variables `age` and `sex`. Note that the names are lower case and that ExaStat is case sensitive. The IPUMS web documentation uses upper case for all variables. The IPUMS documentation for `age` can be found at:

<http://www.ipums.umn.edu/usa/pdemographic/agea.html>.

To get some information about these variables in the ExaStat Census **DataFile**, copy and paste the following code to the ExaStat command line and press enter:

```
IAShowInfo("age, sex");
```

You should see something like the following (except, of course, for the specific file name and path):

```
Name: C:\ExaStat\CensusDemo\CensusIp20001.xdf
Number of rows: 14583731
Number of variables: 265
Number of blocks: 487
```

Column Information:

```
Col 1: 'age', ('Age'), Int Codes
Col 2: 'sex', ('Sex'), UInt (Max/Min=1,0) Labels Codes
```

The first part of the display gives information about the entire data file. The “Number of blocks” gives some information about the way the **DataFile** is arranged. Each column is stored in blocks of rows. In this file there are 487 blocks for each column, which means there are a little less than 30,000 rows in each block. The most efficient way to read from a **DataFile** is by block, and this the default. Thus, when processing data, the default for this **DataFile** is to process it in “chunks” of about 30,000 rows at a time.

The “Column Information:” section gives the name, the description, if any, the data type, and maximum and minimum values, if they have been set. Thus, for the variable `age`, its description is just (`'Age'`), and it is stored as an **Int**, which is a signed integer. The variable `sex` is stored as a **UInt**, which is an unsigned integer (no negative values), it has a maximum value of 1 and a minimum value of 0, and it has both Value Labels and Value Codes. For the Census **DataFile(s)**, most continuous (cardinal) variables are stored as **Ints**, and categorical variables are stored as **UInts**.

To see more information, use the following statement:

```
IAShowInfo("age, sex", true);
```

Setting the second argument to 'true' causes any "Value Info:" to be displayed. This information consists of value labels, value codes or both. The `age` variable has numeric value codes (these are codes are part of the IPUMS variable information for `age`), while `sex` has both codes and labels (the latter being "Male" and "Female"). ExaStat allows both value labels and value codes to be attached to any variable.

## ***Descriptive Statistics Using the Interactive Analysis Functions***

Next, let us compute descriptive statistics for these variables. We will use the function `IADStats()`. The first argument of this function is a "formula" describing the model (using syntax that is similar to the formula notation in R and S-PLUS; see the appendix [ExaStat's Formula Notation](#)). There are four additional, optional, arguments:

- The second argument allows selection of specific observations for the analysis (for example, `"(age >= 20) && (age <= 65)"`).
- The third argument allows for the declaration and definition of new variables.
- The fourth argument allows for arbitrary "transformation" statements.
- The fifth argument allows specification of a name for the analysis.

For the time being, we will omit those additional arguments.

Copy the following statement to the command line and press enter:

```
IADStats("age + sex + age:sex");
```

You should see the following results (these have been edited to fit on the page):

```
Rows Processed: 14583731
```

```
Descriptive Statistics for:
```

```
age + sex + age:sex, pweight = perwt (Total Obs: 14583731, Missing: 0)
```

```
-----
```

Name	Mean	StdDev	Min	Max	SumOfWeights
age	35.8221	22.3681	0	93	281111530
sex	0.5101	0.4999	0	1	281111530
age:sex	35.8221	22.3681	0	93	281111530

```
Cell counts for:
```

```
sex
```

```
Male                137711402
Female              143400128
```

```
Statistics by category (2 categories):
```

Category	Mean	StdDev	Min	Max	SumOfWeights
age for sex=Male	34.4790	21.5936	0	93	137711402
age for sex=Female	37.1118	23.0138	0	93	143400128

“Rows Processed” indicates the number of actual rows of the **DataFile** that have been processed. There are 14,583,731 rows in the **DataFile**. This field is updated during the computations to show progress (rows are processed in blocks of about 30,000 rows each for this **DataFile**). The “SumOfWeights” is the sum of the weight variable **perwt**, and is an estimate of the number of people in the U.S. at the time of the census (281,111,530).

Descriptive statistics are computed for the 3 terms: **age**, **sex**, and **age:sex**. The latter is the interaction between the age and sex variables. The variable **sex** is categorical, with two levels “Male” and “Female.” Thus, it is like a “factor” variable in R. The variable **age** is a continuous variable. It is stored as an integer with values in the range from 0 to 93. The mean, standard deviation, min and max of the three terms are presented in a table, and then additional results are presented for each categorical variable. For the categorical variable **sex**, the weighted counts within each of the categories are given. Thus, the estimate is that there were 137,711,402 males and 143,400,128 females in the U.S. at the time of the census.

For the term **age:sex**, descriptive statistics are presented for the continuous variable **age** within each of the sex categories. Thus, the estimated mean age of males is about 34.5 years and for females is about 37.1 years.

You will probably notice if you re-run this statement that it takes much less time to execute the second time than it did the first time. The main difference is that the operating system accesses the data much faster the second time; this is aided by the way in which ExaStat stores the data in the file. The difference is more pronounced on systems with slower hard drives.

## ***Computing the Age/Sex Distribution***

Next, let us obtain the age/sex population distribution for each year of age. The easiest way to do this is just to tell ExaStat to treat **age** as a categorical variable, with each year of age treated as a separate category. This approach will work because the minimum and maximum values for this variable are known to ExaStat ahead of time (through its value codes).

One way to designate that a continuous variable should be treated as categorical is by putting ‘@’ in front of the name in the formula. Thus **IADStats("@age:sex")** will give us the desired results. However, instead of using the function **IADStats()**, let us use the function **IACube()** instead. This will compute the counts within each cell of a “Cube,” which in ExaStat is a type of multi-dimensional array. Note that a “Cube” object is really a hyper-cube, in that it can have more than 3 dimensions. Cubes are discussed further in the appendix [\*Cubes and Cube Regressions\*](#).

Thus, we can use:

```
Analysis ao = IACube("@age:sex");
```

When you execute this statement the weighted counts of males and females in each age group from 0 (age less than 1) to 93 are computed and displayed. For instance, the estimate is that there were 1,932,061 males and 1,853,728 females less than 1 year old at the time of the census. Note that age 89 is the last age for which accurate numbers are given.

Note also that there are more males than females at every age up to age 34, with the reverse being true thereafter, reflecting the fact that more males than females are born, but males die at a faster rate at every age (these numbers also reflect migration, of course).

## ***Graphing the Age/Sex Distribution in R***

While a table like this is nice, it would be much better to also see a graph of the results. ExaStat does not have any graphics, but R has excellent graphics. Therefore, we will export the results to R, and use R to create some graphs.

To do this, we need to get the results into a form suitable for graphing. This will require a few lines of code.

The return value from `IACube()` is an **Analysis** object, which is a high-level object that knows how to compute and store the results for several types of models. Most of the other interactive analysis functions, such as `IADStats()`, also return an **Analysis** object. We can use the following statement to extract the results from the **Analysis** object and put them into a **DataSet**:

```
DataSet ds = ao.GetAsDataSet("CohortSize");
```

In the **DataSet** "ds" there is a variable corresponding to each of the cube dimensions, and each row of the **DataSet** corresponds to a cell of the cube. In this case there will be variables named `@age` and `sex`, plus a variable corresponding to the cell counts. The default name of the counts variable is "Counts," but a different name can be specified in `GetAsDataSet()`. This variable is called "CohortSize" here.

To see what the **DataSet** ds looks like, use the dot command `".?"`:

```
.? ds
```

The `".?"` is a special interactive mode command that displays the object (it is equivalent to `cout << ds`).

These results correspond to the standard type of table that we associate with Census data. You will notice that when the **DataSet** is displayed, information about each column is shown first, before the data table is printed.

To clean things up a little bit, rename `@age` as `Age`.

```
ds.RenameVar ("@age", "Age");
```

Let us also include only those ages below 90:

```
ds = ds[ ds["Age"] < 90 ];
```

In this statement, `ds["Age"]` represents the column `Age`. A `DataSet` is an array of variables (columns), and so the indexing `[]` operator can be used to access its elements. Since `Age` is the first column, we could also access it by using `ds[0]` or `ds(1)`. (See *Introduction to Programming in ExaStat* for further information about indexing.)

The expression `ds["Age"] < 90` results in a `LogicalVar` with a 1 (true) in each selected row and a 0 (false) in all other rows, and when that is passed to the indexing operator `[]` the selected rows are placed in a new `DataSet`, overwriting "ds."

Now, let us create a new variable `Year` denoting year of birth:

```
ds("Year") = 2000 - ds["Age"];
```

Note that we are using the indexing operator `()` on the left hand side of this statement. That is because it will create the variable `Year` if it does not exist, while the `[]` operator will not.

As a final step, remove the value labels from the variable `sex`. This will make it easier to control what the graph looks like in R.

```
ds["sex"].SetValueLabels();
```

Calling the method `SetValueLabels()` with no argument removes any existing labels.

Now, write the data to a text file that can be imported into R for graphing:

```
ds.WriteToTextFile("AgeSex.txt");
```

This creates a tab delimited file with column names in the first row. Since no explicit directory is specified, it will be written to the default data directory, which is the "CensusDemo" directory. The default data directory was specified to be the `CensusDemo` directory when interactive mode was started. You can also set it explicitly by calling `SetDefaultDataDir("/ExaStat/CensusDemo")`. Also, the extension ".txt" could have been omitted as well, since that is the default text file extension.

To graph the age distribution data, open R, and paste the following R code on its command line (make sure you have set the working directory in R to the Census Demo directory, such as \ExaStat\CensusDemo; this is done automatically if you start R using the R shortcut installed with the Census Demo):

```
AgeSex <- read.table('AgeSex.txt', header=TRUE);
AgeSex$sex <- as.factor(AgeSex$sex);
levels(AgeSex$sex) <- c('Male', 'Female');
require(lattice);
xyplot(CohortSize~Age, data=AgeSex, groups=sex,
scales = list(x = list(at=seq(0,90,5))),
main='2000 U.S. Cohort Size by Age and Sex',auto.key=TRUE);
```

Note: on some computer screens the default Trellis plot symbols are difficult to see. The following R code changes the symbols (this code is also in the file TrellisSymbols.r, which you can source in):

```
sup.sym <- trellis.par.get("superpose.symbol")
sup.sym$pch <- c(16,17)
trellis.par.set("superpose.symbol", sup.sym)
```

Expand the graph to full screen. Note the greater number of males at younger ages and of females at older ages, as mentioned above. Note also the bulge in the middle of the graph corresponding to the baby boom, and the echo (or perhaps echos) at younger ages. Note also the large drop in cohort size for both men and women at the leading edge of the baby boom (between ages 53 and 54 – it helps to refer to the numeric results) followed by a slight rise and then, for the most part, a steady decline. The big exception to the steady decline is the sharp drop in cohort size for women between ages 64 and 65 (but a sharp rise for men between those two ages), followed by a six year rise for women. (It would obviously be helpful to have grid lines on this graph. I have not been able to get them positioned properly, which is probably due to my unfamiliarity with the `xyplot` function.)

The size of each cohort in the year 2000 represents historical trends in birth, death, and migration rates. To think about the results historically it is useful to graph cohort size against year of birth. To do this, execute the following code in R:

```
xyplot(CohortSize~Year, data = AgeSex, groups = sex,
scales = list(x = list(at=seq(2000,1910,-10))),
main='2000 U.S. Cohort Size by Year of Birth and
Sex',auto.key=TRUE);
```

Note again the sharp decline in cohort size for females born during the years 1928 through 1935 (ages 72 to 65 in 2000).

These two graphs can also be put into a single PDF file and viewed using the Adobe Reader. The file "AgeSex.r" contains the code to do this, and it can be run from within ExaStat by executing the following code:

```
ExecRFile("AgeSex.r");
```

This may take a little while. When the Adobe Reader opens up, press Ctrl-0 (Ctrl-Zero) to size the graphs to fit the window. Note that there are two pages, with a graph on each page.

To summarize, the following code has been used to compute the U.S. age/sex population distribution in the year 2000, and to graph the results in R:

```
Analysis ao = IACube("@age:sex");  
DataSet ds = ao.GetAsDataSet("CohortSize");  
ds.RenameVar("@age", "Age");  
ds("Year") = 2000 - ds["Age"];  
ds = ds[ ds["Age"] < 90 ];  
ds["sex"].SetValueLabels();  
ds.WriteToTextFile("AgeSex.txt");  
ExecRFile("AgeSex.r");
```

## Wage Income by Age and Sex in 2000

This part of the demo looks at the relationship between wage income, age, and sex in the year 2000, and explores how variables such as education, race, and marital status, number of children, and disability affect that relationship.

### *Income, Age, and Sex*

Execute the following code:

```
ao = IACube("incwage~@age:sex", "wkswork1 > 20");
```

The IPUMS variable `incwage` (<http://www.ipums.umn.edu/usa/pincome/incwagea.html>) is pre-tax wage and salary income for the previous year, and `wkswork1` (<http://www.ipums.umn.edu/usa/pwork/wkswork1a.html>) is the number of weeks worked. Both are continuous variables that are stored as integers. Thus, we are selecting those people who worked more than 20 weeks during the previous year.

In the formula "`incwage~@age:sex`", `incwage` is treated as a dependent variable, and `IACube()` will compute the weighted average wage income for each age for each sex. The counts of the numbers of people in each category (that is, the sums of `perwt`) are also computed and displayed. An `Analysis` object is again returned, and overwrites

the existing object “ao” (actually, the interpreter does not require that you specify that “ao” is an **Analysis** object, since it knows that this is the type of object returned by `IACube()`; C++ compilers are not so forgiving however).

All age groups have been included in this computation (as long as people are working enough). If we want to explicitly restrict our analysis to people of certain ages, we could do so in the selection statement. However, in computing this cube, it is equivalent and faster to select for the ages we want after we have done the computations.

Let us use the same formula and compute a regression. We could use the `IAREg()` function to do this, but it is somewhat faster to use `IACubeReg()`. In ExaStat, a “Cube regression” is a regression in which the first independent variable must be categorical (see the appendix [Cubes and Cube Regressions](#)). The computations take advantage of this fact, and the regression is also computed without an intercept. Furthermore, if we use `IACubeReg()` we can call the method `GetAsDataSet()` to extract the cube (table) of results in a `DataSet`.

Let us also explicitly exclude ages outside the range from 20 to 65:

```
aoReg = IACubeReg("incwage~@age:sex", "(wkswork1 > 20) &&  
(age >= 20) && (age <= 65)");
```

Now we get standard regression output. The regression coefficients are, of course, equivalent to the results we obtained with `IACube()` (that is, they are the mean of `incwage` in each category), because we do not have any other independent variables in the regression. We get estimates of the standard errors of the coefficients, as well as t statistics and p values, and the cell counts are also displayed. Note that these standard errors are computed using the typical assumption that the error terms are independent, which is not a valid assumption for the Census sample (see the appendix [About the IPUMS Census Data](#)); the standard errors are probably underestimated.

In computing a regression, the **Analysis** object pre-analyzes the model to detect and deal with structural collinearities (singularities), and will also detect computational collinearities when it is processing the results. Columns are dropped to remove the collinearities, and are marked as “dropped” in the display of results.

Note that all age categories are displayed, even though some of them have been excluded by the selection criteria, and some would in any case be excluded from the regression because there is no one in those age categories with any wage income. Since we have told ExaStat to treat age as categorical, it will display all categories. In this case, the variable `age` has value codes, and ExaStat reads those to determine the categories.

We can exclude the empty age groups by creating a new variable `Age` (using a capital ‘A’ to distinguish it from `age`). Since we are excluding all age groups below 20 (that is, ages 0 through 19), subtracting 20 will give us a variable whose value is 0 for age group 20, and 45 for age group 65. If we assign value labels “20, 21, ... 65” to this new

variable, we will have what we want. The following ExaStat code thus creates the variable **Age** from the variable **age** (do not try to execute this code on the command line, however, because the variable **age** does not exist from the point of view of the command line interpreter):

```
FactorVar Age = CreateFactor( age-20, seq(20,65,1) );
```

The function call `seq(20,65,1)` creates a numeric vector containing the sequence from 20 to 65 by ones, and this will be converted into a **StringVar** in `CreateFactor()`. To create the new variable **Age** so that it can be used in an analysis (ExaStat does not allow expressions or function calls in a formula), we can put this code in quotes as the third argument in the interactive analysis functions. In ExaStat, **FactorVar** is just a synonym for a vector of unsigned integers with value labels attached. It is not really a separate class.

The following code will thus give us the same regression results as obtained above, but the extra age groups will be excluded from the display of the results:

```
Analysis aoWI = IACubeReg("incwage~Age:sex", "(wkswork1>20)
&& (age>=20) && (age<=65)", "FactorVar Age =
CreateFactor(age-20, seq(20,65,1));");
```

Copy and paste the following statements to ExaStat's interactive mode command line to extract, clean up, and export the results for graphing in R.

```
// Convert the Cube results to a DataSet and write
// it to a tab delimited text file for use by R
DataSet dsWI = aoWI.GetAsDataSet("WageIncome");
dsWI = dsWI.RemoveMissings();

// Compute female/male wage ratios
DataSet dsFemale = dsWI[ dsWI["sex"]=="Female" ];
DataSet dsMale = dsWI[ dsWI["sex"]=="Male" ];
dsFemale("WageRatio")=dsFemale["WageIncome"]/
dsMale["WageIncome"];

dsWI["sex"].SetValueLabels();
dsWI.WriteToTextFile("WageAgeSex.txt");
dsFemale.WriteToTextFile("WageAgeSexRatios.txt");
```

Now, execute the following code in R to create the graph of wage income by age and sex:

```
WageAgeSex = read.table('WageAgeSex.txt', header=TRUE);
WageAgeSex$sex <- as.factor(WageAgeSex$sex);
levels(WageAgeSex$sex) <- c('Male', 'Female');
require(lattice);
```

```
xyplot(WageIncome~Age, data = WageAgeSex, groups = sex,
scales = list(x = list(at=seq(20,65,5))),
main='Wage Income By Age and Sex',auto.key=TRUE);
```

The difference between the male and female wage trajectories over the life cycle is striking, with the gap increasing rapidly starting in the mid 20's. Both male and female incomes peak in the mid to late 50's (of course, it is important to keep in mind that this represents different age cohorts at one point in time, not individuals tracked over their life cycles).

The following code graphs the same results, but they are presented as female/male wage ratios.

```
WageAgeSexRatios = read.table('WageAgeSexRatios.txt',
header=TRUE);
WageAgeSexRatios$sex <- as.factor(WageAgeSexRatios$sex);
levels(WageAgeSexRatios$sex) <- c('Male', 'Female');
require(lattice);
xyplot(WageRatio~Age, data = WageAgeSexRatios,
scales = list(x = list(at=seq(20,65,5))),
main='Female/Male Wage Income Ratios By Age');
```

The female/male wage ratio increases up until the mid 20's and then declines steadily through the rest of the working years.

## ***Weeks Worked***

An obvious question is the extent to which the difference between annual wages for men and women can be accounted for by differences in the number of weeks worked. The IPUMS variable `wkswork1` is a continuous integer variable with values from 0 to 52. To get descriptive statistics, treating `wkswork1` as categorical, use:

```
IADStats("@wkswork1");
```

To look at the pattern of weeks worked over the life cycle for men and women, use the following code in ExaStat:

```
Analysis aoWks = IACubeReg("wkswork1~Age:sex", "(age >= 20)
&& (age <= 65)", "FactorVar Age = CreateFactor( age-20,
seq(20,65,1) )");
```

```
DataSet dsWks = aoWks.GetAsDataSet("WeeksWorked");
dsWks["sex"].SetValueLabels();
dsWks = dsWks.RemoveMissings();
dsWks.WriteToTextFile("WeeksWorked.txt");
```

This can be graphed in R using:

```
WeeksWorked = read.table('WeeksWorked.txt', header=TRUE);
WeeksWorked$sex <- as.factor(WeeksWorked$sex);
levels(WeeksWorked$sex) <- c('Male', 'Female');
require(lattice);
xyplot(WeeksWorked~Age, data = WeeksWorked, groups = sex,
scales = list(x = list(at=seq(20,65,5))),
main='Weeks Worked Per Year By Age and Sex',auto.key=TRUE);
```

The number of weeks worked per year for women on average is less than that for men at all ages. Notice also the dip in average weeks worked during the reproductive years.

## ***Average Wages per Week***

We can compute a person's average wages per week by dividing annual wages by weeks worked. The following statement will do this:

```
FloatVar IncWagePerWk = incwage/wkswork1;
```

There are a couple of technical caveats. Both `incwage` and `wkswork1` are integers, and integer division gives an integer result (we are assigning the result to a `FloatVar`, but all the numbers will be integral values). Also, integer division by 0 will result in an error. Neither of these issues is a really problem here, as long as we make sure that (`wkswork1 > 0`). Nevertheless, to be on the safe side we can convert `incwage` into a `FloatVar` before it is divided by `wkswork1` by using `FloatVar(incwage)`. When a floating point number is involved in a division, the result is floating point and division by 0 results in a value that is treated as “missing” by ExaStat. Also, the fractional part of the result is retained. The following ExaStat code is thus a little better:

```
FloatVar IncWagePerWk = FloatVar(incwage)/wkswork1;
```

For the rest of this demo we will continue to select only those people who worked more than 20 hours per week and are between the ages of 20 and 65, and so it makes sense to create a subsample of the data that contains these selections. Let us also select just the variables we will be using, and create the variables `Age` and `IncWagePerWk` at the same time. This is essentially the same as the file `CensusWageExtract.xdf` that can be downloaded from the ExaMetrix web site, but the latter file contains two additional variables (`Race` and `MarSpPres`) that we will create below. After creating the new file `CensusWageExtract1.xdf`, we specify that it should be the default data source for the Interactive Analysis functions. The following code will do this. If you are using servers, the code will be distributed so that the appropriate file will be created on each server (you must, of course, have write permission on each server). All rows will be processed on each server, so that if you start out with identical files on each server you will end up with identical extracts.

This code can take a relatively long time (2 to 5 minutes) unless you have very fast hard drives, because in addition to the reading and processing of the data, several hundred megabytes of data must be written to the disk.

```
String sNewFileName = "CensusWageExtract1";
String sColsToSelect = "age, sex, incwage, educrec,
wkswork1, racblk, racasian, racwht, marst, disabwrk, perwt,
nchild";
String sRowSelections = "(age >= 20) && (age <= 65) &&
(wkswork1 > 20)";
String sNewVars = "FactorVar Age = CreateFactor( age-20,
seq(20,65,1) ); FloatVar IncWagePerWk = FloatVar(incwage)
/wkswork1;";
IASubsample(sNewFileName, sColsToSelect, sRowSelections,
sNewVars);
IASetDataSource(sNewFileName);
```

The function `IASubsample()` is discussed in more detail in the appendix [Creating New Data Files from the Census Data](#).

Note that in we are creating two new variables in `sNewVars`. The statements creating them are separated by a semi-colon. Note also that in the row selection statement `"(age >= 20) && (age <= 65) && (wkswork1 > 20)"` the three logical expressions are separated by the logical “and” symbol `&&`. They could be separated by semicolons instead, which would have the effect of treating the expressions as separate statements. The result would be the same, but the code would be slightly slower. However, sometimes complicated statements will generate a "symbol not defined" error, and using semicolons will usually correct this problem. See the documentation of the Data Analysis functions in `InteractiveAnalysis.h` for more information.

Now, let us look at the relationship between `IncWagePerWk`, `age`, and `sex`, to see how that compares with the `incwage` relationship. To do this, we can compute a cube regression with two dependent variables, separating the names with a comma on the left hand side of the formula. There is no limit on the number of dependent variables. (However, multiple dependent variables are only allowed in `IACubeReg()` and `IAREg()` at this time.) The following code computes this model and then extracts and exports the results:

```
Analysis aoWIEx = IACubeReg("IncWagePerWk, incwage ~
Age:sex");
DataSet dsReg =
aoWIEx.GetAsDataSet("WagesPerWeek,WageIncome");
dsReg = dsReg.RemoveMissings();
dsReg["sex"].SetValueLabels();
dsReg.WriteToTextFile("WageAgeSexEx.txt");
```

In the method `GetAsDataSet()`, meaningful names are given to each of the dependent variables, with the names separated by commas (semi-colons could be used as well). Copy and paste the following code to the command line in R to see a graph of these results:

```
WageAgeSexEx = read.table('WageAgeSexEx.txt', header=TRUE);
WageAgeSexEx$sex <- as.factor(WageAgeSexEx$sex);
levels(WageAgeSexEx$sex) <- c('Male', 'Female');
require(lattice);
xyplot( WagesPerWeek + WageIncome~Age,
data = WageAgeSexEx, groups = sex, as.table=TRUE, scales =
list(relation = "free"),
main='Weekly and Annual Wages By Age and Sex\n(With
Independent Income Scales)',auto.key=TRUE);
```

What is striking is how little difference there is between the life cycle trajectories of wage income and wage income per week for men and women (this remains the case even if the selection “`wkswork1 > 20`” is not made).

This can be investigated further by looking at the ratios between average female and male wages and wages per week. The following code puts the results from the analysis above into separate `DataSet`'s for males and females, and then computes the female/male ratios for both wages and wages per week (since we have already stripped the value labels from the `sex` variable, we cannot use `dsReg["sex"] == "Male"` and `dsReg["sex"] == "Female"` in this code):

```
DataSet dsMale = dsReg[ dsReg["sex"] == 0]; // males
DataSet dsFemale = dsReg[dsReg["sex"] == 1]; // females
DataSet dsRatios;
dsRatios("WeeklyWageRatio") = dsFemale["WagesPerWeek"]/
dsMale["WagesPerWeek"];
dsRatios("WageRatio") = dsFemale["WageIncome"]/
dsMale["WageIncome"];
dsRatios("Age") = dsMale["Age"];
dsRatios.WriteToTextFile("WageRatios");
```

Use the following R code to graph the ratios:

```
WageRatios = read.table('WageRatios.txt', header=TRUE);
require(lattice);
xyplot( WageRatio + WeeklyWageRatio ~ Age, data =
WageRatios,
scales = list(x = list(at=seq(20,65,5))),
main='Female/Male Wage Ratios By Age\nFor Annual and for
Weekly Wages',auto.key=TRUE);
```

The ratio of female to male earnings is slightly greater for weekly wages than for annual wages (note that we have already selected people who work at least 20 weeks per year), but the pattern over the life cycle is almost identical. The same patterns hold if we do not exclude those who work less than 20 hours per week, but the disparity between the two ratios increases.

## ***Income Quantiles by Age and Sex***

The income distribution is heavily skewed, and thus for some purposes the median is a more useful measure than the mean. The usual way to compute the “exact” sample median, or other quantiles, is to sort the data. For the Census data, we also have to deal with the fact that the data must be weighted by the sampling weight. A brute force approach to getting the income quantiles would be to sort the variable `incwage`, rearrange the weight `perwt` in the same order, and then compute cumulative sums of `perwt`. Doing this for every age for both males and females would be very time consuming.

However, a fast and straightforward way to obtain close approximations to the quantiles for every age and sex is to cut `incwage` into small categories, interact the resulting variable with all age and sex categories, compute the category counts, and then obtain the cumulative sums of these counts within age and income categories.

To cut `incwage` into intervals of width \$500 between \$0 and the maximum recorded income of \$354,000 (the Census does not retain true values for very high income individuals to protect privacy), we can use the `Cut()` method:

```
FactorVar IncWageCat = incwage.Cut( seq(0,354000,500) )
```

This results in 708 categories. The following statement computes the desired cell counts (using the same selections as used above in computing average income), resulting in a Cube with  $708 \times 46 \times 2 = 65,136$  cells (only a small part will be printed to the screen). Note that the second argument, which allows specification of row selections, is empty (“”) because we are using the file `CensusWageExtract1.xdf`, in which the desired selections have already been made.

```
SetRowDisplayMax(10); // limit the amount displayed
Analysis aoWageDist = IACube("IncWageCat:Age:sex", "",
"FactorVar IncWageCat = incwage.Cut( seq(0,354000,500));");
SetRowDisplayMax(200); // reset display limit to default
```

The function `ComputeWagePercentiles()`, created for this demo, computes percentiles, given the `Analysis` object and a list of percentiles. This function is defined in the file `CensusDemoFns.cpp`, which is loaded by the interpreter when the “ExaStat Census Demo” shortcut is used to start interactive mode. You can look in that file to see what the function does. To summarize, cumulative sums of the `Counts` vector are computed for each age and sex combination, and then the frequencies at the upper bound

of each income category are computed by dividing the cumulative sums by the total count within the group. The quantiles are approximated by linearly interpolating between the income cut points.

```
DataSet dsPercentiles = ComputeWagePercentiles( aoWageDist,
"10, 25, 50, 75, 90, 95");
dsPercentiles.WriteToTextFile("QuantilesAll");
```

The following R code plots each of the specified quantiles by age and sex in separate panels:

```
QuantilesAll = read.table('QuantilesAll.txt', header=TRUE);
QuantilesAll$sex <- as.factor(QuantilesAll$sex);
levels(QuantilesAll$sex) <- c('Male', 'Female');
require(lattice);
xyplot( Quantile.10 + Quantile.25 + Quantile.50 +
Quantile.75 + Quantile.90 + Quantile.95 ~ Age, data =
QuantilesAll, groups= sex,
main='Wage Income Quantiles By Age and Sex\n(Common Income
Scales)',auto.key=TRUE);
```

The absolute divergence of the male and female income quantiles with age is much more pronounced at the higher quantiles. That is, the male income distribution becomes more highly skewed with age than does the female distribution.

The following plot shows the same data, but allows different income scales in each panel.

```
xyplot( Quantile.10 + Quantile.25 + Quantile.50 +
Quantile.75 + Quantile.90 + Quantile.95 ~ Age, data =
QuantilesAll, scales = list(relation = "free"), groups=
sex, main='Wage Income Quantiles By Age and
Sex\n(Independent Income Scales)',auto.key=TRUE);
```

The relative patterns of male and female median income with age (median income is shown in the lower right hand panel) do not appear to be that different from the relative patterns of the other quantiles (except for the lowest quantiles). They also appear to be similar to the relative patterns of mean incomes.

## ***Income, Age, Sex, and Education***

Let us look at how education affects income, and how it affects the relationship between income, age, and sex. We will switch back to using just `incwage` as the dependent variable, and continue using the data file `CensusWageExtract1.xdf`, created in the previous section (`CensusWageExtract.xdf` can also be used).

The IPUMS variable `educrec` categorizes educational attainment into 10 categories, including an N/A category (<http://www.ipums.umn.edu/usa/peducation/educreca.html>).

Use `IADStats()` to get counts of the number of people in each category:

```
IADStats("educrec");
```

You can also use:

```
IACrossTab("educrec");
```

Essentially the same information is computed, but the display shows percentages in each category.

One way to include `educrec` in the regression is as an additive term. Doing this imposes the restriction that the relationship between income and education is unaffected by age and sex.

```
aoWI = IACubeReg("incwage~Age:sex + educrec");
```

Note that category counts are given for the “Cube” portion of the regression, but not for other regressors.

The coefficient for “`educrec='N/A or No schooling'`” has been dropped because there are no observations in this category, and the coefficient for “`educrec='4+ years of college'`” has been dropped because one of the indicator variables for `educrec` must be dropped due to collinearity with the “Cube” portion of the regression. The last argument in the `IACubeReg()` function allows control over whether the first or last indicator (dummy) variable in a set is dropped. The default is that it is the last. However, if it turns out that the last category must be dropped because it is empty, ExaStat will try dropping the first as part of its efforts to correct collinearities.

The results indicate that average yearly income (for all ages and both sexes) is \$19,796.25 less for those who have some 1-3 years of college than for those with 4+ years, and that those with “None or Preshool” have \$34,281.50 less.

A problem with these results is that they assume that the relationship between income and education is unaffected by age and sex. That does not seem reasonable, and since we have enough data, we can relax this restriction by estimating the model that allows the full interaction of `educrec` with age and sex. This is a very large regression (920 independent variables), but it takes about the same time to compute as the previous regression:

```
aoWIASE = IACubeReg("incwage~Age:sex:educrec");
```

Only 200 rows of results are displayed, because of the limit set above. To redisplay all of the results, use:

```
SetRowDisplayMax(1000);  
aoWIASE.DisplayResults();  
SetRowDisplayMax(200);
```

Use the following code to extract the results:

```
DataSet dsWIASE = aoWIASE.GetAsDataSet("WageIncome");  
dsWIASE = dsWIASE.RemoveMissings();  
dsWIASE["sex"].SetValueLabels();  
dsWIASE["educrec"].SetValueLabels();  
dsWIASE.WriteToTextFile("WageAgeSexEd.txt");
```

And use the following code to graph the results in R:

```
WageAgeSexEd = read.table('WageAgeSexEd.txt', header=TRUE);  
WageAgeSexEd$sex <- as.factor(WageAgeSexEd$sex);  
levels(WageAgeSexEd$sex) <- c('Male', 'Female');  
WageAgeSexEd$educrec <- as.factor(WageAgeSexEd$educrec);  
levels(WageAgeSexEd$educrec) <- c('None', 'Grade 1-4',  
'Grade 5-8', 'Grade 9', 'Grade 10', 'Grade 11', 'Grade 12',  
'1-3 College', '4+ College');  
require(lattice);  
xyplot( WageIncome~Age|educrec, data = WageAgeSexEd, groups  
= sex, ylim=c(0,80000),  
main='Wage Income By Age, Sex, and Education',  
auto.key=TRUE);
```

Notice how the wage trajectories for males and females increasingly diverge as educational levels rise. There are benefits to education for both men and women, but much more for men at ages 30 and above.

However, it is not clear from this graph whether or not the relative differences between male and female incomes actually increase with education. One way to investigate this is to allow each panel to have its own scale:

```
xyplot( WageIncome~Age|educrec, data = WageAgeSexEd, groups  
= sex, scales = list(relation = "free"),  
main='Wage Income By Age, Sex, and Education, Independent  
Scales', auto.key=TRUE);
```

From this graph it looks as though the relative differences rise with age, but do not change that much as education rises. This impression is reinforced by graphs of the wage ratios; the following ExaStat code calculates the ratios from the results computed above:

```

DataSet dsMale = dsWIASE [dsWIASE ["sex"] == 0];
// Use female DataSet to hold ratios
DataSet dsFemale = dsWIASE [dsWIASE ["sex"] == 1];
dsFemale("WageRatio") = dsFemale["WageIncome"]/
dsMale["WageIncome"];
dsFemale.WriteToTextFile("WageAgeSexEdRatios");

```

The following R code graphs these ratios:

```

WageAgeSexEd = read.table('WageAgeSexEdRatios.txt',
header=TRUE);
WageAgeSexEd$sex <- as.factor(WageAgeSexEd$sex);
levels(WageAgeSexEd$sex) <- c('Male', 'Female');
WageAgeSexEd$educrec <- as.factor(WageAgeSexEd$educrec);
levels(WageAgeSexEd$educrec) <- c('None', 'Grade 1-4',
'Grade 5-8', 'Grade 9', 'Grade 10', 'Grade 11', 'Grade 12',
'1-3 College', '4+ College');
require(lattice);
xyplot( WageRatio~Age|educrec, data = WageAgeSexEd,
main='Female/Male Wage Ratios By Age and Education',
auto.key=TRUE);

```

## ***Adding Race***

Another variable commonly looked at when analyzing wages is race. People were allowed to specify more than one racial category in the 2000 Census. The IPUMS variables `racblk`, `racasian`, and `racwht`, are categorical with two categories, “No” and “Yes,” representing Black, Asian, and White. They are not mutually exclusive. To see their descriptive statistics (for the `CensusWageExtract1.xdf` file), use:

```

IADStats("racblk * racasian * racwht");

```

The “\*” between the variable names in the formula specifies that all combinations of interactions the variables should be computed. This shows the full three-way table as well as each of the marginal tables.

For this analysis, we will create a single race variable with 4 categories: Black, Asian, White, and Other. There is no unique way to do this, either logically or computationally. One way to create such a variable (call it `Race`) is use the following algorithm:

```

if (racblk == "Yes")
  Race = "Black";
else if (racasian = "Yes")
  Race = "Asian";
else if (racwht = "Yes")
  Race = "White";

```

```
else Race = "Other"
```

If we switch the order of the “if” statements, we will obtain a different categorization, since a small percentage of people say “Yes” to more than one category. However, let us choose one and use it (we can always try it another way to see what the difference is).

There are several ways to implement this algorithm in ExaStat. We could actually write a loop over the data that would use code very much like that shown above and that would be efficient, but doing so is beyond the scope of this demo. Instead, we will use following approach:

```
LogicalMatrix lmRace = (racblk=="Yes") & (racasian=="Yes")  
& (racwht=="Yes");
```

```
FactorVar Race = CreateFactor( lmRace, "Black, Asian,  
White", "Other" );
```

The first statement creates a **LogicalMatrix** with 3 columns. Each row has a value of “true” corresponding to each “Yes,” and a “false” otherwise. The second statement converts this matrix into the variable **Race**. The following code combines these two statements to create the **Race** variable, and then uses it compute a Cube regression with the full interaction of **Age**, **sex**, **educrec**, and **Race**, with 3680 coefficients. Then the results are extracted and written to a file.

```
aoWIASER = IACubeReg("incwage~Age:sex:educrec:Race", "",  
"FactorVar Race = CreateFactor( (racblk=='Yes') &  
(racasian=='Yes') & (racwht=='Yes'), 'Black, Asian, White',  
'Other' );");
```

```
DataSet dsWIASER = aoWIASER.GetAsDataSet("WageIncome");  
dsWIASER = dsWIASER.RemoveMissings();  
dsWIASER["sex"].SetValueLabels();  
dsWIASER["educrec"].SetValueLabels();  
dsWIASER["Race"].SetValueLabels();  
dsWIASER.WriteToTextFile("WageAgeSexEdRace.txt");
```

And use the following code to graph the results in R:

```
WageAgeSexEdRace = read.table('WageAgeSexEdRace.txt',  
header=TRUE);  
WageAgeSexEdRace$sex <- as.factor(WageAgeSexEdRace$sex);  
levels(WageAgeSexEdRace$sex) <- c('Male', 'Female');  
WageAgeSexEdRace$educrec <-  
as.factor(WageAgeSexEdRace$educrec);  
levels(WageAgeSexEdRace$educrec) <- c('None', 'Gr 1-4', 'Gr  
5-8', 'Gr 9', 'Gr 10', 'Gr 11', 'Gr 12', '1-3 Col', '4+  
Col');
```

```

WageAgeSexEdRace$Race <- as.factor(WageAgeSexEdRace$Race);
levels(WageAgeSexEdRace$Race) <- c('Other', 'Black',
'Asian', 'White');
require(lattice);
xyplot( WageIncome~Age|educrec + Race, data =
WageAgeSexEdRace, groups = sex, ylim=c(0,80000),
main='Wage Income By Age, Sex, Education, and Race',
auto.key=TRUE);

```

This plot displays a large amount of information. Each row corresponds to a racial grouping, and each column to an educational attainment category. Each panel plots the average annual wage income for males and females separately. Income rises with educational attainment for each racial group for both males and females, but the absolute disparity between males and females is most pronounced for highly educated Asians and especially whites. It also appears that biggest disparities across racial groups are among highly educated men, with white men, and to a lesser extent Asian men, making significantly more than blacks or others.

## ***Adding Marital Status, Number of Children, and Disability***

It is commonly noted that unmarried women have incomes more comparable to men's than do married women, and that married men make more than unmarried ones. Having children at home also seems likely to affect the incomes of both women and men, as does having a disability. The 2000 Census has information on all of this and in this section we look at some ways of including it in the analysis.

### **Marital Status**

The IPUMS variable “marst” records current marital status (<http://www.ipums.umn.edu/usa/pdemographic/marsta.html>).

The following code summarizes this variable:

```
IADStats("marst");
```

There are 5 categories. We can use this variable to create a simpler variable with just two categories “No spouse” and “Spouse”:

```
FactorVar MarSpPres = CreateFactor( (marst=='Married,
spouse present'), 'No spouse, Spouse' );
```

The cube created by interacting `Age`, `sex`, `educrec`, `Race`, and `MarSpPres` has  $46 \times 2 \times 10 \times 4 \times 2 = 6,480$  cells. This is a lot, but we have enough data so that most of the cells contain reasonable counts.

## Number of Children at Home

The IPUMS variable `nchild` is a continuous variable which “counts the number of own children (of any age or marital status) residing with each individual. `nchild` includes step-children and adopted children as well as biological children”

(<http://www.ipums.org/usa/pfamily/nchilda.html> ).

The following code tabulates `nchild` as a categorical variable for the current data source:

```
IACrossTab("@nchild");
```

The range is from 0 to 9.

## Work Disability Status

The variable `disabwrk` “indicates whether respondents have any lasting physical or mental health condition that causes difficulty working, limits the amount or type of work they can do, or prevents them from working altogether. This does not include temporary health conditions, such as broken bones or pregnancies.”

(<http://www.ipums.umn.edu/usa/pdisability/disabwrka.html>)

The following code summarizes `disabwrk` for the current data source:

```
IADStats("disabwrk");
```

There are 5 disability categories, but two of them were not used in the 2000 Census. Computing and displaying the results as a crosstab shows the percentages in each category.

```
IACrossTab("disabwrk");
```

## Adding Race and Marital Status to the DataFile

The function `IAAddVars()` makes it easy to add new variables to an existing `DataFile`. This function is really just a thin wrapper around `IASubsample()`. The following code will add the variables `Race` and `MarSpPres` to the `CensusWageExtract1 DataFile`.

```
IASetDataSource("CensusWageExtract1");  
IAAddVars("FactorVar Race = CreateFactor( (racblk=='Yes') &  
(racasian=='Yes') & (racwht=='Yes'), 'Black, Asian, White',  
'Other' ); FactorVar MarSpPres = CreateFactor(  
(marst=='Married, spouse present'), 'No spouse, Spouse'  
");
```

The new variables are appended to the existing file “in place.” Since the entire file does not have to be rewritten, this is a faster operation than the use of `IASubsample()` to create a new file.

Use the following code to see a summary of the information about this `DataFile`:

```
IAShowInfo();
```

## Regression Models of the Joint Effects

By using complete interactions, we have avoided making assumptions about the shapes of relationships and the nature of interactions. However, there are not enough observations to continue taking full interactions of all of the variables. The Cube formed by interacting `Age`, `sex`, `educrec`, and `Race` has 3680 categories. If in addition we were to fully interact `MarSpPres` (2 categories), `nchild` (10 categories if treated as categorical, and `disabwrk` (2 meaningful categories that are not empty) we would have 147,200 categories.

Thus, we have to start making some assumptions about shapes and relationships. We can use regression models with some interacted terms and some additive terms, as is typically done in modeling relationships. Having a huge data set just allows us to make fewer assumptions.

For example, a possible model is:

```
incwage ~ Age:sex:educrec:Race:MarSpPres + nchild:Age:sex + disabwrk:Age:sex
```

That is, we fully interact `Age`, `sex`, `educrec`, `Race`, and the binary `MarSpPres` (for a total of 7,360 cells/parameters), but interact `nchild` and `disabwrk` only with `Age` and `sex`. This implies, for instance, that the effect on wage income of the number of own children at home, conditional upon the effects of the other variables, does not vary with education, race, or marital status, but does vary with age and sex. If our primary interest is in the relationships among the variables in the cube portion of the model, perhaps this is not a bad assumption.

Note that we are interacting the raw variable `disabwrk` with `Age` and `sex`, even though `disabwrk` has two empty categories. ExaStat detects the empty cells and drops those terms from the regression. It also detects the fact that this entire term is collinear with the cube portion of the regression and drops terms appropriately.

When the model consists only of a cube portion, the results can be visualized by directly graphing the regression parameters, which are the mean values (and the expected values, given the model) of wages within each category or cell. When there are additive terms in

the model, the parameters no longer have this simple interpretation. To graph the results, we need to use the estimated model to get the expected values of the dependent variable conditional upon specified values of the independent variables. The function `IAEvaluateCubeReg()` does that. It uses the estimated model to predict the expected value of the dependent variable for all possible values of the variables in the cube portion of the regression, and for specified values of all of the other variables. For instance, we can set `nchild = 0` (no children) and `disabwrk = 'No disability that affects work'` (which is equivalent to `disabwrk = 1`).

The function `plotwagemodel()` is a special function created for this demo that allows the specification of models like this, and that will process and then plot the results. For example:

```
IASetDataSource("CensusWageExtract1");
Analysis ao = plotwagemodel("incwage ~
Age:sex:educrec:Race:MarSpPres + nchild:Age:sex +
disabwrk:Age:sex", "WageASERM_ND01", "nchild = 0, disabwrk
= 'No disability that affects work'", "Expected Wage Income
by Age, Sex, Education, Race, and Marital Status,\nGiven No
Children and No Disability");
```

The second argument is a name to be given to the model (this name is used as the base of the names of files containing the results). The third argument allows the specification of the values for all variables not in the cube portion of the regression. If this argument is omitted, the expected values are computed as follows: continuous variables (such as `nchild`) are given their sample mean values; results are computed for every possible value of categorical variables (such as `disabwrk`), and then the weighted average of those results is computed, weighted by the sample frequencies for those categories (this is done for all possible values of the variables in the cube). A fourth argument can be used to specify the title for the plot, as is shown here. If it is not specified, a title is constructed from the model formula.

The function `plotwagemodel()` is defined in the file `CensusDemoFns.cpp`, which is loaded by the interpreter when the "ExaStat Census Demo" shortcut is used to start interactive mode. You can look in that file to see what is going on.

The plot of the results is put in a PDF file, and will be displayed automatically in Adobe Reader. To change the way that the graph is displayed, you can modify the code for `plotwagemodel()` (and its associated functions).

There are two overloads of this function. The first version, illustrated above, takes a model formula as its first argument and returns an **Analysis** object. The second version, shown below, takes an **Analysis** object, such as computed above, and uses it to compute the expected values of the estimated model. The model is not re-estimated; it is just re-evaluated. For instance, we can see what the plot looks like for people with 3 children (instead of no children) with the following code:

```
plotwagemodel(ao, "WageASERM_ND31", "nchild = 3, disabwrk = 1");
```

A word of warning: if you forget to change the name of the model, and if you have not closed the Adobe Reader, you will not get a plot because the PDF file will be locked and cannot be overwritten.

In this example, no title has been specified, and so a title is constructed. Note also that the value of `disabwrk` is set to 1, which is equivalent to 'No disability that affects work'. To see the value labels and codes for `disabwrk`, use:

```
IAShowInfo("disabwrk", true);
```

The `plotwagemodel()` functions can be used to estimate and plot any of the `incwage` cube regression models discussed in this document, plus any others that use the same set of independent variables. You can use it to explore different models.

## Appendices

### *Using Multiple Computers*

The interactive analysis (“IA”) functions and the underlying **Analysis** class methods allow the computations to be distributed across multiple computers. One computer is the “client” and the rest of the computers are “servers.” The computers must be connected on a network and the client must be able to communicate with each of the servers. ExaStat and some or all of the data must be installed (or at least be accessible) on each of the computers. ExaStat must be started in client mode on the client and in server mode on each of the servers, and the client must be told the names of the servers to use. There is no specific limit on the number of servers, but to my knowledge no one has tried using more than 3.

The servers are used to process data and to compute summary information that is sent to the client. The client is used as both a client and a server. As a client, it tells the servers what computations to do, it monitors their progress, and it puts the results together when the servers have finished with their parts. Thus, it usually makes sense to use the most powerful computer as the client. Having more than one processor is more advantageous on the client than on servers for most types of analysis.

In more detail, the required steps are as follows:

1. **Installing ExaStat and the Census Demo:** The installation procedure for ExaStat is the same on the client and on the servers. It is recommended that you install the Census Demo on the servers as well, although all that is needed is to create an `ExaStat\CensusDemo` directory and to copy the `StartCensusServer.bat` and “Start

Census Server” shortcut to the ExaStat directory on each server. There is no need to install Visual Studio or any other IDE or compiler on the servers.

2. **Making Sure Servers can be Accessed from the Client:** The client must be able to access each of the servers by name or by IP address. You should be able to see the servers from the Windows Explorer (for instance, in *My Network Places\Entire Network*). The client communicates with the servers using RPC (remote procedure calls). Firewalls typically block such calls; however, you should be able unblock them. On Windows XP, for instance, the firewall will prompt you to allow the calls to be unblocked the first time you start ExaStat as a server.
3. **Installing Data on Each Computer:** Each server must have access to the data it will be using. The easiest and best way to do this is to put the entire data file on each server (whichever of the files you will be using). The file must be located in ExaStat’s default data directory on that computer, which will be the ExaStat\CensusDemo directory if you start the server as suggested below. The advantage of having all of the data on each server is that this allows you to change at runtime the amount of data to be processed on each computer. However, space constraints may preclude this. The appendix [Creating New Data Files from the Census Data](#) describes how to split the main file into pieces that can then be placed on each computer; if this is done, use the command `IASetUseAllData(true)` on the client to specify that each computer should process its entire data file. (It is possible to have detailed control over what rows of data are used on each computer, but that is beyond the scope of this demo. It is also possible use other data configurations, such as having all computers read data from a single data server, but unless you have an extremely fast data server and extremely fast communication between that computer and the computers that will be doing the computations you are better off using a local file.)
4. **Starting the ExaStat Servers:** Use the “Start Census Server” shortcut on the server to start ExaStat in server mode. This will set the default data directory to the ExaStat\CensusDemo directory, which is where the Census data file should be placed. When you start ExaStat in server mode, a command window will open, and ExaStat will start listening for commands from an ExaStat client. To shut down the server, just close the command window.
5. **Starting the ExaStat Client:** Start the client using the “ExaStat Census Demo” shortcut. This starts ExaStat in interactive mode and sets the default data directory to the ExaStat\CensusDemo directory. You may want to resize the window. Use the command `IAUseCensusServers( )` from the command line to specify the servers to use and their relative speeds. For instance:  
`IAUseServers("server1, server2", "10, 20")` specifies that the servers named “server1” and “server2” should be used, and that server2 is roughly twice as fast as server1 (the default server weight is 10, and that is what is assigned to the client in its role as a server). The server weights are used to determine the number of rows of data each computer will be assigned to process, unless `IASetUseAllData(true)` has been specified. Since different types of analyses use different resources differently (disk, memory, CPU), the relative speeds of the computers may vary from one type of analysis to another. Thus, you may find that you will want to change the server weights while you are exploring a data file.

Calling `IAUseCensusServers()` with no arguments will turn off the use of servers.

## **About the IPUMS Census Data**

The Census data used in this demo comes from the *Integrated Public Use Microdata Series* (IPUMS) 5% 2000 U.S. Census sample produced and maintained by the Minnesota Population Center at the University of Minnesota [1] (<http://www.ipums.org>). This sample is one of several high-precision samples of the American population produced by the remarkable IPUMS project.

This IPUMS sample is a stratified 5% sample of households. The IPUMS file Ip20001.Z has been converted to an ExaStat **DataFile** named CensusIp20001.xdf. Household data has been matched with and assigned to the associated person records. There are 14,583,731 rows in this **DataFile**. Each row corresponds to a person, and each person in the sample on average represents about 20 people in the U.S. population at the time of the Census. The sample represents 281,111,530 individuals in the U.S. population at the time of the Census.

There are 265 variables. Documentation for each of the variables can be found at <http://www.ipums.umn.edu/usa/vars.html>. Variable names in the ExaStat **DataFile** are all lower case but are upper case in the IPUMS documentation. The SPSS dictionary provided on the IPUMS web site was used to obtain variable labels, value labels, and value codes. I can give no assurances about the accuracy of the conversion from the IPUMS raw data to the ExaStat **DataFiles**, other than that I know of no problems. You are free to use these files (subject to ExaStat's license and the restrictions imposed by IPUMS <http://www.ipums.org/usa/cite.html>), but you do so at your own risk.

A smaller **DataFile** (CensusDemoData.xdf) has been extracted from CensusIp20001.xdf for use in this demo. It contains only the variables required for the demo, but has the same number of rows. You may use either file for the demo (see below).

The variable `perwt` is the frequency weight for each person record. This variable “indicates how many persons in the U.S. population are represented by a given person in an IPUMS sample.” (<http://www.ipums.umn.edu/usa/ptechnical/perwta.html>). Although the average value of `perwt` is a little less than 20, the range is from 0 to 320, meaning that some people in the sample have a 0 weight and some a weight of 320. In order to obtain representative statistics for the entire population, it is necessary to weight analyses by `perwt`, and that is done in all the analyses in this demo.

The use of the weight `perwt` allows the computation of means and sums and regression coefficients that are representative of the entire population. However, computations of variances by the “IA” functions are not correct. This is because they are computed under the usual assumption that this is a simple random sample of individuals, with

uncorrelated error terms. However, this assumption is not correct, since this is a stratified sample of households. The effect is probably to cause the computed variances to be smaller than the true variances. See [http://www.ipums.umn.edu/usa/chapter3/standard\\_error\\_project.pdf](http://www.ipums.umn.edu/usa/chapter3/standard_error_project.pdf) for a discussion of this.

## ***Creating New Data Files from the Census Data***

There are four Interactive Analysis (IA) functions for creating new **DataFiles** from an existing one and for adding new variables to an existing **DataFile**:

1. **IASubsample()** Columns and rows may be selected, new variables may be created, and existing ones may be modified. In addition, specific blocks of the **DataFile** may be selected. The default is that the source file is not modified, but this function allows the target and source files to be the same.
2. **IAExtractDataFileBlocks()** Specific blocks and specific columns may be written to a new **DataFile**. A **DataFile** is stored in “blocks” of rows for each column, and the most efficient way to read and write is by block. The Census file CensusIp20001.xdf has 487 blocks, with a little less than 30,000 rows in each block. The source file is not modified by this function.
3. **IASplitDataFile()** Splits an existing **DataFile** into a specified number of pieces; specific columns may be selected; it does not modify the source file.
4. **IAAddVars()** New variables may be appended to an existing **DataFile**; the source file is modified.

Documentation for these functions can be found in the file InteractiveAnalysis.h, which is installed in the ExaStat directory. In addition, the file CensusDemoFns.cpp, which is installed in the ExaStat\CensusDemo directory, contains two examples of the use of **IASubsample()**. These examples are in the functions **CensusDemoData()** and **CensusWageExtract()**, which can be used to create the files CensusDemoData.xdf and CensusWageExtract.xdf respectively.

The input or source **DataFile** for all of these functions is the current default interactive analysis data source, which is set by calling **IASetDataSource()**.

If you are using more than one computer, the code will be executed on each computer and new **DataFiles** with the specified names will be created on each computer. You must, of course, have permission to write on each computer, and you must have enough room for the resulting **DataFiles**. The entire source **DataFile** will be processed on each computer. Thus, if you have identical copies of the source file on each computer, you will end up with identical copies of the new files. If you do not want the computations to be distributed, disable the use of servers first by using the statement:

```
IAUseServers();
```

The following code will create CensusDemoData1.xdf from CensusIp20001.xdf :

```
IASetDataSource("CensusIp20001");  
String sColsToSelect = "age, sex, incwage, educrec,  
wkswork1, racblk, racasian, racwht, marst, nchild,  
disabwrk, perwt";  
IASubsample("CensusDemoData1", sColsToSelect);
```

A simple example of the use of `IAExtractDataFileBlocks()` is:

```
IAExtractDataFileBlocks("CensusExtract1", 1 /*start  
block*/, 10 /* num blocks*/, "age, sex, incwage");
```

This will create the `DataFile` "CensusExtract1" from the current interactive analysis data source. The first 10 blocks (about 300,000 rows) of the variables `age`, `sex`, and `incwage` will be written to the output file. If you omit the last argument, all variables will be transferred to the new file.

A simple example of the use of `IASplitDataFile()` is:

```
IASplitDataFile( 3 /* num pieces */, "MyExtract", "age,  
sex, incwage");
```

This will create the 3 files MyExtract1.xdf, MyExtract2.xdf, and MyExtract3.xdf from the current interactive analysis data source. The files will contain the variables `age`, `sex`, and `incwage` and together will have all of the blocks (and rows) of the source file. The first file will have the first third of the blocks, and so on.

The main reason for splitting the file by blocks is for distribution across multiple computers. By splitting it this way you can put smaller files on each computer, which is helpful if you are have limited disk space on one or more computers. The main disadvantage to splitting the data this way is that you do not have the flexibility of changing the server weights (and thus the amount of data processed by each server) during an analysis. And having a smaller file will probably not affect the speed of an analysis because ExaStat only reads the data it needs in any case.

If you split the source data file in this way and place different pieces on different servers, use the following statement on the client to tell the interactive analysis functions to use all of the data in each file on each computer.

```
SetUseAllCensusData(true);
```

Also, the interactive analysis functions assume that the data source will have the same name on each server, so make sure to rename the files after you have copied them to the servers.

An example of the use of `IAAddVars()` is:

```
IASetDataSource("CensusWageExtract1");  
IAAddVars("FactorVar HaveChildren = (nchild != 0)");
```

This will create the variable `HaveChildren`, which has the value 1 if `nchild` is not 0, and is 0 otherwise, and adds it to the file `CensusWageExtract1.xdf`. When ExaStat adds a new variable to a `DataFile`, it does not need to re-write the whole file, so it is feasible to add new variables to a huge data file such as `CensusIp20001.xdf`. It is also possible to add new rows without re-writing the whole file, but that is beyond the scope of this demo.

### ***ExaStat's Formula Notation***

ExaStat uses a formula notation similar to that of R and S-PLUS for specifying models. For regression, “~” separates dependent variables from independent variables, “+” separates independent variables, and a “,” separates dependent variables. Formulas are always enclosed in quotations. Thus:

```
"y1, y2 ~ x1 + x2"
```

specifies the regression of `y1` and `y2` on `x1` and `x2` and a constant (R and S-PLUS do not allow multiple dependent variables).

The default in a regular regression, but not a Cube regression, is to include an intercept. To exclude an intercept, use -1:

```
"y1, y2 ~ -1 + x1 + x2"
```

A colon ‘:’ between variables denotes an interaction:

```
"c1:d1:d2"
```

is thus the interaction of `c1`, `d1`, and `d2`. The interaction of two categorical variables results in a categorical variable containing the full set combinations of categories, and adds a coefficient to the model for each category. The interaction of two continuous variables is the same as the multiplication of the two variables. The interaction of a continuous and a categorical variable adds a coefficient for the continuous variable for every level of the categorical variable. An asterisk ‘\*’ between categorical variables adds all subsets of interactions of the variables to the model.

A ^ denotes a power:

```
"y ~ 1 + x + x^2"
```

is thus the regression of `y` on a constant (the 1 could be omitted), on `x`, and on `x` squared.

ExaStat (unlike R and S-PLUS) does not allow data transformations within a formula, though it allows arbitrary data transformations as part of the analysis.

Continuous variables can be given aliases that are treated as categorical in a specific model, as long as the high and low values of the variable are known ahead of time. Within a formula, the '@' symbol can be used to create such an alias. Thus `@age` is the variable `age` with each integral value treated as a separate category.

ExaStat allows for different types of weights. Sampling or probability weights are specified as shown in:

```
"incwage ~ sex, pweight=perwt"
```

For these weights, degrees of freedom are computed from the number of valid observations, not from the sum of the weights.

## ***Cubes and Cube Regressions***

In ExaStat a **Cube** is a type of multi-dimensional array. It may have any number of dimensions, and thus is really a hyper-cube. Each dimension typically has value labels.

The statement `IACube("sex:MarSpPres")` computes a **Cube** object containing the cell counts for all `sex` and `MarSpPres` combinations (the return value is an **Analysis** object, which contains the **Cube**). The first dimension of the **Cube** has the categories "Male" and "Female" and the second dimension has the categories "No spouse" and "Spouse." This **Cube** is displayed as follows:

sex vs. MarSpPres

sex	No spouse	Spouse
Male	26212731	41694488
Female	25299780	32372156

ExaStat has several methods for manipulating **Cube** objects. For instance, it is possible to aggregate over, or into, specified dimensions, and it is possible to extract subcubes. **Cube** objects can also be converted into their equivalent "flat" form, resulting in a **DataSet** with a column for every dimension in the **Cube** plus the data columns (in this case, the **Counts** column):

Row	sex	MarSpPres	Counts
[ 1, .]	Male	No spouse	26212731
[ 2, .]	Female	No spouse	25299780
[ 3, .]	Male	Spouse	41694488
[ 4, .]	Female	Spouse	32372156

It is usually the case that the **Cube** representation is more compact and is easier to manipulate than is the **DataSet** representation.

The statement:

```
IACube("incwage ~ sex:MarSpPres");
```

will compute the mean values of **incwage** within each of the **sex:MarSpPres** categories (the computations are the same as for **IACrossTab("incwage ~ sex:MarSpPres")** and for **IACube("incwage:sex:MarSpPres")**, but the results are displayed differently). The result is again a **Cube**.

Using essentially the same formula, but computing a regression with no intercept instead, gives coefficients that also are just the category mean values for **incwage**:

```
IAREg("incwage ~ -1 + sex:MarSpPres");
```

ExaStat makes it easy to extend Cube computations to regression computations by allowing "Cube regressions." For instance:

```
IACubeReg("incwage ~ sex:MarSpPres");
```

This gives the same results as **IAREg("incwage ~ -1 + sex:MarSpPres")**, but the computations are somewhat different.

The main requirement for a Cube regression is that the first independent variable is categorical; it can be the interaction of a set of categorical variables. ExaStat takes advantage of this fact in computing the regression, because one part of the moment matrix is diagonal. This makes it easier to compute huge models such as:

```
IACubeReg("incwage ~ Age:sex:educ:Race:MarSpPres +  
nchild:Age:sex + disabwrk:Age:sex");
```

which has 7,912 coefficients, of which 1040 are dropped due to empty categories and collinearities. ExaStat pre-analyzes models to detect collinearities, and also detects them computationally and removes them by dropping coefficients.

An explicit intercept is always omitted in a Cube regression. Since the first term in the regression is categorical, it is equivalent to a complete set of dummy variables, and thus is collinear with a constant term. One of the dummies could be dropped, but the interpretation of the coefficients is typically easier if the intercept is dropped instead. The R-squared in a Cube regression is, however, computed as if an explicit intercept was included.

As has been discussed in the course of this document, ExaStat makes it easy to extract the results of a Cube regression in a form that is analogous to the results of a simple Cube computation, and that makes it easy to visualize the results.

## References

[1] Steven Ruggles, Matthew Sobek, Trent Alexander, Catherine A. Fitch, Ronald Goeken, Patricia Kelly Hall, Miriam King, and Chad Ronnander. *Integrated Public Use Microdata Series: Version 3.0* [Machine-readable database]. Minneapolis, MN: Minnesota Population Center [producer and distributor], 2004.

[2] W. S. Cleveland. *The Elements of Graphing Data*. Chapman and Hall, New York, 1985, 1994.