Auditor's Guide to the Use of SQLite

Using the SQLite database system to perform analytical audit procedures

Auditor's Guide to the Use of SQLite

Overview of SQLite

SQLite is an open source, public domain, database system which supports most of the SQL92 standards (unsupported standards are described at http://www.sqlite.org/omitted.html). Details of the SQLite system can be found at www.sqlite.org.

SQLite is in the public domain and can be used for any purpose, including commercial – see license at http://www.sqlite.org/copyright.html. SQLite is particularly suited for audit analysis as it can handle fairly large data volumes, has fast transaction rates, is simple to install and maintain, cost-effective to use and complies with the essential SQL92 standards. (Note: EZ-R Stats, LLC has no affiliation or relationship with SQLite.)

Types of audit tasks suitable for SQLite

SQLite is quite suitable for a variety of audit tasks, the details for which are provided below. In general, to determine if SQLite is suitable for a specific task, including audit tasks, a narrative is maintained at: http://www.sqlite.org/whentouse.html.

For small to mid-size applications, e.g. database tables of less than 20 million rows, and used simultaneously by three or fewer auditors, the system can be quite useful and effective.

Examples of common audit procedures that can be addressed using SQLite include:

Data summarization
Data extraction
Population sampling
Frequency distributions
Population statistics
Outlier identification
Top/Bottom 10
Benford Distributions (fraud or errors)
Relative Value (transaction errors)
Duplicates identification
Match / merge
Same, same same
Fraud / error detection

Inherent in database systems is the ability to "drill down" using where clauses.

SQLite also supports "sub-selects" which greatly simplifies many audit tasks. All of the above are described in greater detail below.

Overview of typical audit steps

The analytical process used in most audits will include the following eight steps:

1. Identification of data to be analyzed (i.e. data sources and data elements)
2. Transforming that data into a format suitable for import into a database
3. Define the tables needed to hold the data
4. Identify key fields which should be indexed to facilitate queries
5. Load the data into the tables
6. Design and test the queries
7. Review the output of the queries
8. Repeat/ refine the steps above

Step 1 – Identify data to be analyzed

Data of audit interest may reside in a variety of platforms, including:

Other databases, e.g. DB2, MS-Access, Oracle, etc.
Flat files (text files, mainframe QSAM etc., magnetic tapes)
Excel files
Paper documents

Step 2- Transform the Data

For databases, the needed data must be exported. The database may either contain an export facility, or else a query can be written against the database to output the data in the desired format.

For flat files, the data can be either fixed length or may contain variable width fields. In some cases a data conversion routine may need to be developed.

Step 3 – Define the Tables

Once the data of audit interest is known, as well as its structure, the SQLite database tables can be defined. This information will be needed in order to develop the DML to create the table structures.

Step 4 – Identify key columns

To speed access to data, columns which will be used in queries to locate data need to be identified. Examples would include customer account numbers, supplier numbers, employee identification numbers, store numbers, etc.

This information can then be used to build the appropriate database indices to speed the queries.

Step 5 – Load the data

Data is loaded into SQLite using the copy command (for version 2). The data source must have the columns separated by an identifier such as a tab or a space. The SQLite copy command can recognize a variety of column separators and row separators.

Details on the SQLite copy command are documented at http://www.sqlite.org/lang_copy.html.

Step 6 – Design and test the queries

Custom queries need to be developed to obtain the data of audit interest. Numerous examples are provided in order to serve as a "template" for custom audit queries.

Step 7 – Review the output of the queries

Once the queries are run, the auditor needs to review the results of the query to determine that the query results are reasonable, and to determine what further audit tasks need to be performed.

Step 8 – Repeat / refine the steps above

The audit process will typically be cyclical, because as queries are run, the results will lead to further questions and investigation. This in turn may necessitate obtaining additional data or columns, adding additional indices, "drilling down", etc.

Types of Audit Procedures Suitable for SQLite

Population Statistics

Population statistics can be obtained through the use of aggregation queries which summarize, count and analyze numeric columns. These facilities are a part of the SQL92 standard. Population statistics can be obtained either at the global level (entire table), or at the desired level of detail through the use of the "group by" command.

Simple examples will illustrate for a database table named "Invoices" which consists only of the following columns:

VendorNumber
VendorType
PlantLocation
InvoiceAmount

To obtain the basic population statistics for count, totals, minimum, maximum, average and range, the following SQLite query could be processed:

Select count(*) count, sum(InvoiceAmount) totals, min(InvoiceAmount) minimum, max(InvoiceAmount) maximum, avg(InvoiceAmount) average, (max(InvoiceAmount) – min(InvoiceAmount)) range from invoices.

If a breakdown were desired by VendorType, the VendorType column would be added to the query, along with a "Group By' Clause:

Select count(*) count, sum(InvoiceAmount) totals, min(InvoiceAmount) minimum, max(InvoiceAmount) maximum, avg(InvoiceAmount) average, (max(InvoiceAmount) – min(InvoiceAmount)) range, **VendorType** from invoices **group by VendorType**.

[added statements have been indicated in bold].

Sample Extraction

To extract a sample, a where clause can be added which tests is the current random number that is uniformly distributed between 0 and 1 is less than the desired sample selection percentage. For example, assume that from a population consisting of 5,000 observations, a random sample of 25 (1/2%) is to be drawn. For each observation, a random number will be generated. If that random number is between 0 and .005, then the observation will be selected. (We could also use and other range with a width of .005).

The query would then be:

Select *, ranuni() rnd from invoices where rnd < .005;

The query could also be coded to check for any other interval, e.g. 050 - .055. That query would be:

Select *, ranuni() rnd from invoices where rnd between .050 and .055;

Identification of Outliers (Top/Bottom 10)

Often, the auditor will wish to review the 10 largest (or smallest) transactions in the table. This can be accomplished by sorting the table by the numeric variable and then selecting the first N (e.g. 10) from that sorted list. Example queries using the hypothetical "invoices" table will illustrate:

To select the 10 largest invoices:

Select * from invoices order by InvoiceAmount desc limit 10;

This query sorts the invoices table by InvoiceAmount in descending order and selects the first 10 (i.e. the 10 largest).

To select the *smallest* 10, the sort sequence would be reversed (this can be achieved either by replacing the sort key word "desc" with "asc" or else omitting it entirely:

Select * from invoices order by InvoiceAmount asc limit 10;

Identification of Duplicates

To identify duplicates, one strategy is to first obtain a count of each transaction grouped by the criteria which identifies a duplicate. For example to find a duplicate with the same vendor number, same invoice date and same invoice amount, a count could first be obtained by using the query:

Select count(*), vendornumber, invoicedate, invoiceamount from invoices group by vendornumber, invoicenumber, invoiceamount.

Duplicates will have a count larger than 1.

A strategy to accomplish this is to combine a group of queries into a "database transaction" which is composed of several steps:

1. create a table with the counts of the transactions
2. join that table with the invoices table to create a table o duplicates where the count is greater than 1

This could be accomplished with the following database transaction which is composed of several steps:

```
Begin transaction;
-- drop the counts table, if it exists
Drop table counts;
-- create the table of counts
Create table counts as select count(*) count, vendornumber, invoicedate, invoiceamount
from invoices group by vendornumber, invoicenumber, invoiceamount.
-- drop the dups table, if it exists
Drop table dups;
--create a table of duplicates
Create table dups as select * from invoices I, counts C where I.vendornumber =
c.vendornumber and I.invoicenumber = c.invoicenumber and I.invoiceamount =
c.invoiceamount;
Commit;
```

This query will create the table dups which will contain all rows duplicated by the columns specified.

"Same, Same, Same" Analysis

Same, same, same analysis is similar to the analysis of duplicates above, except that it can address situations where all columns are equal, except the for the last. An example will illustrate. Suppose you wish to review a list of vendors, and identify any two vendors which meets the following criteria:

Same vendor number
Same city
Same zip code
Different IRS Number (Taxpayer Identification Number)

To obtain this information using database queries, one approach is to first identify all rows where a count of the "same, same, same" is greater than one and store that information in a table. The information in that table can then be joined with information in the source table, and in turn obtain a count using all four criteria. Then any such row with a count of exactly one will contain the information that is needed. The SQL code would be as follows:

```
Begin transaction;
--drop a work table containing counts
Drop table workcount;
--create a work table containing counts
Create table workcount as select count(*) count, vendor, city, zip from vendors
```

```
        Group by vendor, city, zip having count(*) > 1;
--now join this information with the source data to pick up the IRS number
Create table targetitems as select * from vendors V, workcount W where V.vendor =
W.vendor and V.City = W.City and V.Zip = W.zip;
--now do a count including the IRS number and select only those with a count of one
Create table target as select count(*) ccount, vendor, city, zip, irsnumber from targetitems
  Group by vendor, city, zip, irsnumber having count(*) = 1;
Commit;
```

The table named target should now contain all rows where the vendor city and zip are the same but the irsnumber is *different*.

Red Flag Indicators – Benford Distributions

When you reviewing a population of transactions, there are often numeric elements which are expected to be random in nature as well as items which are not random (i.e. there should be no pattern). For example, if you are examining disbursements within a certain time period, you would expect that the check numbers would be within a sequential range, and therefore not entirely random. However, the amounts of the checks should be somewhat random, unless there are a large number of recurring payments for the same amount.

In any event, a distribution of the first one or two digits in the numbers will conform with the "Benford" distribution, if the numbers are in fact random. A visual review of the actual distribution versus the expected distribution can be used to determine if the population is in fact a random population. However, it also possible to compile a statistic which quantifies the extent to which the actual distribution conforms with the expected Benford distribution. This amount can be computed by summing the actual percentage of the distribution obtained, subtracting the expected distribution percentage and then squaring it. The sum of all such numbers then comprises the "Benford" index. This amount can range from a low value of 0 (i.e. perfect correlation) to as high as 2.00 (which means that population consists entirely of leading digits of either 9 or 99). In practice, reasonable size populations that are randomly distributed will have a Benford index in the neighborhood of .5. Skewed populations will have Benford indices of 1.2 or higher.

In any event, this index can be used to target which sub-populations for which a further review may be merited. Often, the auditor may wish to obtain an overall Benford index for the population as a whole and then obtain the index at a more detailed level, e.g. by vendor or by employee etc. Such queries can be illustrated below:

Obtain a benford index for the population as a while:

```
Select Benford(InvoiceAmount) from invoices;
```

Obtain a benford index at the vendor level:

Select Benford(InvoiceAmount), VendorNumber from Invoices group by VendorNumber;

The second query may provide information that facilitates targeting unusual transactions for closer review.

Frequency Distributions

A common analytical task is to obtain frequency distributions for aspects of the population which are of audit interest. The purpose may be to focus on the components of the population which are the most (or least common), have the most numeric weight, etc. Another purpose is to enhance the auditor's understanding of the distribution of the population.

Below are some examples of typical analytical questions and how they might be determined using database queries.

Determine which vendors have the most number of invoices (top 50):

Select count(*) ccount, vendornumber from invoices order by ccount desc limit 50;

Determine the vendors having the largest invoice value:

Select sum(InvoiceAmount) totalInvoice, vendornumber from invoices group by vendornumber order by totalInvoice desc ;

Relative Value

Often, the auditor will find that groups of transactions will tend to be fairly tightly clustered around a particular value or range of values. An example might include the unit cost of a widget. If all unit costs for widgets (as derived from invoices) were ranked from top to bottom, you would not expect to see a large difference, as expressed by a ratio, between the most costly item and the second most costly item. Similarly, there should also not be a large gap between the least costly item and the next to the least costly item. In instances where relative value is a large ration, the auditor may want to make further inquiries or perform further review to determine the underlying cause.

To obtain this information using a database query, the numeric column from a table should be sorted and then the first two (i.e. the top 2) items compared. This can be done with a select query with a limit clause. For example:

Select unitcost from invoices order by unitcost desc limit 2;

Processing SQLite Queries

There are two primary methods to process SQLite queries:

1. Using a GUI interface
2. Using a DOS command line

Each method is outlined below.

Using SQLite with the DOS Command Line

The SQLite command line program can be downloaded from the SQLite web-site http://www.sqlite.org/download.html. The narrative below assumes that the program has been copied to a directory on the path, and the name has not been changed.

The sqlite program (sqlite.exe) can be run from the DOS command line either interactively or can use command indirection by accepting commands from a text file. Complete documentation is available at http://www.sqlite.org/sqlite.html.

With command indirection, the text of the queries is stored in a text file, using notepad or other text processor and the output can either be displayed or else stored in a text file as well. For example assume that the text of the query has been stored in a file named "sqlquery.txt:. Then the sqlite command can be processed by issuing the DOS command "sqlite <sqlquery.txt >sqlresults.txt". This causes the query to be run and the output placed in the specified text file.

Using SQLite with GUIs

SQLite commands can also be run using a variety of tools which provide a much easier to use interface. Some tools are free, others have a charge.

We have used the public domain software called SQLite Database Browser (version 1.1) which is available from http://sqlitebrowser.sourceforge.net. This tool allows the display of SQLite database tables, schema, provides for execution of SQL and import of data from certain files.

By googling the text "SQLite query program", you may find other tools that meet your requirements. Examples include:

SQLite Analyzer - http://www.kraslabs.com/sqlite_analyzer.html
SQLite Analyzer - http://www.itlocation.com/en/software/prd57552,,.htm
Terra Informatica - http://terrainformatica.com/sqlitedb/editions.whtm

(Note: EZ-R Stats, LLC has no affiliation or relationship with any of the products or vendors above.)

Importing Data from Other Systems

In order to import data into SQLite, the most common approaches are to either:

1. Connect to SQLite using ODBC and then export from another database or file system to an ODBC database. Note: an SQLite ODBC driver may be obtained from http://www.ch-werner.de/sqliteodbc/.
2. Export the data into a flat file with columns separated by tabs and then use the SQLite "copy" command which is available in version 2.
3.

Database systems

Database systems such as MS-Access permit the export of data into ODBC format as well as the export of data into tab separated file format. This capability is also available with other database systems such as MS-SQL Server, Oracle, MySQL, etc.

To export directly from MS-Access into SQLite, first install the SQLite ODBC driver mentioned previously. Then, in Access, open the Access database, select and view the table to be exported, choose the menu option File | Export. Select the type of export as ODBC database and specify the filename. Then select the ODBC datasource and proceed with the export.

Flat files and text files

Many database systems allow the export of data into tab separated format. For example, MySQL permits a select command with an "INTO" clause.

Excel

Excel files can be imported into SQLite by saving a sheet as a tab separated file and then importing the data into SQLite using the SQLite copy command.

Table Creation

SQlite is a "typeless" database system, i.e. SQLite does not enforce data type constraints. Any data can be inserted into any column. Thus, when a table is defined, the key requirements are to distinguish between columns which intended as numeric vs. columns which are not. By defining a column as numeric, SQLite can properly sort that column.

To create a table, it is necessary to use standard DML language. An example is provided below:

```
Create table customer (
        Customerid varchar(20),
        AccountBalance number(10),
        StreetAddress1 varchar(20),
        StreetAddress2 varchar(20),
        City varchar(20),
        State char(2),
        ZipCode Number(5)
);
```

Note that even though the column widths are specified, these amounts are not enforced by SQLite.

Overview of SQLite tables

SQLite tables conform with most of the standards of SQL92, with some exceptions. These exceptions are documented at http://www.sqlite.org/omitted.html. The most significant exception is that SQLite is a "typeless" database system, meaning that column values are generally accepted, regardless of how the column has been defined.

Note that SQLite does not support most of the "ALTER" commands, so it is difficult to make changes to established tables. However, there are some software tools available which can address this issue.

Overview of Data Manipulation Language (DML)

With the exception of the ALTER command, SQLite DML follows SQL92 standards. The auditor uses DML to create tables, drop (erase) tables, create and drop table indices, delete all (or certain) rows.

Need for indices

Access to large tables will generally be greatly speeded up if the proper indices are established. Generally it is advisable to define indices for all columns which will be used to focus a query. This includes columns such as vendor number, employee number, city, zip code or any other element of data that will be used as part of a query to locate specific data.

Note that when a table is first created, it will not generally not contain indices. Generally indices will be defined once the table has been fully loaded, as this is likely the most efficient approach. If an index is not defined, then often the database will need to read the entire table in order to fulfill a query. For a large table, this can be a time consuming process.

The syntax to drop and create an index is as follows:

Drop index ndx_indexname;

Create index ndx_indexname on table(columnname).

SQLite follows the SQL92 standards.

Drilling Down with Where Clauses

Most of the queries previously discussed can be further refined by limiting their scope with a "where" clause. The where clause identifies additional conditions which must be met for the specified query to select any particular row.

The where clause will specify one or more conditions which must be met. The conditions may be combined with "and" or "or" conditions. Further, a column value may be specified as lying "between" a range of values.

Examples of where clauses include:

Where invoicenumber between 20000 and 22000

Where amount > 100 and invoicediscount = .03

Where employeename like 'Bla%' and dept between '001' and '005'

Where city in ('Raleigh', 'Greensboro','Cary')

Joins – Doing Match / Merges

Matching table rows on key fields
Identifying table rows which do NOT match

A query may obtain data from one or more tables by "joining" them on columns which have matching values. For example, suppose that you have a table with county codes and the name of the county. Another table has the county code, amount and date paid, but does not include the name of the county.

In order to include the name the auditor can "join" the tables by specifying a query such as:

Select * from invoices I, counties C where I.County = C.County;

In English, this query says give me all the information from the invoices table and also the name of the county for the county code specified in the invoice table. Implied in this

query is that if the county code is not defined in the counties table, then it will not be included in the joined query.

Use of Subselects

Occasionally the auditor needs to make a selection where a column value takes on a series of values. For example, suppose you wish to obtain all invoices from suppliers with their headquarters in Raleigh. However, the invoices table lists only the supplier number, not the city where the supplier is located. That information is contained separately in the suppliers table.

In order to accomplish this, the auditor can run a query using a sub-select. For example, to identify all the supplier number with their headquarters in Raleigh, you would run the following select command:

Select suppliernumber from suppliers where headquarters = 'Raleigh';

This query would provide a list of the desired supplier numbers.

To obtain a list of invoices from suppliers in Raleigh, the auditor would issue the command:

Select * from invoices where suppliernumber in (select suppliernumber from suppliers where headquarters = 'Raleigh');

Creating Tables from Select Queries

A table may be created either with data manipulation language (which establishes the structure of the table but does not insert any data) or else created from one or more other tables using the create command.

An example happened when we needed to create a table of potential duplicate payments. This was achieved by specifying that the table be created as the result of a select query with certain conditions:

Create table dups as select count(*) count, invoicenumber from invoices group by invoicenumber having count > 1;

Exporting Data Out of SQLite

Data can be exported from SQLite simply by running a query and redirecting the output to a file. For example, coding the query:

Select * from invoice where invoiceamount > 100;

And saving it to a file called invoicequery.sql.

Then, run the sqlite batch command named "sqlite.exe" as follows:

Sqlite <invoicequery.sql >invoiceextract.txt, will cause the desired data to be extracted from SQlite and written to a tab separated file.

Further Analysis of Data Exported from SQLite

Once data has been exported from SQLite, it is then possible to perform further analysis on the extracted data using EZ-R Stats for Windows. An overview of the analytical procedures, with example scripts is listed below:

Benford Analysis

BENFORD ANALYSIS

Obtaining Benford Distributions

Compare Actual distribution with expected

Useful in identifying potentially fraudulent numbers.Frank Benford, a physicist at GE Research Laboratories in the 1920's, discovered that numbers with low first digits occurred more frequently in the world and calculated the expected frequencies of the digits in tabulated data. Digital analysis is well suited to finding errors and irregularities in large data sets when auditors need computer assisted technologies to direct their attention to anomalies. Auditors can use Benford's discovery in business applications ranging from accounts payable to program management problems. In addition, subset tests identify small lists of serious anomalies in large data sets, making an analysis more manageable. Benford's law provides an analysis method that can help alert auditors to possible errors, potential fraud, manipulative biases, costly processing inefficiencies or other irregularities.

- Natural occurrence rates of leading digits in numbers.
- One or two digits
- Based upon logarithmic scales
- Useful in identification of errors (intentional or unintentional)
- □Benford score□ Range from 0 to about 1.9

Syntax

```
proc benford data=DATAFILE dec=DECIMALS;
var VARIABLENAME;
out = OUTPUTRESULT;
```

Parameters Used

There are four primary parameters for Benford analysis:
DATAFILE - the datafile to be analyzed
VARIABLENAME - the numeric variable to be analyzed
OUTPUTRESULT - name of the file to store the results of the analysis
DECIMALS - number of leading digits to be analyzed (must be 1 or 2)

Example Script

```
*
*;
* benexp1.ezs;
* test of benford analysis procedure;
* generated data - employee expense reports;
* initial version 8-12-05;
libname rest 'c:/ezs/tab';
libname tst 'c:/ezs/tab/tested';
proc benford data=tst.expRep dec=2;
var ExpenseAmount;
out = tst.eR1;
*dec = 2;
run;
```

Example Output

```
Begin benford processing: Wed Nov 2 22:07:31 2005
Benford statistics for value in file: /ezs/libin\series.tab
computed Benford variance is 0.47004
record count for file /ezs/libin\series.tab variable value   927905
digits      expectpct  expectamt actual      diff
     1      0.30103    279327    137246     142081
     2      0.17609    163395     90810      72585
     3      0.12494    115931    106170       9761
     4      0.09691     89923    103786     -13863
     5      0.07918     73472    120601     -47129
     6      0.06695     62120    102714     -40594
     7      0.05799     53811     95266     -41455
     8      0.05115     47464     85727     -38263
     9      0.04576     42458     68276     -25818
Largest difference was for digit 1 amount was 142081
May want to look at trans with digit(s) beginning with 1
End of benford processing: Wed Nov 2 22:07:35 2005
```

Benford Analysis for - /ezs/libin\series.tab

Digit	Frequency
Digit 1 - actual	137246
Digit 1 - expected	279327

Digit 2 - actual	90810
Digit 2 - expected	163395
Digit 3 - actual	106170
Digit 3 - expected	115931
Digit 4 - actual	103786
Digit 4 - expected	89923
Digit 5 - actual	120601
Digit 5 - expected	73472
Digit 6 - actual	102714
Digit 6 - expected	62120
Digit 7 - actual	95266
Digit 7 - expected	53811
Digit 8 - actual	85727
Digit 8 - expected	47464
Digit 9 - actual	68276
Digit 9 - expected	42458

Means (p10)

MEANS

Basic Dataset Statistics

Population Means
This procedural provides a statistical summary of the key attributes for each column in the data. These attributes include count, min, max, mean, standard deviation, range, variance, skewness and kurtosis. This procedure can serve as a springboard to other procedures such as the Univariate procedures or the Domain procedure

The univariate procedure provides information as to quartiles, top and bottom five observations, etc. This procedure requires a sort of the data.

The Domain procedure provides more detail as to the composition of a particular colmun of data, such as the codes used. For example, a dataset may have a code established for a unit of measure. In order to see what codes are being used, the domain procedure could be be specified.

All procedures

Syntax

```
proc means data=DATAFILE;
var VARIABLENAME;
out = OUTPUTRESULT;
```

Parameters Used

There are three primary parameters for the Means command:
DATAFILE - the datafile to be analyzed
VARIABLENAME - the numeric variable to be analyzed
OUTPUTRESULT - name of the file to store the results of the analysis

Example Script

```
*
*;
* means2.ezs;
* test of means analysis procedure;
* tested on 6-18-05;
libname test 'c:\cloop\bls\tab';
libname drug 'c:\ezs\tab\tested';
proc means data=test.cewenb;
var seqno;
out = drug.stats2;
run;
```

Example Output

Means statistics for /ezs/libin\cewenb.tab

Variable	Range	N	Variance	Mean	STD	Min	Max	Skewness	Kurtosis
area	0	3012	0	37000	0	37000	37000	-na	-nan
datatype	0	3012	0	0	0	0	0	-na	-nan
size	0	3012	0	0	0	0	0	-na	-nan

own	8	3012	1.6801	4.26793	1.296187	0		8	34.25969	-116.928
naics	999989	3012	6.2E+10	220310.1	249091.1	10	9	999	1.66472	-0.71123
year1	0	3012	0	2003	0	2003		003	-na	-nan

The "R" System (p11)

THE R SYSTEM

The R System

Running R System Commands
Statistical summary. The R system contains an extensive library of algorithms and analytical tools. The system is distributed by the CRAN Institute, which can be reached at their web siteThe CRAN Institute

The R System is a public domain system which is maintained by contributors from all over the world, many in research or educational instituions. Much of the functionality covers a variety of topics, most in the statistical area.
One of the many advantages of the R system is that fairly extensive analytical procedures can be performed with relatively little coding, through the use of templates and standard analytical procedures which have been written in the R language.

A spin-off of the R system has been commercialized and is also widely used.

Syntax

```
proc means data=DATAFILE;
var VARIABLENAME;
out = OUTPUTRESULT;
```

Parameters Used

There are three primary parameters for using the R System:
DATAFILE - the datafile to be analyzed
VARIABLENAME - the numeric variable to be analyzed
OUTPUTRESULT - name of the file to store the results of the analysis

Example Script

```
*;
* r3.ezs;
* test of R system calling procedure;
* tested on 6-18-05;
libname test 'c:\ezs\tab';
libname drug 'c:\ezs\tab\tested';
proc r;
infile 'c:/test/r/test3.r';
```

```
out = drug.r3;
run;
```

Example Output

```
R : Copyright 2004, The R Foundation for Statistical Computing
Version 1.9.1 (2004-06-21), ISBN 3-900051-00-3
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.

> #
> # test3a.r
> # read in the claims header tab separated table and do some basic stats
> #
> library(fBasics)

fBasics: Markets, Basic Statistics, Date and TimeNo timezone information
found, using default of GMT
> claims <-
read.table("c:/test/r/claims.tab",sep="\t",nrows=600,header=TRUE)
> attach(claims)
> summary(claims)
TXN ICN FINPAY POPPAY
Min. :0.0000 Min. :5.200e+13 NCXIX:600 NCXIX:600
1st Qu.:1.0000 1st Qu.:5.200e+13
Median :1.0000 Median :5.200e+13
Mean :0.7817 Mean :5.242e+13
3rd Qu.:1.0000 3rd Qu.:5.200e+13
Max. :1.0000 Max. :3.020e+14

CKDATE RANBR CLMAGE CLMTYPE
Min. :20031104 Min. : 0 Min. : 1.000 D:600
1st Qu.:20031112 1st Qu.:1625635 1st Qu.: 1.000
Median :20031112 Median :1625635 Median : 1.000
Mean :20031115 Mean :1625966 Mean : 1.025
3rd Qu.:20031118 3rd Qu.:1631727 3rd Qu.: 1.000
Max. :20031126 Max. :1637472 Max. :16.000

HFDOS HTDOS MID LASTNAME
Min. :20030929 Min. :20030929 900863388O: 32 BOONE : 63
1st Qu.:20031031 1st Qu.:20031031 900134141L: 19 BRIGGS : 51
Median :20031105 Median :20031105 244508101R: 16 BRADFORD : 44
Mean :20031087 Mean :20031087 241500460K: 14 ADKINS : 41
3rd Qu.:20031113 3rd Qu.:20031113 242700669Q: 14 AUTREY : 38
Max. :20031120 Max. :20031120 946956733L: 13 ARMSTRONG : 32
(Other) :492 (Other) :331
FIRSNAME MI DOB PROVNUM
KIMERAN : 32 :214 Min. : 0 Min. :5041
CARRIE : 21 E : 56 1st Qu.:19270325 1st Qu.:5041
TAMMY : 19 M : 47 Median :19391209 Median :5041
FRANCES : 18 B : 37 Mean :19437126 Mean :5041
NELLIE : 16 L : 35 3rd Qu.:19700609 3rd Qu.:5041
BETTY : 14 A : 34 Max. :20030620 Max. :5041
```

```
(Other) :480 (Other):177 NA's : 1
BILLAMT PAIDAMT PROVTYP PROVCNTY
Min. : 586 Min. : 0.0 Min. : 26.00 Min. :100
1st Qu.: 1547 1st Qu.: 606.8 1st Qu.: 26.00 1st Qu.:100
Median : 4052 Median : 1617.5 Median : 26.00 Median :100
Mean : 7608 Mean : 4356.7 Mean : 27.62 Mean :100
3rd Qu.: 7983 3rd Qu.: 5817.5 3rd Qu.: 26.00 3rd Qu.:100
Max. :161054 Max. :160752.0 Max. :999.00 Max. :100
NA's : 1
UBIND GENDER PROVSPEC X
Mode:logical : 1 Min. :84 Mode:logical
NA's:600 F:442 1st Qu.:84 NA's:600
M:157 Median :84
Mean :84
3rd Qu.:84
Max. :84
NA's : 1
> sink("c:/test/r/results3a.txt")
> mean(claims)
> min(claims)
```

Histogram (p13)

HISTOGRAM

Compiling Histogram Statistics

Determining how data is distributed

Developing histogram data from tab separated file. Useful in determining the type of distribution and the identification of possible outlier values.

A histogram provides a birds eye overview of the distribution of data in the file. Often, the distribution details can provide the analyst with ideas on which types of further analysis may be desireable.

Output from the histogram is written both as data which can be imported into other programs such as MS-Excel, as well as output in html format for ease of viewing.

The syntax for the analytical specifications is fairly straightforward. Required are the starting value (i.e. lowest value for which histogram values are to be obtained, the number of "bins" to be used (maximum is 100), and the width of each bin. (all bin widths are uniform).

Output from the procedure is both tabular and visual, as presented in HTML.

Syntax

```
proc histo data=DATAFILE start=START width=WIDTH n=N;
var VARIABLENAME;
out = OUTPUTRESULT;
```

Parameters Used

There are six primary parameters for the Histogram procedure:
DATAFILE - the datafile to be analyzed
VARIABLENAME - the numeric variable to be analyzed
OUTPUTRESULT - name of the file to store the results of the analysis
START - the lowest number for the range of data to collect histogram results
WIDTH - the width of each bin in the histogram
N - number of bins (maximum of 100)

Example Script

```
*;
* histo1.ezs;
* test of parameter options on proc statement;
* tested 07-02-2005;
*;
libname test 'c:/ezs/tab';
libname drug 'c:/ezs/tab/tested';
proc histo data=test.maud2002 chart vertical start=.1 width=.5 n=30;
out=drug.h;
var QUAN1;
run;
```

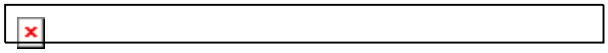
obs	start	width	count	sc
0	0.1	0.5	3901	278
1	0.6	0.5	1663	118
2	1.1	0.5	1516	108
3	1.6	0.5	581	41
4	2.1	0.5	305	21
5	2.6	0.5	188	13
6	3.1	0.5	127	9
7	3.6	0.5	116	8
8	4.1	0.5	93	6
9	4.6	0.5	104	7
10	5.1	0.5	81	5
11	5.6	0.5	49	3
12	6.1	0.5	29	2
13	6.6	0.5	32	2
14	7.1	0.5	40	2
15	7.6	0.5	33	2
16	8.1	0.5	29	2
17	8.6	0.5	49	3
18	9.1	0.5	44	3
19	9.6	0.5	21	1
20	10.1	0.5	21	1
21	10.6	0.5	19	1

22	11.1	0.5	17	1
23	11.6	0.5	17	1
24	12.1	0.5	13	0
25	12.6	0.5	16	1
26	13.1	0.5	4	0
27	13.6	0.5	6	0
28	14.1	0.5	5	0
29	14.6	0.5	6	0

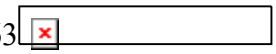
Histogram

obs count

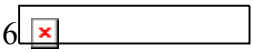
00 3901



01 1663



02 1516



03 581

04 305

05 188

06 127

07 116

08 93

09 104

10 81

11 49

12 29

13 32

14 40

15 33

16 29

17 49

18 44

19 21

20 21

21 19

22 17

23 17

24 13

25 16

26 4

27 6

28 5

29 6

Same, Same, Same (p14)

SAME, SAME, SAME

Identifying Unusual Transactions

Selecting the Needle in the Haystack

Identifying unusual transactions from tab separated file. Often, unusual transactions can be identified when unexpected combinations are identified or located. For example, same invoice number, same vendor, same amount. Or same employee name, same employee number, same store number. There are other combinations possible as well, such as same, same, same, different, etc.

The same, same, same procedure is flexible in how the processing si done. First, the analyst must determine which variables will be involved in the identification process (for example, in the instances cited above the variables would be invoice number, vendor, amount). There can be up to five variables involved, or as few as a single variable. In each instance, all but the last variables are SAME, and the final variable may be either SAME or DIFFERENT. For example, in a review or payroll data, you might want to identify the same employee last name, same street address and same city, but DIFFERENT employee number. In this instance the selection type would be coded as "SSSD".

A much different (and possibly simpler) instance would be where you wanted to identify any duplicate items in a single column. In this instance the selection type would be "S".

There are a number of examples of the "same, same same" processing included in the test files provided.

Syntax

```
proc sss data=DATAFILE;
var VARLIST;
out = OUTPUTRESULT;
```

Parameters Used

There are three parameters for the Same Same Same (SSS) procedure:

DATAFILE - the datafile to be analyzed
VARLIST - the variables that determine the control break
OUTPUTRESULT - name of the file to store the results of the analysis

Example Script

```
*;
* sss1.ezs;
* test of same same same procedure;
* identify same vendor number, same invoice number, DIFFERENT invoice
date;
libname test 'c:\test\model';
libname ss 'c:\ezs\tab\SSS';
libname drug 'c:\ezs\tab\tested';
proc sss data=ss.TestData sstype="SSD";
var VendorNo InvoiceNo InvDate;
out = drug.sss;
run;
```

Example Output

InvoiceNo	PaidDate	InvAmount	REC
1630	9/17/2004	1177.18	This
1630	9/17/2004	1177.18	Last
1630	9/17/2004	1177.18	This
1630	9/17/2004	1177.18	Last
2331	4/29/2004	2381.32	This
2331	4/29/2004	2381.32	Last
2331	4/29/2004	2381.32	This
2331	4/29/2004	2381.32	Last
3387	5/23/2004	4424.31	This
3387	5/23/2004	4424.31	Last
3387	5/23/2004	4424.31	This
3387	5/23/2004	4424.31	Last
3387	5/23/2004	4424.31	This
3387	5/23/2004	4424.31	Last
3387	5/23/2004	4424.31	This
3387	5/23/2004	4424.31	Last

Spike (p15)

SPIKE ANALYSIS

Identifying Unusual Transactions

For Data File Analysis
Identifying unusual transactions from tab separated file. Sharp, short-term fluctuations which can be characterized as spikes, compared with the longer term trend or moving average.

Hurdles/Challenges

There are several potential challenges which the auditor must overcome.

- How can data be distilled to enable a meaningful analysis, and
- How can volatility best be measured to ensure that the unusual patterns can be highlighted.

Compounding the difficulty of performing an effective analysis of spikes are the difficulties in gathering and summarizing the information from what is often a very large repository of data.

Audit Procedures

Generally the auditor will take the following steps as part of the overall process of identifying and assessing spikes: Identify the quantitative measure

to be reviewed (e.g. revenue dollars, units shipped, service units billed, etc.) Identify the time periods that should be used (e.g. monthly, quarterly, weekly, daily etc.) Identify the subgroup level at which the analysis should be conducted (e.g. factory location, branch of bank, customer / patient level, etc.) Perform an initial assessment to identify the ranges of variation that can be expected to be identified.

Detail Steps

Once the quantitative measures have been identifies, along with the subgroup level to be analyzed, the next step is to derive summarized information at the level of the subgroup and for the time period selected. For example, an audit of employee expense reports might be based upon expense reports submitted at the operating unit level, with a time period of one month. In this situation the auditor would need to obtain a summary of the number of expense reports and dollar amount of reported expense by operating unit by monthly reporting period.

Analysis of Range of Variation

In order to present a fairer assessment of variation, larger percentage variations should be weighted more than smaller variations. For example, if an operations volume is steadily rising over the audit period (i.e. consistent, uniform growth), then this raises less of an audit flag than of the volume fluctuates by larger percentage during fewer periods. In order to assign a measure to the volatility of measures during a period, the sum of the squares of the percentage changes measured from one time period to the next time period should be divided by the number of measurements less one. This formula has some similarity to the traditional measure of the variance, but uses as its basic measure the percentage change during the period, instead of the difference between the value of the period and the average of all periods. As a result, this approach results in larger values assigned to operations whose results fluctuate more sharply.

An example of an actual case study will illustrate: In order to identify potentially fraudulent health care claims, the auditor decides to summarize a population of healthcare claims by health care provider and month that the first day of service was provided. Both the number of claims, as well as the dollar amount of the claims, are to be summarized by provider and by month. From this summarized population, a "spike value" is assigned programmatically by ordering the data by provider and month of service and then summarizing the count of claims and dollar amount of claims by provider and by month of service. For each provider, a "spike value" is computed by summing the squares of the percentage changes and then dividing by one less than the number of periods identified. These scores can then be ranked in descending order and the top 5% or top 10 providers then reviewed in more detail. Typically the auditor would want to look at the providers with the most variability in further detail. Typically, the auditor would then focus on these identified providers and extract and chart the data on a more frequent time period, e.g. daily or weekly in order to visualize the underlying activity.

Syntax

```
proc spike data=DATAFILE;
var VARIABLENAME;
by BYVAR;
out = OUTPUTRESULT;
```

Parameters Used

There are four parameters for the Spike procedure:

DATAFILE - the datafile to be analyzed
VARIABLENAME - the numeric variable to be analyzed
OUTPUTRESULT - name of the file to store the results of the analysis
BYVAR - the variable name for the sort sequence

Example Script

```
*;
* spike1.ezs;
* test of spike analysis procedure;
* validated on 7-1-05;
libname test 'c:\temp\spike';
libname ss 'c:\test\ccode\cSSS';
libname drug 'c:\test\model\tested';
proc spike data=test.clmspk6;
var CCOUNT;
by BILLPROV;
out = drug.res6;
run;
```

Example Output

CTL-BREAK	SPIKEVAL	MOVEAVG	HIPCT	COUNT
7997565	0.93163015	1.2	3	131
8990239	0.61845756	1.5	2.5714286	161
89902C0	1.82847222	1.9	0	10
89902FU	2.32101913	17.1	1.5688073	571
89902XA	2.39124787	2.1	1.9411765	64
8995978	0.48194444	1.3	0.875	20

Outlier (p16)

OUTLIERS

Identifying Unusual Transactions

Observations with Values well outside from the norm

Identifying possible outliers from tab separated file. In small datasets, outliers may reside at two standard deviations or more from the population mean. However, once a single outlier is removed from the population, the other data points previously thought to also be outliers, may now fall within the normal range, as the population mean and standard deviation can be narrowed due to the removal of an outlier. This procedure selects the single most extreme

value, in terms of deviation from the population mean and then recomputes the population statistics to identify if any remaining extreme values may also be outliers.

Syntax

```
proc outlier data=DATAFILE;
var VARIABLENAME;
out = OUTPUTRESULT;
```

Parameters Used

There are three parameters for the Outlier procedure:

DATAFILE - the datafile to be analyzed
VARIABLENAME - the numeric variable to be analyzed
OUTPUTRESULT - name of the file to store the results of the analysis

Example Script

```
*;
* outlier.ezs;
* test of outlier procedure;
* validated on 7-8-05;
libname test 'c:\ezs\tab\tested';
libname wic 'c:\cloop\wic\tab';
proc outlier data=wic.trans;
var PAIDAMT;
out = test.out2;
run;
```

Example Output

```
Begin outlier processing: Sun Nov 6 08:07:09 2005
outlier statistics for PAIDAMT in file: /ezs/libin\trans.tab
begin the load outlier data routine...
Loaded 4999 records of outlier data
N is 5000
Variance is 322.67609
Mean is 21.96337
STD is 17.96318701271
nUnderMin99 is 0 pct is 0.00000
nUnderMin95 is 0 pct is 0.00000
nOverMax99 is 82 pct is 0.01640
nOverMax95 is 339 pct is 0.06780
min obsno is 4389 value 1.79000
max obsno is 3097 value 160.44000
Under Min 95 0 Under Min99 0
Over Max 95 1 Over Max99 1
N is 4999
```

Variance is 318.90319
Mean is 21.93567
STD is 17.85786077162
nUnderMin99 is 0 pct is 0.00000
nUnderMin95 is 0 pct is 0.00000
nOverMax99 is 85 pct is 0.01700
nOverMax95 is 342 pct is 0.06841
min obsno is 4389 value 1.79000
max obsno is 3098 value 160.44000
Under Min 95 0 Under Min99 0
Over Max 95 1 Over Max99 1
N is 4998
Variance is 315.12725
Mean is 21.90796
STD is 17.75182382332
nUnderMin99 is 0 pct is 0.00000
nUnderMin95 is 0 pct is 0.00000
nOverMax99 is 90 pct is 0.01801
nOverMax95 is 350 pct is 0.07003
min obsno is 4389 value 1.79000
max obsno is 3083 value 127.61000
Under Min 95 0 Under Min99 0
Over Max 95 1 Over Max99 1
N is 4997
Variance is 312.95350
Mean is 21.88681
STD is 17.69049191292
nUnderMin99 is 0 pct is 0.00000
nUnderMin95 is 0 pct is 0.00000
nOverMax99 is 92 pct is 0.01841
nOverMax95 is 351 pct is 0.07024
min obsno is 4389 value 1.79000
max obsno is 297 value 120.93000
Under Min 95 0 Under Min99 0
Over Max 95 1 Over Max99 1
N is 4996
Variance is 311.05189
Mean is 21.86698
STD is 17.63666322574
nUnderMin99 is 0 pct is 0.00000
nUnderMin95 is 0 pct is 0.00000
nOverMax99 is 92 pct is 0.01841
nOverMax95 is 352 pct is 0.07046
min obsno is 4389 value 1.79000

max obsno is 298 value 120.93000
Under Min 95 0 Under Min99 0
Over Max 95 1 Over Max99 1
N is 4995
Variance is 309.14873
Mean is 21.84715
STD is 17.58262571816
nUnderMin99 is 0 pct is 0.00000
nUnderMin95 is 0 pct is 0.00000
nOverMax99 is 94 pct is 0.01882
nOverMax95 is 352 pct is 0.07047
min obsno is 4389 value 1.79000
max obsno is 3897 value 115.57000
Under Min 95 0 Under Min99 0
Over Max 95 1 Over Max99 1
N is 4994
Variance is 307.45103
Mean is 21.82838
STD is 17.53428167503
nUnderMin99 is 0 pct is 0.00000
nUnderMin95 is 0 pct is 0.00000
nOverMax99 is 93 pct is 0.01862
nOverMax95 is 354 pct is 0.07089
min obsno is 4389 value 1.79000
max obsno is 3898 value 115.57000
Under Min 95 0 Under Min99 0
Over Max 95 1 Over Max99 1
N is 4993
Variance is 305.75196
Mean is 21.80961
STD is 17.48576436523
nUnderMin99 is 0 pct is 0.00000
nUnderMin95 is 0 pct is 0.00000
nOverMax99 is 93 pct is 0.01863
nOverMax95 is 354 pct is 0.07090
min obsno is 4389 value 1.79000
max obsno is 3899 value 115.57000
Under Min 95 0 Under Min99 0
Over Max 95 1 Over Max99 1
N is 4992
Variance is 304.05149
Mean is 21.79083
STD is 17.43707230643
nUnderMin99 is 0 pct is 0.00000

```
nUnderMin95 is 0 pct is 0.00000
nOverMax99 is 93 pct is 0.01863
nOverMax95 is 355 pct is 0.07111
min obsno is 4389 value 1.79000
max obsno is 3900 value 115.57000
Under Min 95 0 Under Min99 0
Over Max 95 1 Over Max99 1
N is 4991
Variance is 302.34964
Mean is 21.77204
STD is 17.38820399442
nUnderMin99 is 0 pct is 0.00000
nUnderMin95 is 0 pct is 0.00000
nOverMax99 is 95 pct is 0.01903
nOverMax95 is 355 pct is 0.07113
min obsno is 4389 value 1.79000
max obsno is 3084 value 109.38000
Under Min 95 0 Under Min99 0
Over Max 95 1 Over Max99 1
computed outlier variance
record count for file /ezs/libin\trans.tab variable PAIDAMT is 5000
observation 297 value 120.93000 is outlier
observation 298 value 120.93000 is outlier
observation 3083 value 127.61000 is outlier
observation 3084 value 109.38000 is outlier
observation 3097 value 160.44000 is outlier
observation 3098 value 160.44000 is outlier
observation 3897 value 115.57000 is outlier
observation 3898 value 115.57000 is outlier
observation 3899 value 115.57000 is outlier
observation 3900 value 115.57000 is outlier
End of outlier processing: Sun Nov 6 08:07:09 2005
```

Sample (p17)

SAMPLES

Select Random Samples

CMA, Random
Selecting random samples from populations. Three types of samples are provided:

- random samples
- interval samples
- cumulative monetary amount (also known as dollar unit sampling)

Syntax - Interval Sample

```
proc sample data=DATAFILE type=SAMPLETYPE ival=INTERVALAMT;
out = OUTPUTRESULT;
```

Parameters Used - Interval Sample

There are three parameters for the Interval sample procedure:

DATAFILE - the datafile to be analyzed
SAMPLETYPE - always "interval"
INTERVALAMT - a decimal specifying the interval to be used for sampling

Example Script - Interval Sample

```
*;
* isample.ezs;
* test of interval sample procedure;
* validated on ;
libname test 'c:\ezs\tab\tested';
libname fs 'c:\cloop\misc\tab';
proc sample data=fs.feesched type=interval ival=300;
out = test.isamp;
run;
```

Example Output - Interval sample

Syntax - Random Sample

```
proc sample data=DATAFILE type=interval seed=SEED;
out = OUTPUTRESULT;
```

Parameters Used - random Sample

There are three parameters for the Random sample procedure:

DATAFILE - the datafile to be analyzed
SAMPLETYPE - always "random"
SEED - the numeric seed to be used to select the first random number

Example Script - Random Sample

```
*;
* rsample.ezs;
* test of CMA sample procedure;
```

```
* validated on 8-2-2005;
libname test 'c:\ezs\tab\tested';
libname fs 'c:\cloop\misc\tab';
proc sample data=fs.feesched type=random seed=7345;
out = test.rsamp;
run;
```

Example Output - Random sample

Syntax - CMA Sample

```
proc sample data=DATAFILE type=cma r=R j=J;
out = OUTPUTRESULT;
var VARNAME;
```

Parameters Used - CMA Sample

There are five parameters for the CMA sample procedure:

DATAFILE - the datafile to be analyzed
SAMPLETYPE - always "cma"
R - R Factor which is 1,2 or 3 (1 results in fewer samples, 3 - more samples)
J - J Amount to be used
VARNAME - the numeric field, amount to be tested

Example Script - CMA Sample

```
*;
* csample.ezs;
* test of CMA sample procedure;
* validated on ;
libname test 'c:\ezs\tab\tested';
libname fs 'c:\cloop\misc\tab';
proc sample data=fs.feesched type=cma r=3 j=5000;
out = test.csamp;
var FSFEFGAM;
run;
```

Example Output - CMA sample

Gap (p. 18)

GAP

Identifying Gaps

Within Sequentially Numbered Items
Identifying possibly missing items within a sequentially numbered series. An example would include missing check numbers from a series of checks. Another example would include a review of sequentially numbered transactions such as purchase orders.

Syntax

```
proc gap data=DATAFILE;
var VARIABLENAME;
out = OUTPUTRESULT;
```

Parameters Used

There are three parameters for the Gap procedure:

DATAFILE - the datafile to be analyzed
VARIABLENAME - the numeric variable to be analyzed
OUTPUTRESULT - name of the file to store the results of the analysis

Example Script

```
*;
* gap2.ezs;
* identify missing invoice numbers;
libname test 'c:\ezs\tab\tested';
libname gap 'c:\ezs\tab';
proc gap data=gap.GapData;
out = test.gapseq2;
var InvNo;
run;
```

Example Output

```
Start the process: Fri Oct 07 15:48:23 2005
Gap analysis for InvNo in file: c:\ezs\tab\GapData.wrk
gap between 0 and 2051
gap between 2079 and 2083
gap between 2253 and 2256
gap between 2869 and 2871
Detected a total of 2056 missing items
End of process: Fri Oct 07 15:48:23 2005
```

Relative Value (p.19)

RELATIVE VALUE

Comparing High Dollar Transactions

Identifying transactions of unusual amount
Computes the ratio of the highest dollar transaction with the amount the next to the highest dollar transaction. High ratios indicate the potential for miskeying or overstatement. For example, invoices from one particular vendor may tend to be bunched around a central figure, but one invoice was miskeyed, for example $1,995 instead of $19.95.

Syntax

```
proc rv data=DATAFILE;
var VARIABLENAME;
out = OUTPUTRESULT;
by VENDORNUMBER;
```

Parameters Used

There are four parameters for the RV procedure:

DATAFILE - the datafile to be analyzed
VARIABLENAME - the numeric variable to be analyzed
OUTPUTRESULT - name of the file to store the results of the analysis
VENDORNUMBER - the field used to identify a control break, e.g. supplier number, employee number etc.

Example Script

```
*;
* RV.ezs;
* test of relative value - RV;
* last tested on 7-16-05;
libname test 'c:\ezs\tab\tested';
libname rv 'c:\test\ccode\cRS';
proc RV data=rv.dhs1;
out = test.rv1;
var CMALAMAM;
by CMBLPRNO;
run;
```

Example Output

GROUP	RVVALUE	HIGH	NEXT	C	UNT
3400002	5.47686	71005.48	12964.64		25
3400002S	1.375	5396.1	3924.43		5
3400007	1.7062	3009.57	1763.9		2
3400008	1.88366	9140.95	4852.77		21
3400014S	1.42857	4288.49	3001.94		8
3400020	2.62934	7887.57	2999.83		12
3400022	1.39718	4769.16	3413.41		10

3400028	1.04257	14282.63	13699.46	68
3400030	1.51519	438059.5	289112.5	90
3400035	1.0738	3918.89	3649.56	20
3400040	4.22585	118275.6	27988.58	93
3400040S	2.05741	6952.61	3379.3	3
3400040T	1.10526	19701.18	17824.88	3
3400042	1.5242	6059.8	3975.73	14
3400047	2.0223	57458.36	28412.32	19
3400047S	1.08008	2730.08	2527.67	4
3400061	1.11057	32467.22	29234.62	23
3400061S	13.01077	109128.7	8387.57	4
3400061T	1.21411	9581.59	7891.86	2
3400064	1.32161	4268.24	3229.57	7
3400065	1.29375	4556.78	3522.16	8
3400068	1.49049	7491.95	5026.51	13
3400069A	4.9357	73151.8	14820.97	57
3400069F	5.73321	12554.25	2189.74	4
3400071	2.90024	11321.09	3903.5	12
3400075	1.35221	6187.37	4575.76	9
3400075S	3.50001	6710.77	1917.36	3
3400085	1.46488	2512.11	1714.89	6
3400087	1.62134	4140.22	2553.58	5
3400091	1.05566	7689.18	7283.77	72
3400091S	1.46099	5021.27	3436.89	11
3400096	1.07267	3091.97	2882.49	9
3400099	1.06864	3245.12	3036.68	11
3400099S	1.02871	3355.38	3261.75	4
3400106	1.10134	2675.55	2429.36	4
3400107	1.21131	4031.61	3328.31	13
3400113	1.47157	42441.69	28841.01	51
3400113S	2.30123	9164.63	3982.5	4
3400120	2.08783	3493.32	1673.18	4
3400123	1.05655	4268.24	4039.79	9
3400126	1.36477	5384.97	3945.71	26
3400127	1.22663	2332.81	1901.8	2
3400130	2.64267	10947.73	4142.68	11
3400132	1.22683	3483.37	2839.33	19
3400133	1.02877	4521.77	4395.31	8
3400141	2.28165	22898.72	10036.04	43
3400141S	1.14286	4092.86	3581.25	5
3400143	1.03031	3306.96	3209.69	9
3400143S	1.08656	1757.72	1617.69	3
3400147	1.27273	6523.51	5125.61	29
3400147S	1.14364	9319.3	8148.8	11
3400151	1.03087	5872.78	5696.94	38
3400151S	1.51216	6334.95	4189.34	3
3400158	1.50664	3229.57	2143.56	3
3403026	2.98298	8668.44	2905.97	2

3404001	1.03333	12138.05	11746.5	9
3404002	1.66667	3966.75	2380.05	8
3404003	1.1	8140.55	7400.5	6
3404004	1.26087	9571.45	7591.15	4
3404016	8.58567	7710.79	898.1	2
4200005	1.32161	4372.35	3308.34	3
4200064	1.02505	3308.34	3227.48	2
4900017	1.02505	2590.95	0	1

Univariate (p.23)

UNIVARIATE

Obtain univariate statistics

Display the field names for each column of data in the dataset. Assumes the dataset has fields separated by tabs, with linefeeds at the end of each line of text. This format for data is used by all EZ-R Stats procedures.

Proc Univariate provides the following statistics:

- **Number of observations in the population**
- **Median value**
- **mode value**
- **Percentiles as follows- First, 90th, 95th, 99th**
- **Quartiles**
- **Range**
- **Mean**
- **Variance**
- **Standard Deviation**
- **Minimum Value**
- **Maximum value**
- **Sum of Squares**
- **Skewness**
- **Kurtosis**

Syntax

```
proc univariate data=DATAFILE;
var VARIABLENAME;
out = OUTPUTRESULT;
```

Parameters Used

There are three parameters for the Univariate procedure:

DATAFILE - the datafile to be analyzed

VARIABLENAME - the numeric variable to be analyzed
OUTPUTRESULT - name of the file to store the results of the analysis

Example Script

```
*;
* uni2.ezs;
* test of univariate procedure;
* last run on 8-12-05;
libname test 'c:/ezs/tab/tested';
libname uni 'c:/ezs/tab';
proc univariate data=uni.dhs1;
out = test.uni2;
var CMNTPDAM;
run;
```

Example Output

```
Start the process: Sun Oct 2 17:35:10 2005
Univariate statistics for CMNTPDAM in file:
c:/test/ccode/cUnivariate/srtwk.tab
Items to process 1134
now start analysis: Sun Oct 2 17:35:10 2005
even items so median is midpoint of 567 and 566
mode is 1950.15000 with a count of 39
first percentile at item 11 amt 793.35000
90th percentile at item 1020 amt 6807.29000
95th percentile at item 1077 amt 8227.03000
99th percentile at item 1122 amt 9728.27000
lowest quartile bottom is 0.00000
25th percentile at item 283 amt 2057.66000
50th percentile at item 567 amt 2945.91000
75th percentile at item 850 amt 4247.77000
N is 1134
Range is 10483.61000
Mean is 3543.20499
Variance is 4131175.309771916
STD is 2032.52928878575
Min is 409.01000
Max is 10892.62000
SumX2 is 18917199651.18383
SumX3 is 114200609256905.00000
SumX4 is 819957979578231400.00000
skewness is 6.763664760
kurtosis is -7.567479884
End of process: Sun Oct 2 17:35:10 2005
```

Linear Regression (p.24)

LINEAR REGRESSION

Determine relationships between variables

Display the field names for each column of data in the dataset. Assumes the dataset has fields separated by tabs, with linefeeds at the end of each line of text. This format for data is used by all EZ-R Stats procedures. Performs a linear regression analysis to determine of their is a linear relatioship between two variables. Values can range from -1 (inverese relationship), 0 - no relationship to 1 (perfect correlation). This enables the analyst to quantify the extent there is a relationship between two variables.

Syntax

```
proc lr data=DATAFILE;
var VARIABLENAME;
out = OUTPUTRESULT;
```

Parameters Used

There are three parameters for the Linear Regression procedure:

DATAFILE - the datafile to be analyzed
VARIABLENAME - the numeric variable to be analyzed
OUTPUTRESULT - name of the file to store the results of the analysis

Example Script

```
*;
* lr.ezs;
* test of linear regression procedure;
* last run on 7-17-05;
libname test 'c:\ezs\tab\tested';
libname wic 'c:\test\ccode\cLR';
proc lr data=wic.lrdata;
out = test.lr;
var X Y;
run;
```

Example Output

```
xy pairs is 40
Coefficient of determination 0.96739
Coefficient of correlation 0.98356
Standard error of estimate 0.57253
```

Pareto (p.26)

PARETO

Quantify the frequency of certain variables

Pareto analysis is the classical 80/20 rule. For example, 20percent of the transactions account for 80% of the value, etc. Postulated by the Italian Statistician Pareto.

Syntax

```
proc pareto data=DATAFILE;
var VARIABLENAME;
out = OUTPUTRESULT;
by BYVAR;
```

Parameters Used

There are four parameters for the Pareto procedure:

DATAFILE - the datafile to be analyzed
VARIABLENAME - the numeric variable to be analyzed
OUTPUTRESULT - name of the file to store the results of the analysis
BYVAR - the sort sequence variable name

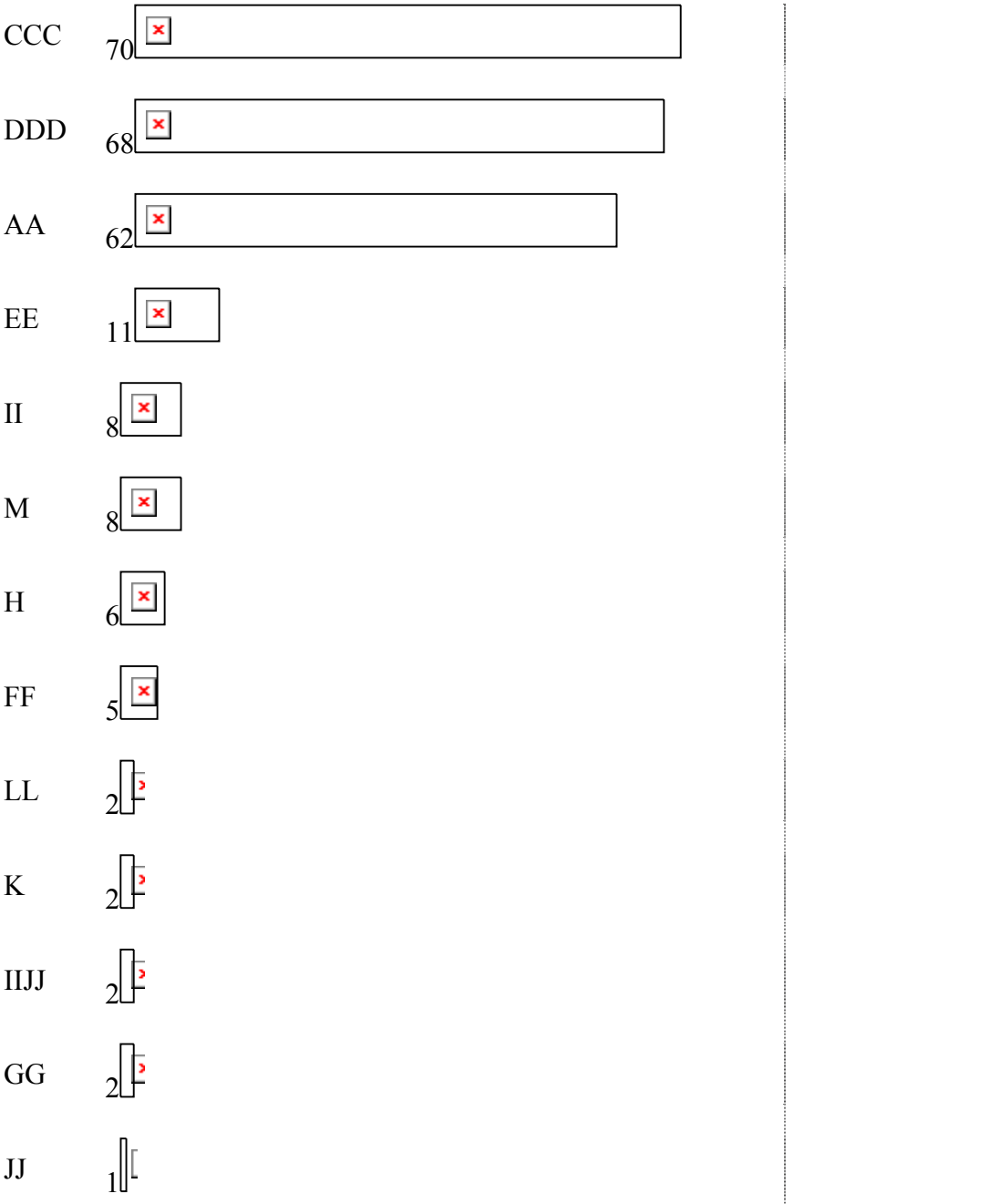
Example Script

```
/*
test pareto
pareto1.ezs
*/;
* last tested 07-24-05;
libname test 'c:\ezs\output';
libname bls 'c:\ezs\tab\bls';
libname par 'c:\ezs\tab\pareto';
proc pareto data=bls.cewenb;
var own;
by area;
out=par.bls;
run;
```

Example Output

Pareto Analysis

prov AMT



Conclusions from Example Data

The example above shows that type 58 is the largest component, at 47.13%. The first two components, i.e. 57 and 58 make up 77.3% of the population, but comprise only 20% of the values.

Domain (p.27)

ODBC

Extract data from database using ODBC

Extract data from a database using ODBC. Query is run against the database by specifying a connection to an ODBC driver.

Syntax

```
proc SQL data=DATAFILE db=DBNAME;
connect CONNECT;
SQL;
```

Parameters Used

There is four parameters for the ODBC procedure:

DATAFILE - where the query results will be stored
CONNECT - the ODBC connection string
DBNAME - always ODBC when running a select against an ODBC database
SQL - the text of the SQL select command

Example Script

```
*
* odbctest1.ezs
* test the select function for odbc
* tested on 7-2-05;
*;
libname temp 'c:\ezs\output';
libname drug 'c:\test\model';
libname test 'c:\test\model\tested';
proc sql data=temp.todbc1 type=odbc;
connect "DRIVER={Microsoft Access Driver
(*.mdb)};DBQ=c:\temp\nc\provider.mdb;uid=admin;pwd=";
SELECT count(*) as cCount from provider;
run;
```

Example Output

```
agglvl AMT PCT CUMPCT
58 6058.0000 0.4713 0.4713
57 3878.0000 0.3017 0.7730
56 1907.0000 0.1483 0.9213
55 673.0000 0.0524 0.9737
```

54 190.0000 0.0148 0.9885
53 109.0000 0.0085 0.9970
52 21.0000 0.0016 0.9986
51 11.0000 0.0009 0.9995
96 8.0000 0.0006 1.0001
0.0000 0.0000 1.0001
50 0.0000 0.0000 1.0001

Conclusions from Example Data

The example above shows that type 58 is the largest component, at 47.13%. The first two components, i.e. 57 and 58 make up 77.3% of the population, but comprise only 20% of the values.

Domain2 (p.20)

DOM2

Domain for a single variable

Determine the domain of values for a single column of data.

This procedure can help familiarize the analyst with the data contained in the variable being analyzed. A frequency distribution is presented for all of the values contained for the variable name specified.

In the example below, data has been obtained from the Bureau of Labor Statistics. One of the variable names is "agglvl" which is the "aggregration level". The purpose of this procedure is to determine which values are contained in this variable name and their frequency. This can be accomplished by running the "DOM2" procedure, and specifying the variable name. In addition, an html report is produced which shows the counts and frequencies in a graphical manner for easier viewing of the population data.

Syntax

```
proc dom2 data=INPUTFILE ;
out = OUTPUTFILE;
var VARNAME;
```

Parameters Used

There are three parameters for the DOMAIN procedure:

INPUTFILE - the name of the variable to be stored in the registry
OUTPUTFILE - the Registry value to be assigned
VARNAME - the name of the variable to be analyzed

Example Script

```
*;
* dom2.ezs;
* test of domains procedure;
* tested on 8-12-05;
libname test 'c:\cloop\bls\tab';
libname drug 'c:\ezs\tab\tested';
proc dom2 data=test.cewenb;
out = drug.dom2;
var agglvl;
run;
```

Example Output

Domain Analysis

agglvl	COUNT
58	1341
57	899
56	472
55	189
54	62
53	36
52	7
51	4
96	1
50	1

Acknowledgements

We would like to acknowledge the following resources which the auditor may find useful in performing their work:

The SQLite Organization: http://www.sqlite.org

The Comprehensive R Archive Network (CRAN) for their "R" System: http://cran.r-project.org/

National Institute of Standards and Technology (SQL92) - http://www.itl.nist.gov/fipspubs/fip127-2.htm