CRSP Data Retrieval and Analysis in SAS® Software: Sample Programs and Programming Tips

Donald P. Cram, Graduate School of Business, Stanford University
Stanford, California 94305-5015, Internet: doncram@gsb-ecu.stanford.edu, 415-725-6862
http://www-leland.stanford.edu/~doncram

ABSTRACT

Center for Research in Security Prices (CRSP) data users would prefer to leave behind the Fortran programs that are most commonly used to retrieve and analyze CRSP data. But while a SUGI 20 paper (Ratnaraj & Katzman) explained the theoretical benefits of using SAS® to deliver CRSP and other large databases at a major US business school, and while SAS/ETS®'s PROC DATASOURCE has been available since 1993, it appears that certain supporting central files at each site, general programming tips, and sample SAS/ETS and Base SAS programs are needed. This paper is intended to address these gaps in service to existing and would-be CRSP SAS users. The main contribution is an extended sample program which executes an "event study", one of the most common CRSP data applications. using central files installed at the Stanford Graduate School of Business site.

But first, I define CRSP data and describe its use. CRSP is the Center for Research in Security Prices at the University of Chicago's Graduate School of Business. It provides well-verified daily and monthly data files covering securities traded on the New York, American, and NASDAQ stock exchanges. The NYSE/AMEX data includes all trading days from July 1962 to the end of the last calendar year; the NASDAQ data file covers trading since December 1972. While many sources provide current stock quotes, CRSP data is the definitive source for academic researchers in accounting, finance, and economics.

Beyond the division between daily and monthly data, CRSP data is further divided into two data sets by trading exchange. One is for firms listed on either the New York Stock Exchange (NYSE) or the American Stock Exchange (AMEX). The other file is for firms traded over the counter under the Automated Quotation system of the National Securities Dealers (NASDAQ).

Each security traded has a CUSIP code assigned by Compustat which is the company identifier used by most CRSP analysts. Compustat has occasionally reapplied CUSIP numbers of expired securities, so the CUSIP user must be careful. Infrequently, the CUSIP includes a letter, so CUSIP variables in programs are defined as character strings. In the annual CRSP data release spanning up to Dec. 31, 1994, the securities are sorted by PERMNO, which is a unique numeric identifier assigned by CRSP. Share prices, trading volumes, and investor returns are reported for each security during its life. The CRSP installation at Stanford University Graduate School of Business (GSB) is fairly typical: GSB purchases the NYSE/AMEX daily and monthly data and the NASDAQ daily data. These occupy 2.4 gigabytes of diskspace. For each CRSP data set, a calendar file contains stock market indices and information to decode the larger data files that hold the returns, share prices, trading volumes, etc. for the individual securities. See the *SAS/ETS User's Guide* for a concise list of all variables available, and the *CRSP Stock File Guide* for their full definitions.

How can CRSP data be retrieved?

University sites get annual data updates on tapes from CRSP and load them to mainframe or network systems. ASCII or binary versions of the files might be read by many programming languages, but most users have adapted the sample programs in FORTRAN provided by CRSP. To provide efficient random access to the data, administrators at some schools have processed the CRSP-provided data to create indexed versions that are random access readable by speedier FORTRAN or SAS programs. A major reason that sites such as Stanford GSB have just offered CRSP ascii files directly is that the programs to set up indexed versions are not readily available. A smaller consideration is that indexed versions consume more diskspace than the CRSP-supplied data files.

Gaining SAS access to CRSP, especially on the Unix systems being adopted at many CRSP sites, can represent a huge service improvement for users. For example, until recently CRSP users at Stanford GSB ran FORTRAN programs on a VAX system which, for CUSIP-sorted input lists of CUSIPs and corresponding data ranges, extracted CRSP data while reading painfully slowly through the ascii CRSP files. Programs could take 24 hours to run and the program would cease output if a CUSIP was out of order or not recognizable, or if a date range did not apply correctly for a CUSIP in the particular file being searched. SAS programs are far more forgiving.

Besides being more robust to CUSIP input list anomalies, SAS programs to read CRSP data are easy to understand. Although FORTRAN programs could be written to relegate tasks to subroutines, the sample FORTRAN programs provided by CRSP are typically long and difficult to learn: they are full of comments, format statements, and detailed read statements to process the complicated headers of the CRSP data files.

The PROC DATASOURCE command in SAS, by contrast, simply attaches the appropriate CRSP files, and is modified by WHERE, RANGE, and KEEP or DROP statements to chop away data that is not of interest. A simple SAS program to print two firms' daily returns for two securities (IBM and Disney,common) for 10 years is succinct:

```
filename NYSEDCAL '/crsp1/udx/CALENDAR.DAT';
filename NYSEDDAT '/crsp1/udx/DATA1.DAT';

proc datasource filetype=crspdcs infile=(NYSEDCAL NYSEDDAT)
out=IBMDIS1;
 where cusip='45920010' or cusip='25468710';
 range from '01jan84'd to '31dec93'd;
 keep cusip date ret;
```

The PROC DATASOURCE statement references the NYSE daily files as installed at the Stanford site and names the SAS dataset it will create; the 'dcs' in the file type tells SAS that the data is Daily, in Character format, and that Security data rather than the index data is wanted (see the *SAS/ETS User's Guide* for a key to all CRSP filetypes). The WHERE modifying statement reduces the data to just IBM and Disney; the RANGE statement cuts the data to the desired time period; the KEEP statement drops series variables other than those listed plus indexing variables that PROC DATASOURCE will not permit dropping (PROC DATASOURCE will not allow the user to drop the date ID variable or any of what are called BY-GROUP variables: these are CUSIP, PERMO, COMPNO, ISSUNO, HEXCD, and HSICCD; WHERE statements can select based on BY-GROUP variables).

This program required 19 minutes and 50 seconds of CPU time, and then just one second to print to the list file. The extract took that long because the CRSP data files accessed are ASCII rather than binary, and non-indexed, hence not randomly accessible. With an indexed, SAS permanent dataset installation of the NYSE data, access would have been much faster.

SUPPORTING CENTRAL FILES

A few business school sites have installed SAS permanent dataset versions of the main CRSP data files. These multiply disk usage fourfold and are difficult to install on Unix platforms due to Unix's 2.1 gigabyte disk partition size limit (although Luan (1994) describes an interesting data management alternative using PROC DATASOURCE to create separate files for each security). However, efforts to speed access to the full CRSP data are not necessarily worthwhile. The CPU time saved by speeding access to the raw CRSP data is small relative to the CPU time involved in further analyses by most CRSP users, and supporting alternative database formats is time-consuming for site administrators. So, I recommend letting users apply PROC DATASOURCE to create custom datasets. Site administrators can help users more by simply providing four central files useful in CRSP data analysis. The SAS permanent datasets which I recommend for a CRSP installation are:

- a "CUSIPLIS" file which users emply to look up CUSIPs, PERMNOs, and trading date ranges for company names or stock tickers of interest,
- a "DATEKEY" data set, which contains a correspondence between calendar dates and trading day numbers, useful in common CRSP applications involving different date windows across securities,
- a "CRSPVRET" data set, which contains the CRSP NYSEAMEX value-weighted return, needed in common applications such as stock Beta estimation, and
- a "MERGRETD" data set which contains merged NYSE and NASDAQ data on returns, by far the most frequently used variable, for the most recent years of daily securities data.

Sample SAS programs to create these four resources can be found in a GSB Technical Report by this author.

PROGRAMMING TIPS

To work with their typically large files, CRSP data users need to learn to index, to select data, and to program SAS macros which will run efficiently.

Indexing datasets

Indexing datasets increases dataset size but will cut runtime of subsequent PROC and DATA steps that use BY and WHERE statements. In PROC DATASOURCE statements an output dataset may be indexed by including the word INDEX as follows:

```
libname '/USERDATA';
proc datasource filetype=crspmcs infile=(NYSEMCAL NYSEMDAT)
 out=USERDATA.ALL9192 index;
 range from 01jan91'd to '31dec92'd;
```

That indexes the output dataset ALL9192 by CUSIP, PERMNO, and the other BY-GROUP variables (COMPNO, ISSUNO, HEXCD, and HSICCD). The diskspace used by the ALL9192 dataset which this saves to disk along with its associated index file, is more than that necessary to retrieve the data by CUSIP or PERMNO alone. Alternatively, a dataset can be indexed in the DATA step that creates it, e.g., to re-index solely by CUSIP:

```
data ALL9192 (index = (CUSIP));
 set USERDATA.ALL9192;
```

Also, a dataset can be indexed by the PROC DATASETS procedure, e.g., to re-index by CUSIP and PERMNO:

```
proc datasets;
 modify ALL9192;
 index CUSIP;
 run;
```

Selecting data from data sets

Selecting observations that meet criteria can be done several ways. For input data, say from a text file named

"stocks" with CUSIP, date, stock price, trading volume, and stock return variables, only "subsetting by if" works:

```
data STOCKS1;
 infile 'stocks';
 input cusip date prc vol ret;
 if prc < 5 or cusip = '01234567' then delete;
```

But to construct new datasets from existing datasets by SET or MERGE statements, "subsetting by where" is faster as SAS checks the truth of the WHERE condition "even before the observation is selected into the program data vector" (Jaffe, p. 167). For example:

```
data STOCKS2;
set STOCKS1;
where prc >= 5 and cusip ^='01234567';
```

And, "subsetting by where" can sometimes obviate a data step entirely. You can screen which data are processed within a procedure, rather than creating a subset dataset first:

```
proc means data=STOCKS1;
where prc < 5 or cusip = '01234567';
```

WHERE can also be used elegantly to select observations in a list:

```
data PEERGRP;
set STOCKS1;
where cusip in ('00176510', '02003910');
```

But if you know the observation number of interest in a dataset, say a list of CUSIPs and firm names, you can subset fastest by "pointing". In this case SAS doesn't have to check all the observations in a dataset--it just goes to the relevant one:

```
data ONECUSIP;
 set CUSIPLIS point=1000;
 put '1000th cusip is ' cusip;
 stop;
```

Here, the PUT statement writes to the SAS log file the value of the variable 'cusip' for the 1,000th observation. The STOP statement is then needed, else SAS reads and puts that one observation again and again, forever; SAS has not reached its normal stopping point which is the end of the file. Pointing to a variable place in the dataset such as the value of a macrovariable 'number' can be accomplished with an extra line that sets the pointer within the data step, as in:

```
data _NULL_;
 numb = &number;
 set CUSIPLIS point=numb;
 put cusip;
 stop;
```

Naming a dataset _NULL_means no new dataset is created. This saves time and workspace when you only need to grab a value out of an existing dataset.

To be complete, SAS also offers a PROC SQL which can be used for subsetting, and it is also possible, in some cases, to avoid subsetting through use of a format table lookup. SAS experts may prefer these last alternatives for their elegance or CPU efficiency, but new users may prefer the methods I present here for their economy of programming time.

Programming SAS Macros

To run SAS programs addressing date-rectangular CRSP data sets (ones where the date ranges for all securities are the same), only an elementary knowledge of SAS is required. But for typical applications involving staggered date windows, one needs to learn to apply SAS macrovariables and macro programming. Jaffe (1994) provides a complete treatment of SAS macros, but I will present a brief tutorial here to cover tools needed in the sample event study program.

SAS "macrovariables" may be defined anywhere in a SAS program by a statement such as:

```
%let VLIST = PRC VOL RET;
```

and the character value 'PRC VOL RET' would be invoked thereafter by '&list'. An example usage is:

```
proc print data=IBMDIS1;
var &LIST;
```

Use of a macrovariable saves retyping a fixed sequence of characters. SAS's structured macro language, however, can take arguments and permit real programming of repetitive operations. For example, a macro program to print just the first 10 observations in a dataset:

```
%macro print10(dataset);
title2 "First 10 observations in &dataset";
run;
proc print data = &dataset (obs=10);
run;
%mend;
```

which is invoked by calls naming specific datasets:

```
%print10(STOCKS1);
%print10(CUSIPLIS);
```

Macros can execute program loops that we'll need later to extract staggered event windows of CRSP data. A simple loop macro which runs the print macro 3 times, for datasets STOCKS1, STOCKS2,and STOCKS3:

```
%macro pr3sets;
%do j = 1 %to 3;
title "Stocks &j";
%print10(stocks&j);
%end;
%mend;
```

This macro is invoked by '%pr3sets' with no arguments. It

defines and uses macrovariable "j". Note that double quotes rather than single quotes in the title statement are used so that the SAS program preprocessor knows to replace '&j' with the appropriate macrovariable value.

AN EXTENDED SAMPLE CRSP APPLICATION PROGRAM

An event study explores whether "events" that might affect stock prices, such as accounting earnings announcements, in fact drive unusual stock market returns in date windows around events. When IBM announces earnings higher than expected, its security return is high as investors bid up its price. But since information leaks and diligent research allow some investors to anticipate the higher earnings, the price might be driven up before a formal earnings announcement, and it might also continue to rise after the announcement as related information is gained by investors. We say that a class of events is significant if, on average, cumulative returns in corresponding event windows generally are high or low, relative to market-wide returns.
The measure of a stock's usual responsiveness to market variation is called its Beta.

The programming task is to calculate, for a list of CUSIPs and corresponding event dates, the Cumulative Abnormal Returns during the window of 40 trading days preceding and 20 trading days following the event date, using a market model with each company's Beta calculated on the trading days in the calendar year preceding the event window. The task is complicated by the need to calculate calendar dates for CRSP data retrieval that span a fixed number of trading days around the event dates given, and by the need to run a regression for each firm to calculate its Beta. We apply the program to an input text file named "eventlist.txt" containing pairs of CUSIPs and event dates:

```
45920010 920130
25468710 890601
45920010 920811
```

The programming strategy is first to create the smallest date-rectangular data set containing all the required information that is possible using PROC DATASOURCE. This is just the returns variable for just the companies of interest, and for just the date range from the earliest date required for any CUSIP to the latest date required. Say such a dataset named MINIMAL is created by a single PROC DATASOURCE statement or by extraction from a site-specific MERGRETD or other file. Second, each "event" is evaluated in a macro loop through the listed event - security pairs. I rely upon my site's date translation file named DATEKEY and my site's CRSP NYSE/AMEX value-weighted market index file named CRSPVRET in the sample program. The macro UPDATECU will calculate the dates bracketing the event and Beta calculation windows; the macro BUILD will run the Beta estimation and cumulate abnormal returns during the event window for one firm; the macro MAIN will run UPDATECU and BUILD in a loop through all pairs of CUSIPs and event dates. The program:

```
libname LIBREF1 '/crsp1/sasdata';
libname LIBREF2 '/crsp2/sasdata';
libname LIBREF3 '/scratch';
libname USERDATA '/userdata';

%global numfirm nextcus nextedat nextcdat evstart evend bstart
bend;

data EVENTS 1;   /*Read and count cusip-event date pairs*/
format eventdt date9.;
 infile 'eventlist.txt' end=LAST;
 input cusip $ eventdt yymmdd6;
 if LAST then if cusip = ' ' then do;
  put 'error: no trailing blanks in cusip list permitted';
  endsas;
 end;
 else call symput ('numfirm', _N_);
 run;

data DATEKEY;
 set LIBREF1.datekey;
 format date date9.;

data CRSPVRET;
 set LIBREF1.crspvret;

data MINIMAL;
 set USERDATA.minimal (keep=date cusip ret);
%macro updatecu(number); /*Update the cusip and date range */
data ONECUSIP;
 N = &number;
 set EVENTS2 point=N;
 call symput ('nextcus', cusip);
 call symput ('nextedat', eventdt);
 stop; run;

data _NULL_;
 set DATEKEY;
 where date = &nextedat;
 call symput ('nextcdat', crspdate);
 run;

data _NULL_;
 set DATEKEY;
 where date = &nextedat-40;
 call symput ('evstart', date);
 run;

data _NULL_;
 set DATEKEY;
 where crspdate = %nextcdat+20;
 call symput ('evend', date);
 run;

data _NULL_;
 set DATEKEY;
 where crspdate = &nextdate-405;
 call symput ('bstart', date);
 run;

date _NULL_;
 set DATEKEY;
 where crspdate = %nextcdat - 41;
 call symput ('bend',date);
```

```
run;

%mend updatecu;

%macro build; /* isolate subset of data for a given company, calc
Beta on relevant range; calc CAR on event window; append event
period data to RESULTS dataset */
%put 'in build at nextcus ' &nextcus;

data ONEFIRM;
 set MINIMAL;
 where cusip = symget('nextcus');
 run;

data BETAFIRM; /* isolate Beta estimation period */
 set ONEFIRM end=LAST;
 where date >= & bstart and date <= &bend;
data BETA_MKT;
 set CRSPVRET;
 where date >= &bstart and date <= &bend;

data BETABOTH;
 merge BETAFIRM BETA_MKT;
 by date;

proc reg noprint data=BETABOTH outest=COEFSOUT;
 model ret = crspvret;

data _NULL_;
 set COEFSOUT;
 call symput ('nextalph', intercep);
 call symput ('nextbeta', crspvret);
run;

data EVNTFIRM;
 set ONEFIRM;
 where date >= &evstart and date <= &evend;

data EVNT_MKT;
 set CRSPVRET;
 where date >= &evstart and date <= &evend;

data CAR_CALC;
 merge EVNTFIRM EVNT_MKT;
 by date;
 if _N_ = 1 then car = ret - (%nextalpha + &nextbeta * crspvret);
 else car = ret - (&nextalpha + &nextbeta *crspvret) + holdcar;
 holdcar = car;
 retain holdcar;
 evstart = &evstart;
 evend = &evend;
 alpha = &nextalph;
 beta = &nextbeta;
 run;

proc append base = CARS data =CAR_CALC(drop=holdcar);

%mend build;

%macro main;
%global numfirm nextalph nextbeta betanobs;
%do i = 1 %to &numfirm;
 %put 'main internation ' &i;
 %updatecu(&i);
 %end;
%mend main;

%main;

title 'Cumulation of  Abnormal Returns during Event Window';
run;
proc print data=CARS;

data LIBREF3.CARS;
set CARS;

endsas;
```

A similar program takes 18 minutes to run on a Unix Sparc20 workstation at GSB. The SAS System is robust to data input errors: if an invalid CUSIP - Event Date pair is given then the program simply reports missing values, while a Fortran program would have failed badly. This program yields results including:

OBS	CUSIP	DATE	RET	CRSPVRET	API	α	B
123	45920010	3DEC91	-.0108	0.000226	0.98892	-.0007	1.03891
124	45920010	4DEC91	-.0136	-0.002032	0.97747	-.0007	1.03891
125	45920010	5DEC91	.0027	-0.005455	0.98572	-.0007	1.03891
126	45920010	6DEC91	-.0138	0.003665	0.96831	-.0007	1.03891
127	45920010	9DEC91	-.0435	-0.002043	0.92821	-.0007	1.03891

which reflect IBM performing below what the market model would predict, during the event period.

Other common CRSP data applications are variations upon the above program. If only the cumulative abnormal return of the entire event window is desired, replace the PROC APPEND statement above by:

```
data CAR2;
 set CAR_CALC;
 if _N_ = 61;
 drop ret crspvret holdcar;

proc append base = CARS data = CAR2;
```

If an abnormal performance index were desired, then replace the CAR_CALC data step by:

```
data API_CALC;
 set EVNTBOTH;
 if _N_ = 1 then api = 1 + ret -
  (&nextalph +&nextbeta *crspvret);
 else api = (1 + ret - (&nextalph +
  &nextbeta * crspvret))*holdapi;
 holdapi = api;
 retain holdapi;
 alpha = &nextalph;
 beta=&nextbeta;
 run;

proc append base = API data = API_CALC(drop holdapi);
```

Even beginner users of SAS can modify sample programs such as this and others in the GSB technical report, and their learning will transfer to other applications.

CONCLUSION

With sample programs and central files installed as I recommend above, CRSP users at a site will find SAS a

better option than FORTRAN programs used previously. The author is confident that using SAS for CRSP access and analysis is a good emerging choice for accounting, economic, and financial researchers.

REFERENCES

Center for Research in Security Prices, *CRSP Stock File Guide: Data Ending Dec. 30, 1994*, Chicago, IL: CRSP 1995.

Cram, Donald P., *Technical Report No. 80: CRSP Data Retrieval and Analysis in SAS Software for Users and Administrators*, Stanford Graduate School of Business, 1996.

Jaffe, Jay A., *Mastering the SAS System*, 2nd Ed. Van Nostrand Reinhold, 1994.

Luan, Peter C. C., "Accessing Restructured CRSP Data Using SAS PROC DATASOURCE: A Downsizing Experience", in *Proceedings, The Third Annual Conference of the Western Users of SAS Software*, September 1995.

Ratnaraj, Paul J. and Carol M. Katzman, "Managing Large Financial Data with Ease: CRSP, Compustat, etc. with SAS", *SUGI 20 Proceedings*, 1994.

SAS Institute, Inc., *SAS/ETS User's Guide, Version 6*, 2nd Ed., Cary, NC: SAS Institute, Inc., 1993.