Lecture 3 [Working with Variables]

Gwinyai T.

May 15, 2017

3 Working with Variables

Successful inquiry in the social sciences presents many challenges, because of the vagaries of the subject matter. A normative economic model, for example, inevitably requires that some assumptions are made. After all, the end product is an abstraction of how market participants behave. Before one can critically evaluate whether the premises enhance understanding of observed outcomes, variables that account for reality need to be defined. Time stamps of events, metrics quantifying their impact, and qualitative descriptors are necessary, but not suffcient, inputs to good modeling.

3.1 Date-Time Constants

SAS has its own way of counting days (the number since January 1st, 1960), and its own way of measuring intraday time (seconds since midnight). It may take a bit of getting used to, but hopefully upon completing today's exercises, the versatility of this system for finance applications will become apparent. The integers 20731 and 20954 correspond to October 4, 2016 and May 15, 2017, respectively. Opening and closing auction times for the New York Stock Exchange are 9:30 AM EST (34200 seconds past midnight) and 4:00 PM EST (57600 seconds after midnight), respectively.

Code reusability is a virtue. One step toward achieving it is eliminating fixed values inside a program's procedural sections. Inserting dates directly into a query, say, causes dependencies that may be unsuitable for future trials. To avoid this practice, known as hard-coding, the next block of instructions defines global variables (visible to all parts of the program). Soft-coding through *%let* statements allows for the selection of different time periods and stocks, while obviating the need to tinker with the underlying architecture of the application.

```
/*  Global Variables  */
%let startDate = 20731;
%let endDate = 20954;
%let open = 34200;
%let close = 57600;
%let permnoList = (92203, 47466, 77418, 90199, 11850, 91926);
```

The previous execution does not generate any output. It merely stores the assigned values in memory for the duration of the SAS session. With the frame of reference communicated, a SQL query can then be directed at the CRSP database. While your code is deploying, take a moment to look at the data definitions again.

```
/*  8 Bits (w.d)
    Nested Query
*/
proc sql;
  create table _stock01
    as select
      a.permno, comnam,
      date format yymmdd10.,
      &open as open format time12.3,
      &close as close format time12.3,
      abs(prc)/cfacpr as P format 8.2, ret as rj format percentn8.2
    from a_stock.dsf as a,
      (select
        monotonic() as rowIndex,
        permno,
        comnam from a_stock.stocknames
      where permno in &permnoList
      group by permno having max(rowIndex) = rowIndex) as b
  where b.permno = a.permno
      and &startDate <= a.date <= &endDate
  order by a.permno, date;
quit;
```

Representing dates as a 4-digit year, 2-digit month and 2-digit day should be quite familiar to you at this point. For legibility purposes, dashes separate years from months and months from days, bringing the number of elements to ten. The on-screen appearance of a time item is governed by similar logic. The total width of the variable here is twelve elements: 2-digit hours, 2-digit minutes, 2-digit seconds and two colons comprise the first eight. An additional colon with three trailing digits brings the number of bits in the variable to capacity, and permits tags that are accurate to the millisecond.

The data item *prc* on the CRSP daily stock file (*dsf*) reports the last traded price for a security. If one is not available, the midpoint of the bid and ask is used instead. Entries in such cases are prefixed with a minus sign. So, the absolute value of *prc* is taken as a correction. Stock dividends (redistributions) and splits (redenominations) affect market quotes. The cumulative factor to adjust prices (*cfacpr*) normalizes the effects of these corporate actions. With the chosen format, the maximum price per share that can be displayed sans computer-generated style modifications (truncating decimals etc.) is 99999.99. Pull permno 17778 and make a note of what happens. Since only one US firm is affected by the 8 bit limitation on P, as of this writing, the code shall remain untouched. Formatting changes are purely cosmetic, and therefore just a matter of personal taste. Now, you know the author's.

Recall how in lecture 1.3 the preferred name for a company was its most recent? The novelty in the current rendition is that the search for each name

is performed in a self-contained block before the inner join is called. Sequential processing of this sort is called nested querying, and it can significantly reduce the time to arrive at a final result. This is a hardware issue. Creating a table forces SQL to read from- and write to- the hard disk drive whenever an action is requested. Select clauses without moorings only need one read from the hard drive. Thereafter, activity takes place in the computer's random access memory, which is considerably faster.

```
/*  Date-Time Manipulations */
proc sql;
  select
    comnam,
    date,
    (date - 1) as dateLagDay format yymmdd10.,
    (open - 1) as openLagSecond format time12.3,
    (open - .001) as openLagMilli format time12.3,
    intnx('week', date, +1) as dateLeadWeek format yymmdd10.,
    intnx('month', date, +1) as dateLeadMonth format yymmdd10.,
    intnx('month', date, +1, 'E') as dateLeadMonthEnd format
    yymmdd10.,
    intck('day', &startDate, &endDate) as calendarDays,
    count(distinct date) as tradingDays
  from _stock01;
quit;
```

Arithmetic operations can be performed on date-time variables as a way to transform them, or calculate intervals. Increments, decrements and differences should all conform to the data type's unit of measure. Adding one to a date variable advances its value by a single day, whereas adding the same integer to a time variable changes it by one second. Finer demarcations may be computed with fractional quantities, as shown by *openLagMilli*. SAS also has its own set of procedures for these purposes. The *intnx(<interval>, <date>, <increment>, **B**eginning/**E**nd)* command takes a date, either advances or rewinds it by a specified number of plain English intervals, and gives a final answer at the beginning or end of the desired period.

intck(<interval>, <start>, <end>) uses similar language to *intnx()*. The output, however, is the number of intervals that have elapsed between the start and end of the period under study. The final point of which to be aware concerns the existence of non-synchronicities between the Gregorian Calendar and the trading calendar. On Friday October 7, 2016, Exxon Mobil ended the day at $86.74. The stock closed out Monday October 10, 2016, up 1.96%, at a price of $88.44 per share. That return calculation references the percentage change after the passage of one unit of time, even though three actual days elapsed between price measurements. Likewise, from the presentation of lecture 1 in 2016 to the submission of the final homework assignment in 2017, fewer price points (154) than days (223) were witnessed.

3.2 Numeric Variables

Exxon Mobil's rise from that Friday to the following Monday is a holding period return, the generic representation of which is:

$$\frac{(P_1 - P_0)}{P_0}$$

Had the company paid a weekend dividend, that income would be included in the equation:

$$\frac{(P_1 + D_1 - P_0)}{P_0}$$

Calculating the intervening holding period returns that cumulated to yield 1.96% would be possible with higher frequency data. Additional insights about the economic factors moving the stock and its industry could be ascertained by matching more finely spaced price ticks with contemporaneous news items. Conjunctively, the patterns of intraday activity provide answers about market microstructure. Over shorter horizons, the mechanics of trade start to matter more for price discovery, and thus are of interest to investors and regulators.

Average returns can be calculated in one of two ways. The arithmetic mean sums individual returns and divides them by the number of observations:

$$\frac{(r_1 + r_2 + ... + r_n)}{n}$$

The geometric average takes the n^{th} root of their product:

$$\sqrt[n]{(1 + r_1)(1 + r_2)...(1 + r_n)} - 1$$

```
/*  Mean and Standard Deviation (per stock) */
proc sql;
  create table _stock02
    as select
      permno, comnam,
      date,
      P,
      rj,
      avg(rj) as rjMean format percentn8.3
      label = 'Arithmetic Average Return',
      exp(sum(log(1 + rj)))**(1/count(date)) - 1 as rjGeo
      format percentn8.3 label = 'Geometric Average Return',
      std(rj) as rjSigma format percentn8.3
      label = 'Sample Standard Deviation'
    from _stock01
    group by permno;
quit;
```

Open up *_stock02* and scroll down to Monday March 27, 2017 for the sixth company. The week starts out in unremarkable fashion before an encouraging 4.11% midweek jump, a precipitous 23.44% Thursday decline, and a modest 2.19% correction to conclude. What happened? Disappointing fourth quarter sales, competition from rivals; Nike and Under Armour, and pessimistic forecasts for the athletic apparel industry conspired to hand Lululemon Athletica its steepest percentage drop in eight years.

Quantifying the economic significance of price fluctuations begs caution. LULU's arithmetic average return over that tumultuous week (from Tuesday onward) works out to -4.453%, and if taken as representative, implies a stock price of $\$64.11(1-0.04453)^4 = \$53.43$ at the end of day four. Adjusting the Monday price by the geometric average instead gives $\$64.11(1-0.05159)^4 = \$51.87$. Take note that the multiplicative property of logarithms is used in the SQL code to determine the geometric mean. The latter result is in agreement with what actually occurred, illustrating the shortcomings of the simple average as a measure of central tendency.

Summary statistics can also prove misleading if the time dimension is not considered carefully. Irrespective of computation method, the 154-day return averages would provide little guidance to an investor facing the XOM and LULU situations discussed. Rolling measures are far more informative for real-time decision-making. The SAS *expand* procedure is ably equipped to handle these scenarios. *expand* takes a time series and interpolates, extrapolates, or calculates moving statistics as the user sees fit. Observations must be sorted by date, with no duplicates allowed, unless they occur in a separate time series (by group). *transformin* is a pre-processing directive that alters the input variable according to the user's wants, and *transformout* does the same to the output. The choice of a moving average window of four days is deliberate. Check LULU's dramatic week in *_stock03* for confirmation of the figures in the previous paragraph.

```
/*  SAS-Specific Time Series Routines */
proc expand data = _stock02 (keep = permno comnam date rj rjSigma)
    out = _stock03 method = none;
  id date;
  by permno;
    convert date = dateLagDay
      / transformout = (lag 1);
    convert date = dateLeadDay
      / transformout = (lead 1);
    convert rj = rjMovAvg4
      / transformout = (movave 4);
    convert rj = rjMovGeo4
      / transformin = (+1)
        transformout = (movgmean 4 -1);
    convert rj = rjMovStd4
      / transformout = (movstd 4);
quit;
```

3.3 Character Variables

A strong command of character manipulations will make you a better finance programmer. Strings are closer to natural language than interpreted code and the underlying hardware that ultimately executes a trade, or runs the center piece regression of an academic study. Navigate to the latest filings section of the Security and Exchange Commission's website. Open any document (10-K, 13F, 144/A...) and read a few lines. Internally, you are aware that you are looking at a corporation's annual report, an institutional investor's quarterly statement of shares held, a privately placed sale of securities, or whatever it is on which you happened to click. If you go back to the root directory of your selection, there should be at least one file with the suffix *XML*. Inside it, you should see words that correspond to the content you were looking at just now, but also many seemingly extraneous entries. Crudely put, XML is one giant string that binds the information you acquired reading that beautifully prepared document (with company logo and all) to the machine, and vice versa. The importance of facility with strings in finance is self-evident.

Character sequences can be combined and reduced in ways analogous to mathematical addition and subtraction. Concatenation is a concept already encountered in Lecture 2.2.6 with tables. String concatenation builds new data items by adjoining old ones. Substringing, in contrast, is the process of removing alphanumeric characters (including blanks) from existing entries. These two techniques are demonstrated next as fictitious email addresses are assigned to the six students.

The CRSP/Compustat Merged Database is exactly what it sounds like. The universes of CRSP and Compustat do not overlap perfectly, because their origin and intent differ. Thus far, CRSP has met all requirements for stock data. However, the field containing company websites is found in Compustat. As a start, Compustat's firm-level identifier, *gvkey*, is extracted from Exxon through to Lululemon. *gvkey* is similar to *permco*, but not *permno*. A database designed to catalog accounting data would naturally have the entire enterprise as its primary key, while one focused on markets should drill down to the individual security level. The reader is encouraged to explore the manuals and reason out why the second condition in the SQL *where* clause restricts the search for a match to within *startDate* and *endDate*. Developing a habit of following hyperlinks is intrinsic to success in this course.

```
1  /*  CRSP Compustat Merge  */
2  proc sql;
3    create view _ccm01
4      as select distinct
5        a.permno,
6        a.comnam,
7        b.gvkey
8      from _stock01 as a, a_ccm.ccmxpf_lnkhist as b
9    where a.permno = b.lpermno
10       and (linkDT <= &startDate and &endDate <= linkEndDt or
       linkEndDt = .E);
11 quit;
```

Armed with *gvkey*, the redundant sounding *hweburl* (historical, web, uniform resource locator) is now the target. The creation of the *_ccm01* object should have been near instant; *proc sql* reports query runtimes in the SAS *Log* screen. A *create view* command imposes even less of a burden on system resources than an untethered *select*. Instructions on the desired appearance of the table are stored in memory, but the entries are only populated when the view is opened, or called in a subsequent SQL procedure.

_ccm02 is an inner join predicated on *gvkey* and the dates of interest. With these constraints, the website retrieved is the one in vogue during that time frame. Calling a join runs the risk of distrupting the ordinal ranking by *permno* in *_ccm01*. Bear in mind that SQL, the farmer (Lecture 2.2.5), does not care about the placement of fruit in the trays, unless compelled to do so with a summary function. There is nothing to summarize here. However, accurate use of the *monotonic()* function is needed for merging on the identification variable, and blindly applying it could result in the wrong numbers going to each stock/website. *order by* is of no help, because it only affects the presentation of the finished table to the next SQL call, whereas *monotonic()* operates on the current one. Therein lies the solution. Since SQL *alter* is hierarchically the same as SQL *create*, the assignment of *ID* can be postponed until after the first query completes. See below:

```
/*  Sequential Queries  */
proc sql;
  create table _ccm02
    as select
      permno,
      a.gvkey,
      comnam,
      b.hweburl
    from _ccm01 as a, A_ccm.comphist as b
    where a.gvkey = b.gvkey
      and (hchgDt <= &startDate and &endDate <= hchgEndDt or
    hchgEndDt = .E)
  order by permno;
  alter table _ccm02
    add ID int label = 'Identification Number';
  update _ccm02
    set ID = monotonic();
quit;
```

The web addresses in the table are conformant to the Hyper Text Transfer Protocol, or HTTP. To reduce them into something the Simple Mail Transfer Protocol (SMTP) can recognize, the substring after *www.* is needed. However, not all entries have the *www.* prefix. SQL if-then logic applied on a case by case basis, using the command *case*, can determine website suitability. If the string of characters starting at position one and ending at position four falls into the unwanted category, the new variable *smtp* consists of the substring from five onward. In any other case, the variable *hweburl* is left as is, and its value passed on to *smtp*.

```
/*  $68. = String 68  */
proc sql;
  create table _smtp01
    as select
      ID,
      case
        when substr(hweburl, 1, 4) = 'www.'
          then substr(hweburl, 5, 68)
        else hweburl end as smtp
    from _ccm02
  order by ID;
quit;
```

The author would not be surprised to learn that the reader's professional email contact is some combination of first and last name "@" employer. With the SQL string concatenation operator, "||", you can create a facsimile thereof for the six students. Performing the next join requires a copy of *_student01* in *Work*. For SMTP purposes, the concatenation presents a slight problem. Since names and websites vary in length, the email address will carry all of the trailing blanks from its constituent parts. The bit sizes in *_student01* were settled upon in Lecture 1.2, in part to accommodate a first name. The *varchar(68)* for *hweburl* can be found in the CRSP documentation, or by right-clicking the column from within the table-viewing window, and inspecting its attributes. Therefore, the SAS *kcompress()* function is called to eliminate whitespace. Finally, if you are so inclined, you may conveniently snitch on the members of the table. They apparently have email accounts with the companies they should be impartially covering.

```
/*  Trailing Blanks */
proc sql;
  create table _student01smtp01
    as select
      a.ID,
      firstName,
      lastName,
      kcompress(firstName||lastName||'@'||smtp) as email
    from _student01 as a, _smtp01 as b
    where a.ID = b.ID
  order by ID;
quit;

/*  Fin!  */
```

Verify your results, save your code and exit SAS.