SHAPING DATA LONG TO WIDE:

Counting Common Characteristics and Making Them into Variables

Part I. Data has multiple records per account

VIEWTABLE: Work.Cards

	acct_num	card_num	last_name	first_name
1	12345	12345	Smith	John
2	12345	12346	Smith	Mary
3	12345	12347	Smith	Jack
4	15794	15728	Lopez	Scott
5	15794	15790	Castro	Verali
6	25678	25800	Orlik	Jorge
7	25678	25805	Pasko	Barbara
8	25678	25807	Orlik	Brandon
9	25678	25808	Orlik	Gwenda
10	45678	45900	Brown	Kent
11	45678	45902	Clarke	Abbi
12	89234	89237	Morin	Vince
13	93467	93467	White	Donna

and user count is required

(Data has to be sorted by the common characteristic in question)

OUTPUT

VIEWTABLE: Work.Cards

	acct_num	card_num	last_name	first_name
1	12345	12345	Smith	John
2	12345	12346	Smith	Mary
3	12345	12347	Smith	Jack
4	15794	15728	Lopez	Scott
5	15794	15790	Castro	Verali
6	25678	25800	Orlik	Jorge
7	25678	25805	Pasko	Barbara
8	25678	25807	Orlik	Brandon
9	25678	25808	Orlik	Gwenda
10	45678	45900	Brown	Kent
11	45678	45902	Clarke	Abbi
12	89234	89237	Morin	Vince
13	93467	93467	White	Donna

VIEWTABLE: Work.Test_firstlast_1

	acct_num	total_count
1	12345	3
2	15794	2
3	25678	4
4	45678	2
5	89234	1
6	93467	1

COUNTING
1. First and Last – Assigning Value 1

```
data test_firstlast_1;
        set cards;
        by acct_num;
        if first.acct_num then total_count=0;
        total_count=sum(total_count,1);
        retain total_count;
        if last.acct_num then output;
        drop card_num last_name first_name;
run;
```

COUNTING
1. First and Last – Assigning Value 2

```
data test_firstlast_2;
      set cards;
      by acct_num;
      if first.acct_num then total_count=0;
      total_count+1;
      if last.acct_num then output;
      drop card_num last_name first_name;
run;
```

1. First and Last – Assigning V 1 and 2 Differences

SUM function – excludes missing values, requires retain statement

SUM statement - assigns zeros to missing values, does not require retain

Both can assign initial zero value to total_count

COUNTING
1. First and Last – Automatic Assignment

```
data test_firstlast_3;
      set cards;
      by acct_num;
      total_count+1;
      if last.acct_num then do;
      output; total_count=0; end;
      drop card_num last_name first_name;
run;
```

COUNTING
2. Do Until

```
data test_dountil;
do until (last.acct_num);
      set cards;
      by acct_num;
          total_count=sum(total_count,1);
end;
       drop card_num last_name first_name;
run;
```

1. First and Last and 2. Do until Differences

DOW – Do Loop of Whitlock – Do Until precedes Set to ensure that data is processed as it is read in, values of total_count are reset to missing at the beginning of each loop

It is not possible to construct Do While (not Last.acct_num) loop as its condition is evaluated at the top and it is terminated before the result can be outputted

COUNTING
3. Proc SQL

```
proc sql;
create table test_procsql as
        select acct_num,
                count(*) as total_count
        from cards;
        group by acct_num
        order by acct_num;
quit;
```

1. First and Last, 2. Do until and 3. Proc SQL Differences

Proc SQL – does not require preliminary sorting and is time effective pulling large volumes of data

Numeric functions in Proc SQL can be used with character variables to include them into grouped dataset (e.g. max, min)

Part II. Data has multiple records per account

VIEWTABLE: Work.Cards

	acct_num	card_num	last_name	first_name
1	12345	12345	Smith	John
2	12345	12346	Smith	Mary
3	12345	12347	Smith	Jack
4	15794	15728	Lopez	Scott
5	15794	15790	Castro	Verali
6	25678	25800	Orlik	Jorge
7	25678	25805	Pasko	Barbara
8	25678	25807	Orlik	Brandon
9	25678	25808	Orlik	Gwenda
10	45678	45900	Brown	Kent
11	45678	45902	Clarke	Abbi
12	89234	89237	Morin	Vince
13	93467	93467	White	Donna

and one record per account is required

(Data has to be sorted by the common characteristic in question)

OUTPUT

VIEWTABLE: Work.Cards

	acct_num	card_num	last_name	first_name
1	12345	12345	Smith	John
2	12345	12346	Smith	Mary
3	12345	12347	Smith	Jack
4	15794	15728	Lopez	Scott
5	15794	15790	Castro	Verali
6	25678	25800	Orlik	Jorge
7	25678	25805	Pasko	Barbara
8	25678	25807	Orlik	Brandon
9	25678	25808	Orlik	Gwenda
10	45678	45900	Brown	Kent
11	45678	45902	Clarke	Abbi
12	89234	89237	Morin	Vince
13	93467	93467	White	Donna

VIEWTABLE: Work.Cardsbyaccount_transpose

	acct_num	last_name1	last_name2	last_name3	last_name4	first_name1	first_name2	first_name3	first_name4
1	12345	Smith	Smith	Smith		John	Mary	Jack	
2	15794	Lopez	Castro			Scott	Verali		
3	25678	Orlik	Pasko	Orlik	Orlik	Jorge	Barbara	Brandon	Gwenda
4	45678	Brown	Clarke			Kent	Abbi		
5	89234	Morin				Vince			
6	93467	White				Donna			

TRANSPOSING
1. Proc Transpose

```
proc transpose data=cards out=cardsbyaccount1
        (drop=_NAME_ _LABEL_)  prefix=last_name;
      var last_name;
      by acct_num;
run;
proc transpose data=cards out=cardsbyaccount2
        (drop=_NAME_ _LABEL_)  prefix=first_name;
      var first_name;
      by acct_num;
run;
```

1. Proc Transpose (Cont'd)

```
proc sort data=cardsbyaccount1; by acct_num; run;

proc sort data=cardsbyaccount2; by acct_num; run;

data cardsbyaccount_transpose;
      merge cardbyaccount1(in=a)
                  cardbyaccount2(in=b);
      by acct_num;
      if a=b;
run;
```

1. Proc Transpose (Cont'd)

Proc Transpose needs prior sorting but will create the necessary number of columns to accommodate all records being transposed

Proc Transpose cannot transpose several variables into one row, instead they have to be transposed one by one

TRANSPOSING
2. Transposing with Arrays

```
proc means data=test_firstlast1 noprint missing;
        var total_count;
        output out=max_obs (drop=_FREQ_ _TYPE_)
        max=max_total_count
run;

data _null_
        set max_obs;
        call symput ('N' ,Trim(Left(max_total_count)));
run;
```


Call symput adds leading blanks to a macro, Trim and Left are used to remove them

2. Transposing with Arrays (Cont'd) Using First/Last and Do Loops 1

```
data cardsbyaccount_array_1;
    set cards;
    by acct_num;
    array ln{&N}  $ last_name1-last_name&N;
    array fn{&N}  $ first_name1-first_name&N;
        if first.acct_num then i=1; else i+1;
            ln{i}=last_name; fn{i}=first_name;
        if last.acct_num;
        if i lt &N then do i=i+1 to &N;
            ln{i}=' '; fn{i}=' '; end;
retain acct_num last_name1-last_name&N first_name1-first_name&N;
keep acct_num last_name1-last_name&N first_name1-first_name&N;
run;
```

2. Transposing with Arrays (Cont'd)
Using First/Last and Do Loops 1

First/Last and Do Loops need a value for maximum records to be transposed, which requires an additional step to get and set N as a macro variable

First/Last and Do Loops need specific instructions to fill the excess records with blanks if number of existing records is less than N

2. Transposing with Arrays (Cont'd)
Using First/Last and Do Loops 2

```
data cardsbyaccount_array_2;
      retain acct_num;
      array ln{&N}  $ last_name1-last_name&N;
      array fn{&N}  $ first_name1-first_name&N;
         do i=1 to &N until (last.acct_num);
      set cards;
      by acct_num;
            ln{i}=last_name; fn{i}=first_name;
       end;
keep acct_num last_name1-last_name&N first_name1-first_name&N;
run;
```

2. Transposing with Arrays (Cont'd)
Using First/Last and Do Loops 2

The second version of First/Last and Do Loops is shortened by combining Do To with Unil (Last) and taking advantage of built in features like assigning missing values to records short of N and incrementing the count by 1 for the next loop

2. Transposing with Arrays (Cont'd) Using initial missing values and First/Last

```
data cardsbyaccount_array_3;
    set cards;
    by acct_num;
    retain acct_num;
    array ln{&N}  $ last_name1-last_name&N;
    array fn{&N}  $ first_name1-first_name&N;
      if first.acct_num then do; i=1;
        do j=1 to &N; ln{i}=' '; fn{i}=' '; end;
      end;
      else i+1; ln{i}=last_name; fn{i}=first_name;
      if last.acct_num then output;
retain acct_num last_name1-last_name&N first_name1-first_name&N;
keep acct_num last_name1-last_name&N first_name1-first_name&N;
run;
```

2. Transposing with Arrays (Cont'd)
Using initial missing values and First/Last

Initial missing values and First/Last (*Jesse Coull's approach*) instructs SAS to create all N records for each account, which then are filled with the existing data or remain blank

This approach has additional advantages if one of the variables in the dataset is a time variable as it can be incorporated into counter

2. Transposing with Arrays (Cont'd) Using initial missing values and First/Last

```
data cardsbyaccount_array_3;
      set cards;
      by acct_num;
      array bal{&N}  $ balance1-balance&N;
      array st{&N}  $ status1-status&N;
         if first.acct_num then do;
            do j=1 to &N; ln{i}=' '; fn{i}=' '; end;
         end;
         index=&N-(intck('month',Date_Entered,today());
         bal {index}=acct_balance; st{index}=acct_status;
         if last.acct_num then output;
retain acct_num last_name1-last_name&N first_name1-first_name&N;
keep acct_num last_name1-last_name&N first_name1-first_name&N;
run;
```

Conclusions and Contact Info

Conclusions:

1. Counting can be the end goal or the intermediate step in data transformation process;
2. Once the maximum value of records is known the ARRAY statement can be used to transform data more quickly and efficiently.

Iryna Nekhayevska, ATB Financial
INekhayevska@atb.com
economics@consultant.com