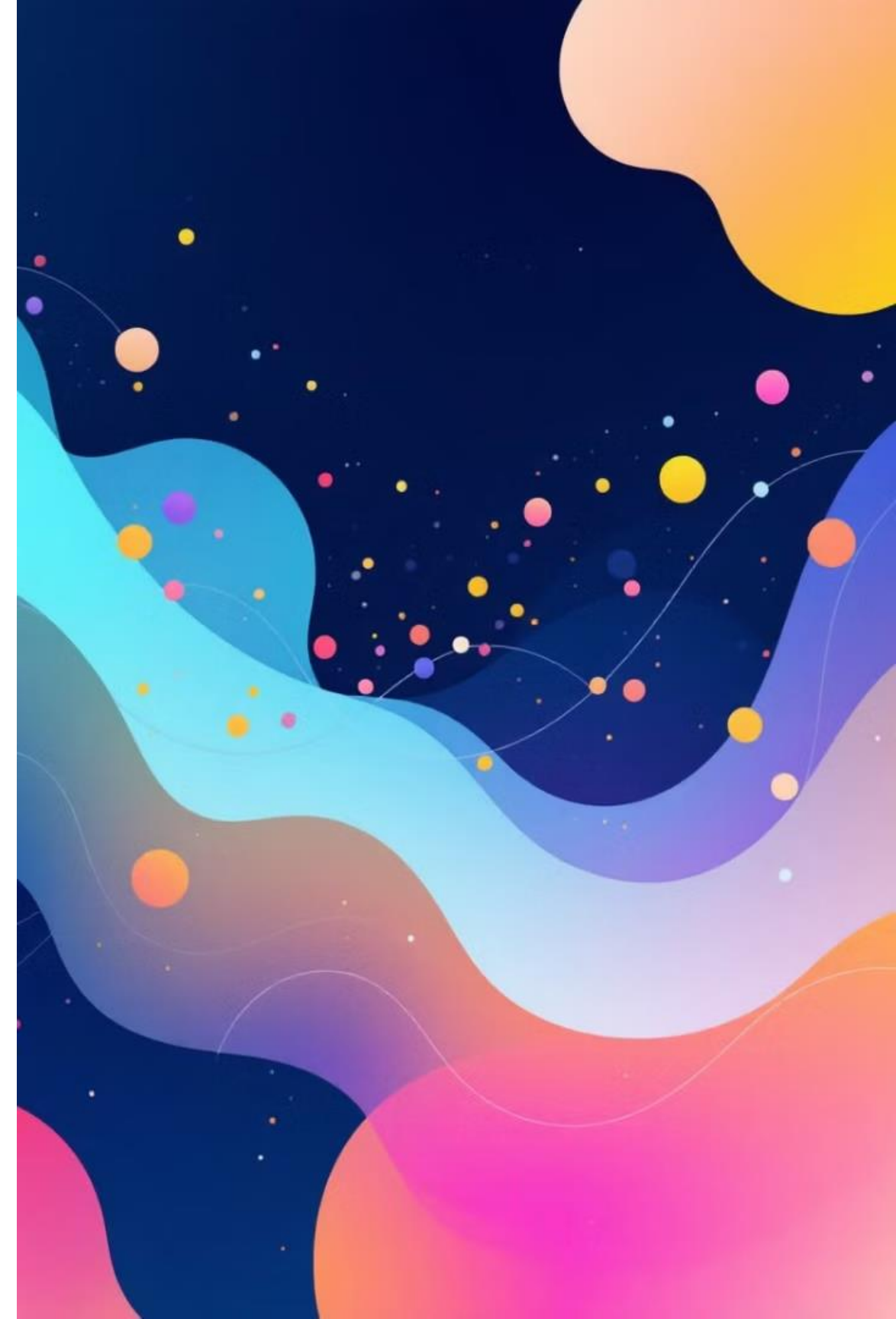


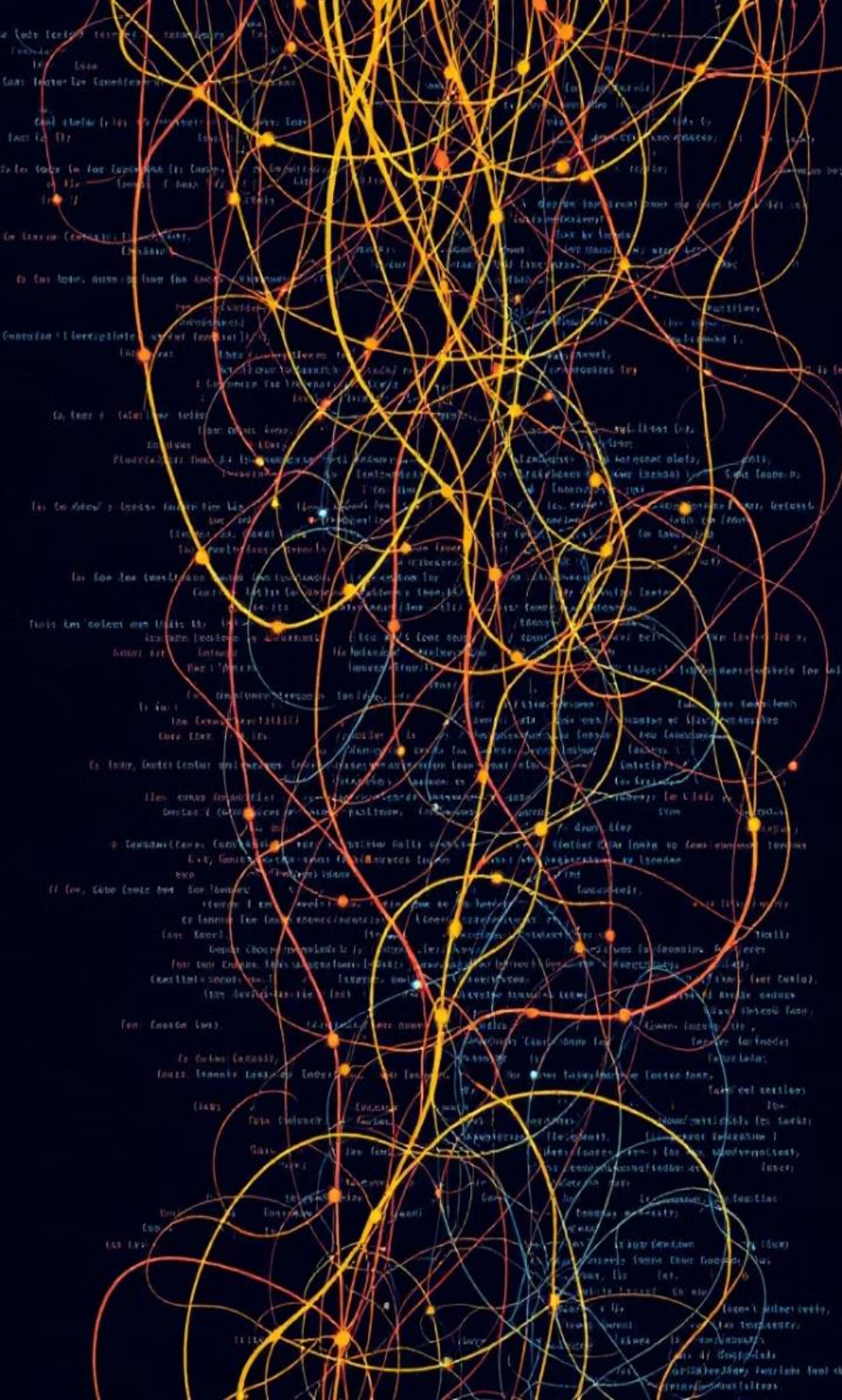
Write Concise ADaM Programs with If-less Programming Techniques

By Lei Zhang and Wei Shao, Bristol Myers Squibb

ADaM is a widely adopted standard for analysis dataset creation in clinical trials. This presentation examines methods to eliminate conditional statements through if-less programming techniques to enhance the readability, reusability and maintainability of ADaM programs.

LZ Presented by Lei Zhang





The Challenges In ADaM Programming

Excessive Use of Conditionals

Conditional statements like IF-THEN-ELSE or SELECT-WHEN are excessively used to derive ADaM variables.

Reusability Issues

While over 95% of ADaM specifications are similar, ADaM programs filled with conditional statements can be challenging to modify and reuse across different studies.

Code Complexity

Excessive use of conditional statements can make an ADaM program unnecessarily cumbersome, difficult to read, and error-prone.

Example: Derive ADSL Variables Using Conditional Statements

```
/* ENRLFL */
if not missing (RFICDT) then ENRLFL = 'Y';
else ENRLFL = 'N';

/* RANDFL */
RANDFL="N";
if missing(RANDDT) then do;
    if ENRLFL = "Y" then RANDFL = "Y";
end;

/* SAFFL */
If ENRLFL='Y' and not missing(TRTSDA) then SAFFL='Y';
else SAFFL='N';

/* BRTHDT */
BRTHDT=.;
if not missing(BRTHDTC) then do;
    if length(strip(BRTHDTC))=10 then do;
        BRTHDT=input(strip(BRTHDTC), yymmdd10.);
    end;else do;
        if length(strip(BRTHDTC))=7 then do;
            BRTHDT=input(strip(BRTHDTC)||'-15', yymmdd10.);
        end;else do;
            if length(strip(BRTHDTC))=4 then do;
                if input(BRTHDTC, 4.) <= (year(RFICDT) - 3) then BRTHDT=input(strip(BRTHDTC)||'-07-01', yymmdd10.);
            end;
        end;
    end;
end;

/* AGEGR1N */
if not missing(AAGE) then do;
    if AAGE < 18 then AGEGR1N=1;
    else if 18 <= AAGE < 65 then AGEGR1N=2;
    else AGEGR1N=3;
end;

/* REGION1N */
If REGION1="US/CANADA" then REGION1N=1;
else if REGION1="EUROPE" then REGION1N=2;
else if REGION1="REST OF WORLD" then REGION1N=3;
else REGION1N=4;
```

If-less Programming Techniques

If-less programming aims to minimize the use of conditional statements, because they often lead to verbose code. By utilizing if-less approaches, programmers can reduce code complexity, decrease program size, and enhance readability and reusability of their ADaM programs.

1

Replace if-else with formats

Use PUT() or INPUT() functions with custom formats

2

Use IFN() and IFC() functions

Replace simple if-else statements with these two functions

3

Use mapping macro functions

Develop inline mapping macros to simplify complex conditional logic

Approach 1: Using Formats Via PUT() and INPUT()

Map values based on predefined formats, good for single-variable transformations



Advantages

- Every SAS programmer knows how to do that
- Can map a large set of values
- Can utilize additional format features



Limitations

- Can't handle multiple input variables, such as many-to-1, and many-to-many mappings
- Requires separate format definitions, which makes the code fragmented.

Example: Using Formats Via PUT() and INPUT()

```
PROC FORMAT;
  Invalue region1n_infmt
    "US/CANADA"=1
    "EUROPE"=2
    "REST OF WORLD"=3
    Other=4
  ;
  value agegr1n_fmt
    1 = "<18"
    2 = ">=18 AND <65"
    3 = ">=65"
  ;
Run;

DATA ADSL;
  ...

  /* REGION1N */
  REGION1N=INPUT(REGION1, REGION1N_INFMT.);

  . . .
  /* AGEGR1 */
  AGEGR1=PUT(AGEGR1N, AGEGR1N_FMT.);
  ...
Run;
```

Approach 2: Use IFN() and IFC() Functions

Replaces simple if-else statements with functional equivalents.

Advantages

- Handle multiple input variables, that is, ***many-to-1*** mapping
- One-line code instead of *if-then-else* or *select-when* code blocks.
- Enables function composition.

Limitations

- Nested IFC() calls become cumbersome and hard to read when multiple choices exist.
- Can be challenging for complex nested conditions.
- Require additional helps from other functions such as WHICHN() and CHOOSEN() for select-when implementation.

Example: Use IFN() and IFC() Functions

```
DATA ADSL;  
...  
    /* ENRLFL */  
    ENRLFL=IFC(NOT Missing(RFICDT), "Y", "N");  
  
    /* RANDFL */  
    RANDFL=IFC(ENRLFL = "Y" and not missing(RANDDT), "Y", "N");  
  
    /* REGION1N */  
    REGION1N=CHOSEN(1+WHICH (REGION1, "US/CANADA", "EUROPE", "REST OF WORLD"),4,1,2,3);  
  
...  
RUN;
```

Approach: Using Inline Mapping Macro Functions

Use the inline mapping macros that encapsulate the composite or combination of IFN(), IFC(), WHICHN(), CHOSEN(), CHOOSEC(), and COMPARE() with simplified syntax and consistent mapping logic and expressions.

Advantages

- Generalization of approach 2 with simplified syntax and usage.
- Create an ADaM variable in ***Var=%F(. . .)*** form instead of conditional statements.
- Mapping rules defined in macro functions are sequentially listed making it clear how the variable is derived. 95% adam variables can be derived with this approach.
- Inline macro functions, which can be combined and used like any built-in SAS functions.

Limitations

- Can't perform ***1-to-many***, or ***many-to-many*** mapping.
- Become cumbersome when the no. of conditions to be listed is greater than 8.
- User-defined functions (by PROC FCMP) is a better choice when you want to produce ***1-to-many***, or ***many-to-many*** mappings.

Inline Mapping Macro Function Overview

%N2C()

Maps one or multiple numeric values to a character value based on a list of num-to-char mapping rules.

%C2N()

Maps one or multiple character values to a numeric value based on a list of char-to-num mapping rules.

%C2C()

Maps one or multiple character values to a different character value based on a list of char-to-char mapping rules.

%COND2N() and %COND2C()

Map one or multiple conditions or logical expressions to a numeric or character value based on mapping rules.

Example 1: Implementation of Inline Mapping Macro (%N2C)

```
/*$ N2C - Purpose: - Mapping a numeric value to a character value.*/
%Macro N2C(
    NUMEXP      /* Numeric expression */
    ,RULE1, RULE2, RULE3, RULE4, RULE5, RULE6, RULE7, RULE8 /* Mapping RULEs */
    ,OTHER=" " /* Value to be returned if no RULEs are satisfied */
    ,SEP=# /* DLM between the first and second part of a mapping RULE */
);
%LOCAL IDX N K MAXK XRULE RULE_L_EXP RULE_R_EXP COND_LST EXP_LST;
%let N=8;
%let IDX=0;
%DO %WHILE (&IDX LT &N);
    %LET IDX=%EVAL(&IDX + 1);
    %if %length(&&RULE&IDX) EQ 0 %then %goto leave;
%End;
%leave:
%if &IDX EQ &N %then %let MAXK=&N; %else %let MAXK=%eval(&IDX-1);
%if &MAXK=0 %then %do;
    %PUT ERROR: Empty numeric-to-character mapping list.;
    %return;
%end;
%let K=&MAXK;
%DO %UNTIL(&K=0);
    %let xRULE=&&RULE&K;
    %let RULE_L_EXP=%sysfunc(scan(&xRULE, 1, &sep, Q));
    %let RULE_R_EXP=%sysfunc(scan(&xRULE, 2, &sep, Q));
    %if %length(&RULE_R_EXP)=0 %then %do;
        %let RULE_R_EXP=&RULE_L_EXP;
        %let RULE_L_EXP=&K;
    %end;
    %if &K=&MAXK %then %do;
        %let COND_LST=(&NUMEXP EQ &RULE_L_EXP);
        %let EXP_LST=&RULE_R_EXP;
    %end; %else %do;
        %let COND_LST=(&NUMEXP EQ &RULE_L_EXP)%str(,) &COND_LST;
        %let EXP_LST=&RULE_R_EXP%str(,) &EXP_LST;
    %end;

    %let K=%EVAL(&K-1);
%End;
Choosec(1+whichn(1, &COND_LST),&OTHER,&EXP_LST)
%Mend;
```

Example 2: Implementation of Inline Mapping Macro (%COND2N)

```
/*$ COND2N - Purpose: Mapping a condition to a numeric value. */
%Macro COND2N(
    RULE1, RULE2, RULE3, RULE4, RULE5, RULE6, RULE7, RULE8 /* Mapping rules */
    , OTHER=. /* Value to be returned if no rule is satisfied */
    , SEP=# /* DLM between the first and second part of a mapping rule */
);
    %LOCAL IDX N K MAXK XRULE ENDFLAG RULE_L_EXP RULE_R_EXP COND_LST EXP_LST;
    %let N=8;
    %let IDX=0;
    %DO %WHILE (&IDX LT &N);
        %LET IDX=%EVAL(&IDX + 1);
        %if %length(&&RULE&IDX) EQ 0 %then %goto leave;
    %End;
    %leave:
    %if &IDX EQ &N %then %let MAXK=&N; %else %let MAXK=%eval(&IDX-1);
    %if &MAXK=0 %then %do;
        %PUT ERROR: Empty character-to-numeric mapping list.;
        %return;
    %end;
    %let K=&MAXK;
    %DO %UNTIL(&K=0);
        %let xRULE=&&RULE&K;
        %let RULE_L_EXP=%sysfunc(scan(&xRULE, 1, &sep, Q));
        %let RULE_R_EXP=%sysfunc(scan(&xRULE, 2, &sep, Q));

        %if %length(&RULE_R_EXP)=0 %then %let RULE_R_EXP=&K;

        %if &K=&MAXK %then %do;
            %let COND_LST=&RULE_L_EXP;
            %let EXP_LST=&RULE_R_EXP;
        %end; %else %do;
            %let COND_LST=&RULE_L_EXP%str(,) &COND_LST;
            %let EXP_LST=&RULE_R_EXP%str(,) &EXP_LST;
        %end;

        %let K=%EVAL(&K-1);
    %End;
    Chosen(1+whichn(1, &COND_LST),&OTHER,&EXP_LST)
%Mend;
```

Example: Rewrite ADSL Variables With Inline Mapping Macros

```
DATA ADSL;
...
/* ENRLFL */
ENRLFL=IFC(NOT Missing(RFICDT), "Y", "N");

/* RANDFL */
RANDFL=IFC(ENRLFL = "Y" and not missing(RANDDT), "Y", "N","N");

/* SAFFL */
SAFFL=IFC(ENRLFL='Y' and not missing(TRTSDA), "Y", "N","N");

/* REGION1N */
REGION1N=%C2N(REGION1, "US/CANADA", "EUROPE", "REST OF WORLD", other=4);

/* BRTHDDT */

BRTHDTC=%COND2C(lengthn(BRTHDTC)=10 # strip(BRTHDTC)
               ,lengthn(BRTHDTC)=7 # strip(BRTHDTC)||'-15'
               ,lengthn(BRTHDTC)=4 & input(BRTHDTC, 4.)<=(year(RFICDT) - 3) # strip(BRTHDTC)||'-07-01'
               ,other=' ')

BRTHDT=input(BRTHDTC, ??yymmdd10.);

/* AGEGR1N */
AGEGR1N=%COND2N( 0 < AAGE<18 # 1 ,18<= AAGE <65 # 2,65 <=AAGE # 3);

/* LSTCTDT */
LSTCTDT=%COND2N(
    (nmiss(DTHDT)=0)#DTHDT
    ,(nmiss(DISCDT)=0)#DISCDT
    ,(nmiss(LSTDDT)=0)#LSTDDT+14
    , OTHER=LSTVDT);
...
RUN;
```



Benefits of Inline Mapping Macro Functions



Versatility

Can be used in data steps, PROC SQL, or PROC FCMP without side effects.



Modularity

The mapping logic and action for each ADaM variable is completely encapsulated within a mapping macro call



Readability

Simple syntax in parameters strings all rules in a sequence, eliminating complex nested function calls.



Maintainability

Concise expressions are easy to understand, modify, and reuse.

Conclusion

Importance of If-less Programming

Adopting if-less programming practices is essential for programmers to develop concise and reusable ADaM programs.

Benefits

These techniques facilitate the easy modification and reuse of ADaM programs and macros, reducing development time and effort.

Future Impact

- Embracing these approaches leads to the creation of more robust and reusable ADaM programs for across-study use.
- AI agents can use them to create more human and machine readable code.

Get in Touch with Your Questions

Have specific queries or need further clarification on if-less programming techniques? Feel free to reach out to us at Lei.Zhang2@bms.com or Wei.Shao2@bms.com. We are here to assist you.

