CHECKERS, We are Keeping the Table

JJ Hantsch, Takeda Global Research & Development Center, Inc., Deerfield, IL

Abstract

The CHECKERS macro is a validated SAS macro for improving the practice of verifying clinical trial tables and listings. CHECKERS reads in unmodified RTF files, received from in-house statistical programmers or CROs and compares their content to a user-created dataset from the raw datasets. This parallel analysis either confirms or denies the validity of the original table’s programming.

CHECKERS does this complex task in a staged fashion. That is, the instantiation of macro parameters can be accomplished a few at a time. The staging is controlled by a single parameter. Initial, exploratory work progresses stepwise; follow-up work (or subsequent runs on subsequent versions of the table) can be executed swiftly. To facilitate rapid reruns of all tables reporting on a clinical trial, multiple calls to CHECKERS, set to pass/fail mode, can be placed in a script.

There are three advantages to using CHECKERS: greater productivity, greater accuracy and greater efficiency. First, CHECKERS eliminates eye-checking, the laborious process of visually comparing two files displayed side-by-side. Eye-checking can lead to many false positive errors. Instead, CHECKERS generates a list of only non-matching entries in the table, which allows the verifier to focus on discrepancies. Secondly, CHECKERS allows a 100% check of every table: every number is confirmed rather than the typical 5-10%. The third advantage is that in the event of any change being made to the source data, a complete revalidation of all tables and listings can be delivered quickly.

Introduction

The table, listing and graph (abbreviated TLG) is the medium of exchange of information between pharmaceutical companies and regulatory agencies. Producing TLGs is one of the major functions of statistical programmers. Because of this import of TLGs, the quality standards are extraordinarily high. When producing the TLGs which summarize a clinical trial, most pharmaceutical companies produce a validation plan, in which they marshal their programmer resources to validate the programs which produce the most important TLGs most carefully. The gold standard for quality control (QC) is independent programming. In this, an independent programmer, the validation programmer, reads the original instructions from a statistician, accesses the same original dataset, recreates the results of the original programmer of the table, listing or graph and then compares, item-for-item, the two results. Obviously, this is very detailed and time-consuming work and many pharmaceutical companies use independent programming either sparingly or farm it out to either junior programmers or contract programmers. I have written a macro, CHECKERS, which automates the comparison of a TLG and a dataset derived from the original data sources and then displays only the disparities.
between the two. An experienced user, using the macro, can greatly reduce the time required to validate that is confirm the accuracy and validity of the programming, a TLG. Thus, more tables and listings can be 100% checked and the accuracy of the validation process is greatly increased.

**The Problem**

A programmer assigned to independently program another’s work on a specific table receives the same instruction set from a statistician. This consists of a table shell, a spec and a location/description of the source datasets. The table shell is a document which demonstrates the general layout of the desired table. The spec is a document which identifies how to fill in the numbers on the table. These must both be written in English, neither SAS code nor even a first approximation of SAS code, i.e. pseudo-code. The location of the source datasets is also documented.

The validator’s task is to confirm each entry within the original programmer’s table or listing. Typically, this is done by 1) recreating the portion of the programming which results in a final dataset, 2) sorting this dataset into a similar order as displayed, 3) printing the results out in as simple a manner as possible and 4) comparing number for number or character for character between the table and the validator’s output. Since tables can frequently reach dozens of pages long and involve hundreds of subjects leading to thousands of data points, verifying a table frequently involves checking a randomly selected subset of the data points. Many programmers employ a rule of thumb to check a minimum of 5% of all numbers, to follow 5% of subjects all the way through the table and check all summarization numbers (totals, sub-totals, etc). This checking is often called ‘eye-checking’ as one moves one’s eye back and forth checking one number on one page to its match on another page.

There are several obvious limitations to this process. First, if you require statisticians to write in English you will find that statisticians can not write in English. Of course, the same can be said of statistical programmers. Most of us describe statistical issues in terms of the tools we employ to resolve them, that is, we’d rather talk war stories about coding than describe the intricacies of the requirements. Our ability to do so is at least one reason we are in this field, our adeptness at it is why we continue to be employed and it is difficult to abandon this and rephrase the issues into English completely devoid of pseudo-code. Second, unless required to, very few of us actually perform the extra work of ordering the elements of a table in a specified order. If we are only looking at a few numbers and they can be found with reasonable effort, then the requirements are met even if this results in imperfect match of the TLG’s display. The 5% rule is a minimum standard and yet in practice for longer tables, it is seldom met. A 1000 page table would require every element relating to 5% of the subjects to be generated and checked. Checking fifty pages by hand would require hours of mind-numbing work. First pages, last pages, significant events and a random sampling of the others are one abbreviation of the 5% rule. For short tables, all results may be checked. Eye-checking is difficult and often inaccurate. On a page full of separate columns with dates and four digit values, repeated values are the bane of accuracy. Finally, every time
new data arrives, or a correction occurs or a new derivation is approved, all TLGs which are affected in any way must be re-checked completely. Eye-checking the same tables in the conventional manner multiple times makes mistakes easier to overlook.

Solution

CHECKERS resolves each of these limitations in validation programming, apart from the language issue that is. To utilize CHECKERS, which matches on a per line basis, the user must match the sort order of the original table. This in of itself makes errors more likely to be detected. If a table and a dataset of different lengths are attempted to be matched, CHECKERS will identify errors. Even a cursory examination will reveal differences in subgroups which may belie important differences in the conduct of the two different analyses. Matching the sort order will by itself, align the subgroups and highlight any such differences.

CHECKERS always conducts 100% matches. Checking every data point means no errors just slip by. Even small, innocuous errors, which can be significant in subsequent interpretation/ analysis will be discovered and indicated by CHECKERS. They can then be identified early and resolved. CHECKERS also eliminates eye checking. There is no fatigue in having the computer match results. Repeated values and similar spellings of different words do not matter to CHECKERS; the match must be exact or it is reported. Lastly, CHECKERS can be put into pass-fail mode which is designed to be rapidly and repeatedly called with minimal user intervention and minimal output. Because the same table may be run more than a dozen times during the course of conducting and reporting on a single clinical trial, CHECKERS can respond to this by allowing rapid execution and rapid reporting on the same table.

Staging

CHECKERS has a pass-fail mode. In practice, the vast majority of program runs utilizing CHECKERS are in pass-fail mode. Pass-fail mode minimizes the output to a single message (either pass or fail); all diagnostic messages are suppressed. A program script can be employed to run the validation programs for all of the tables and listings for a clinical trial. I find this is very convenient when the inevitable data reload occurs. It is possible to include the source directory’s path as a parameter in a script so that with each reload, a single change will permit the validation script to be rerun immediately. This will validate or identify the problems with all of the uploaded data within a few minutes.

Most pharmaceutical companies perform multiple scheduled reloads of data for each clinical trial, but only revalidate a set number of times. This can allow erroneous results to linger in a clinical trial database. With CHECKERS in pass-fail mode and a script which runs first the dataset creation programs, the table/ listing programs and then the validation programs, the last containing calls to CHECKERS, data errors can be detected within minutes of receipt and attention can be directed to them. In practice, this is frequently a “while I’m on the phone with the responsible parties” operation and a reload can be fully checked and resolved of any errors in the same day. CHECKERS utilizing validation programs are written for the first (often either a test or 50%) data transfer and then included in the script in pass-fail mode. The script is then run for every
subsequent data transfer, scheduled or not, within a few minutes. Moving my validation
time to the first data transfer and making subsequent transfers and validation much less
time consuming is also helpful for my workload management.

The programming work of writing validation programs and employing
CHECKERS on this first data transfer is where CHECKERS staged nature comes in
handy. CHECKERS operates under the ‘divide and conquer’ maxim. Each individual
stage in this initial exploratory work is summarized by a single question. CHECKERS
answers each question in turn. The first stage asks the question, “What table and dataset
are to be compared?” CHECKERS displays multiple lines of both. If the table or dataset
is incorrect, the first stage’s macro parameters can be adjusted until the correct ones are
displayed. The second stage asks the question “What lines of in the table are data and
what should be ignored?” CHECKERS can be set to display either the lines of the table
which contain data or the lines which do not contain data and are being excluded. Again,
if these are incorrect, then the second stages parameters can be adjusted. The third stage
asks the question, “Which columns on the page are data and which are not?”
CHECKERS displays a small number of lines with identifying marks (XXXXXXXX)
indicating which columns and how many columns are selected as data. The fourth stage’s
question is “What lines don’t match between the table and the dataset?” CHECKERS
displays the first non-matching lines in this mode. While running in this mode,
CHECKERS does the work of actually checking the results of the table against the
dataset and providing feedback. This is where any errors either, in data interpretation,
table programming or validation programming are identified and resolved. The last stage
is CHECKERS pass-fail mode. The pass message and the fail message are purposely set
up to be visually very distinctive and draw attention to any failing validation attempt.

CHECKERS parameters

There are four sets of parameters for controlling CHECKERS. A single
parameter, STAGE, controls the stage of analysis. The only permitted values are: ONE,
ONEB, TWO, THREE, FOUR and FIVE. Stage ONEB is a bonus stage. Stage FIVE is
the pass/fail mode which will be the most common mode for CHECKERS to run in.

Parameters INFILE and MYDSET identify the two comparators. INFILE is the
location reference of the rtf file to be read in. Note that in this paper, examples use UNIX
locations. MYDSET is a SAS dataset located in the work directory. It is intended to be a
dataset on which statistical analysis and sorting has been completed. CHECKERS
assumes that the MYDSET is sorted in an ordering identical to that of the INFILE as it
matches line for line.

STOPS is a list of user-defined stop words for deleting lines from the read in
table. Title lines, headers, footers, solid lines across the page and column headings, etc.
are all irrelevant to matching the items in the table. CHECKERS has a pre-set list of stop
words: Note:, Listing, Table, Safety, Concurrent, Contd., Development, __________, Page & Protocol all trigger the line to be ignored. The list the user enters with STOPS allows
additional control lines to be ignored. A blank entry will function.
Explanation of the table: the pink boxes, such as Preferred are items which are matched to elements of the STOPS parameter. These are stop words and they indicate that that entire line is not to be included in the dataset. Great care must be taken so as not to eliminate a real data line with an inadvertently matching word. The blue boxes, such as 8 (2.2%) are data items which are located by the SPACER parameter. The values in the blue boxes are all that will be considered for matching. They are located by a pointer location.

CHECKERS matches line for line so a missing second line of an 1100 line table would generate 1099 lines of output. To control this problem, LINES and FIRST are parameter inputs to an embedded proc print command within CHECKERS. They control how many lines and which counted line is first to be displayed. If there are forty lines to display and wish to observe only the last ten, then LINES=10 (to display only ten lines) and FIRST=30 (to start displaying at the thirtieth line) would be the appropriate values. These parameters are useful because they limit the amount output; the default values are 25 and 1. With the parameters at the default settings only 25 lines are generated.

Pass-Fail Call
%checkers(stage=FIVE, infile="'home/jhantsch/T1_1_1.rtf'", stops="Note:, Listing, Table, Safety, Concurrent, Contd., Development, ________, Page, Protocol'", spacer="53, 71, 87, 101, 120", lines=25, first=1, mydset=last2);

The output will be: “All results match. Zero (0) errors for table: ‘home/jhantsch/T1_1_1.rtf’”
**Exploratory Calls** (one example for each stage)

%checkers(stage=ONE, infile="'home/jhantsch/T1_1_1.rtf'", stops=, spacer=, lines=10, first=1, mydset=last2);

This will reprint the first ten lines of the rtf input file T1_1_1.rtf and the first ten records on my dataset, last2.

%checkers(stage=ONEB, infile="'home/jhantsch/T1_1_1.rtf'", stops="Note:, Listing, Table, Safety, Concurrent, Contd., Development, ________, Page, Protocol", spacer=, lines=25, first=1, mydset=last2);

This will print out the first 25 non-duplicate lines from T1_1_1.rtf which will be excluded. This mode is very useful for identifying data lines which are inadvertently excluded because they match stop words.

%checkers(stage=TWO, infile="'home/jhantsch/T1_1_1.rtf'", stops="Note:, Listing, Table, Safety, Concurrent, Contd., Development, ________, Page, Protocol", spacer="53, 71, 87, 101, 120", lines=10, first=1, mydset=last2);

This will print out only the lines of data; in this case only the first 10 lines of data. The value of LINES is 10. Below are just two lines.

<table>
<thead>
<tr>
<th>Bundle Branch Block</th>
<th>1 ( 0.3%)</th>
<th>0 ( 0.0%)</th>
<th>0 ( 0.0%)</th>
<th>1 ( 0.1%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cardiac Flutter</td>
<td>0 ( 0.0%)</td>
<td>1 ( 0.3%)</td>
<td>0 ( 0.0%)</td>
<td>1 ( 0.1%)</td>
</tr>
</tbody>
</table>

%checkers(stage=THREE, infile="'home/jhantsch/T1_1_1.rtf'", stops="Note:, Listing, Table, Safety, Concurrent, Contd., Development, ________, Page, Protocol", spacer="53, 71, 87, 101, 120", lines=25, first=1, mydset=last2);

This will print out five lines of data with XXXXXX locating where the pointers in SPACER are that locate data. For example:

<table>
<thead>
<tr>
<th>Bundle Branch Block</th>
<th>1 ( 0.3%)</th>
<th>0 ( 0.0%)</th>
<th>0 ( 0.0%)</th>
<th>1 ( 0.1%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cardiac Flutter</td>
<td>0 ( 0.0%)</td>
<td>1 ( 0.3%)</td>
<td>0 ( 0.0%)</td>
<td>1 ( 0.1%)</td>
</tr>
</tbody>
</table>

This can prove quite helpful for locating where the data is on the line and adjusting to read just the desired values.

%checkers(stage=FOUR, infile="'home/jhantsch/T1_1_1.rtf'", stops="Note:, Listing, Table, Safety, Concurrent, Contd., Development, ________, Page, Protocol", spacer="53, 71, 87, 101, 120", lines=10, first=20, mydset=last2);

This will display ten lines of errors starting with the 20\textsuperscript{th} error line. If there are no errors a brief congratulatory message will be printed instead.
Conclusion

CHECKERS can save you time and make your validation efforts more productive, more accurate and more efficient. More productive means 100% of the TLGs are validated. More accurate means 100% of the data is checked. More efficient means reruns can be accomplished in a minute or less. Using CHECKERS, an experienced statistical programmer can effortlessly perform a 100% validation of 100% of the tables reporting on a clinical trial, even through multiple data transfers including the inevitable last-minute one.

Contact

J.J. Hantsch
Senior Statistical Programmer
Mail Stop #133
Takeda Global Research & Development
One Takeda Parkway
Deerfield, IL 60015

jhantsch@tgrd.com