PhUSE 2022

Paper DV07

Pixel Plots: Identify Missing Data, See High Level Patterns and Break Axis Constraints

Paul Grimsey, Roche Products Ltd, Welwyn Garden City, UK

ABSTRACT

Before starting analysis, it is useful to know if all your data is present and if there is anything unexpected or needing further investigation. By representing each received result by a single coloured cell, pixel plots can help you to quickly identify missing results and unexpected patterns in the data. This visualisation gives the Data Analyst / Scientist a very good feel for how much data is present and for which patient visits. Limitations of plot axes are overcome by exporting to Excel, which removes issues found when plotting too much data into static plots. This paper will show how to create pixel plots in R and SAS, along with template code for both languages for the basic visualisation, using a single SDTM dataset. More complex real life examples of the visualisation will also be presented, highlighting where this visualisation can be a very useful addition into the Data Scientist's toolkit.

INTRODUCTION

How do you know that you have all of your data? It is a simple question but the answer may not be so simple. If there are a number of different functions involved in the data flow chain, delays and potentially loss of data can occur on its way to you. The main purpose of traditional tables, listings and graphics (TLGs) is to show the content of the data, so finding what is not there is not always obvious. Answering the question "do we have all of the data?" was the original inspiration for creating this visualisation. However, finding what data is missing from an ongoing clinical study is much harder than just displaying what data is currently available. Key decision points, such as: interim analysis; dose escalation; dose decision; or ongoing safety monitoring, often necessitate data analysis during the course of a clinical study. After quite some time working on an over-engineered solution to this problem, it became apparent that a much-simplified version of this visualisation would still be very useful. By simplifying this visualisation, more applications became apparent and some of these are examples presented in the *Other Examples of Pixel Plots* section.

WHAT ARE PIXEL PLOTS?

Pixel plots are the representation of a 2-dimensional data set. In these plots, each pixel refers to a different value in a data set[1]. While the plots described in this article may not strictly be pixel plots, this classification is a very close match. An example of the visualisation, shown in Figure 1, plots patient ID on the x-axis and the scheduled visit (including scheduled time-point) on the y-axis. The user then assigns colours to a dataset value, or a set of values,

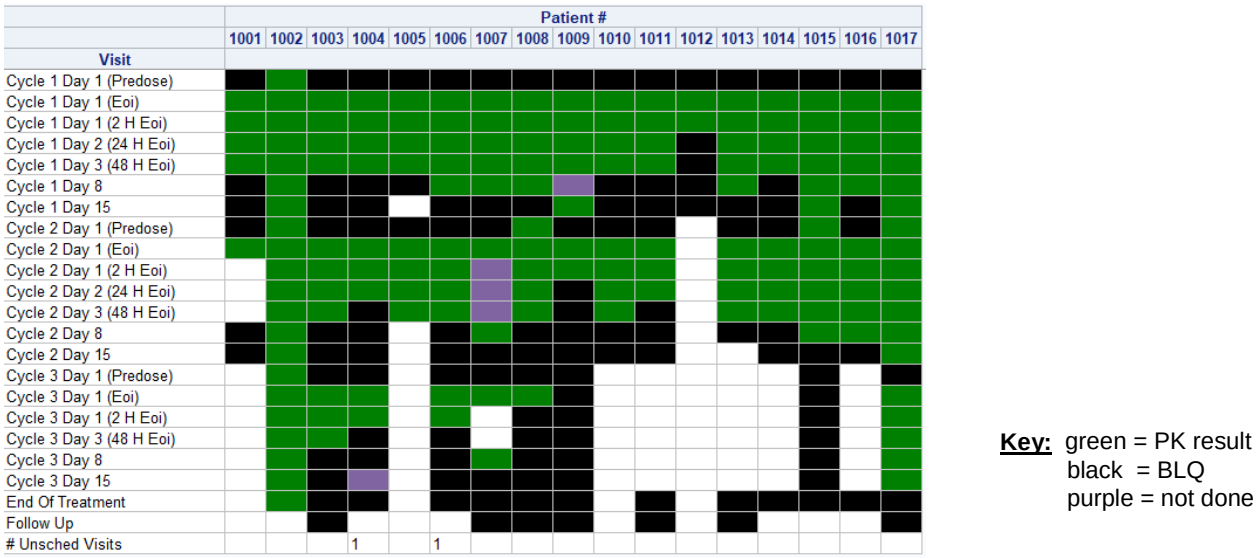


Figure 1: The basic visualisation, plotting patient number of the x-axis and scheduled visit and timepoint on the y-axis, using a single SDTM as a data source.

which are then plotted as cells. Absence of data can be displayed as either empty cells, or coloured cells, depending upon the needs of the visualisation. Figure 1 displays a section of a basic visualisation, showing received Pharmacokinetic (PK) data for a clinical study. In this visualisation, numeric results are displayed as green cells, BLQ values as black cells and results databased as "not done" are shown as purple cells.

WHY ARE PIXEL PLOTS USEFUL?

Pixel plots allow you to see exactly how much data is present. You can also see where data has not yet been received. Once identified, you can investigate to see if these gaps are justified. Data issues may also be identified, depending on how the cells are coloured. For example, in Figure 1, patient 1002 has a PK concentration before the drug was administered, which should not be possible and needs further investigation. By including some simple cell colouring, trends can be identified by the user. For example, the colour of the cell could be changed when:

- a patient develops anti-drug antibodies
- there is an unexpected dosing regimen
- a patient achieves a given efficacy response
- a patient triggers an adverse event of interest.

STATIC PLOTS VS OUTPUT TO EXCEL

Initially these visualisations were created in SAS as static plots using PROC SGPLOT. A number of limitations to this approach soon became apparent. If there were very little data, the spaces between the plot symbols (cells) would be very large, making it difficult to read. Figure 2 shows an example of a visualisation for 2 patients, both having 2 results (displayed as blue cells).

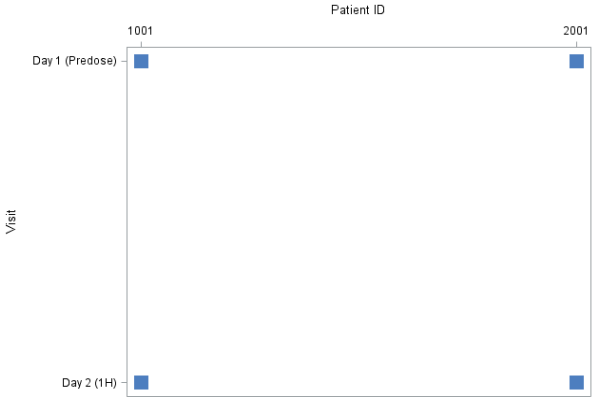


Figure 2: PROC SGPLOT representation of 4 results for 2 patients over 2 visits. Note the large spacing between the "cells".

The symbol size here can be adjusted manually in the code but it is not efficient to do this at every data delivery. By contrast, having lots of data can cause the merging of symbols, making the plot unreadable. There can also be

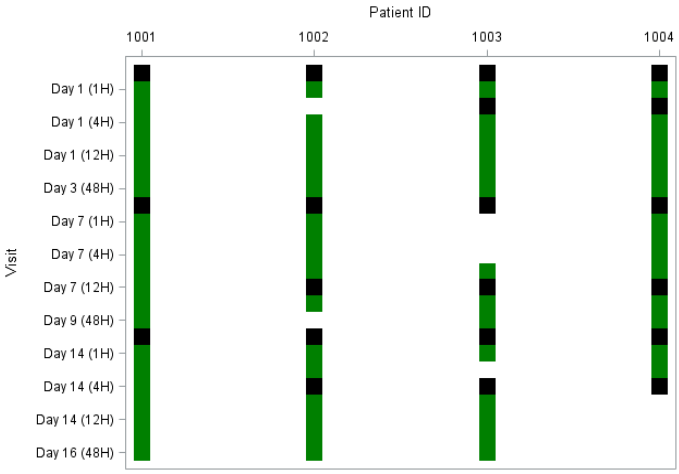


Figure 3: PROC SGPLOT representation of result data for 4 patients with 24 visits. Note the loss of cell definition and loss of information on the x-axis.

loss of information on the axes in order to fit in the data. Figure 3 shows some dummy data for four patients, with green cells indicating a PK concentration and black cells showing BLQ results.

Figure 3 shows data for predose, 1, 2, 4, 8, 12, 24 and 48 hours post dose for days 1, 7 and 14. The amount of data shown in this example has resulted in the "cells" merging. Information has also been lost on the y-axis, for example the predose, 2H, 8H and 24H time points are not shown. Again, this could be adjusted manually in the code but it is far from efficient. Outputting the visualisation to Excel overcomes all of these limitations as:-

- The cells are always the same size and no truncation of the data occurs.
- If there is more data than can be displayed on a page then the zoom function, row and column sliders can be used to easily navigate the visualisation.
- The header row of the visualisation can be frozen in the code, so that you always see the columns headers.
- Data can be split over multiple worksheets e.g. for different treatment groups or parameters (see Figure 4 for an example):

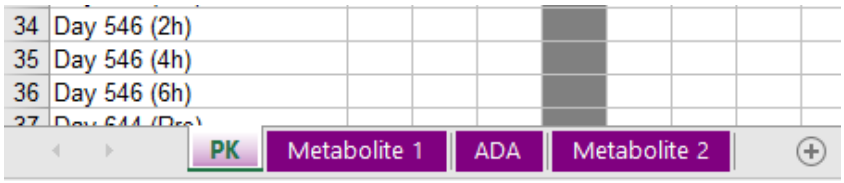
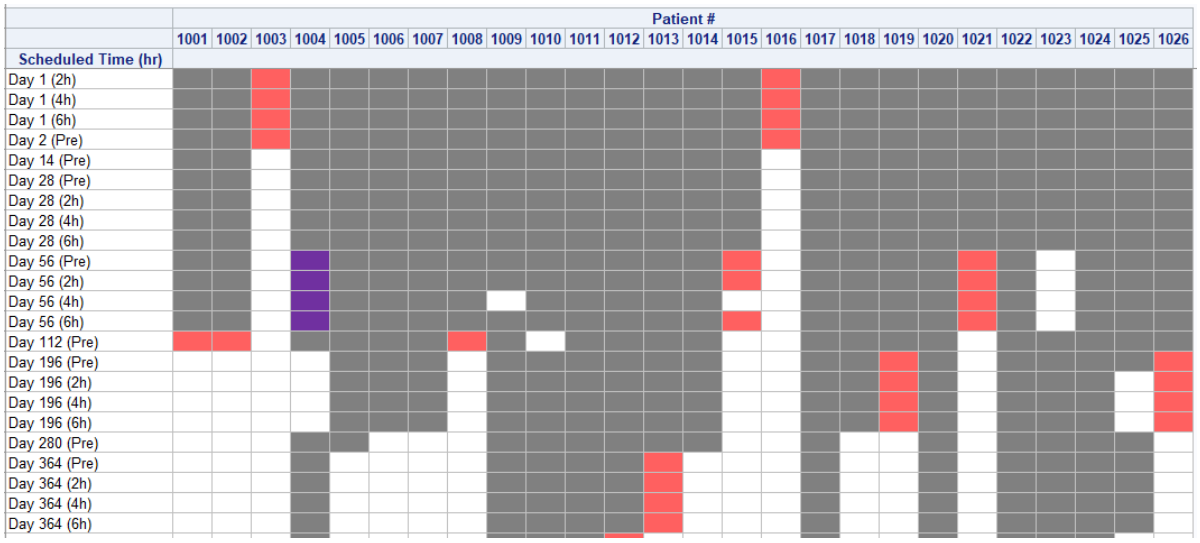


Figure 4: Splitting the visualisation over multiple worksheets.

OTHER EXAMPLES OF PIXEL PLOTS

So far, we have seen a basic example of the visualisation, which uses a single SDTM dataset in its creation. These visualisations can be as simple, or complex as required. Be aware that these visualisations can be difficult to maintain if they are trying to do too many things. Below shows some other examples that the author has created whilst supporting a variety of different clinical studies. These examples illustrate just a few applications for this type of visualisation and maybe the reader is able to think of many more.

1) **New Data Visualisation**



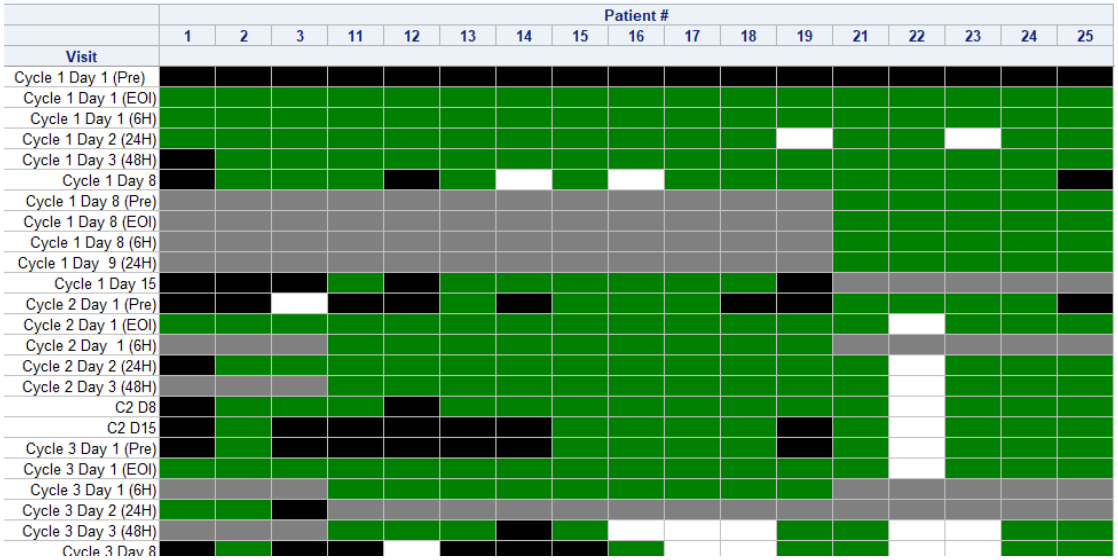
Key: grey = unchanged data
red = new data
purple = data changed since the last delivery

Figure 5: Visualisation of new and updated data.

This is an extremely useful visualisation, which the author uses each time new PK/PD data is received. Older data is represented by a dull colour, so as not to draw the attention. The new data is represented using a brighter colour, so it can be easily identified. Removed, or changed data since the last data load can be shown using another colour which stands out. The visualisation requires combining the old data with the new data to create a dataset, which

identifies what data is: unchanged; new; changed or removed. It is a very good way of communicating newly received data with other functions that are also analysing the data. Figure 5 shows an example of this type of visualisation.

2) Basic Visualisation Greying Out Non-Protocol Visits

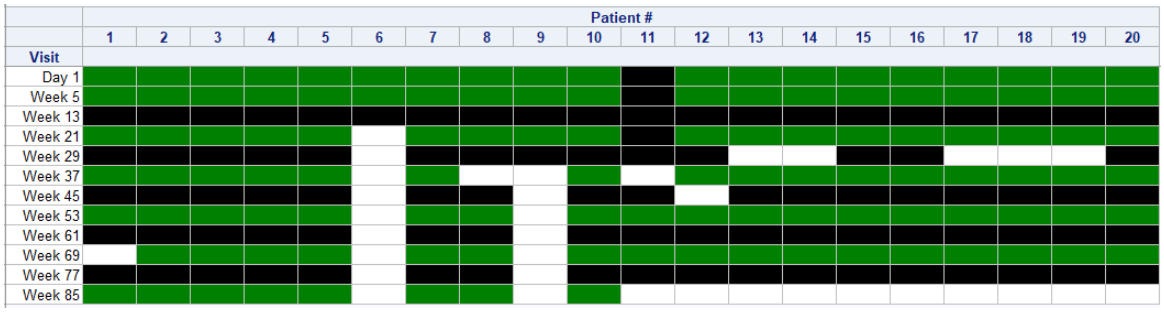


Key: green = PK concentration
black = BLQ
grey = visits unexpected as per protocol
blank = no data received

Figure 6: The basic visualisation, greying out visits which are not in the patient Schedule of Assessments.

The basic visualisation has many benefits, such as seeing the entirety of the day, identifying gaps and seeing high level trends. One drawback is that there can be a lot of blank space. If the visualisation shows patients with different protocol Schedule of Assessments (SoAs), then the visualisation can be misleading. If a patient is never going to attend a visit due to their SoA, then this should not be shown as a blank cell. The above example shows the visits that are not expected as grey cells. This visualisation requires a dataset representation of the SoA from the protocol, which is combined with the result dataset. Figure 6 shows an example of this with 3 different patient cohorts, each with a different protocol SoA: 1) patients 1 – 3, 2) patients 11 – 19; 3) patients 21 – 25.

3) Dosing Visualisation

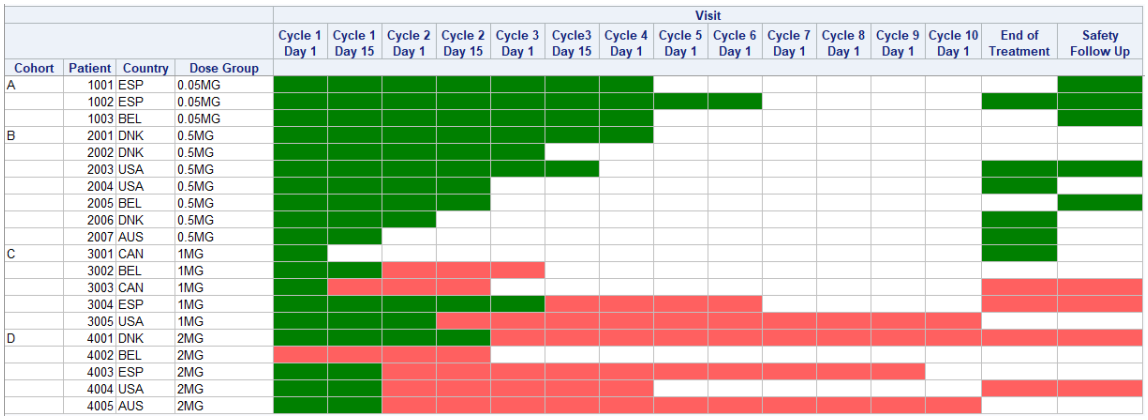


Key: green = active treatment
black = placebo

Figure 7: Basic visualisation, showing dosing data from EX.

Figure 7 shows another example of the basic visualisation, which only requires a single SDTM dataset, in this case EX. This visualisation was created to spot unexpected dosings within an 800 patient Phase III study. The different treatment groups were split over multiple worksheets to reduce the amount of data displayed at a time. The visualisation was extremely valuable as it allowed missing doses and an unexpected dosing schedule to be found. The dosing schedule for this study is as follows: an active loading dose is given on Day 1 and then another active dose at Week 5, further active treatments are then given every 16 weeks (Q16W) thereafter. Treatments for the whole study are given as Q8W and so non-active dosings in the Q16W are given as placebo, so as not to unblind. Here an unexpected dosing regimen can be seen for patient 11, who should have received active treatment before placebo but does not receive active treatment until Week 53. Being able to spot this strange dosing allowed the programmers to use this patient to help ensure derivations using dosing dates were correct.

4) Anti-Drug Antibody (ADA) Visualisation

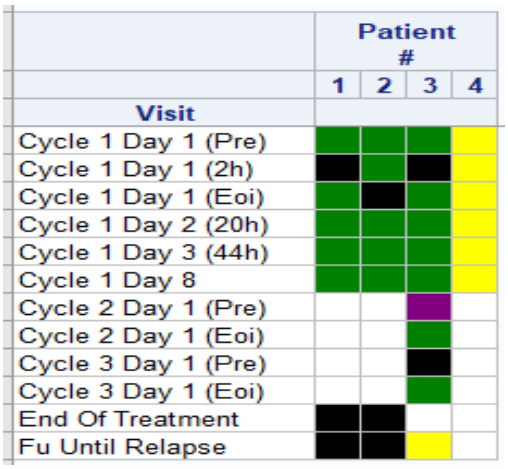


Key: green = ADA negative
pink = ADA positive

Figure 8: Visualisation showing where patients become ADA positive.

Figure 8 shows if and when patients become ADA positive during the study. Pink cells show visits that are ADA positive and green cells show visits that are ADA negative. The Clinical Pharmacologist had been creating this visualisation by hand for presentation to the project team at each data delivery and asked if this could be achieved programmatically. In previous figures, patient IDs have been shown on the x-axis and scheduled visits along the y-axis. This specific visualisation required the scheduled visits on the x-axis and patient ID on the y-axis. Additional demographic information was also requested which came from DM.

5) Visualisation Showing Missing or Expected Data



Key: green = PK concentration
black = BLQ
purple = not done
yellow = visit taken but no PK results received

Figure 9: Basic visualisation highlighting where data is potentially missing.

The purpose of this visualisation is to get a full picture of the data. As with the basic example, green and black cells depict what data is present and purple cells represent samples databased as not done. As before, this data comes from a single dataset, in this case PC. The addition to this visualisation are the yellow cells, which depict what data is not yet present but expected. By creating a dataset of expected PK visits from the protocol SoA and combining this with the subject visit dataset (SV), it is possible to determine which PK visits have been taken. If these expected visits do not have any PK sample data associated with them, they are coloured yellow in the visualisation (see Figure 9). This means that 3 data sources are required to produce the visualisation: PK concentrations (PC); PK SoA dataset (user-defined); subject visit dataset (SV).

The concept is quite simple: If you know what you have and what you are expecting; you can see what is missing. In reality, this type of visualisation can take a lot of work and require ongoing maintenance. Most of the work involved is in creating the user-defined SoA dataset. For early phase studies, protocol amendments and addition of new patient cohorts are likely to impact the SoA dataset. Even once you have a stable SoA dataset, you may have to account for visit names in SV not matching those in PC. Patients may also take visit / time point combinations, which are not in the protocol. All of this can result in a "messy" visualisation that needs continual maintenance.

The visualisation showing missing / expected data is getting closer to answering the original question of "do we have all of the data?" but it is still not fully there. There are arguments for and against this amount of checking by functions downstream of the Data Management and Sample Management departments. However, the purpose of this paper is to provide a potential solution and not to discuss whether or not it should be needed.

DATA STRUCTURE FOR CREATING THE VISUALISATION

The basic visualisation shown in Figure 1 is created from a single SDTM dataset. This section will describe the steps taken in its creation. Firstly, process the SDTM dataset to create the following five columns:-

1) ID
2) REPVIS
3) VISITORD
4) RESULT
5) TEST

Here are recommendations for creating these five columns:-

ID

Take the numeric part of the patient identifier (e.g. from USUBJID). This column displays the patient number in the visualisation

REPVIS

Combine the scheduled visit (e.g. from VISIT) and scheduled time-point (e.g. from PCTPT) to form a single string. REPVIS displays the patient visit information in the visualisation.

VISITORD

Each REPVIS value must have a unique value of VISITORD. VISITNUM and xxTPTNUM (e.g. PCTPTNUM) can be combined to provide this unique ordering, or it can be derived in a user-defined manner. VISITORD is used is for ordering REPVIS only and is not displayed in the visualisation

RESULT

This column contains the information that defines the cell colour in the visualisation. Figure 10 shows a user-defined derivation where: non-missing instances of PCSTRESN are set to "R"; PCSTRESC values of "BLQ" are set to "B"; PCSTAT values of "NOT DONE" are set to "N" (not shown in Figure 10).

TEST

This column is used to split the data over multiple worksheets in Excel. Figure 10 shows a single test but this column could be used to split by lab parameter, treatment group, or anything else that makes sense.

	1	2	3	4	5
	ID	REPVIS	VISITORD	RESULT	TEST
1	10001	CYCLE 1 DAY 1 (PREDOSE)	2001	B	RO1234567
2	10001	CYCLE 1 DAY 1 (6 HOURS POST DOSE)	2002	R	RO1234567
3	10001	CYCLE 1 DAY 1 (END OF INFUSION)	2005	R	RO1234567
4	10001	CYCLE 1 DAY 2 (24 HOURS POST DOSE)	3003	R	RO1234567
5	10001	CYCLE 1 DAY 3 (48 HOURS POST DOSE)	4004	B	RO1234567
6	10001	CYCLE 1 DAY 8	5000	B	RO1234567
7	10001	CYCLE 1 DAY 15	7000	B	RO1234567
8	10001	CYCLE 2 DAY 1 (PREDOSE)	8001	B	RO1234567
9	10001	CYCLE 2 DAY 1 (END OF INFUSION)	8005	R	RO1234567
10	10001	CYCLE 2 DAY 2 (24 HOURS POST DOSE)	9003	B	RO1234567
11	10001	CYCLE 2 DAY 8	11000	B	RO1234567
12	10001	CYCLE 2 DAY 15	12000	B	RO1234567

Figure 10: Example of the dataset structure required to create the basic visualisation.

Programming the visualization can start now that the dataset has been created. The two sections below present two programming strategies applicable to create the pixel plot from Figure 1 in R and SAS.

HOW TO CREATE THE VISUALISATION IN R

The following shows a strategy for creating the visualisation in R and sample code is provided in Appendix I:-

1) **Transpose the Dataframe**

The dataframe created in the previous section contains ID, REPVIS, VISITORD, RESULT and TEST as columns. This dataframe needs to be transposed so that the column required to be displayed on the x-axis has its values as column names. The column required to be displayed on the y-axis remains as a column. Figure 11 shows an example of the transposed dataframe with ID as column names and RESULT populated as its value. REPVIS, VISITORD and TEST remain as columns.

	REPVIS	VISITORD	TEST	10001	10002	10003	21001	21002	21003	21004
1	CYCLE 1 DAY 1 (PREDOSE)	2001	RO1234567	B	B	B	B	B	B	B
2	CYCLE 1 DAY 1 (6 HOURS POST DOSE)	2002	RO1234567	R	R	R	R	R	R	R
3	CYCLE 1 DAY 1 (END OF INFUSION)	2005	RO1234567	R	R	R	R	R	R	R
4	CYCLE 1 DAY 2 (24 HOURS POST DOSE)	3003	RO1234567	R	R	R	R	R	R	R
5	CYCLE 1 DAY 3 (48 HOURS POST DOSE)	4004	RO1234567	B	R	R	R	R	R	R
6	CYCLE 1 DAY 8	5000	RO1234567	B	R	R	R	B	R	*NA*

Figure 11: Example of transposed R dataframe.

Duplicate data can cause an issue with the transpose. See the section *Handling Duplicate Data in R* for more details.

2) **Define the Output to Excel**

A number of different R packages exist for outputting a dataframe to Excel. The author has tried some of these packages and found openxlsx to be the most intuitive and easiest to find online help for, for this type of application. All of the Excel formatting required by the user must be defined within the code. Here are some of the areas that the user will need to define:-

- The colours are used to represent dataframe values
- The colour of the cells and which cells the colouring applies to
- The column widths and row heights
- Freezing the top row

3) **Create a Loop for Multiple Worksheets**

In order to display each unique TEST as a separate worksheet, the code from 2) is run for every value of TEST. The

list of unique values of TEST is stored in a vector from the reporting dataframe. Next, a for loop is used to run the code from 2), using the values in the vector to output the dataframe as separate worksheets.

4) **Create a Dataframe for the Visualisation Information**

It is possible to use openxlsx to show header information, or print text above the dataframe, to display information regarding the title, key and program location. A simpler way is to include this information into a separate user-defined dataframe and display it as the final worksheet (see Figure 12):

	A	B	C	D	E	F	G
1	Key.Values						
2	Study: WP12345						
3							
4	Key:-						
5	Green = Received						
6	Black = BLQ						
7	Purple = Not Done						
8							
9	This visualisation shows each sample per patient as a coloured cell (colour codes given above)						
10	Unscheduled visits show the number of unscheduled samples only.						
11							
12	Program location: \myprojects\WP12345\programs\l_pk_samples.r						

Figure 12: Example of a user-defined dataframe that contains information about visualisation, including key and program location.

One benefit to this approach is that a lot of information can be included, without detracting from the visualisation. The downside is that the key is not shown next to the visualisation.

Note: The author is not an expert in R programming, so the reader may be aware of other, more efficient ways to achieve the same outcome.

HOW TO CREATE THE VISUALISATION IN SAS

The following shows a strategy for creating the visualisation in SAS. Appendix II provides sample code:-

1) **Create a SAS FORMAT for displaying REVIS**

To allow the correct order of REPVIS in the PROC REPORT step, VISITORD is included in the COLUMN statement. To display the values of REPVIS, instead of the numeric VISITORD value, create a FORMAT, which contains the unique values of REVIS and VISITORD. Note: there must be a 1:1 mapping between REPVIS and VISITORD.

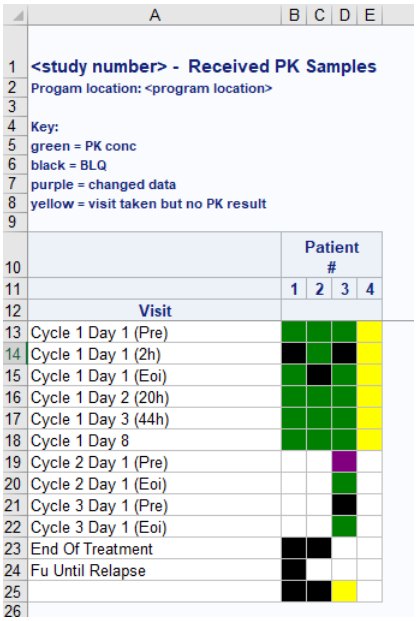
2) **Output to Excel using PROC REPORT and ODS**

Much of the work required to create the visualisation is accomplished using PROC REPORT and ODS. Here are some examples:-

- Output each unique TEST to a separate worksheet using ODS and a BY statement
- Data transposition using the ACROSS option of the DEFINE statement in PROC REPORT.
- Cell background and foreground colours are controlled by applying FORMATs to RESULT.
- Header rows are frozen using an ODS EXCEL statement.

3) **Information About The Visualisation**

As with the R strategy, it should be possible to create a dummy dataset to display this information but this option has not been pursued. A simple alternative is to use the TITLE statement to provide the visualisation title, key, program location and any other useful information. One drawback of this approach is that all of the information is placed above the visualisation and it is advised to keep the number of additional lines to a minimum. It is not recommended to use the FOOTNOTE statement as this will be displayed off the screen, when there is a lot of data. Figure 13 shows an example of how this could look, using the *Missing or Expected Data Visualisation* as an example:



	A	B	C	D	E
1	**<study number> - Received PK Samples**				
2	Progam location: <program location>				
3					
4	Key:				
5	green = PK conc				
6	black = BLQ				
7	purple = changed data				
8	yellow = visit taken but no PK result				
9					
10		Patient #			
11		1	2	3	4
12	Visit				
13	Cycle 1 Day 1 (Pre)				
14	Cycle 1 Day 1 (2h)				
15	Cycle 1 Day 1 (Eoi)				
16	Cycle 1 Day 2 (20h)				
17	Cycle 1 Day 3 (44h)				
18	Cycle 1 Day 8				
19	Cycle 2 Day 1 (Pre)				
20	Cycle 2 Day 1 (Eoi)				
21	Cycle 3 Day 1 (Pre)				
22	Cycle 3 Day 1 (Eoi)				
23	End Of Treatment				
24	Fu Until Relapse				
25					
26					

Figure 13: Example of using SAS TITLE statements to provide information about the visualisation.

CREATING THE VISUALISATON IN PYTHON

It is possible to use the R strategy to create the visualisation in python. The dataframe creation, transpose and output to Excel can all be achieved using the pandas library. As with R, a for loop can be used to output each unique TEST to a worksheet. Pandas will do most of the job but it doesn't appear to handle automatic column widths, or cell bordering. More investigation is required to produce a similar quality visualisation in python.

DUPLICATE DATA

When receiving clinical trial data it is possible to receive more than one result per unique scheduled visit and timepoint. It may be that there are duplicated results, or there are different results at exactly the same time-point. Below are some points for consideration concerning these types of data issues:

Handling Duplicate Data in R

Any duplicate record of REPVIS per ID must be removed before the dataframe transpose, otherwise the script will fail. The example code outputs these duplicate rows to an external file for the user to review. The resulting visualisation will not know which observation is correct and will pick one to report. This may not be an issue if both RESULT values are identical. If the RESULT values are not identical, the user should decide which observation should be kept.

Handling Duplicate Data in SAS

In SAS, the appearance of duplicate data does not cause the script to fall. A new line is created for each sample that is repeated by scheduled visit and time-point. Figure 14 shows an example of this for two results at the Cycle 1 Day 2 visit:

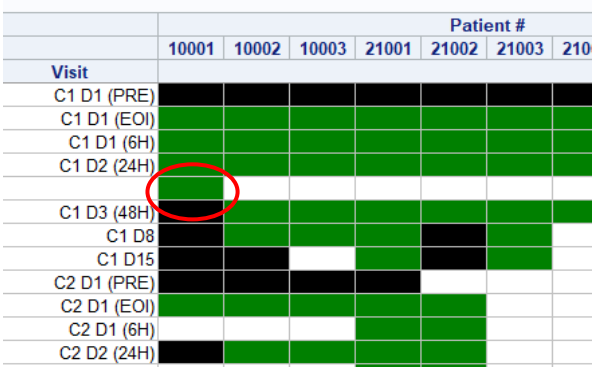


Figure 14: An example of how SAS displays duplicate data in the visualisation

UNSCHEDULED VISITS

Unscheduled visits do not work neatly with this type of visualisation, as:-

- A patient can have more than one unscheduled visit
- These visits usually have the same name
- They occur at different dates / times

It is possible to display each unscheduled visit on a new line, in a similar manner to the above illustration from *Handling Duplicate Data in SAS*. This may not make sense in the visualisation if you do not know when the unscheduled visit happened. One solution is to rename each unscheduled visit to include the actual visit day, or cycle number in the string e.g. Unscheduled (Cycle 1 Day 1). This ensures correct ordering of the unscheduled visit but can make the visualisation harder to read, as most patients will not have this visit. One solution, which is provided in the template scripts, is to total the number of unscheduled visits and display these at the end of the visualisation (see Figure 15):

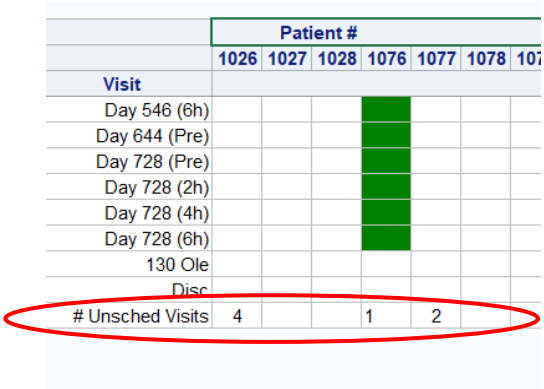


Figure 15: An example of totalling unscheduled visits in the visualisation.

Other types of non-scheduled visits also appear in clinical trials e.g. Follow-Up, Early Withdrawal, End of Treatment. These visits usually have unique names and so do not cause this issue. If there are examples of these that do not have unique names, the same strategy as for unscheduled visits can be applied.

CONCLUSIONS

Pixel plots are useful visualisations for showing how many results are present and for which patient visits. By using simple cell colouring, they can also show trends in the data, or data issues. These visualisations can be developed simply by using a single dataset, or to show messages that are more complex by combining more than one dataset. The more complex the visualisation, the more code maintenance will be required, so it is recommended to keep the visualisation as simple as possible. These visualisations can provide the Data Scientist with information that may not be easily found from other types of TLGs and are a good visual way of communicating the data with other functions.

REFERENCES

[1] Adapted from an article entitled “Create 2D Pixel Plot in Python” from: https://www.geeksforgeeks.org/

ACKNOWLEDGMENTS

I would like to thank my Clinical Pharmacology colleagues who provided me feedback on how to make these visualisations more user-friendly and provided the initial question that started this work.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Paul Grimsey
Roche Products Ltd
6 Falcon Way, Shire Park
Welwyn Garden City / AL7 1TW
+44 (0) 1707 36 5871
paul.grimsey@roche.com

APPENDIX I R SAMPLE CODE FOR THE BASIC VISUALISATION

R session info:
R version: 4.0.5 (2021-03-31)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 10 x64 (build 19044)

Package versions;
Tidyr v1.3.1
haven v2.4.3
reshape2 v1.4.4
stringr v1.4.0
openxlsx v4.2.4

```
################################################################################
# l pk samples.r
################################################################################
# Author  : P.Grimsey
# R Version: 4.0.5
################################################################################
# Purpose : Visualisation of received PK samples
#
# Inputs  : PC.sas7bdat
# Outputs : l_pk_samples_<date>.xlsx ; duplicated_data.csv
#
# Comments : This code accompanies Phuse Paper DV07 from 2022.
#            Please read the paper for more details on this visualisation.
################################################################################

#rm(list = ls())


# !!!!! USER DEFINED !!!!! #
studyno <- "<!!! your study number !!!>"
progloc <- "<!!! your program location !!!>"
progname <- "l_pk_samples"
id.column.width <- 8 # length of columns in final output

# Add libraries
library(tidyverse)
library(haven)
library(stringr)
library(openxlsx)


#-------------------------------------------------------------------------------#
# !!!!! User defined Objects / Dataframes for output file - update as required !!!!! #
#-------------------------------------------------------------------------------#

# Dataframe that contains all of the information on cell colouring used in the visualisation
Key.Values <- c(paste("Study: ", studyno),
                "",
                "Key:-",
                "Green = Received",
                "Black = BLQ",
                "Purple = Not Done",
                "",
                "This visualisation shows each sample per patient as a coloured cell (colour codes given
above)",
                "Unscheduled visits show the number of unscheduled samples only.",
                "",
                paste("Program location: ", progloc, "\\", progname, ".r"))
Key.Values <- data.frame(Key.Values)

# User-defined styles for formatting colours in Excel
Received <- createStyle(fontColour = "#00BA38", bgFill = "#00BA38")
BLQ      <- createStyle(fontColour = "#404040", bgFill = "#404040")
Not.Done <- createStyle(fontColour = "#CC0066", bgFill = "#CC0066")
Negative <- createStyle(fontColour = "#404040", bgFill = "#404040")
borders  <- createStyle(borderStyle  = openxlsx_getOp("borderStyle", "thin"),
                        border       = "TopBottomLeftRight")


#-----------------#
# Data Processing #
```

```
#-----------------#

pc <- read_sas("pc.sas7bdat") %>%
  select(STUDYID, USUBJID, PCTEST, PCTESTCD, PCSTRESC, PCSTRESN,
         PCSTRESU, VISIT, PCTPT, PCSTAT, PCREFID, PCSCAT,
         VISITNUM, PCTPTNUM) %>%
  mutate(ID = as.numeric(substring(USUBJID, 9))) %>%
  mutate(TEST = PCTEST)

# get scheduled visits and process
pc.sched <- pc %>%
            filter(!grepl("UNSCHED", VISIT, ignore.case = TRUE)) %>%
               mutate(VALUE = case_when(
                 !is.na(PCSTRESN)       ~ 'R',
                 PCSTRESC == "BLQ"      ~ 'B',
                 PCSTAT   == "NOT DONE" ~ 'N',
               ))  %>%
               mutate(VISITORD = case_when(
                  !is.na(PCTPTNUM) ~ VISITNUM * 1000 + PCTPTNUM,
                  is.na(PCTPTNUM)  ~ VISITNUM * 1000
               )) %>%
               mutate(REPVIS = case_when(
                 PCTPT == "" ~ VISIT,
                 PCTPT != "" ~ paste(VISIT, " (", PCTPT, ")")
               ))


# get unscheduled visits
pc.unsched <- pc %>%
  filter(grepl("UNSCHED", VISIT, ignore.case = TRUE)) %>%
    mutate(VISITORD = 9999999) %>%
    mutate(REPVIS = "# Unsched Visits") %>%
    select(ID, TEST, REPVIS, VISITORD)


# count the number of rows per patient for unscheduled visits
unsched.summ <- pc.unsched %>%
                  count(ID, TEST, REPVIS, VISITORD, name = "COUNT")


# join the scheduled and unscheduled visit data
all.data <- bind_rows(pc.sched, unsched.summ) %>%
             mutate(RESULT = case_when(
                      VALUE != "" ~ VALUE,
                      !is.na(COUNT)  ~ as.character(COUNT)
             ))

# Keep only columns of interest
repdf <- all.data %>% select(REPVIS, VISITORD, RESULT, ID, TEST)

# find any duplicated rows duplicated and export to a csv
duplicated.rows <-  repdf[duplicated(repdf) | duplicated(repdf, fromLast=TRUE), ]
write.csv2(duplicated.rows, file = ".\\duplicated_data.csv")

# keep only unique rows and transpose data for reporting
repdft <- repdf %>% distinct(REPVIS, VISITORD, RESULT, ID, TEST) %>%
             spread(key = ID, value = RESULT) %>%
             arrange(VISITORD)

# remove VISITORD
final <- repdft %>% select(-VISITORD)

# create vector for unique TESTs
uniq.tests <- unique(final[c("TEST")])
uniq.tests.v <- uniq.tests$TEST

# get the number of unique tests #
uniq.tests.n <- length(uniq.tests.v)

# get unique VISIT / order combinations and print to console
uniqord <- as.data.frame(unique(repdf[c("VISITORD", "REPVIS")])) %>%
             arrange(-VISITORD)
print("Unique Visit and Order - Please review to check the ordering is correct!", quote = FALSE)
print(uniqord)


#-------------------------------#
# Create Visualisation in Excel #
#-------------------------------#
```

```
wb <- createWorkbook() # create a workbook

  for(ronum in uniq.tests.v)
  {
addWorksheet(wb, ronum, gridLines = TRUE) #add a worksheet to the workbook
writeData(wb, ronum,  assign(ronum,final %>% filter(TEST==ronum))) # write dataframe(s) into the worksheet of
the workbook

# apply the formatting for the cell colour
conditionalFormatting(wb, ronum, cols = 2:ncol(final),
                      rows = 2:nrow(final), rule = "R", style = Received,
                      type = "contains")

conditionalFormatting(wb, ronum, cols = 2:ncol(final),
                      rows = 2:nrow(final), rule = "B", style = BLQ,
                      type = "contains")

conditionalFormatting(wb, ronum, cols = 2:ncol(final),
                      rows = 2:nrow(final), rule = "N", style = Not.Done,
                      type = "contains")

# apply the borders
addStyle(wb, ronum, style = borders, cols = 1:ncol(final), rows = 1:nrow(final)+1, gridExpand = TRUE)


# relabel Cell A1
writeData(wb,
          sheet = ronum,
          x = "Visit / Patient No.",
          xy = c(1,1)
)

# set width of 1st column to be automatic
setColWidths(
  wb,
  ronum,
  # cols = 1:ncol(final),
  cols = 1,
  widths = "auto"
)

# set width of ID columns to be user defined
setColWidths(
  wb,
  ronum,
  cols = 2:ncol(final),
  widths = id.column.width
)

# freeze first row
freezePane(
  wb,
  ronum,
  firstRow = TRUE,
)

# Hide 2nd column
setColWidths(wb, sheet = ronum, cols = 2, hidden = rep(TRUE, length(cols)))

  }

# Worksheet for Key
addWorksheet(wb, "Key", gridLines = TRUE)
writeData(wb, "Key", Key.Values, startRow = 1, startCol = 1)

# output final Excel visualisation
saveWorkbook(wb, str_glue(studyno, "_", progname, ".xlsx"), overwrite = TRUE)



########## END OF PROGRAM ##########
```

APPENDIX II SAS SAMPLE CODE FOR THE BASIC VISUALISATION

```
/*************************************************************************
* l_pk_samples.sas
*************************************************************************
* Author : P.Grimsey
* SAS version : 9.4
*************************************************************************
* Purpose : Visualisation of received PK samples
*
* Inputs  : PC.sas7bdat
* Outputs : l_pk_samples_<date>.xlsx
*
* Called macros : fmtmerge
*                 (for the fmtmerge code, see Appendix I in CT01 paper (Phuse 2017)):
* https://phuse.s3.eu-central-1.amazonaws.com/Archive/2017/Connect/EU/Edinburgh/PAP_CT01.pdf)
*
* Comments : This code accompanies Phuse Paper DV07 from 2022.
*            Please read the paper for more details on this visualisation.
*************************************************************************/


*** !!!!! USER DEFINED - please change for your study !!!!! ***;
%let progname = l_pk_samples;
%let study    = <!!! your study number !!!> ;
%let progloc  = <!!! your program location !!!>;
%let output   = <!!! your output location !!!>;
%let sdtmloc  = <!!! your SDTM location !!!>;
%let title1   = title1 j = left              "&study -  Received PK Samples";
%let title2   = title2 j = left height = 8pt "Progam location: &progloc./&progname..sas";
%let title3   = title3 j = left height = 8pt " ";
%let title4   = title4 j = left height = 8pt "Key:  ";
%let title5   = title5 j = left height = 8pt "green = PK conc";
%let title6   = title6 j = left height = 8pt "black = BLQ";
%let title7   = title7 j = left height = 8pt "purple = Not done";

%include "<!!!your SAS macro location!!!>/fmtmerge.sas";  /* !!! see program header for FMTMERGE code !!! */


*------------------------*;
* Set up the environment *;
*------------------------*;

options mprint mlogic validvarname = upcase;

*** libnames ***;
libname sdtm   "&sdtmloc." access = readonly;
libname output "&output.";

*** program formats ***;
proc format;
  value $rescol
    "R" = "green"
    "B" = "black"
    "N" = "purple"
    ;
  value $res1c
    "Received" = "R"
    "BLQ"      = "B"
    "Not done" = "N"
    ;
run;

*----*;
* PK *;
```

```
*----*;

*** process the raw data and output unscheduled visits separately ***;
data _pc
     _unsched;
  set sdtm.pc (keep = studyid usubjid pctest pctestcd pcstresc pcstresn pcstresu visit
                      pctpt pcstat pcrefid pcscat visitnum pctptnum pcdtc);

  id = input(scan(usubjid, -1, "-"), best.);

  * subset unscheduled visits for counting *;
  if upcase(visit) in: ("UNSCHED") then output _unsched;

  * make a reporting visit + ordering variable *;
  else do;

    * visit ordering *;
    if not missing(pctptnum) then visitord = (visitnum * 1000) + pctptnum;
                             else visitord = (visitnum * 1000);

    * visit label *;
    if not missing(pctpt ) then
         repvis = strip(propcase(visit)) || " (" || strip(propcase(pctpt )) || ")";
    else repvis = strip(propcase(visit));

    * value *;
    attrib value length = $10.;
         if pcstresn ne .        then value = "Received";
    else if pcstresc = "BLQ"      then value = "BLQ";
    else if pcstat  = "NOT DONE" then value = "Not done";

    output _pc;

  end;

run;

*** count unscheduled visits ***;
proc sql;
  create table unsched as
  select id, pctest, visit, count(*) as count
  from _unsched
  group by id, pctest, visit
  order by id
  ;
quit;

*** reporting dataset ***;
data pc;
  set _pc
      unsched (in = unsched);

  attrib result length=$1;
  result  = put(value, $res1c.);

  if unsched then do;
    repvis = "# Unsched Visits";
    visitord = 999999999;               /* !!! check this is appropriate for your study !!! */
    result = strip(put(count, best.));
  end;

run;

proc sort data = pc;
  by pctest visitord id;
```

```
run;

*** Unique visit ordering - used for debugging ***;
title "unique visit ordering";
proc sql;
  select distinct visit, pctpt, repvis, visitord
  from pc
  order by visitord
  ;
quit;

*** make a format for visit ***;
proc sql;
  create table repvis as
  select distinct visitord, repvis as repvis
  from pc
  order by visitord, repvis
  ;
quit;
%fmtmerge(fname = repvis, fstart = visitord, ftype = n);


*----------------------------*;
* Create the report in Excel *;
*----------------------------*;

ods excel file = "&output.\&progname._&sysdate..xlsx"
   options(embedded_titles = 'yes'
                 tab_color = 'purple'
            frozen_headers = 'yes'
            sheet_interval = 'bygroup'
                sheet_name = "key");

options nobyline;

ods excel options(sheet_name="#byval1");

&title1;
&title2;
&title3;
&title4;
&title5;
&title6;
&title7;

proc report data = pc
 style(summary)=Header;
 by pctest;
  columns visitord id,result dummyvar;
  define visitord / group  order = data     "Visit" left format = repvis.;
  define id       / across order = internal "Patient #";
  define result   / display ' ' style(column) = {background = $rescol.
                                                 foreground = $rescol.};

  define dummyvar / computed noprint;
    compute dummyvar;
    dummyvar = 1;
  endcomp;

run;

ods excel close;


********** END OF PROGRAM **********;
```