The following example illustrates the execution of a SET statement within a DATA step resulting in the changes to the structure of the union of two datasets with common/identical variables in both.

```
DATA SET blue yellow; RUN;
```

Here BLUE and YELLOW have identical variables and hence the resultant GREEN has similar number of variables and all observations of BLUE and YELLOW.

However, any difference in the number of variables across the two datasets can result in the following shapes. The variables not present in either of the datasets (BLUE or YELLOW) will result in missing values within the resultant GREEN.

The example code of 1 and 2 is as follows:

```
DATA 1 and 2.
```

- **SAS PROCEDURES**

**PROC TRANSPOSE**

Next moving on to SAS® procedures, a commonly used procedure for data transformation is the TRANSPOSE procedure (PROC TRANSPOSE). You can see how the data gets transformed using the example code.

```
PROC transpose data= in. out= if; Run;
```

The ID statement within the TRANSPOSE procedure transforms the values of TRY observations in dataset V into two distinct variables in dataset Y. The BY statement retains the order of the VISIT variable and the VAR statement assigns the value of COUNT against the corresponding VISIT observations across the two TRY variables.

```
PROC FORMAT:
```

Another example of SORT procedure (PROC SORT) has been explained in the paper associated with this poster – PP22: Understanding SAS®, the SAS® way!

All of us would be able to identify the following pictures and would have certainly associated these when we all started learning the, our first step to learning the English language. A picture is worth thousand words or SQL, the key

<table>
<thead>
<tr>
<th>Raw Data (A)</th>
<th>Analytic/Derived Data (B)</th>
<th>Outputs (C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>28</td>
<td>29</td>
<td>30</td>
</tr>
</tbody>
</table>

We could only wish that we had a magic wand and say the famous words 'Abracadabra' and hey presto!! we have raw data (A) transformed to analytic/derived data (B) and repeat the same to get all sizes and fancy output (C) of the end... job done with a breeze!

However, the reality is that we have to use SAS. That is the version of a magic wand in reality but not without using our brains. Right, so where do we begin then, with the first (DATA) step.

```
DATA STEPS:... What is really a DATA step?
```

I relate this to fixing any given dataset using various tools and techniques either using various statements, functions or arrays of (x Read and modify) and changing either in terms of the number of variables or number of observations.

The following examples illustrate the execution of a MERGE statement within a DATA step resulting in the changes to the structure of the merging datasets.

```
DATA green; merge blue yellow; by vars...
```

Here BLUE and YELLOW have the same number of observations and hence the resultant GREEN has the same number of observations and variables of BLUE and YELLOW.

```
DATA green: merge blue (in= in) yellow (in= in) by vars; Run;
```

Here the number of observations in BLUE is less than the number of observations in YELLOW. However as the IN operator is used conditionally against YELLOW, GREEN has the same number of observations as BLUE and variables of BLUE and YELLOW.

```
DATA green: merge blue (in= in) yellow (in= in) by vars; Run;
```

Here the number of observations in BLUE is less than the number of observations in YELLOW. However as the IN operator is used conditionally against YELLOW, GREEN has the same number of observations as YELLOW and variables of BLUE and YELLOW.

```
DATA green; set blue yellow; Run;
```

Here BLUE and YELLOW have identical variables and hence the resultant GREEN has similar number of variables and all observations of BLUE and YELLOW.

```
DATA green; set blue yellow; Run;
```

Here BLUe and YELLOW have identical variables and hence the resultant GREEN has similar number of variables and all observations of BLUE and YELLOW.

```
DATA green; set blue yellow; Run;
```

Here BLUe and YELLOW have identical variables and hence the resultant GREEN has similar number of variables and all observations of BLUE and YELLOW.

```
DATA green; set blue yellow; Run;
```

Here BLUe and YELLOW have identical variables and hence the resultant GREEN has similar number of variables and all observations of BLUE and YELLOW.

The example code of 1 and 2 is as follows:

```
DATA 1 and 2.
```

- **DO LOOPS**

Given that we touched base on ARRAYS earlier, it cannot function without the use of DO LOOPS. One of the most under-rated and underestimated are DO UNTIL and DO WHILE. So what are the functionalities of both and the key difference between the two?

What action will be performed repeatedly if I say DO WHILE you hear the music on the radio and do UNTIL the radio is switched off? Yes, dancing on the floor which becomes an iterative step of the DO LOOP.

In simple words, for the DO WHILE loop, dancing on the floor will only happen if the condition (of the music on the radio) is true. So, do we move on, or does the floor continue to happen until the condition (of the music on radio) is not true? Yes, until the radio is switched off. Essentially...

```
Do While The Condition = True
```

- **SAS® FUNCTIONS**

Now let us talk about a bit about functions. The most common ones that are used in any data manipulation procedures are the PUT (numeric to character conversion) and INPUT (character to numeric).

Now you might be aware that using the first function one can store a numeric value as character within a SAS® dataset. Say if we have a dataset with just two variables x and y, how can one identify which of these variables is numeric or character without even looking at the variable properties. The hint is the alignment of the values. Any SAS® dataset has only two possibilities of storing any value i.e as numeric or character. Any numeric variable will have its value right aligned and any character variable will have its value left aligned within a SAS® dataset. Hence, generally I have been able to associate the PUT and INPUT functions with the “shift” in the alignment of values when the data conversion happens, a right directed character to numeric function by default (numeric to character) for INPUT and PUT function respectively. I could clearly remember this pair of functions and their functionalities by relating to the following road hazard sign a.k.a, the numeric to character to numeric conversion sign.

```
y = PUT(x, best2); x = INPUT(y, best2);
```

Now can anybody guess which SAS® functions can mend the broken heart?

```
Oh God!! When COMPRESS is used
```

```
When COMPRESS is used
```

Looking at the second picture we are almost but not quite there, the COMPRESS function helps in compressing between two character variables but loses all spaces existing between any two character values.

```
ONE THING THAT CAN'T DO WITHOUT...
```

We have discussed several functionalities of SAS®. Now let me test your knowledge with just one question. What is the lone that SAS® can’t do without? You would have guessed it from the “rinning” image. Any DATA STEP or a SAS® procedure cannot execute without a “RUN” statement. It would be like using a gun without a bullet. You cannot hit bulls-eye!