Phuse 2021 - Replicating R figures into SAS

Antonio Rodríguez Contestí

March 2021

1 Abstract

One key feature of a Statistical Software are the graphics capability since they allow to explore and understand the data. When reading throw comparison between R and SAS, a common comment is "base on Graphics R is the winner"[9, 3, 6] so to proof if that was right, I decided to replicate some of complicated plots in R and check whether they were doable or not into SAS. All graphs where doable into SAS, but in some cases, due to the lack of flexibility of some of the tools some alternative approaches where needed and we are going to share the code of them. In terms of capability, both are equally powerful, but R seems to do it better in terms of flexibility for the user.

2 Introduction

I was navigating into LinkedIn and I saw a really interesting post about Rain-Cloud plots posted by Adrian Olszewski, that always share really interesting STATs posts. As usual, the output was produced in R, and all of those comments of SAS vs R comparisons "base on Graphics R is the winner" [9, 3, 6] came to my mind, so I contacted Adrian, and very kindly, he share about 20 kind of plots (all R plots are provided by him), and also share different libraries in R that could be of interest. I started the task to try to replicate them into SAS. Some of them where trivial but some of them, without thinking a bit outside the box where not doable. Here are a selection of some of those plots.

3 Raincloud plot

The Rain Cloud plot it is a mix of Box-plot (half), density and scattered data for continues data and for discrete data dot plot or finger plots. An example on continues data can be seen on Figure 1.

A Box-plot alone hides multiple modes, fixed with a density plot. Also box-plot can be degenerated into a line when the median equals the quantiles.

On first glance, it looks as an easy figure, but there are no options to display only half a box-plot on SAS, neither to select where do we want to display the density plot when more than one group present.

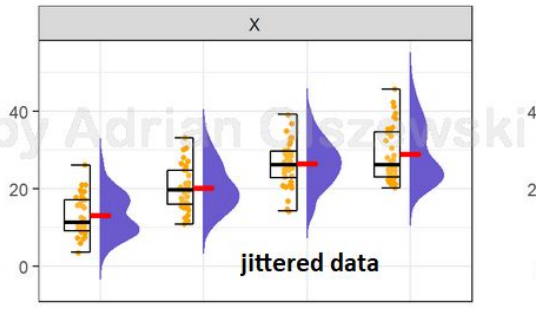


Figure 1: RainCloud produced using R

Let's produce this figure step by step.

First, we are going to produce the scatter plot with jittering. Due to the needs for the others figures, we are going to use a numerical x-axis, so we are going to add a numerical version of species.

```
PROC FORMAT;
   INVALUE Species
     'Setosa' = 1
     'Versicolor' = 2
     'Virginica' = 3;
RUN;

DATA Iris;
  SET SASHELP.Iris;
  Species_n = INPUT(Species,Species.);
RUN;
```

To create the jittered data plot we use the following code to obtain Figure 2.

```
PROC TEMPLATE;
  DEFINE STATGRAPH RainCloud;
    BEGINGRAPH;
      LAYOUT OVERLAY;
        SCATTERPLOT X = Species_n Y = SepalLength / JITTER = AUTO
                    JITTEROPTS = (AXIS = X WIDTH = 0.25 );
   ENDLAYOUT;
   ENDGRAPH;
 END;
RUN;

PROC SGRENDER DATA = Iris TEMPLATE = RainCloud;
RUN;
```

Once we have this, we can proceed to the next step, the box-plot. By default, there is no option to place half a box-plot or move the whiskers to sides, that would be equivalent, so we are going to be in need to create our own box-plot from scratch.

We are going to base our construction into [4], simply modifying to adapt to

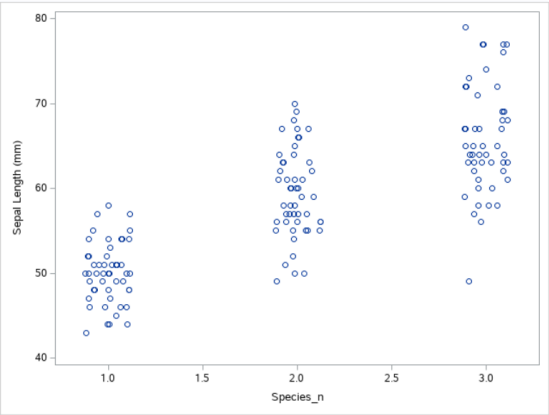


Figure 2: Scatter plot with Jitter

GTL language instead of SGPLOT and move around the whiskers, to get this code.

```
%MACRO DrawBoxPlots;
  /* draw median line */
  VECTORPLOT X = XBoxRight Y = median XORIGIN = XBoxLeft
                    YORIGIN = median / ARROWHEADS = FALSE;
  /* draw quantile box */
  VECTORPLOT X = XBoxRight Y = q1 XORIGIN = XBoxLeft
                           YORIGIN = q1 / ARROWHEADS = FALSE;
  VECTORPLOT X = XBoxRight Y = q3 XORIGIN = XBoxLeft
                           YORIGIN = q3 / ARROWHEADS = FALSE;
  VECTORPLOT X = XBoxLeft Y = q3 XORIGIN = XBoxLeft
                           YORIGIN = q1 / ARROWHEADS = FALSE;
  VECTORPLOT X = XBoxRight Y = q3 XORIGIN = XBoxRight
                           YORIGIN = q1 / ARROWHEADS = FALSE;
  /* draw whiskers */
  VECTORPLOT X = XWhiskerRight Y = YWhiskerTop
             XORIGIN = XWhiskerRight YORIGIN = q3 /
                                          ARROWHEADS = FALSE;
  VECTORPLOT X = XWhiskerRight Y = YWhiskerTop
             XORIGIN = XWhiskerLeft YORIGIN = YWhiskerTop /
                                          ARROWHEADS = FALSE;

  VECTORPLOT X = XWhiskerRight Y = YWhiskerBottom
             XORIGIN = XWhiskerRight YORIGIN = q1 /
                                          ARROWHEADS = FALSE;
  VECTORPLOT X = XWhiskerRight Y = YWhiskerBottom
             XORIGIN = XWhiskerLeft YORIGIN = YWhiskerBottom /
                                          ARROWHEADS = FALSE;
%MEND;
```

To prepare the data for this graph we are going to use a PROC MEANS and set all needed parameters

```
PROC MEANS DATA =  Iris NOPRINT;
 BY Species_n Species;
 VAR SepalLength;
 OUTPUT OUT = BoxPlotStats  MEAN = mean  MEDIAN = median
        Q1 = q1  Q3 = q3 QRANGE = qrange  MIN = min MAX = max;
RUN;

DATA GraphData;
  SET Iris
      BoxPlotStats(IN = box);
  IF box THEN DO;
      XBoxLeft              =  Species_n  -  0.2;
      XWhiskerLeft          =  Species_n  -  0.05;

      /*Right side of the boxplot will be slightly left too */
      XBoxRight             =  Species_n  - 0.01;
      XWhiskerRight         =  Species_n  - 0.01;
      YWhiskerTop           =  (1.5*qrange)  +  q3;
      YWhiskerBottom        =  q1  - (1.5*qrange);
      IF max  LE  YWhiskerTop    THEN YWhiskerTop     =  max;
      IF min  GE  YWhiskerBottom THEN YWhiskerBottom  =  min;
  END;
RUN;
```

Since we need to handmade the box, that's why we set up the axis as a numerical. With this done, we can produce the two sets of data overlapped using following template and obtaining figure 3.

```
PROC TEMPLATE;
  DEFINE STATGRAPH RainCloud;
    BEGINGRAPH;
      LAYOUT OVERLAY;
        SCATTERPLOT X = Species_n Y = SepalLength / JITTER = AUTO
                    JITTEROPTS = (AXIS = X WIDTH = 0.25 );
        %DrawBoxPlots;
   ENDLAYOUT;
   ENDGRAPH;
 END;
RUN;

PROC SGRENDER DATA = GraphData TEMPLATE = RainCloud;
RUN;
```

We can improve the figure 3 simply adding a bit of offset into the scatter plot X coordinate to avoid the overlapping.

Now only the Violin or density plot is left. The problem is that SAS despite having Density statement into GTL, it do not enable to offset the origin, so we can not place them in 1, for example. To fix that, we are going to create our own density function, as we did with the half box plot, using HIGHLOW statement.

To do so, we are going to use the PROC KDE[5]

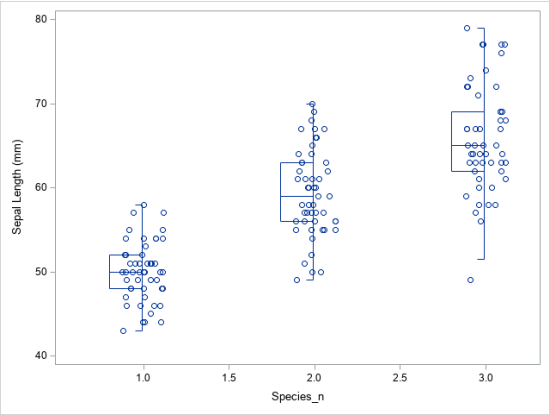


Figure 3: Scatter plot with Jitter and half box plot

```
/* Density */
PROC KDE DATA = Iris;
BY Species_n;
UNIVAR SepalLength / NGRID = 1000 UNISTATS PERCENTILES PLOTS = NONE
          OUT = Density (RENAME = (value = SepalLength) DROP = var);
RUN;
```

This is going to give us values between 0 and the point with maximum concentration for each Species. Since we want them to be visible in our scale, we can convert them into scale 0-0.25 (we can not go to much further, to avoid overlap between groups). To do that, we will calculate the maximum of this density and apply the conversion, later on we will set the origin to the corresponding X coordinate.

```
PROC SQL NOPRINT;
  SELECT MAX(density) INTO :max_dens
  FROM Density;
QUIT;

DATA Density_scaled;
  SET Density;
  Origin = Species_n;
  Density = Species_n + Density * 0.25/&max_dens.;
RUN;
```

We can join all the data, and apply the Offset on the Scatter, creating a new variable to contain the X axis information less certain amount with the following code

```
DATA GraphData;
  SET Iris (IN = iris)
      BoxPlotStats(IN = box)
      Density_scaled;
  IF iris THEN DO;
    Scatter_X = Species_n - 0.375;
  END;
  IF box THEN DO;
      XBoxLeft              =  Species_n  -  0.2;
      XWhiskerLeft          =  Species_n  -  0.05;

      /*Right side of the boxplot will be slightly left too */
      XBoxRight             =  Species_n  - 0.01;
      XWhiskerRight         =  Species_n  - 0.01;
      YWhiskerTop           =  (1.5*qrange)  +  q3;
      YWhiskerBottom        =  q1  - (1.5*qrange);
      IF max  LE  YWhiskerTop    THEN YWhiskerTop     =  max;
      IF min  GE  YWhiskerBottom THEN YWhiskerBottom  =  min;
  END;
RUN;
```

Once we have the data, we can easily modify the shell to use the new var on X-axis for the scatter and add the density using following code, obtaining the figure 4.

```
PROC TEMPLATE;
  DEFINE STATGRAPH RainCloud;
    BEGINGRAPH;
      LAYOUT OVERLAY;
        SCATTERPLOT X = Scatter_x Y = SepalLength / JITTER = AUTO
                    JITTEROPTS = (AXIS = X WIDTH = 0.25 );
        %DrawBoxPlots;
        HIGHLOWPLOT Y = SepalLength HIGH = Density LOW = Origin /
                    DISPLAY = (FILL) TYPE = BAR BARWIDTH = 1
                    INTERVALBARWIDTH = 1 DATATRANSPARENCY = 0.5;
   ENDLAYOUT;
   ENDGRAPH;
 END;
RUN;

PROC SGRENDER DATA = GraphData TEMPLATE = RainCloud;
RUN;
```

Now, only minor tweaks are needed. We need to include the means and the proper axis. To do so it's enough to add a couple of SCATTERPLOT and play a bit with the offsets min and max to get everything where we want. Obtaining finally our figure 5.

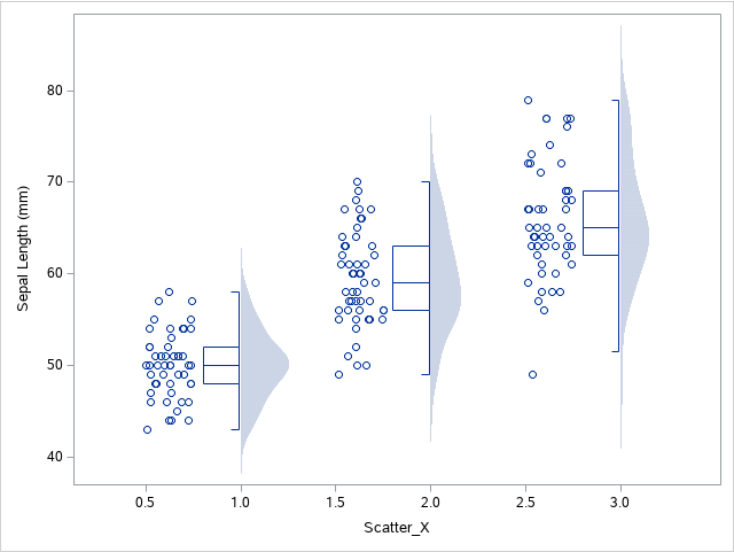


Figure 4: Scatter plot with Jitter, half box plot and density

```
PROC TEMPLATE;
  DEFINE STATGRAPH RainCloud;
    BEGINGRAPH;
      LAYOUT OVERLAY / XAXISOPTS = (DISPLAY = (LINE)
                       OFFSETMIN = 0.05 OFFSETMAX = 0.05)
                  X2AXISOPTS = (OFFSETMIN = 0.22 OFFSETMAX = 0.1);
        SCATTERPLOT X = Scatter_x Y = SepalLength / JITTER = AUTO
                       JITTEROPTS = (AXIS = X WIDTH = 0.25 );
        %DrawBoxPlots;
        HIGHLOWPLOT Y = SepalLength HIGH = Density LOW = Origin /
                    DISPLAY = (FILL) TYPE = BAR BARWIDTH = 1
                    INTERVALBARWIDTH = 1 DATATRANSPARENCY = 0.5;
        SCATTERPLOT X = Species  Y = Mean / XAXIS = X2
                  MARKERATTRS = (SIZE = 0 ) DISCRETEOFFSET = -0.5;
        SCATTERPLOT X = Species  Y = Mean / XAXIS = X2
                   MARKERATTRS = (SIZE = 0) DISCRETEOFFSET = 0.25;
        SCATTERPLOT X = Species  Y = Mean / XAXIS = X2
                              MARKERATTRS = (COLOR = RED SIZE = 5);
    ENDLAYOUT;
    ENDGRAPH;
  END;
RUN;

PROC SGRENDER DATA = GraphData TEMPLATE = RainCloud;
RUN;
```

Alternatively, to not need to play with the offset, we can create a format and display only the marks that are needed, in our case 1, 2 and 3 that should be equivalent to each species, using something as the following code

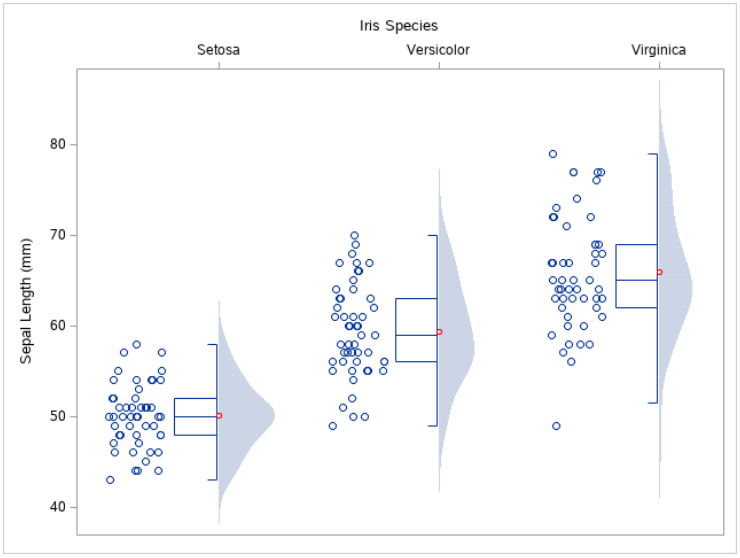


Figure 5: Final Rain Cloud in SAS

```
LAYOUT  OVERLAY / XAXISOPTS = (LINEAROPTS = (
                                  TICKVALUEFORMAT = Species_n.
                                  TICKVALUELIST = (1 2 3)
                               )
                    OFFSETMIN = 0.05 OFFSETMAX = 0.05)
```

4 Pie chart embeded into other plot

The figure that we are going to replicate on this section can be seen on figure 6. It is a 3 section plot containing survival data. First cell containing an histogram alike data, a second one containing box plot with fringe to indicate when each event happens and last but not least a series of scatters and lines with a pie chart.

We are going to generate the whole plot, but the reason why I bring this figure, is due to the pie chart. SAS has some functionalities, specially the LAYOUT REGION, that allow for those kind of outputs, but they can only be standalone, and without any type of text (other than using annotation functionality). Due to that reason, we are going to create our own pie chart, in the same way that we created our own half histogram.

To work on this data, we are going to use the SASHELP.BMT (subset to first 49 records), that has some time to event data with censor value. Let's first create the pie chart and work from there. To create the Pie, we are going to run some PROC FREQ to get the frequency of each event.

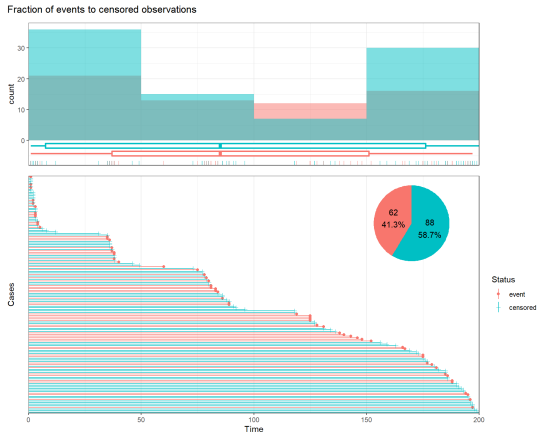


Figure 6: Figure containing a Pie chart produced on R

```sas
DATA Survival;
  SET SASHELP.Bmt;
  IF _N_ < 50;
RUN;

PROC FREQ DATA = Survival;
  TABLE Status / OUT = Freqout OUTCUM;
RUN;
```

Now that we have the data, we can use polar coordinates to transform the percentage to a given point into a circumference. We are going to transform each percentage to radiant, starting from $\frac{\pi}{2}$ (Top part of the circumference) and rotate anti-clock wise.

```
DATA Pie;
  SET Freqout;

  /* Origin of arc calculation in percentage*/
  RETAIN Start;
  IF MISSING(Start) THEN Start = 0;
  ELSE Start = Cum_PCT - Percent;

  /* End origin arc in percentage*/
  End      = Cum_PCT;

  /* Circumference parameters */
  Center_X = 2000;
  Center_Y = 2000;
  Radius   = 250;

  /* Label to display */
  Label    = PUT(Percent,4.1)||"%";

  /* To create a smooth circumference for each 0.01% create a dot */
  DO i = ROUND(Start,0.01) TO ROUND(End,0.01) BY 0.01;
    X_Polar = Center_X +
        Radius * COS(CONSTANT("PI")/2-i*2*(CONSTANT("PI"))/100);
    Y_Polar = Center_Y +
        Radius * SIN(CONSTANT("PI")/2-i*2*(CONSTANT("PI"))/100);
    OUTPUT;
  END;
  /* To close the circumference, output the center of the figure */
  X_Polar = Center_X;
  Y_Polar = Center_Y;
  OUTPUT;
RUN;
```

Once we have the data ready, we can produce figure 7 using the following code

```
PROC TEMPLATE;
  DEFINE STATGRAPH PieChart;
    BEGINGRAPH ;
      LAYOUT OVERLAY ;
        POLYGONPLOT X = X_Polar Y = Y_Polar ID = Status /
                    DISPLAY = (FILL) GROUP = Status LABEL = Label;
      ENDLAYOUT;
    ENDGRAPH;
  END;
RUN;

/* Generate the graph */
PROC SGRENDER DATA = Pie TEMPLATE = PieChart;
RUN;
```

Now that the pie chart is up and running, we can display the values of the subjects beneath. To avoid have issues with the range of the figures (lines and dots vs the pie chart), we are going to use main axis X and Y for displaying lines and dots and X2 and Y2 for the Pie chart, this way we can avoid any deformation of the Pie due to the range displayed.

First, we need to join the data, and we also need to have some variable indicating the origin time for all lines, in our case Origin = 1. Also since there is no USUBJID variable, we are going to use _N_ to identify different subjects.

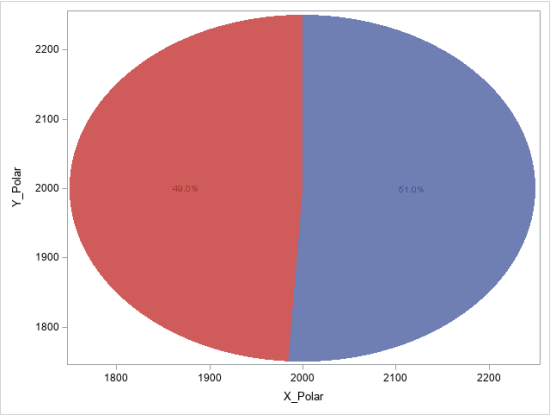


Figure 7: Pie chart section only

To do so we can use the following code.

```
PROC SORT DATA = Survival;
  BY T;
RUN;

DATA All;
  SET Survival (IN = Obs)
      Pie;
  IF Obs THEN DO;
    /* Origin of survival time */
    Origin = 1;
    SubjectID = _N_;
  END;
RUN;
```

Once all is in place, we can use the following template to produce the lowest of the 3 plots forming the figure, as can be seen on figure 8.

```
PROC TEMPLATE;
  DEFINE STATGRAPH PieChart;
    BEGINGRAPH ;
      LAYOUT OVERLAY / Y2AXISOPTS = (DISPLAY = (LINE)
                        LINEAROPTS=(VIEWMIN = 500 VIEWMAX = 2500))
                       X2AXISOPTS = (DISPLAY = (LINE)
                         LINEAROPTS=(VIEWMIN = 0 VIEWMAX = 2500))
                       YAXISOPTS = (DISPLAY = (LINE LABEL)
                                    LABEL = "Cases" REVERSE = TRUE);
        POLYGONPLOT X = X_Polar Y = Y_Polar ID = Status /
                    DISPLAY = (FILL) GROUP = Status LABEL = Label
                                           XAXIS = X2 YAXIS = Y2;
        SCATTERPLOT X = T Y = SubjectID / GROUP = Status;
        HIGHLOWPLOT LOW = Origin HIGH = T Y = SubjectID /
                                                GROUP = Status;
      ENDLAYOUT;
    ENDGRAPH;
  END;
RUN;
```

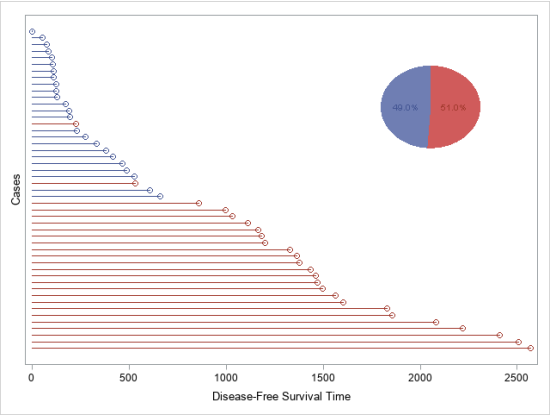


Figure 8: Lowest section of the whole needed figure

We can easily modify the template to include histograms and the fingers without any data modification, we only need to include a different LAYOUT OVERLAY and encapsulate both of them into a LAYOUT LATTICE. All of this can be archived with the following piece of code obtaining figure 9

```
PROC TEMPLATE;
  DEFINE STATGRAPH PieChart;
    BEGINGRAPH ;
    LAYOUT LATTICE / ROWS = 2 ROWWEIGHTS = (0.2 0.8)
                                  COLUMNDATARANGE = UNION;
      COLUMNAXES;
        COLUMNAXIS / DISPLAY=(TICKS TICKVALUES LABEL) LABEL = "Time";
      ENDCOLUMNAXES;
    LAYOUT OVERLAY / YAXISOPTS =(DISPLAY = (LINE));
      BOXPLOT Y=T X = Status / ORIENT=HORIZONTAL BOXWIDTH=.9
                     DATATRANSPARENCY=0.4 GROUP = Status;
      FRINGEPLOT T / GROUP = Status ;
    ENDLAYOUT;
      LAYOUT OVERLAY / Y2AXISOPTS = (DISPLAY = (LINE)
                       LINEAROPTS=(VIEWMIN = 500 VIEWMAX = 2500))
                      X2AXISOPTS = (DISPLAY = (LINE)
                         LINEAROPTS=(VIEWMIN = 0 VIEWMAX = 2500))
                      YAXISOPTS = (DISPLAY = (LINE LABEL)
                                   LABEL = "Cases" REVERSE = TRUE);
        POLYGONPLOT X = X_Polar Y = Y_Polar ID = Status /
                    DISPLAY = (FILL) GROUP = Status LABEL = Label
                                              XAXIS = X2 YAXIS = Y2;
        SCATTERPLOT X = T Y = SubjectID / GROUP = Status;
        HIGHLOWPLOT LOW = Origin HIGH = T Y = SubjectID /
                                                  GROUP = Status;
      ENDLAYOUT;
    ENDLAYOUT;
    ENDGRAPH;
  END;
RUN;
```

Finally, we can add a new row into the LAYOUT LATTICE, and include a histogram. To match with the look of the reference figure 6, the histogram will be updated only on the marks indicated into x-axis, so each 500 units of

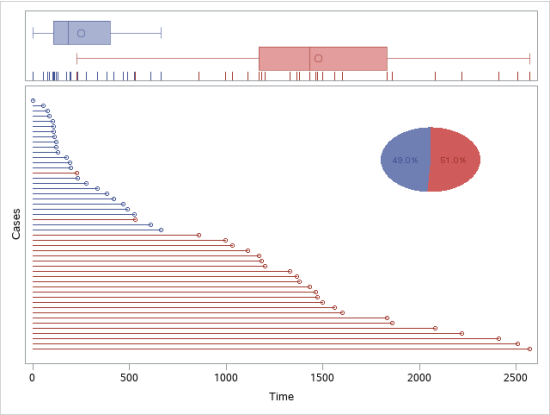


Figure 9: Two out of three of the whole needed figure

time. Due to the last bit of figure added, the pie chart needs some adjustments to preserve the ratio, and we are going to add a legend.

We can do so with a few options into HISTOGRAM and a SIDEBAR with a DISCRETELEGEND, as shown bellow to obtain our final figure 10.

```
PROC TEMPLATE;
  DEFINE STATGRAPH PieChart;
    BEGINGRAPH ;
    LAYOUT LATTICE / ROWS = 3 ROWWEIGHTS = (0.45 0.1 0.45)
                                  COLUMNDATARANGE = UNION;
      COLUMNAXES;
        COLUMNAXIS / DISPLAY=(TICKS TICKVALUES LABEL) LABEL = "Time";
      ENDCOLUMNAXES;
      LAYOUT OVERLAY;
        HISTOGRAM T / BINSTART = 250 BINWIDTH = 500 NBINS = 6
                                    DISPLAY = (FILL) GROUP = Status
                                             DATATRANSPARENCY = 0.6;
      ENDLAYOUT;
    LAYOUT OVERLAY / YAXISOPTS =(DISPLAY = (LINE));
      BOXPLOT Y=T X = Status / ORIENT=HORIZONTAL BOXWIDTH=.9
                       DATATRANSPARENCY=0.4 GROUP = Status;
      FRINGEPLOT T / GROUP = Status ;
    ENDLAYOUT;
      LAYOUT OVERLAY / Y2AXISOPTS = (DISPLAY = (LINE)
                        LINEAROPTS=(VIEWMIN = 1500 VIEWMAX = 2500))
                       X2AXISOPTS = (DISPLAY = (LINE)
                          LINEAROPTS=(VIEWMIN = 0 VIEWMAX = 2750))
                       YAXISOPTS = (DISPLAY = (LINE LABEL)
                                    LABEL = "Cases" REVERSE = TRUE);
        POLYGONPLOT X = X_Polar Y = Y_Polar ID = Status /
                    DISPLAY = (FILL) GROUP = Status LABEL = Label
                                                XAXIS = X2 YAXIS = Y2;
        SCATTERPLOT X = T Y = SubjectID / GROUP = Status;
        HIGHLOWPLOT LOW = Origin HIGH = T Y = SubjectID /
                                 NAME = "Survival"  GROUP = Status;
      ENDLAYOUT;
      SIDEBAR / ALIGN = RIGHT;
        DISCRETELEGEND "Survival" / TITLE = "Status" BORDER = FALSE
                                                       ACROSS = 1;
      ENDSIDEBAR;
    ENDLAYOUT;
    ENDGRAPH;
  END;
RUN;
```

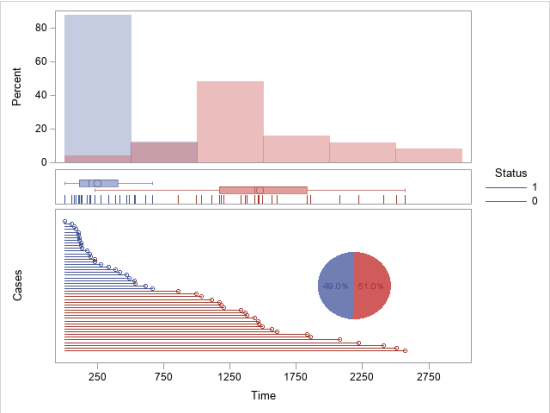


Figure 10: Whole figure summarising survival with a Pie Chart

5 Mosaic with text

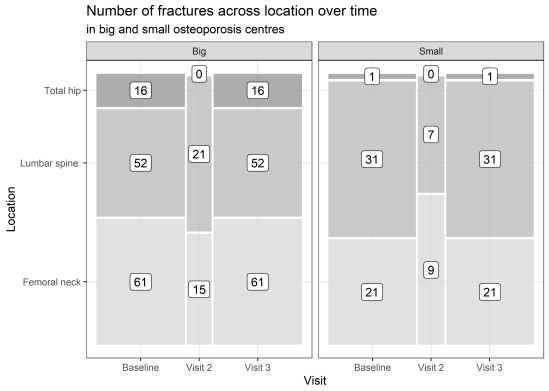


Figure 11: Mosaic plot done on R

The next figure that we are going to discuss is the Mosaic, as can be seen on figure 11.

Similarly to pie chart,this plot can be obtained throw LAYOUT REGION, but did not allow interaction to any other kind of plot, and thus, the percentages or frequencies can not be added in any way other than with annotations [10].

We are going to use a different approach, to avoid the use of annotations, that require an additional data set with the annotations hand placed and will need to be moved around with each new data. Instead we are going to use polygons, similar to the pie chart.

We are going to use Cars data, and display type vs origin. First, we are going to calculate the frequency tables, we can do it as follows.

```
PROC FREQ DATA = SASHELP.Cars;
   WHERE Type ne 'Hybrid';
   TABLES Type / OUT = TypeFreq OUTCUM ;
RUN;

PROC FREQ DATA = SASHELP.Cars;
   WHERE Type ne 'Hybrid';
   TABLES Origin * Type / OUT = CrossFreq (WHERE=(PERCENT ne .))
                                                        OUTPCT;
RUN;
```

Notice that we removed Type = "Hybrid", to be equivalent to the one displayed on bibliography [10] and be able to compare the output. Also we removed in the cross table all categories with missing percentage.

The idea is to display in the x-axis the Type and Origin variable into y-axis. To do so, we need to calculate the coordinates where each one is going to be plotted. We are going to use TypeFreq to decide the x-axis, and the CrossFreq to decide the y-axis. Additionally, we are going to calculate a mid point to place

the axis (Midpoint_X and Midpoint_Y) label and an ID of each different box that we are going to display (MosaicID)

```
DATA TypeFreq;
  SET TypeFreq;
  Start_X = Cum_Pct - Percent;
  End_X   = Cum_Pct;
  Midpoint_X = (Start_X + End_X) / 2;
  KEEP Type Start_X End_X Midpoint_X;
RUN;

PROC SORT DATA = CrossFreq;
  BY Type Origin;
RUN;

DATA CrossFreq;
  SET CrossFreq;
  BY Type;
  RETAIN Cum_Pct 0 MosaicID 0;
  MosaicID = MosaicID + 1;
  IF first.Type THEN Cum_Pct = 0;
  Cum_Pct = Cum_Pct + Pct_Col;
  Start_Y = Cum_Pct - Pct_Col;
  End_Y   = Cum_Pct;
  /* Midpoint on first Y category only */
  IF Type = "SUV" THEN Midpoint_Y = (Start_Y + End_Y) / 2;
  KEEP Type Origin Start_Y End_Y Count MosaicID Midpoint_Y;
RUN;
```

We are going to merge these two data sets together, to have into the same data set x and y coordinates, and being able to create the squares polygons.

```
DATA All;
  MERGE CrossFreq
        TypeFreq;
  BY Type;
  /* Space between blocs */
  Gap = 1;
  /* Create rectangles for each category */
  X = Start_X;
  IF Start_X ne 0 THEN X = X + Gap/2;
  Y = Start_Y;
  IF Start_Y ne 0 THEN Y = Y + Gap/2;
  OUTPUT;
  X = Start_X;
  IF Start_X ne 0 THEN X = X + Gap/2;
  Y = End_Y;
  IF End_Y ne 100 THEN Y = Y - Gap/2;
  OUTPUT;
  X = End_X;
  IF End_X ne 100 THEN X = X - Gap/2;
  Y = End_Y;
  IF End_Y ne 100 THEN Y = Y - Gap/2;
  OUTPUT;
  X = End_X;
  IF End_X ne 100 THEN X = X - Gap/2;
  Y = Start_Y;
  IF Start_Y ne 0 THEN Y = Y + Gap/2;
  OUTPUT;
  LABEL X = "Type"
        Y = "Origin";
RUN;
```

We have now all needed information to create the squares and take a first look to the output. With the following template we can generate figure 12.

```
PROC TEMPLATE;
  DEFINE STATGRAPH MyMosaic;
    BEGINGRAPH;
     LAYOUT OVERLAY;
         POLYGONPLOT X=X Y=Y ID = MosaicID  /  DATATRANSPARENCY = 0.5
                  DISPLAY=(FILL OUTLINE) GROUP = Origin LABEL = Count
                  LABELATTRS=(COLOR = BLACK SIZE = 10)
                  OUTLINEATTRS=(COLOR = BLACK);
     ENDLAYOUT;
   ENDGRAPH;
  END;
RUN;

PROC SGRENDER DATA = All TEMPLATE = MyMosaic;
RUN;
```

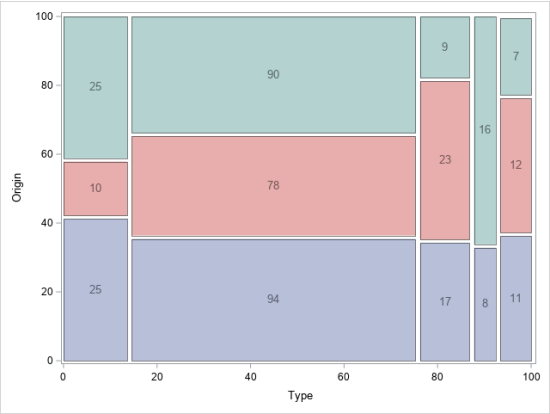


Figure 12: First iteration of the Mosaic

First iteration is almost perfect, the only thing that need improvement are X and Y axis, that currently are displaying the percentages. To get the correct text on each one of the axis, we are going to create formats from the data sets as follows.

```
DATA TypeFmt;
   LENGTH Label $8;
   SET TypeFreq (RENAME = (Type = Label));
   RETAIN FmtName 'Type_n' Type 'n';
   /* Round to avoid issues with overlapping */
   Start = ROUND(Start_X,0.001);
   End   = ROUND(End_X,0.001);
   OUTPUT;
   KEEP Start End Label FmtName;
RUN;

PROC FORMAT LIBRARY=WORK CNTLIN=TypeFmt;
RUN;

DATA OriginFmt;
   LENGTH Label $8;
   SET CrossFreq (RENAME = (Origin = Label));
   WHERE Type = "SUV";
   RETAIN FmtName 'Origin_n' Type 'n';
   /* Round to avoid issues with overlapping */
   Start = ROUND(Start_Y,0.001);
   End   = ROUND(End_Y,0.001);
   OUTPUT;
   KEEP Start End Label FmtName;
run;

PROC FORMAT LIBRARY=WORK CNTLIN=OriginFmt;
RUN;
```

Now that we have formats Origin_n and Type_n, we only need to select the midpoints where they are going to be placed. To avoid do it by hand, we are going to use a bit of SQL to create a list with all midpoints, and later on we will apply the format, this way they will be converted to the desired category. The SQL code is as follows

```
PROC SQL NOPRINT;
  SELECT Midpoint_X INTO :TypeList SEPARATED BY " "
  FROM TypeFreq;
QUIT;

PROC SQL NOPRINT;
  SELECT Midpoint_Y INTO :OriginList SEPARATED BY " "
  FROM CrossFreq WHERE Type = "SUV";
QUIT;
```

With all of this in place, it's enough to update the LAYOUT OVERLAY by the following in the previous template to get figure **??**.

```
LAYOUT OVERLAY /
      XAXISOPTS = (LINEAROPTS = (TICKVALUEFORMAT = Type_n.
                                 TICKVALUELIST = (&Typelist.)
                                 TICKVALUEFITPOLICY = ROTATE))
      YAXISOPTS = (LINEAROPTS = (TICKVALUEFORMAT = Origin_n.
                                 TICKVALUELIST = (&Originlist.)));
```

This method is easier to adapt to new data (only the first category of the plot is indicated), and there is no hand placement for the number of counts, that can be also changed to whatever we prefer (e.g. percentages). Also if needed it

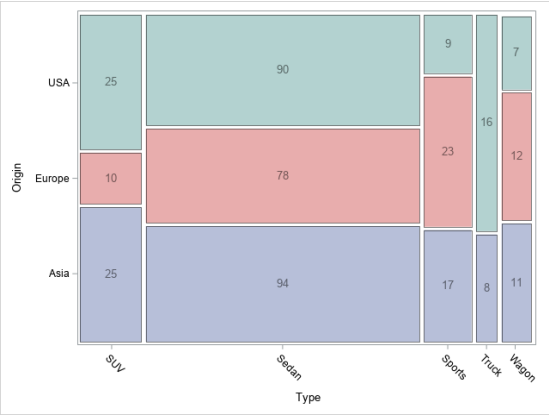


Figure 13: Final mosaic plot produced by SAS

can be combined with any other kind of plots.

6 Worth mention

Those are only a few examples that we tried to replicated. There where about 20 of them. Some of the ones that didn't pass the cut where QQ-plot with CI, that SAS by default do not produce them (formula can be extracted from R code or John Fox book [2]), Kaplan-Meier with number at risk and median alone or in panel (there are a lot of ways to do them, here is a good explanation of how to do it for one KM alone, that can be encapsulated on a LAYOUT PROTOTYPE [7]), alluvial plots (a github and a few articles had bean already posted [1],[8]) and a lot more that .

7 Conclusions

In conclusion, I think that there are no one clear winner in figure related, both have similar capabilities. That said, it's true that it seems that those modifications and more innovative plots are easier to do in R, in the sense that it has more flexible tools. That could be do to different reasons, but probably, it's more related with the main target of each program. SAS has a clear target on Pharma and Banking sector, meanwhile R has a more big audience, due to that, it needs to adapt to a higher spectrum of possibilities and customisation that may not be that crucial to the average SAS user.

References

[1] Ryan Bailey and Shane Rosanbalm. *sas-sankeybarchart*. 2017. URL: https://github.com/RhoInc/sas-sankeybarchart.

[2] John Fox. *Applied Regression Analysis and Generalized Linear Models*. SAGE Publications, 2008.

[3] Sakshi Gupta. *SAS vs R: Which is Better?* Feb. 2020. URL: https://in.springboard.com/blog/sas-vs-r/.

[4] Volker Harm Catalina Mejía Herrera. *Grouped Jittered Box Plots in SAS 9.2 and SAS 9.3*. 2012. URL: https://lexjansen.com/phuse/2012/cs/CS03.pdf.

[5] Sanjay Matange. *Violin Plots*. Oct. 2012. URL: https://blogs.sas.com/content/graphicallyspeaking/2012/10/30/violin-plots/.

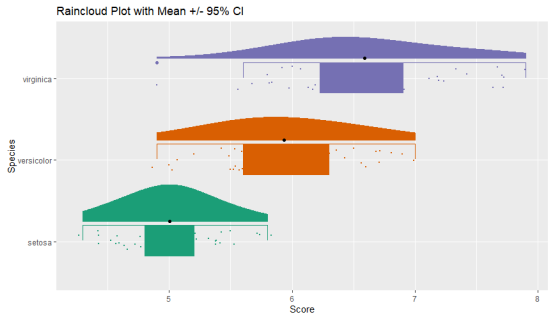
[6] Priya Pedamkar. *Difference Between SAS and R*. URL: https://www.educba.com/sas-vs-r/.

[7] Antonio Rodríguez and Sebastià Barceló. *Guide for Producing Figures with Graphic Template Language (GTL) Using SAS*. Lulu, 2020.

[8] Shane Rosanbalm. "Getting Sankey with Bar Charts". In: *PharmaSUG* (2015).

[9] *SAS vs R: Which One is Better Statistics Language*. Aug. 2019. URL: https://www.codeavail.com/blog/sas-vs-r/.

[10] Rick Wicklin. *How to add an annotation to a mosaic plot in SAS*. July 2019. URL: https://blogs.sas.com/content/iml/2019/07/08/add-annotation-mosaic-plot-sas.html.



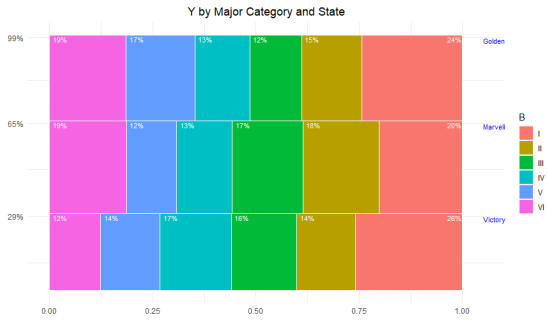
A R codes

In case that the reader is not experienced in R figures creation, we are going to display the code to create equivalent graphs using R and ggplot2 package.

A.1 Raincloud

To generate a raincloud as shown on figure A.1 the following code can be used

```r
# Load packages ----
library(ggplot2)
library(PupillometryR)
library(gghalves)
library(plyr)

#Raincloudsw with mean and confidence interval
summarySE <- function(data = NULL, measurevar, groupvars = NULL,
              na.rm = FALSE, conf.interval = .95, .drop = TRUE) {
  # New version of length which can handle NA's: if na.rm==T, don't
     count them
  length2 <- function(x, na.rm = FALSE) {
    if (na.rm) {
      sum(!is.na(x))
    } else {
      length(x)
    }
  }

  # This does the summary. For each group's data frame, return a
  # vector with N, mean, median, and sd
  datac <- plyr::ddply(data, groupvars, .drop=.drop,
                       .fun = function(xx, col) {
                         c(N      = length2(xx[[col]], na.rm=na.rm),
                           mean   = mean(xx[[col]], na.rm=na.rm),
                           median = median(xx[[col]], na.rm=na.rm),
                           sd      = sd(xx[[col]], na.rm=na.rm)
                         )
                       },
                       measurevar
  )

  # Rename the "mean" and "median" columns
  datac <- plyr::rename(datac, c("mean" = paste(measurevar, "_mean",
     sep = "")))
  datac <- plyr::rename(datac, c("median" = paste(measurevar, "_
     median", sep = "")))

  datac$se <- datac$sd / sqrt(datac$N)  # Calculate SE of the mean

  # Confidence interval multiplier for standard error
  # Calculate t-statistic for confidence interval:
  # e.g., if CI is .95, use .975 (above/below), and use df=N-1
  ciMult <- qt(conf.interval / 2 + .5, datac$N - 1)
  datac$ci <- datac$se * ciMult

  return(datac)
}
myiris = iris[,c(1,5)]
summary_iris <- summarySE(myiris, measurevar = "Sepal.Length",
                          groupvars = c("Species"))

ggplot(iris,aes(x=Species,y=Sepal.Length, fill = Species, colour =
     Species))+
 geom_flat_violin(position = position_nudge(x = .25, y = 0),adjust
     =2)+
  geom_point(position = position_jitter(width = .15), size = .5)+
  geom_half_boxplot( size = .15,position = position_nudge(x = .2, y =
      0)) +
  geom_point(data = summary_iris, aes(x = Species, y = Sepal.Length_
     mean), position = position_nudge(.25), colour = "BLACK")+
  ylab('Score')+xlab('Species')+coord_flip()+guides(fill = FALSE,
     colour = FALSE) +
  scale_colour_brewer(palette = "Dark2")+
  scale_fill_brewer(palette = "Dark2")+
  ggtitle("Raincloud Plot with Mean +/- 95% CI")
```



A.2 Mosaic

To generate a Mosaic plot as shown on figure A.3 the following code can be used

```r
library(MASS)
library(tidyr)
library(dplyr)
library(ggplot2)
data <- oats[oats$N == "0.0cwt",]
data_mosaic2 <-data %>% group_by(V) %>%
  mutate(
    share = Y / sum(Y),
    tot_group = sum(Y)
  ) %>% ungroup()

data_mosaic <-data_mosaic2 %>%
  group_by(B) %>%
  arrange(desc(V)) %>%
  mutate(
    ymax = cumsum(tot_group) / sum(tot_group),
    ymin = (ymax - (tot_group/sum(tot_group)))
  ) %>% ungroup() %>%
  group_by(V) %>%
  arrange(desc(B)) %>%
  mutate(xmax = cumsum(share), xmin = xmax - share) %>%
  ungroup() %>%
  arrange(B)

ggplot(data_mosaic) +
  geom_rect(aes(ymin = ymin, ymax = ymax, xmin = xmin, xmax = xmax,
     fill = B), colour = "white", size = 0.2)+
  theme_light()

labels <- data_mosaic %>%
  filter(B == "I") %>%
  mutate(y = ymax - 0.01, yRange = (ymax - ymin)* 100) %>%
  select(V, xmax, y, yRange) %>%
  ungroup()

value_labels <- data_mosaic %>%
  select(V, B, xmin, xmax, ymax, share) %>%
  mutate(
    x = ifelse(B == "I", xmax, xmin),
    y = ymax - 0.005,
    label = paste0(round(share * 100), "%"),
    hjust = ifelse(B == "I", 1.05, -0.25)
  )

ggplot(data_mosaic) +
  geom_rect(aes(ymin = ymin, ymax = ymax, xmin = xmin, xmax = xmax,
                fill = B), colour = "white", size = 0.2)+
  theme_light()+
  geom_text(
    data = labels,
    aes(x = 1.05, y = y, label = as.character(V)),
    hjust = 0, vjust = 1, colour = "blue",size=3
  ) +
  geom_text(
    data = value_labels,
    aes(x = x, y = y, label = label, hjust = hjust),
    vjust = 1, size = 3, alpha = 1, colour = "white"
  ) +
  scale_y_continuous( breaks = labels$y, limits = c(0, 1),
                      labels = scales::percent)+
  theme_minimal()+
  theme(axis.title=element_blank(),
        plot.title = element_text(hjust = 0.5))+
  ggtitle("Y by Major Category and State")
```

A.3 Pie chart

To generate a Pie chart inside an other plot as shown on figure **??** the following code can be used

```r
library(KMsurv)
library(ggplot2)
library(scatterpie)
library(ggpubr)
data(bmt)
mybmt<- bmt[1:50,c(1,3,4)]
A <- sum(ifelse(mybmt$d1==0,1,0))
B <- sum(ifelse(mybmt$d1==1,1,0))
x <- 2000
y <- 2000
radius = 500
pie <- data.frame(A,B,x,y,radius)
mybmt$d1 <- as.factor(mybmt$d1 + 1)
mybmt <- mybmt[order(mybmt$t2),]
mybmt$id <- (50:1) * 50
histogram <- ggplot(mybmt, aes(x=t2, color=d1, fill=d1)) +  geom_histogram(binwidth=500,center=250,alpha=0.5,position="identity")
boxplot   <- ggplot(mybmt, aes(x=t2, y=d1, color=d1)) +  geom_boxplot()
series    <- ggplot(mybmt, aes(x=t2, y=id, color=d1)) + geom_point(na.rm=TRUE, size=3, pch=18) +  geom_segment(aes(xend=0,yend=id)) +
                  geom_scatterpie(aes(x=x, y=y,r=radius), data=pie, cols=c("A", "B"))
figure <- ggarrange(histogram, boxplot, series,
                    labels = c("A", "B", "C"),
                    ncol =1, nrow = 3)
figure
```