Paper 164-2024

Here Comes the Rain (Cloud Plot) Again

Bartosz Jabłoński, yabwon / Warsaw University of Technology

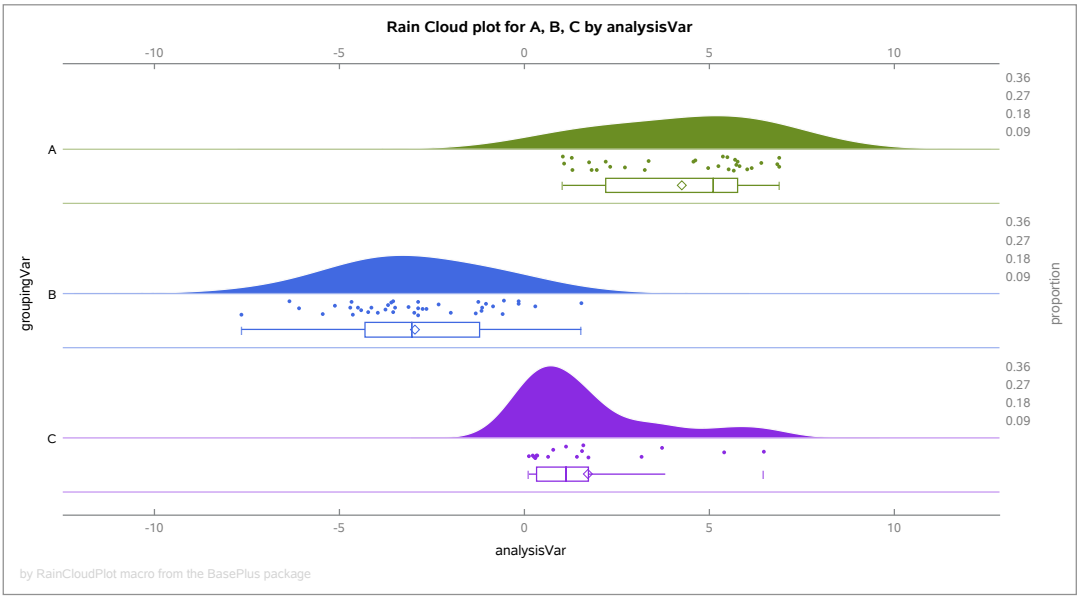
ABSTRACT

Rain cloud plots are very popular and practical tools for data visualization. They enable the display of several components in one figure, including: kernel density estimates, data points, and box-and-whisker plots. This method facilitates the comparison of variable distribution across different categories. During this presentation, a macro called `%RainCloudPlot()` will be introduced and its capabilities for creating rain cloud plots will be presented. Sample code will accompany all examples.

INTRODUCTION or "What Rain Cloud Plots are?"

Rain Cloud Plots are very practical data visualization tools. They were discussed, presented, and implemented at different occasions (e.g., see [**Allen *et al.* 2019**], [**Jablonski 2021(1)**], [**Morioka 2022**], and [**Tsutsugo 2022**]). As the old saying goes: *A picture is worth a thousand words*[1]. Before describing what *rain cloud plots* are first let's start with showing an example of our first[2] rain cloud plot (produced with help of the `%RainCloudPlot()` macro).

Figure 1: My First Rain Cloud Plot



[1] See: `https://en.wikipedia.org/wiki/A_picture_is_worth_a_thousand_words`

[2] Assuming you had never seen one before

A *Rain Cloud Plot* is a plot that enhances visual data analysis by combining three basic plots:

- kernel density estimate plot (the "*cloud*"),
- scatter plot of jittered raw data values (the "*rain drops*"), and
- box-and-whiskers plot (a "*train in the rain*")

in one picture. If we execute a visual data analysis with those three simple plots separately, in each case, we miss some information describing data. For example, the kernel density plot and the scatter plot don't accentuate outliers. On the other hand, the box-and-whisker plot cannot show multimodality in data distribution. When we combine them, those "three musketeers" have each other's back and give us a comprehensive insight.

Computer scientist Edsger Dijkstra is attributed to have said: "A picture may be worth a thousand words, a formula is worth a thousand pictures." With help of the `%rainCloudPlot()` macro, first introduced to the `basePlus` SAS package in [**Jablonski 2021(1)**], all the code that we need to run to generate a graph like in Figure 1 is the following snippet:

code: create basic rain cloud plot

```
%rainCloudPlot(
  ds   /* input data set */
, gr   /* grouping variable */
, vars /* analyzed variable */
)
```

The meaning of those three positional parameters is the following: the firs one `ds` indicates input data set, the second `gr` denotes a grouping variable(s) which provides categories, and the third `vars` indicates analysis variable(s) which data are used to generate the plot.

To learn how to install and use SAS packages, like `basePlus`, see Appendix B - install the SAS Packages Framework and packages.

DATA

We cannot make any plot without data, so let's cook some. Assume we have a data set named `have` with grouping variable `groupingVar` with 3 values "A", "B", and "C", and an analysis variable `analysisVar`, all of it generated by the following code (for the coloring convention check Appendix A - code coloring guide):

code: data set to play with

```
data have;
  call streaminit(42);
  do i = 1 to 3;
    groupingVar = byte(64+i);
    do j = 1 to rand("integer",17,42);
      select(i);
        when(1) analysisVar = rand("uniform", 1, 7);
        when(2) analysisVar = rand("normal", -3, 2);
        when(3) analysisVar = rand("exponential", 2);
        otherwise;
      end;
      output;
    end;
  end;
  drop i j;
run;
```

The log shows that `have` data set has 87 observations and two variables.

the log

```
NOTE: The data set WORK.HAVE has 87 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

THE %RainCloudPlot() MACRO UNVEILED

In this section we are going to disassemble the `%RainCloudPlot()` macro, its parameters, and how they work for our advantage for preparing publication ready graphics. All the following examples were generated with `basePlus` SAS package in version `1.41.0` or later.

We have already learned about three positional parameters the macro has, all other parameters are `key=value` pairs parameters with predefined default values. Let's learn them all[3]!

TITLES AND FOOTNOTES

There is no good plot without meaningful title. To add titles or footnotes to the plot we use the `title=` and `footnote=` parameters.

code: rain cloud plot with custom titles and footnotes

```
%rainCloudPlot(
  have
, groupingVar
, analysisVar
, TITLE = %nrstr(title1 J=C HEIGHT=2 "The Rain Cloud Plot";
                 title2 J=C HEIGHT=1 "Plotting for groups: &list_g.";
                )
, FOOTNOTE =
    %str(footnote1 J=R H=1 C=Gray
         "Data set HAVE, with 87 observations, two variables.";)
)
```

The result of the snippet above is presented in Figure 2.

The meaning of parameters is the following. Both `title=` and `footnote=` expect syntactically correct `TITLEn` or `FOOTNOTEn` statements. These statements contain special characters, e.g., semicolons(`;`) or other, hence the text has to be enclosed in macro quoting function `%STR()`. If, like in our example, we want to call an internal macrovariable `list_g`, that keeps a list of grouping variable values, or any other one, we have to use the `%NRSTR()` macro quoting function.

CLOUDS IN COLORS

Each group value has its own color assigned. Here are 12 predefined colors used, and they are "recycled" when the number of values is bigger than a dozen.

The default list of colors, that is: `BlueViolet`, `RoyalBlue`, `OliveDrab`, `Gold`, `HotPink`, `Crimson`, `MediumPurple`, `CornflowerBlue`, `YellowGreen`, `Goldenrod`, `Orchid`, and `IndianRed`, is presented in Figure 3.

[3]Well, almost all. We skipped a few, see the documentation: `https://github.com/SASPAC/baseplus/` to really see all.

Figure 2: Rain Cloud Plot with customized titles and footnotes

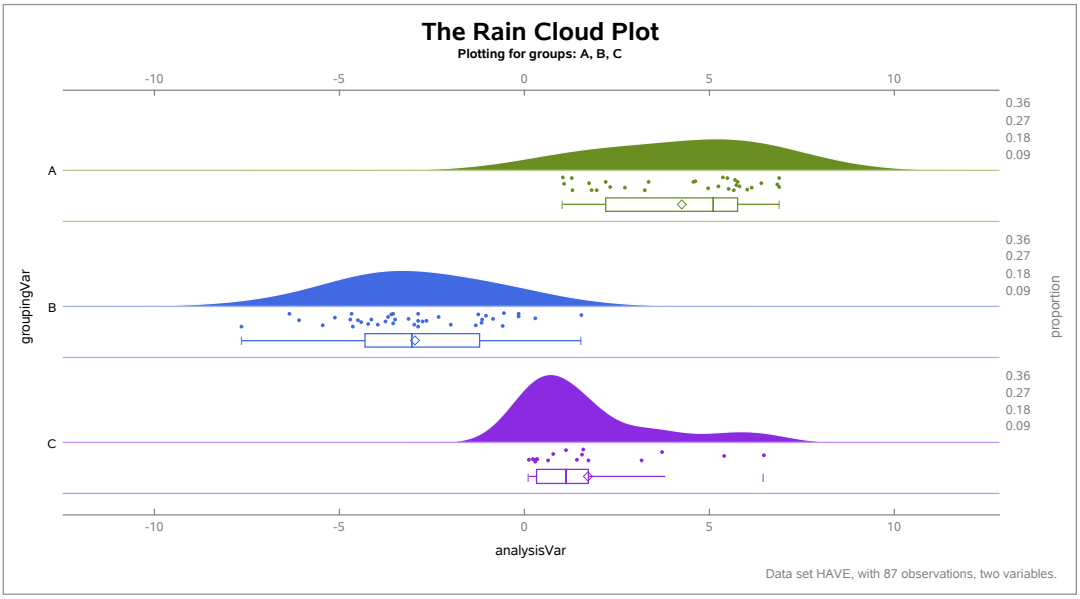
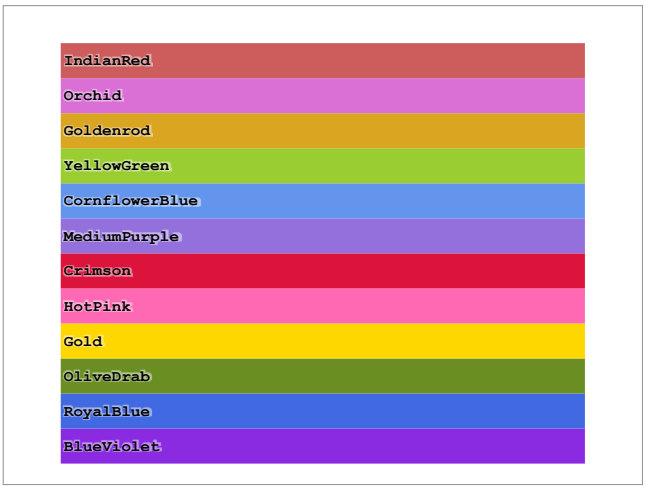


Figure 3: Rain Cloud Plot default colors



If we want to change colors we can do it in two ways. The first, by making the plot gray-scale with the `monochrome=` parameter set to `1`. The effect looks like in Figure 4

The second way is by altering colors list for the plot with the `colorslist=` parameter. Colors can be provided in any "valid" form which is accepted by the `SGPLOT` procedure, e.g., `colorslist = CXFF0011 "very light purplish blue" darkgreen`.

code: rain cloud plot with custom colors list

```
%rainCloudPlot(
  have
, groupingVar
, analysisVar
, title = %nrstr(title1 J=C HEIGHT=2 "The Rain Cloud Plot";
                 title2 J=C HEIGHT=1 "Plotting for groups: &list_g.";)
, footnote = %str(footnote1 J=R H=1 C=Gray
                 "Data set HAVE, with 87 observations, two variables.";)
, COLORSLIST = CornflowerBlue OliveDrab IndianRed
)
```

The effect of using the code above is presented in Figure 5. As we can see, colors from the list are applied to the plot grouping data from the bottom to the top.

Figure 4: Rain Cloud Plot in shades of grey

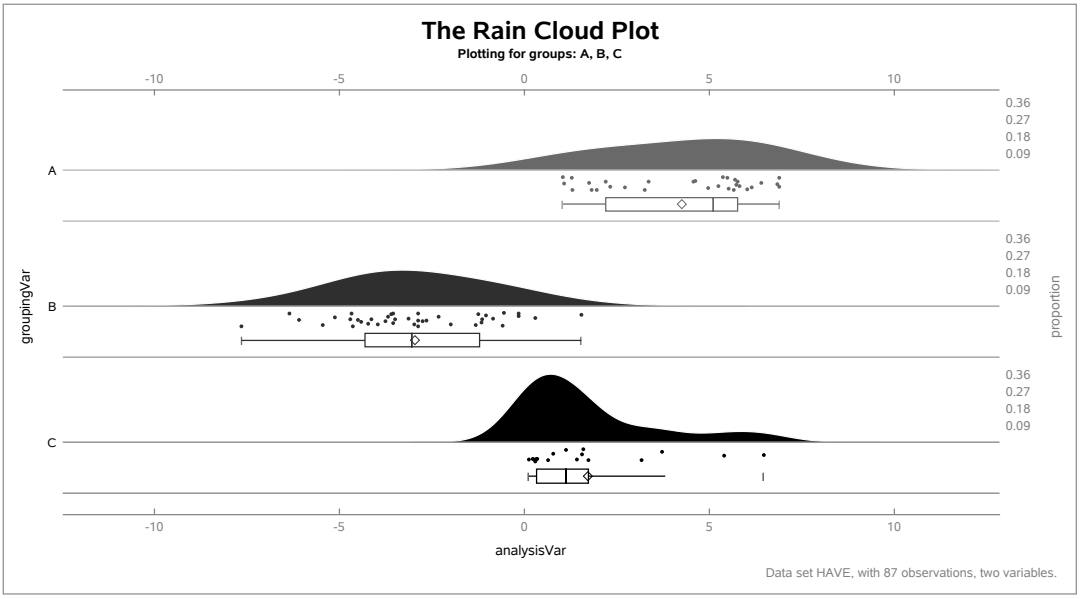
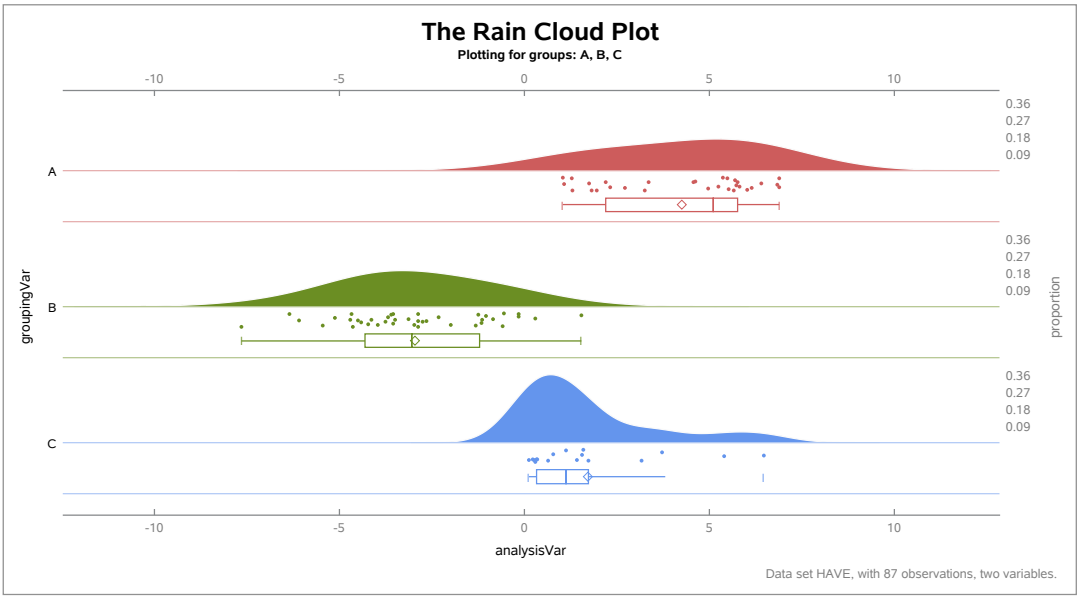


Figure 5: Rain Cloud Plot with custom colors list: CornflowerBlue, OliveDrab, and IndianRed



CLOUDS SHAPES AND RAIN INTENSITY

To change "raindrops" (data points) size we can use `rainDropSize=` parameter, by default it is `5px`.

The shape of "the cloud" (kernel density estimate) can be changed with a few more parameters. These parameters are:

- `KERNEL_K=` and `KERNEL_C=`,
- `VSCALE=` and `VSCALEmax=`.

We describe them in the subsequent paragraph while introducing mathematics that does the job behind the scenes.

The kernel density estimate is calculated by the UNIVARIATE procedure. All the mathematics describing formulas used "under-the-hood" for calculations can be found in SAS documentation. For the time this article is published (September 2024) documentation for the UNIVARIATE procedure, in particular for kernel density estimate, can be found here:
documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/procstat/procstat_univariate_details60.htm

The kernel density estimate is calculated using the following formula:

$$\hat{f}_\lambda(x) = \frac{hv}{n\lambda} \sum_{i=1}^{n} K_0\left(\frac{x - x_i}{\lambda}\right)$$

Where:

- $K_0(\cdot)$ is the kernel function, it is selected by setting KERNEL_K= to one of the following values: normal, quadratic, or triangular (see SAS documentation for precise definitions),
- n is the sample size,
- λ is the bandwidth, calculated as $\lambda = cQn^{-\frac{1}{5}}$, where the Q is the interquartile range and c is scaling factor provided by KERNEL_C= parameter (note: c has to satisfy $0 < c \leq 1$),
- x_i is the i^{th} observation,
- h is width of histogram interval,
- v is vertical scaling factor, its values depend on the VSCALE= parameter selected and are: from 0 to n for VSCALE = count, from 0 to 100 for VSCALE = percent, and from 0 to 1 for VSCALE = proportion.

For better understanding the h and v parameters see [**Wicklin 2024**]. The VSCALEmax= parameter sets upper limit on the right hand side vertical axis scale. If the VSCALEmax= parameter is used the following note is printed the log:

the log - note for VSCALEmax

```
INFO: VSCALE=Percent. Maximum value used: 60
      Recommended values are:
      - VSCALE=PERCENT between 0 and 100.
      - VSCALE=PROPORTION between 0 and 1.
      - VSCALE=COUNT between 0 and N (sample size).
```

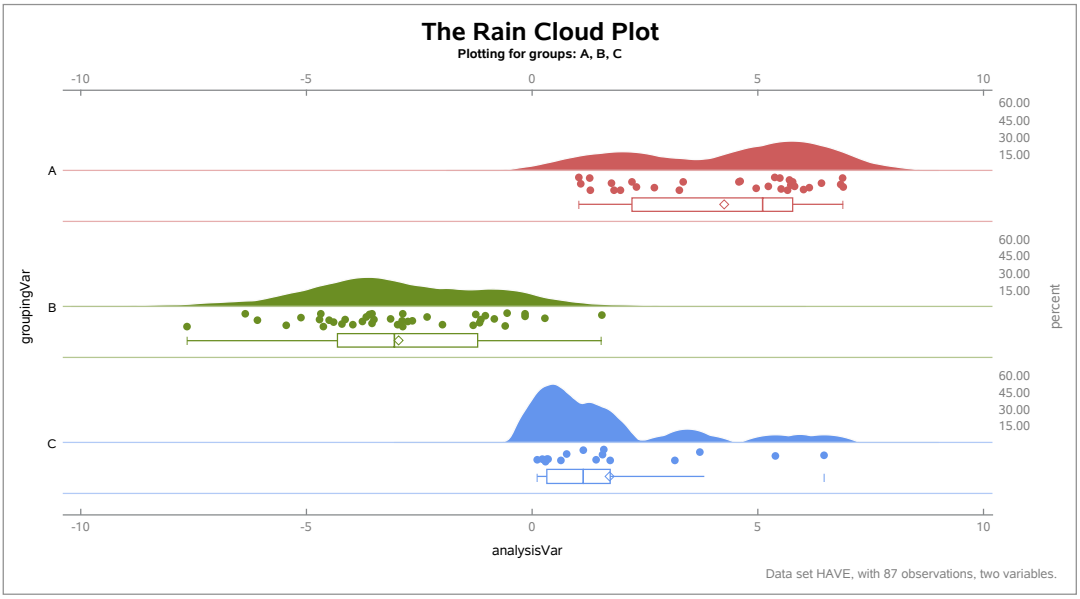
The following snippet changes "cloud shape" and "rain intensity". We increase size of scatter plot symbols from default 5px to 8px. We change kernel properties so that quadratic function with $c = 0.99$ is used, scale is set to percents, and maximum value is set to 60%

code: alternating kernel density plot and scatter plot

```
%rainCloudPlot(
  have
, groupingVar
, analysisVar
, title = %nrstr(title1 J=C HEIGHT=2 "The Rain Cloud Plot";
                 title2 J=C HEIGHT=1 "Plotting for groups: &list_g.";)
, footnote = %str(footnote1 J=R H=1 C=Gray
                 "Data set HAVE, with 87 observations, two variables.";)
, colorslist = CornflowerBlue OliveDrab IndianRed
, RAINDROPSIZE = 8px
, KERNEL_K = quadratic
, KERNEL_C = 0.99
, VSCALE = Percent
, VSCALEmax = 60
)
```

Result is presented in Figure 6 and we can clearly see change in the plot shape.

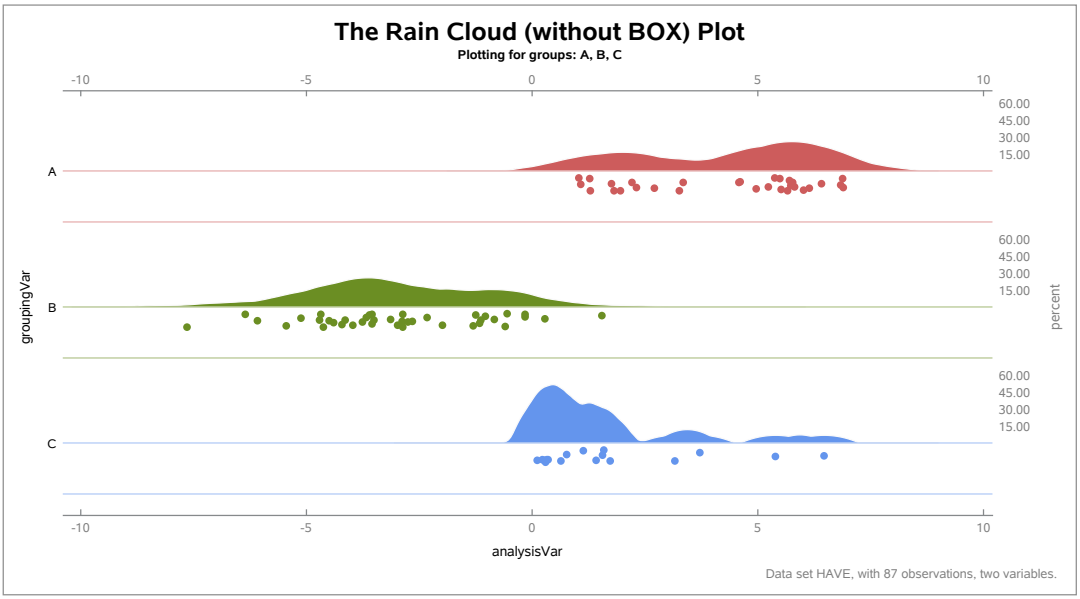
Figure 6: Rain Cloud Plot with customized kernel density plot and scatter plot



WHISKERS IN A BOX

The box-and-whisker plot is displayed by default, but when we set the `boxPlot=` parameter to `0`, the B-and-W will not be printed. Such case is presented in Figure 7.

Figure 7: Rain Cloud Plot without box-and-whiskers plot



Though it is possible to drop the B-and-W plot, we do not recommend this. Removing it reduces our visual insights about simple descriptive statistics. Since we have mentioned descriptive statistics, it is a good idea to explain what they are on the plot so that no one would be confused looking at the graph.

Symbols on the `box-and-whisker` plot have the following interpretation:

- *left vertical bar* indicates the data $minimum$,
- *left whisker line* starts at $max(Q1–1.5IQR,\ minimum)$[4] and ends at lower quartile ($Q1$, left wall of the box),
- *diamond* indicates mean,

[4] IQR is interquartile range

- *vertical bar inside of the box* indicates median,
- *right whisker line* starts at upper quartile ($Q3$, right wall of the box) and ends at $min(Q3{+}1.5IQR,\ maximum)$,
- *right vertical bar* indicates the data $maximum$.

[**Highlight:**] With above setup it may happen that there is a *gap* between the minimum marker and the beginning of the left whisker or there is a *gap* between the end of the right whisker and the maximum marker. See Figure 6, group "C", there are two observations landing in the *gap* on the right side. These two, if we assume the "classic" definition[5], would be considered outliers.

The box-and-whisker plot aesthetics can be modified with use of the following parameters:

- `boxPlotFill=` that indicates, on scale from 0 to 1, box background transparency (default value is 0, and means fully translucent),
- `boxPlotLineSize=` indicates lines thickness, default value is `1px`,
- `boxPlotSymbolSize=` is a two element space-separated list of sizes for the mean (diamond) and min-max (vertical bars), default value is `8px`; if two values are provided, e.g., `16px 8px` the firs one is for the diamond.

The following snippet shows how to alternate the box-and-whisker plot:

code: alternating the box-and-whiskers plot

```
%rainCloudPlot(
  have
, groupingVar
, analysisVar
, title = %nrstr(title1 J=C HEIGHT=2 "The Rain Cloud (without BOX) Plot";
                 title2 J=C HEIGHT=1 "Plotting for groups: &list_g.";)
, footnote = %str(footnote1 J=R H=1 C=Gray
                 "Data set HAVE, with 87 observations, two variables.";)
, colorslist = CornflowerBlue OliveDrab IndianRed
, raindropsize = 8px
, kernel_k = quadratic
, kernel_c = 0.99
, vscale = percent
, vscalemax = 60
, BOXPLOT = 1
, BOXPLOTFILL = 0.5
, BOXPLOTLINESIZE = 2px
, BOXPLOTSYMBOLSIZE = 12px 8px
)
```

Results are displayed in Figure 8

MAKING AXIS LOOK NICE

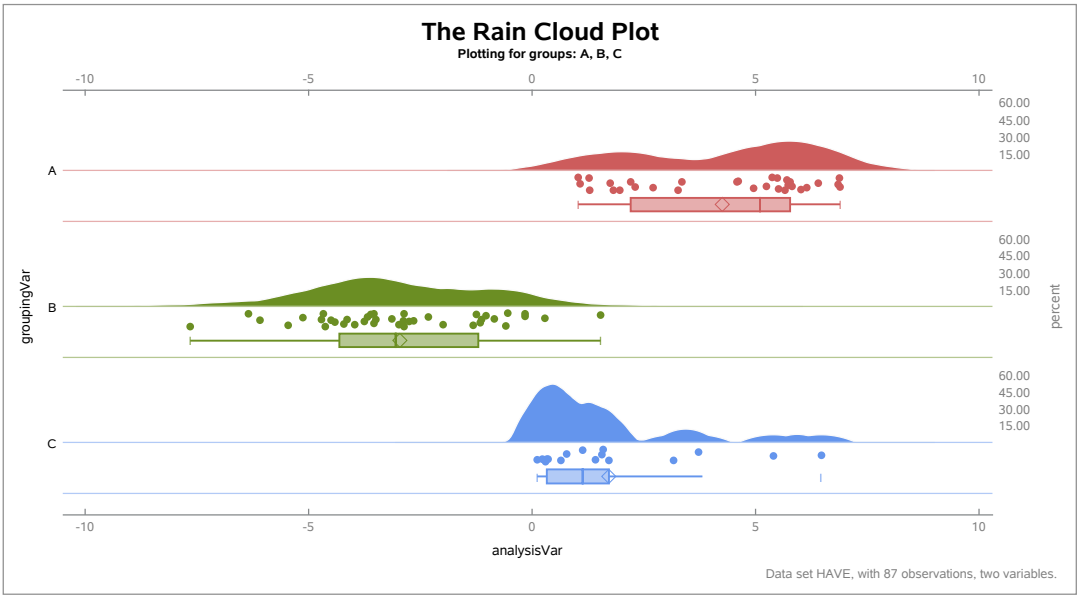
There are ways how the horizontal and two vertical axis look can be polished. There are three groups of options to do it.

For the horizontal axis:

- `xLabels=` When empty (by default) a data variable name is used as label. List of values for labels should be a quoted comma-separated lists enclosed with brackets, e.g., `xLabels=("Analyzed variable")`.

[5] Outliers have values below $Q1{-}1.5IQR$ or values above $Q3{+}1.5IQR$

Figure 8: Rain Cloud Plot with modified box-and-whiskers plot



- `xLabelPos=` indicates position of the label on the axis, default value is `DATACENTER`. Allowed values are `LEFT`, `CENTER`, `DATACENTER`, and `RIGHT`.
- `xLabelAttrs=` by default its value is empty. Contains a list of attributes for axis labels. The list should be a space-separated lists of `key=value` pairs, e.g., `xLabelAttrs=size=12 color=Orange weight=bold`. In case of doubts about what values are allowed see the `XAXIS` statement documentation for the `SGPLOT` procedure.
- `xaxisValueAttrs=` allows to modify axis values attributes, default value is `Color=Grey`. All notes about the `xLabelAttrs=` apply.
- `xaxisTickstyle=` allows to modify axis tick style, default value is `INSIDE`. Allowed values are `OUTSIDE`, `INSIDE`, `ACROSS`, and `INBETWEEN`. *For SAS prior to 9.4M5, should be set to missing!*
- `xBothAxis=` indicates if both horizontal axis should be displayed, default value of `1` tells to show both the bottom and the top one.
- `xaxisValues=`, `xaxisValuesDisplay=`, `xaxisValuesFormat=`[6], and `xaxisValuesRotate=` behave exactly as the `SGPLOT` procedure parameters `Values=`, `ValuesDisplay=`, `ValuesFormat=`, and `ValuesRotate=`. By default values of `xaxisValues...` are missing.
- `xaxisOther=` parameter allows to provide other modifications to horizontal line, such as: add minor thick marks (`MINOR`), add grid lines and their style (`GRID`, `MINORGRID`, `MINORGRIDATTRS=`, and `GRIDATTRS=`, etc.).

For left vertical axis, the one with categories:

- `catLabels=` When empty (by default) a data variable name is used as label. List of values for labels should be a quoted comma-separated lists enclosed with brackets, e.g., `catLabels=("Grouping variable")`.
- `catLabelPos=` indicates position of the label on the axis, default value is `DATACENTER`. Allowed values are `BOTTOM`, `CENTER`, `DATACENTER`, and `TOP`.
- `catLabelAttrs=` by default its value is empty. Contains a list of attributes for axis labels. The list should be a space-separated lists of `key=value` pairs, e.g., `catLabelAttrs=size=10 color=Black weight=normal`. In case of doubts about what values are allowed see the `YAXIS` statement documentation for the `SGPLOT` procedure.

[6] An interesting side note, SAS documentation says that instead using `w.d` format we should use `Fw.d` format

- `catAxisValueAttrs=` allows to modify axis values attributes, default value is `Color=Black`. All notes about the `catLabelAttrs=` apply.
- `formated=` indicates if values of the grouping variable should be formatted, default value of `0` tells to take raw values.

For right vertical axis, the one with scales:

- `y2axis=` indicates if the right vertical axis should be displayed, default value of `1` tells to print values.
- `y2axisLevels=` sets the number of expected levels of values printed on the right vertical axis, default value is `4`.
- `y2axisValueAttrs=` allows to modify right vertical axis values attributes, default value is `Color=Grey`.
- `y2axisFormat=` allows to modify right vertical axis values format, default value `12.2-L`.
- `y2axisLines=` if set to `1`, adds horizontal lines for right vertical axis values, default value `0`.

The following program makes use of all the listed parameters:

code: alternating axis

```
/* format... */
proc format;
value $ FormatForGroup
  "A" = "Uniform[1, 7]"
  "B" = "Normal(-4, 2)"
  "C" = "Exponential(2)"
  other = "Unknown!"
  ;
picture myPercent
  0-100 = "099.9%"
  other = "Wrong value"
  ;
run;
proc datasets lib=work nolist;
  modify have;
    format groupingVar $FormatForGroup.;
  run;
quit;

/* style... */
%let labelStyle = size=10 family="Courier New" color=MidnightBlue weight=bold;
%let valuesStyle = size=8 family="Courier New" style=italic;

/* ...and plot */
%rainCloudPlot(
  have
, groupingVar
, analysisVar
, title = %nrstr(title1 J=C HEIGHT=2 "The Rain Cloud (without BOX) Plot";
                 title2 J=C HEIGHT=1 "Plotting for groups: &list_g.";)
, footnote = %str(footnote1 J=R H=1 C=Gray
                 "Data set HAVE, with 87 observations, two variables.";)
, colorslist = CornflowerBlue OliveDrab IndianRed
, raindropsize = 8px
```

```
, kernel_k = quadratic
, kernel_c = 0.99
, vscale = percent
, vscalemax = 60
, boxplot=1
, boxplotfill = 0.5
, boxplotlinesize = 2px
, boxplotsymbolsize = 12px 8px

, XLABELS = ("Analyzed variable")
, XLABELPOS = CENTER
, XLABELATTRS = &labelStyle.
, XAXISVALUEATTRS = &valuesStyle.
, XAXISTICKSTYLE = ACROSS
, XBOTHAXIS = 0
, XAXISVALUES = (-10 to 10 by 2)
, XAXISVALUESFORMAT = F5.2
, XAXISOTHER = grid gridattrs=(thickness=0)
               minor minorgrid minorgridattrs=(color=cxefefef pattern=dot)

, CATLABELS = ("Grouping variable with formated values")
, CATLABELPOS = TOP
, CATLABELATTRS = &labelStyle.
, CATAXISVALUEATTRS = &valuesStyle.
, FORMATED = 1

, Y2AXIS = 1
, Y2AXISLEVELS = 5
, Y2AXISVALUEATTRS = &valuesStyle.
, Y2AXISFORMAT = myPercent.
, Y2AXISLINES = 1
)
```

Figure 9 presents the result.

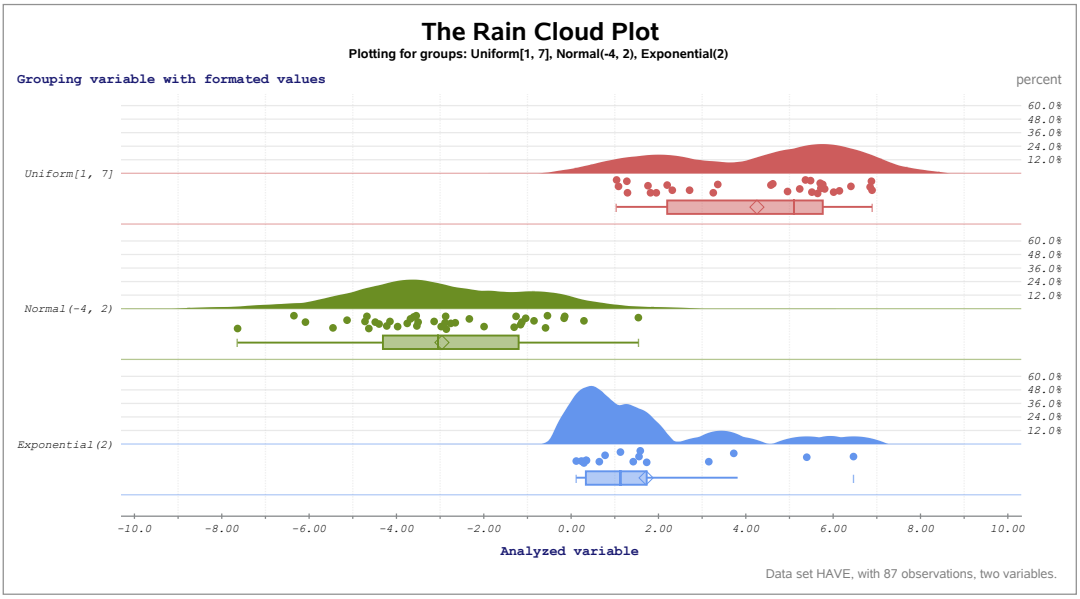
DIMENSIONS

The macro is design to set dimensions of the plot area as `1200` pixels in width and `220 × the number of groups` pixels in height, so `220px×3 = 660px` in our case. Dimensions can be altered with the help of `WidthPX=` and `HeightPX=` parameters, their values should be positive integers.

Data dimensions can be altered too. Vertical data axis (with categories) can be modified by already mentioned `vscalemax=` parameter and by changing number of groups in data. Horizontal data axis can be altered by setting `minRange=` and `maxRange=` values. By default their values are numerical missing, i.e., the dot (`.`), which means that both minimum and maximum are calculated from the data (kernel density estimate usually extends data range).

The following snippet modifies both plot dimensions and displayed data range. Since this code is the same as the previous code example just extended by new parameters, for brevity, "repeated" part of is skipped. Check attached code file to see the program in full.

Figure 9: Rain Cloud Plot with modified axis



code: changing dimensions

```
%rainCloudPlot(
  have
, groupingVar
, analysisVar
  ...
, WIDTHPX = 1200
, HEIGHTPX = 300
, MINRANGE = -8
, MAXRANGE =  8
)
```

Figure 10 has now size of 1200 by 900 pixels (notice different aspect ratio than Figure 9) and horizontal axis data values range is cut to range between -8 and 8.
[**NOTE:**] After the macro executes the ODS Graphics width and height are reset to 800 by 600 pixels.

TECHNICAL PARAMETERS

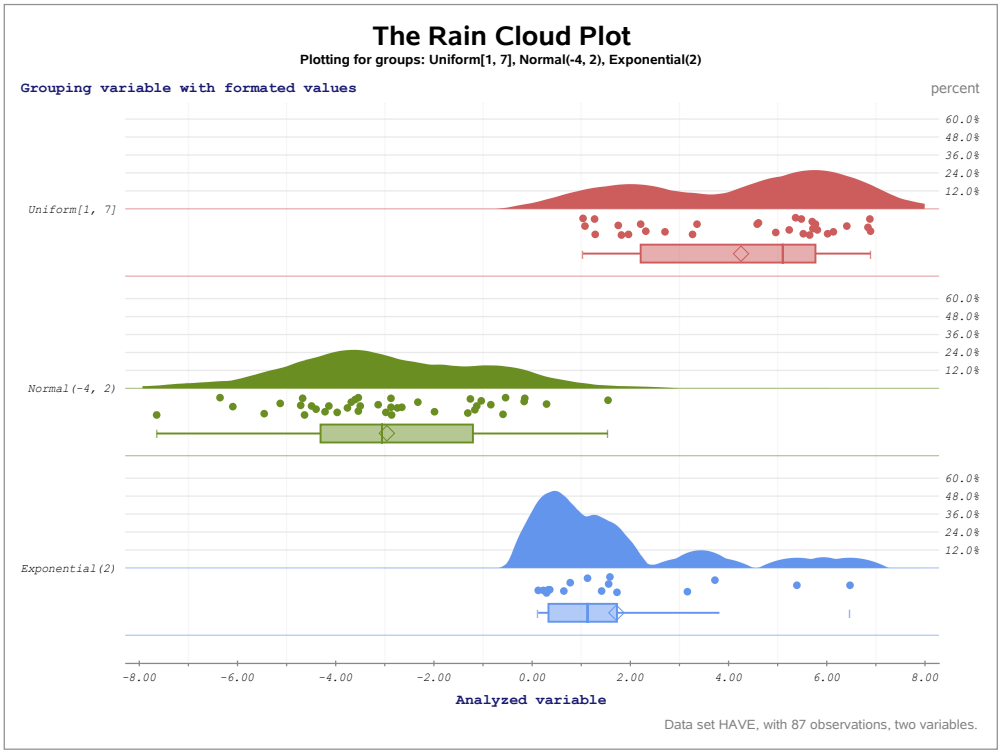
The macro has a few technical parameters which are intended to modify the SGPLOT procedure options, the ODS Graphics options, to do some house-keeping, and display source code generating the plot.

The first one, sgPlotOptions=, allows to add options to the PROC SGPLOT statement. As the default the noautolegend, the noborder, and the subpixel are set.

The next one, odsGraphicsOptions=, allows to add options to the ODS Graphics statement. Three options are always used, the Width=, the Height=, and the AntiAliasMAX= which is calculated based on the number of observations in the plot data set.

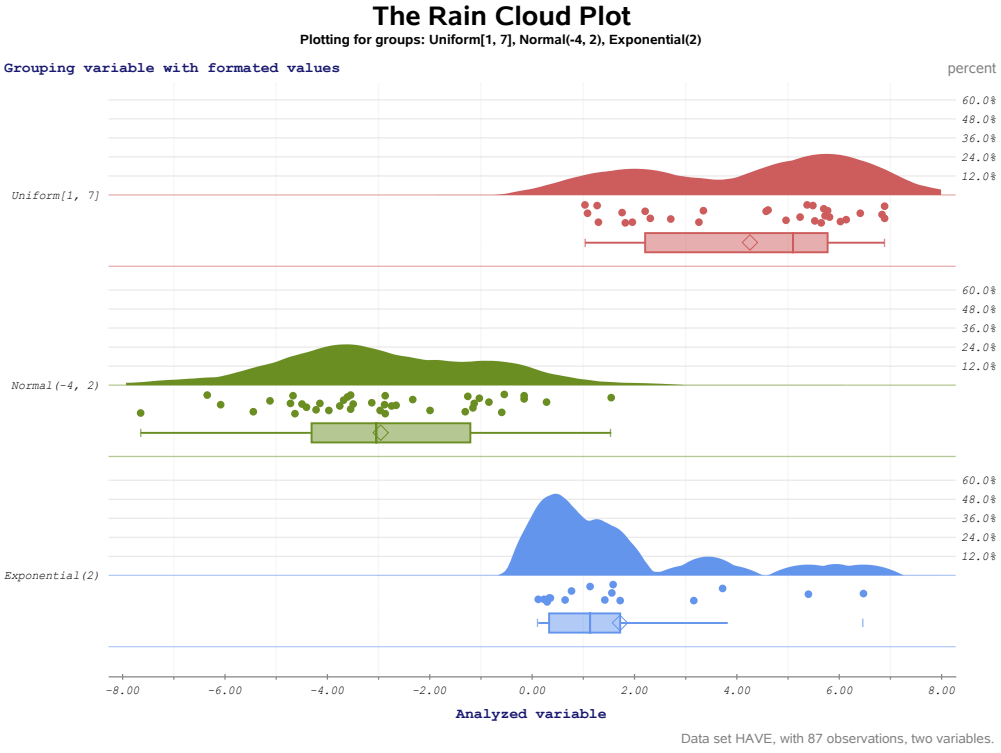
The last two are cleanTempData= that indicates if temporary data should be deleted (yes by default) and codePreview that indicates if the source code should be MPRINTed to the log (no by default).

Figure 10: Rain Cloud Plot with modified size and displayed data range



The rain falls wherever it wants to fall, no borders can stop it. In Figure 11 we can see how the `odsGraphicsOptions = NOBORDER` and `sgPlotOptions = NOAUTOLEGEND NOBORDER NOWALL PAD=0` alters the plot. It may not be visible at first sight but the `PAD=0` expands the plot a bit (look at the distance between the title and top rule, or see a PNG file generated).

Figure 11: Rain Cloud Plot without borders and with zero padding



ADDING SOMETHING EXTRA

From time to time even the most robust plot is not enough, in such situations the annotations can help. The `sganno=` parameter expects a name of an annotations data set. It may be really helpful, if we for example want to highlight that out plot is not done yet, like in this snippet:

code: annotating the plot

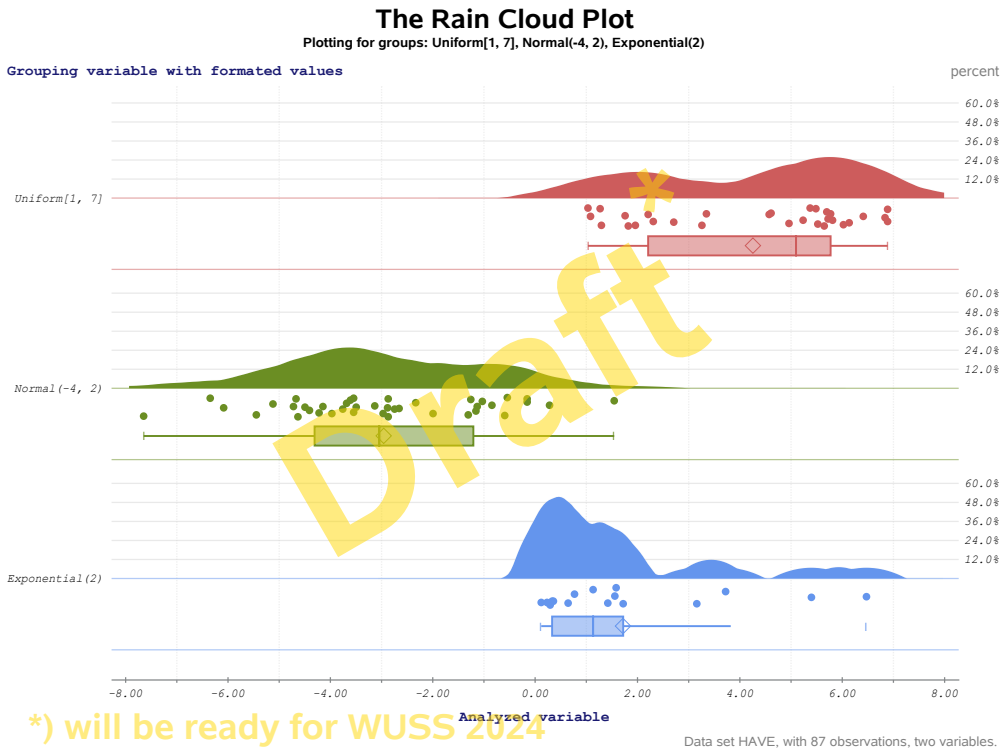
```
/* annotation data set */
data WORK.IN_PROGRESS;
  function="TEXT"; transparency=0.5; width=500; textcolor="GOLD";
  textweight="BOLD"; widthunit="PERCENT"; drawspace="GRAPHPERCENT";

  label="*) will be ready for WUSS 2024";
  textsize=24; anchor="LEFT  "; x1=2; y1=3; rotate=0;
  output;
  label="Draft(*ESC*)sup '*'";
  textsize=128; anchor="CENTER"; x1=50; y1=50; rotate=30;
  output;
run;

%rainCloudPlot(
  have
, groupingVar
, analysisVar
  ...
, SGANNO =  WORK.IN_PROGRESS
)
```

The `WORK.IN_PROGRESS` data set helps us to indicate that Figure 12 is not "finished" yet!

Figure 12: Rain Cloud Plot with annotations



CHANGES IN MEAN

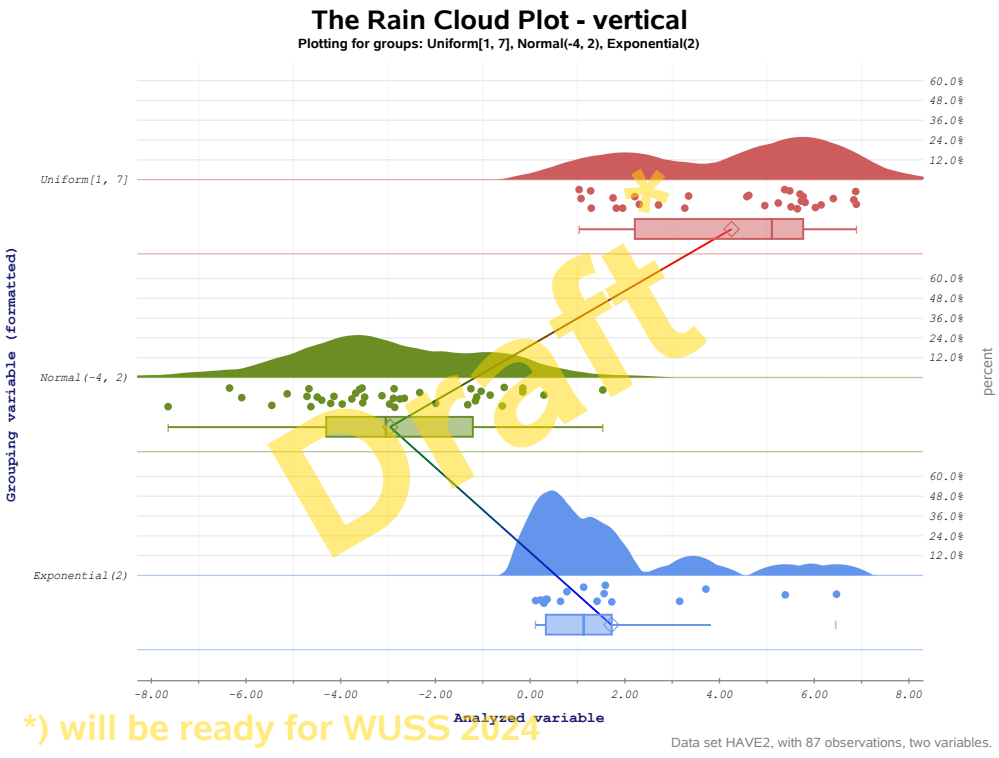
If we want to highlight changes of mean in groups we can use the `meanShiftLine=` parameter. It forces the macro to print line connecting mean points of all box plots. By default the parameter is set to zero, what means the line will not be displayed. Additional counterpart parameters have the following meaning: the `meanShiftStep=` is responsible for how smoothly line color changes; the `meanShiftColors=` parameter provides a list of colors to be used for the line gradient, by default it mimics the value of `colorsList=` parameter.

code: changes in mean highlighted

```
/* annotation data set */
%rainCloudPlot(
  have
, groupingVar
, analysisVar
  ...
, MEANSHIFTLINE = 1
, MEANSHIFTSTEP = 0.1
, MEANSHIFTCOLORS = Blue Green Red
)
```

The effect of how those parameters affect the plot can be seen in Figure 13.

Figure 13: Vertical Rain Cloud with mean shift lines



MULTIPLE GROUPS OR ANALYSIS

Sometimes we have multiple grouping variables against which we want to plot the data, or multiple analysis variables. In such situations all we need to do is to list them, space separated, in the second or third positional parameter of the macro, for example assume we have the following data set:

code: new data

```
data have2;
  set have;
  groupingVar2 = ifc(0=mod(_N_,2),"Even","Odd");
  analysisVar2 = rannor(42);
run;
```

All we need to do is to run the macro with variables listed. If we are using multiple variables the list of values for labels has to be adjusted, *important* to remember is that they *have to be comma separated*! Of course if the analysis variables have different scales setting `minRange=` and `maxRange=` to missing is a good idea.

For example with the following code:

code: two groups and two analysis

```
%rainCloudPlot(
  have2
, groupingVar groupingVar2
, analysisVar analysisVar2
  ...
, xlabels = ("First analyzed variable", "Second analyzed variable")
, catlabels = ("Grouping variable one", "Grouping variable two")
  ...
, minRange = .
, maxRange = .
)
```

we will get four plots, i.e., a Cartesian product of all grouping variables by all analysis variables, just like in Figure 14.

THE CHERRY ON TOP

As the cherry on top we discuss one more parameter that is available.

Ok, so we have our plot precisely polished, every aspect of it is perfect, and then, when it is a really windy day, our boss comes and wants the rain to fall horizontally, i.e., plot has to be oriented vertically... The parameter to save us is `vertical=` and it is a binary indicator telling the macro should the plot be rotated 90 degrees or not.

code: two groups and two analysis

```
%rainCloudPlot(
  have2
, groupingVar
, analysisVar
  ...
, VERTICAL = 1
)
```

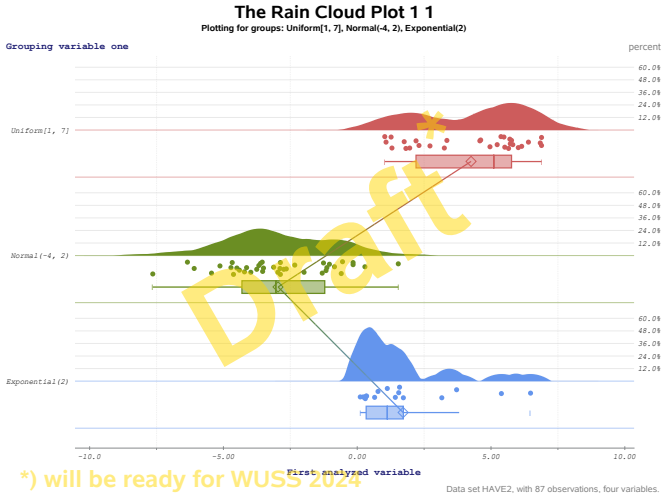
The behavior is depicted in Figure 15.

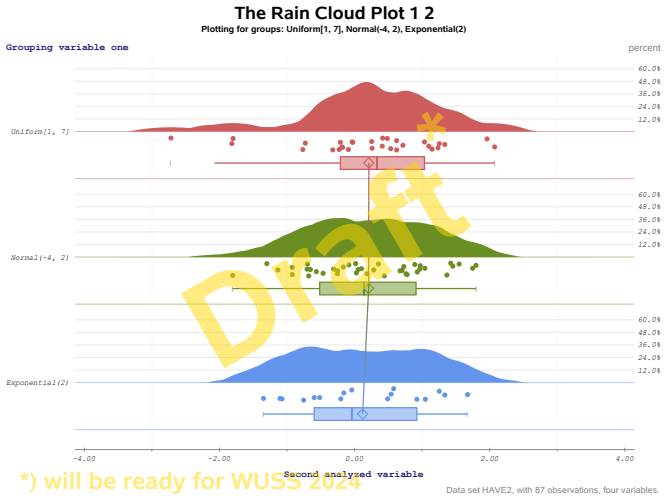
CONCLUSIONS

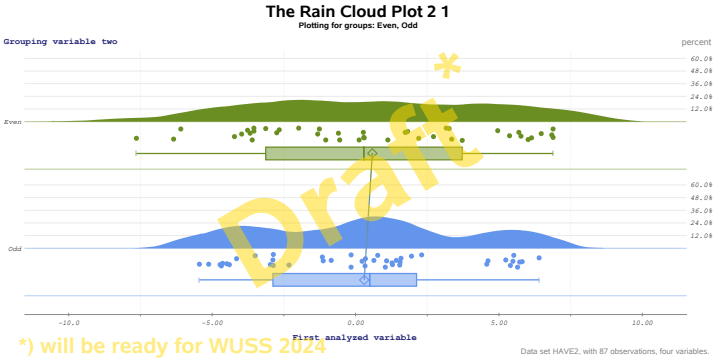
Rain Cloud Plots are very practical tools for visual data analysis. With help of `%RainCloudPlot()` macro they are now very easily "accessible" also in SAS.

The End

Figure 14: Rain Cloud Plot with two groups by two analysis variables







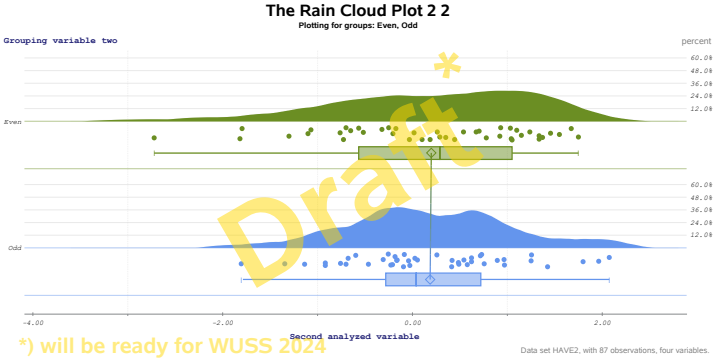
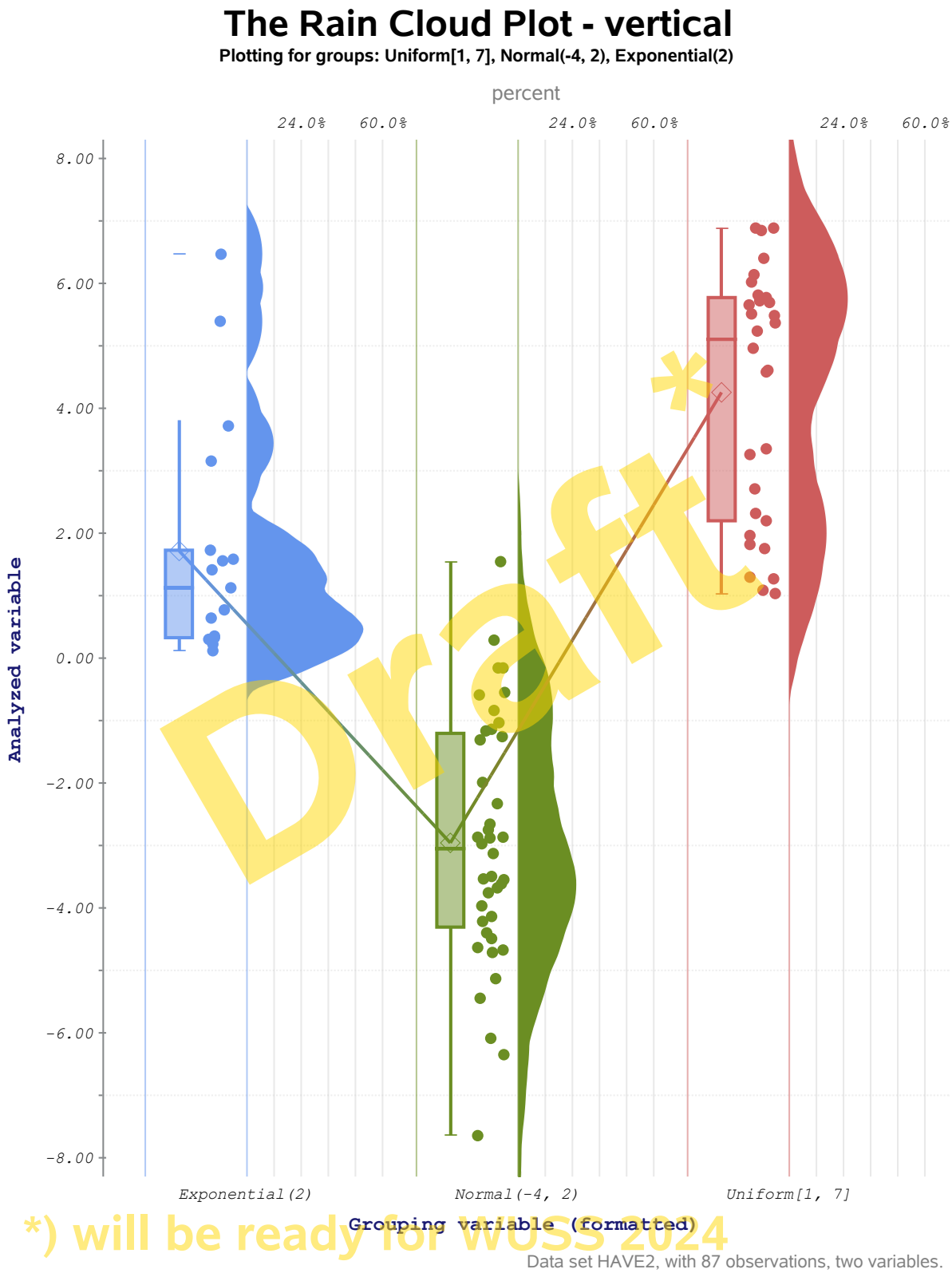


Figure 15: Vertical Rain Cloud Plot



REFERENCES

[Allen *et al.* 2019] Micah Allen, Davide Poggiali, Kirstie Whitaker, Tom Rhys Marshall, Rogier A. Kievit,
"Raincloud plots: a multi-platform tool for robust data visualization",
Wellcome Open Research (https://wellcomeopenresearch.org/articles/4-63/v2), 2019,
First Version Published: 01 Apr 2019, 4:63 (https://doi.org/10.12688/wellcomeopenres.15191.1)
Latest Version Published: 21 Jan 2021, 4:63 (https://doi.org/10.12688/wellcomeopenres.15191.2)
GITHUB: https://github.com/RainCloudPlots/RainCloudPlots

[Jablonski 2020] Bartosz Jabłoński, "SAS Packages: The Way to Share (a How To)", SGF Proceedings, 2020
https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2020/4725-2020.pdf
extended version available at: https://github.com/yabwon/SAS_PACKAGES/blob/main/SPF/Documentation

[Jablonski 2021(1)] Bartosz Jabłoński, "Fun with SAS ODS Graphics: Rain Cloud Plot", SAS communities post, 2021,
https://communities.sas.com/t5/Graphics-Programming/Fun-with-SAS-ODS-Graphics-Rain-Cloud-Plot/td-p/771681

[Jablonski 2021(2)] Bartosz Jabłoński, "My First SAS Package - a How To", SGF Proceedings, 2021
https://communities.sas.com/kntur85557/attachments/kntur85557/proceedings-2021/59/1/Paper_1079-2021.pdf
also available at: https://github.com/yabwon/SAS_PACKAGES/tree/main/SPF/Documentation/Paper_1079-2021

[Tsutsugo 2022] Kosuke Tsutsugo, "SAS plotter", Jul .3, 2022
GITHUB: https://github.com/Superman-jp/SAS_Plotter
Documentation: https://superman-jp.github.io/SAS_Plotter/index.html
Web site (in Japanese): https://picolabs.jp

[Morioka 2022] Yutaka Morioka, "Implementation of Raincloud Plot with SAS and its USE for Clinical Trial Data",
PharmaSUG Japan, 2022, https://www.pharmasug.org/proceedings/japan2022/PharmaSUG-Japan-2022-04.pdf

[Jablonski 2023] Bartosz Jabłoński, "Share your code with SAS Packages a Hands-on-Workshop",
WUSS 2023 Proceedings, 2023, https://www.lexjansen.com/wuss/2023/WUSS-2023-Paper-208.pdf

[Wicklin 2024] Rick Wicklin, "Scale a density curve to match a histogram", Blog post at blogs.sas.com, June 2024,
https://blogs.sas.com/content/iml/2024/06/19/scale-density-curve-histogram.html

CODE

For your convenience all code snippets and a copy of the article were also collected in one place. Visit:

https://github.com/SASPAC/baseplus/tree/main/extras/

to download them if you wish.

ACKNOWLEDGMENTS

The author would like to acknowledge: Sanjay Matange, for all indirect help in his outstanding "Graphically Speaking" blog; Rick Wicklin, for statistical "hints"; Filip Kulon, for "linguistic polishing" that made this paper look and feel as it should!

CONTACT INFORMATION

Your comments and questions are valued and encouraged!
Contact Bart at one of the following e-mail addresses:

yabwon✉gmail.com or bartosz.jablonski✉pw.edu.pl

or via the following LinkedIn profile: www.linkedin.com/in/yabwon or at the communities.sas.com by mentioning @yabwon.

Appendix A - code coloring guide

The best experience for reading this article is in color and the following convention is used:

- The code snippets use the following coloring convention:

code: is surrounded by a black frame

```
In general we use black ink for the code but:
- code of interest is in orange ink so that it can be highlighted,
- and comments pertaining to code are in a bluish ink for easier reading.
```

- The LOG uses the following coloring convention:

the log - is surrounded by a blueish frame

```
The source code and general log text are blueish.
Log NOTEs are green.
Log WARNINGs are violet.
Log ERRORs are red.
Log text generated by the user is purple.
```

Appendix B - install the SAS Packages Framework and packages

To install the SAS Packages Framework and a SAS Package we execute the following steps:

- First we create a directory to install SPF and Packages, for example: `/home/user/packages` or `C:/packages`.
- Next, depending if the SAS session has access to the internet:
  - if it does - we run the following code:

code: install from the internet

```
filename packages "/home/user/packages";

filename SPFinit url
 "https://raw.githubusercontent.com/yabwon/SAS_PACKAGES/main/SPF/SPFinit.sas";
%include SPFinit;

%installPackage(SPFinit)
%installPackage(packageNameYouWant)
```

  - If the SAS session does not have access to the internet we go to the framework repository:

    https://github.com/yabwon/SAS_PACKAGES

    next (if not already) we click the stargazer button [★] ;-) and then we navigate to the SPF directory and we copy the SPFinit.sas file into the directory from step one (direct link: https://raw.githubusercontent.com/yabwon/SAS_PACKAGES/main/SPF/SPFinit.sas). And for packages - we just copy the package zip file into the directory from step one.
- From now on, in all subsequent SAS session, it is enough to just run:

code: enable framework and load packages

```
filename packages "/home/user/packages";
%include packages(SPFinit.sas);
%loadPackage(packageNameYouWant)
```

to enable the framework and load packages. To update the framework or a package to the latest version we simply run:

code: update from the internet

```
%installPackage(SPFinit packageYouWant1 packageYouWant2 packageYouWant3)
```

See [**Jablonski 2020**], [**Jablonski 2021(2)**], and [**Jablonski 2023**] for details.

Appendix C - safety considerations

The SPF installation process, in a "nutshell", reduces to copying the `SPFinit.sas` file into the packages directory. It is the same for a packages too.

You may ask: *is it safe to install*?

Yes, it's safe! When you install the SAS Packages Framework, and later when you install packages, the files are simply copied into the packages directory that you configured above. There are no changes made to your SAS configuration files, or autoexec, or registry, or anything else that could somehow "break SAS." As you saw, you can perform a manual installation simply by copying the files yourself. Furthermore the SAS Packages Framework is:

- written in 100% SAS code, it does not require any additional external software to work,
- full open source (and MIT licensed), so every macro can be inspected.

When we work with a package, before we even start thinking about loading content of one into the SAS session, both the help information and the source code preview are available.

To *read help information* (printed in the log) you simply run:

code: get help info

```
%helpPackage(<packageName>, <*|componentName|license>)
```

To **preview source code of package components** (also printed in the log) you simply run:

code: get code preview

```
%previewPackage(<packageName>, <*|componentName>)
```

The asterisk means "print everything", the `componentName` is the name of a macro, or a function, or a format, etc. you want see.