SOFTWARE TOOL ARTICLE

Raincloud plots: a multi-platform tool for robust data visualization [version 1; peer review: 2 approved]

Micah Allen [1-3], Davide Poggiali [4,5], Kirstie Whitaker [2,6], Tom Rhys Marshall[7,8], Rogier A. Kievit [9,10]

[1]Aarhus Institute of Advanced Studies, Aarhus University, Aarhus, Denmark
[2]Department of Psychiatry, University of Cambridge, Cambridge, UK
[3]Centre of Functionally Integrative Neuroscience, Aarhus University Hospital, Aarhus, Denmark
[4]Department of Mathematics, University of Padova, Padova, Italy
[5]Padova Neuroscience Center, University of Padova, Padova, Italy
[6]Alan Turing Institute, London, UK
[7]Wellcome Centre for Integrative Neuroimaging, University of Oxford, Oxford, UK
[8]Department of Experimental Psychology, University of Oxford, Oxford, UK
[9]MRC Cognition and Brain Sciences Unit, University of Cambridge, Cambridge, UK
[10]Max-Planck Centre for Computational Psychiatry and Aging, UCL/MPI Berlin, London, UK

v1 **First published:** 01 Apr 2019, **4**:63 (https://doi.org/10.12688/wellcomeopenres.15191.1)
Latest published: 01 Apr 2019, **4**:63 (https://doi.org/10.12688/wellcomeopenres.15191.1)

Abstract

Across scientific disciplines, there is a rapidly growing recognition of the need for more statistically robust, transparent approaches to data visualization. Complementary to this, many scientists have called for plotting tools that accurately and transparently convey key aspects of statistical effects and raw data with minimal distortion. Previously common approaches, such as plotting conditional mean or median barplots together with error-bars have been criticized for distorting effect size, hiding underlying patterns in the raw data, and obscuring the assumptions upon which the most commonly used statistical tests are based. Here we describe a data visualization approach which overcomes these issues, providing maximal statistical information while preserving the desired 'inference at a glance' nature of barplots and other similar visualization devices. These "raincloud plots" can visualize raw data, probability density, and key summary statistics such as median, mean, and relevant confidence intervals in an appealing and flexible format with minimal redundancy. In this tutorial paper, we provide basic demonstrations of the strength of raincloud plots and similar approaches, outline potential modifications for their optimal use, and provide open-source code for their streamlined implementation in R, Python and Matlab (https://github.com/RainCloudPlots/RainCloudPlots). Readers can investigate the R and Python tutorials interactively in the browser using Binder by Project Jupyter.

Keywords

data visualization, raincloud plots, R, Python, Matlab, barplots

Open Peer Review

Reviewer Status ✓ ✓

	Invited Reviewers	
	1	**2**
version 1 published 01 Apr 2019	✓ report	✓ report

1 **Lisa M. DeBruine**, University of Glasgow, Glasgow, UK

2 **Elena Allen**, Rodin Scientific, LLC, Albuquerque, USA

Any reports and responses or comments on the article can be found at the end of the article.

Corresponding author: Micah Allen (micah.allen@medschl.cam.ac.uk)

Author roles: Allen M: Conceptualization, Methodology, Project Administration, Resources, Software, Supervision, Validation, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing; **Poggiali D**: Conceptualization, Methodology, Resources, Software, Validation, Visualization, Writing – Review & Editing; **Whitaker K**: Conceptualization, Methodology, Resources, Software, Validation, Writing – Review & Editing; **Marshall TR**: Conceptualization, Methodology, Resources, Software, Validation, Visualization, Writing – Review & Editing; **Kievit RA**: Conceptualization, Funding Acquisition, Project Administration, Software, Supervision, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing

Competing interests: No competing interests were disclosed.

Grant information: MA is supported by a Lundbeckfonden Fellowship (R272-2017-4345), the AIAS-COFUND II fellowship programme that is supported by the Marie Skłodowska-Curie actions under the European Union's Horizon 2020 (Grant agreement no 754513), and the Aarhus University Research Foundation, and thanks Lincoln Colling for insightful statistical discussions. KW is funded by the Alan Turing Institute under the EPSRC grant EP/N510129/1. RAK is supported by the Wellcome Trust (grant number 107392/Z/15/Z).
The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

How to cite this article: Allen M, Poggiali D, Whitaker K *et al.* **Raincloud plots: a multi-platform tool for robust data visualization [version 1; peer review: 2 approved]** Wellcome Open Research 2019, **4**:63 (https://doi.org/10.12688/wellcomeopenres.15191.1)

First published: 01 Apr 2019, **4**:63 (https://doi.org/10.12688/wellcomeopenres.15191.1)

Introduction

Effective data visualization is key to the interpretation and communication of data analysis. Ideally a statistical plot or data graphic should balance functionality, interpretability, and complexity, all without needlessly sacrificing aesthetics. That is to say, the perfect visualization is one which uses as little 'ink' as possible to capture exactly the desired statistical inference in an intuitive and appealing format (Tufte, 1983). As concerns regarding the need for robust, reproducible data science have grown in recent years, so too have calls for more meaningful approaches to plotting one's data. Here we present an open source, multi-platform tutorial for the *raincloud plot* (Neuroconscience, 2018a).

A common visualization method of raw datapoints is the barplot (see Figure 1, left panel) to represent the mean or median of some condition or group via horizontal bars (or lines) and represents uncertainty about the illustrated parameter estimated via 'whisker' errorbars, usually conveying the standard error or 95% confidence interval. This approach has been widely criticized on several counts, including: 1) it is prone to distortion (e.g., by cropping of the Y-axis), 2) it fails to represent the actual data underlying relevant parameter inferences, 3) it often leads to misleading inferences about the magnitudes of statistical differences between conditions (Weissgerber *et al.*, 2015) and 4) it may obscure differences in distributions (and concurrent violations of distributional assumptions in parametric statistics). These limitations are illustrated in Figure 1, below. Indeed, criticism of this approach has reached such a pitched fervor that a movement to "bar bar plots" ("#barbarplots," 2016; Piccinini, 2016) has arisen with many signees pledging to request all such plots be changed to something more informative[1].

To remedy these shortcomings, a variety of visualisation approaches have been proposed, illustrated in Figure 2, below. One simple improvement is to overlay individual observations (datapoints) beside the standard bar-plot format, typically with some degree of randomized jitter to improve visibility (Figure 2A). Complementary to this approach, others have advocated for more statistically robust illustrations such as boxplots (Tukey, 1970), which display sample median alongside interquartile range. Dot plots can be used to combine a histogram-like display of distribution with individual data observations (Figure 2B). In many cases, particularly when parametric statistics are used, it is desirable to plot the distribution of observations. This can reveal valuable information about how e.g., some condition may increase the skewness or overall shape of a distribution. In this case, the 'violin plot' (Figure 2C) which displays a probability density function of the data mirrored about the uninformative axis is often preferred (Hintze & Nelson, 1998). With the advent of increasingly flexible and modular plotting tools such as ggplot2 (Wickham, 2010; Wickham & Chang, 2008), all of the aforementioned techniques can be combined in a complementary fashion.

Indeed, this combined approach is typically desirable as each of these visualization techniques have various trade-offs. Simply plotting raw data can reveal valuable information about individual differences, outliers, and

[1] This raises the question of why such uninformative plots became widespread in the first place. Speculatively, they may simply have been easier to produce before the advent of personal computers and associated statistical software, when plots were typically hand-drawn. Manual plotting of this type was time consuming and error-prone; simply plotting all raw data points would have considerably increased workload and the full-scale plotting of probability distributions may have been beyond the grasp of many researchers.

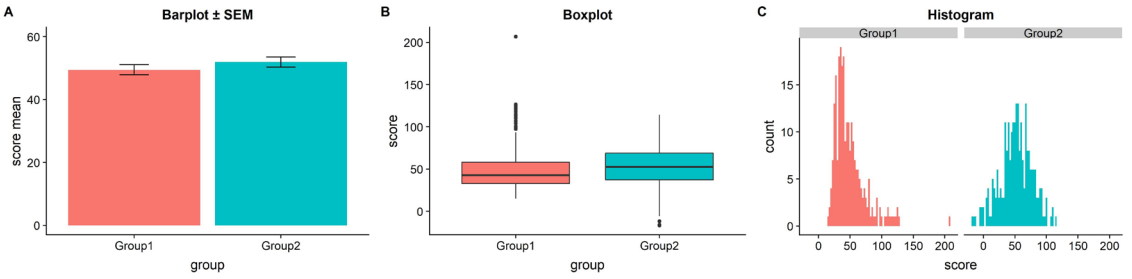


Figure 1. The trouble with barplots. Example reproduced from "Boxplots vs. Barplots" (2016) two simulated datasets with mean = 50, sd = 25, and 1000 observations. **A**) a barplot and errorbars representing +/- standard error of the mean gives the impression that the measure is equivalent between the two groups. In fact, group 1 is drawn from an exponential distribution as seen in **B**) boxplots, and **C**) histograms. The barplot not only obscures the underlying nature of the observations, but also hides the fact that these data are not appropriate for standard parametric inference. See figure1.Rmd for code to generate these figures.

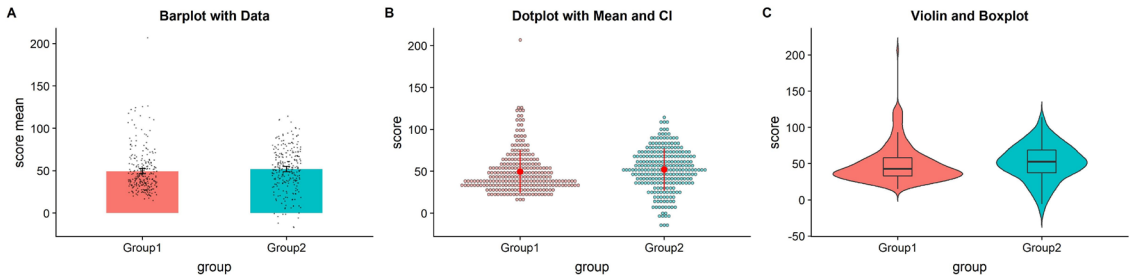


Figure 2. Extant approaches to improved data plotting. A) The simplest improvement is to add jittered raw data points to the standard boxplot and +/- standard error scheme. **B**) Alternatively, dotplots can be used to supplement visualizations of central tendency and error, at the risk of added complexity due to the dependence of such plots on choices such as bin-width and dot size. **C**) A popular recent alternative is the violin plot coupled with boxplots or similar. However, this needlessly mirrors information about the redundant data axis (here, the x-axis). See figure2.Rmd for code to generate these figures.

unexpected patterns within the data. However, human observers are notoriously poor[2] at estimating statistical moments and distributions from raw data (Bobko & Karren, 1979; "Guess the Correlation," 2017; Spence *et al.*, 2016; Zylberberg *et al.*, 2014), and the utility of such plots can be limited when the number of observations is large. In this case the dotplot may be advantageous, as it displays both a histogram of raw data points and the frequency of different binned observations. On the other hand, the interpretation of dotplots depends heavily on the choice of dot-bin and dot-size, and these plots can also become extremely difficult to read when there are many observations. The violin plot in which the probability density function (PDF) of observations are mirrored, combined with overlaid boxplots, have recently become a popular alternative. This provides both an assessment of the data distribution and statistical inference at a glance (SIG) via overlaid boxplots[3]. However, there is nothing to be gained, statistically speaking, by mirroring the PDF in the violin plot, and therefore they are violating the philosophy of minimising the "data-ink ratio" (Tufte, 1983)[4].

To overcome these issues, we propose the use of the 'raincloud plot' (Neuroconscience, 2018a), illustrated in Figure 3. The raincloud plot combines a wide range of visualization suggestions, and similar precursors have been used in various publications (e.g., Ellison, 1993, Figure 2.4; Wilson *et al.*, 2018). The plot attempts to address the aforementioned limitations in an intuitive, modular, and statistically robust format. In essence, raincloud plots combine a 'split-half violin' (an un-mirrored PDF plotted against the redundant data axis), raw jittered data points, and a standard visualization of central tendency (i.e., mean or median) and error, such as a boxplot. As such the raincloud plot builds on code elements from multiple developers and scientific programming languages (Hintze & Nelson, 1998; Patil, 2018; Wickham & Chang, 2008; Wilke, 2017).

Many previous attempts have been made to produce more robust, intuitive, and transparent plots. Our goal here is not to propose a totally novel invention, but rather to make a powerful visualization strategy freely, easily, and transparently available across commonly used platforms. To this end, similar but distinct plotting strategies include beanplots (Kampstra, 2008), estimation plots (Ho *et al.*, 2018), pirateplots (Phillips, 2016), sinaplots (Sidiropoulos *et al.*, 2018), stripcharts (Chambers, 2017), beeswarm plots (Eklund, 2016), and many others. Our hope here is to offer a cross-platform, open science tool which builds upon these approaches and makes robust and transparent data-plotting available to as wide an audience as possible.

Inference-at-a-glance is supported by adding whatever flavor of data summary measure is optimal for the data at hand; typical examples include overlaid boxplots or other illustrations of central tendency such as mean/median and associated confidence intervals. Depending on the analysis at hand, PDF illustration can also be replaced with more advanced options such as posterior probability densities (i.e., as derived from Bayesian inference) or other parameter estimates (Ho *et al.*, 2018).

Thus, raincloud plots offer the user maximum utility and flexibility, ensuring that nothing is 'hidden away' and that the reader has all information needed to assess the data, its distribution, and the appropriateness of any reported

[2] Indeed, try it yourself at http://guessthecorrelation.com/

[3] See http://www.fharrell.com/post/interactive-graphics-less/ for an interactive demonstration of how raincloud-like plots can aid minimal yet powerful inference.

[4] Moreover, for some violin plots are, shall we say - overly provocative ("xkcd: Violin Plots," n.d.).

statistical tests in a visually appealing format. Indeed, as illustrated in Figure 4, raincloud plots can reveal information that even a boxplot plus raw data might hide away, such as a bimodal distribution which may not be readily 'eyeballed' from raw data points.

In terms of general interest, following their introduction raincloud plots have generated substantial enthusiasm on social media amongst scientists from a variety of disciplines (@neuroconscience, 2018b; Neuroconscience, 2018a), and are now available as a default option in at least one statistical plotting software (Wilke, 2017). To further

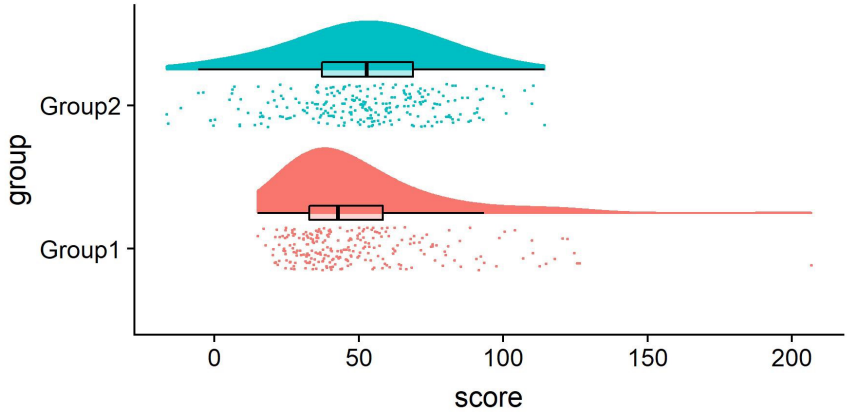


Figure 3. Example Raincloud plot. The raincloud plot combines an illustration of data distribution (the 'cloud'), with jittered raw data (the 'rain'). This can further be supplemented by adding boxplots or other standard measures of central tendency and error. See figure3.Rmd for code to generate this figure.

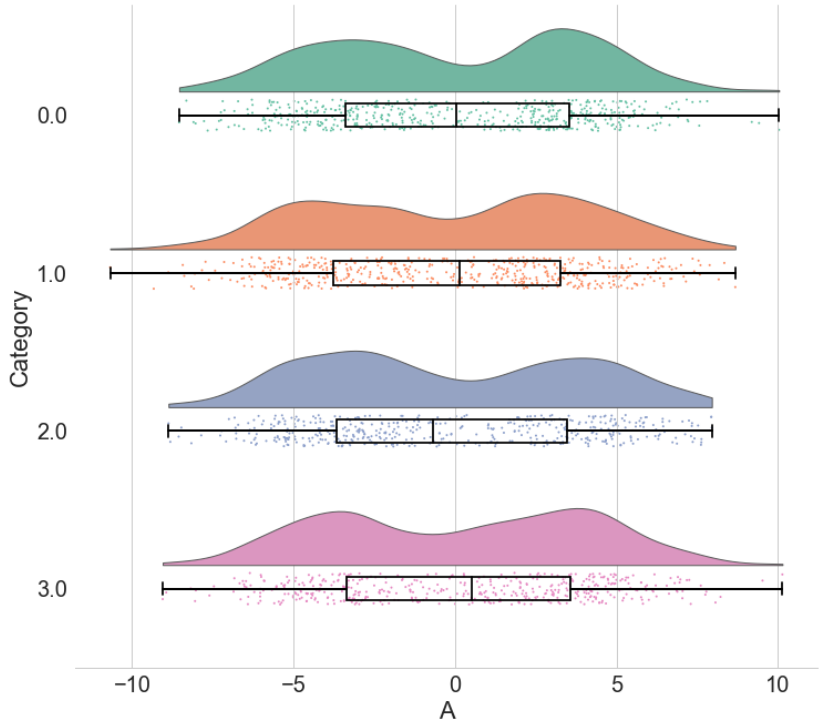


Figure 4. Raincloud plots leave little to the imagination. By replacing the redundantly mirrored probability distribution with a boxplot and raw data-points, the raincloud plot provides the user with information both about individual observations and patterns among them (such as striation or clustering), and overall tendencies in the distribution. As illustrated here, even a boxplot plus raw data may hide bimodality or other crucial facets of the data. See figure4.ipynb for code to generate these figures.

their accessibility and ease-of-use, in the following multi-platform tutorial we provide code and documentation for the step-by-step creation and customization of raincloud plots in R, Matlab, and Python.

Code tutorials: how to make it rain

How to make it rain in R

R (https://www.r-project.org) is a multiplatform, free and open source tool widely used in the statistical community (R Core Team, 2013). Our tutorial includes an associated R-script to create the raincloud function which complements the existing ggplot2 package (Wickham, 2010; Wickham & Chang, 2008), as well as an R-notebook (reproduced below) which walks the user through the simulation of data, illustrates a variety of parameters that can be user modified and shows how to get from barplots to rainclouds.

The code is available at

https://github.com/RainCloudPlots/RainCloudPlots/tree/master/tutorial_R

and can be run interactively in the browser at

https://mybinder.org/v2/gh/RainCloudPlots/RainCloudPlots/master?urlpath=rstudio.

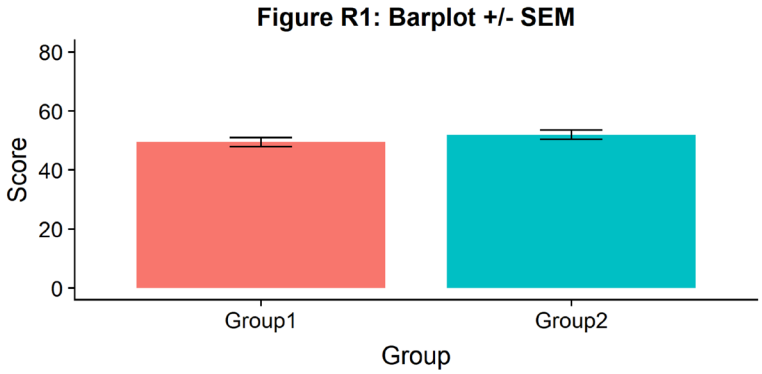
This tutorial will walk you through the process of transforming your barplots into rainclouds, and also show you how to customize your rainclouds for various options such as ordinal or repeated measures data. First, we'll run the included "R_rainclouds" script, which will set-up the split-half violin option in ggplot, as well as simulate some data for our figures:

```
source("R_rainclouds.R")
source("summarySE.R")
source("simulateData.R")
library(cowplot)
library(readr)
# width and height variables for saved plots
w = 6
h = 3
```

```
head(summary_simdat)
##    group   N score_mean score_median       sd       se       ci
## 1 Group1 250   49.45877     42.74587 25.27975 1.598832 3.148958
## 2 Group2 250   51.94353     52.69956 25.06328 1.585141 3.121994
```

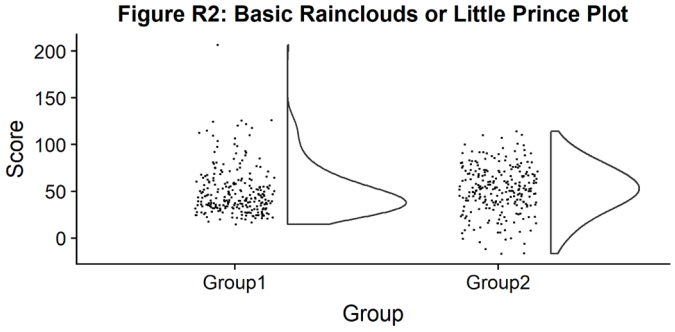
The function gives us two groups of N = 250 observations each; both have similar means and SDs, but group one is drawn from an exponential distribution. Now we'll plot a basic barplot for our simulated data. Note that we're using the 'cowplot' theme (https://github.com/wilkelab/cowplot) to produce simple, uncluttered plots - you should set-up your own theme or other customization options as desired:

```
#Barplot
p1 <- ggplot(summary_simdat, aes(x = group, y = score_mean, fill = group))+
  geom_bar(stat = "identity", width = .8)+
  geom_errorbar(aes(ymin = score_mean - se, ymax = score_mean+se), width = .2)+
  guides(fill=FALSE)+
  ylim(0, 80)+
  ylab('Score')+xlab('Group')+theme_cowplot()+
  ggtitle("Figure R1: Barplot +/- SEM")
  ggsave('1Barplot.png', width = w, height = h)

 p1
```



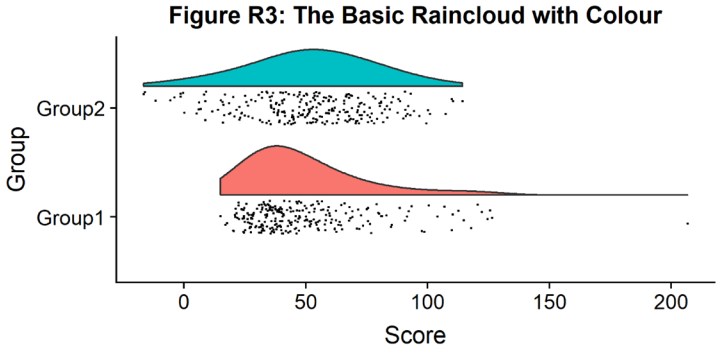
There we go - just needs some little asterisks and we're ready to publish! Just kidding. Let's start our first, most basic raincloud plot like so, using the 'geom_flat_violin' option our function already setup for us:

```
#Basic plot
p2 <- ggplot(simdat,aes(x=group,y=score))+
  geom_flat_violin(position = position_nudge(x = .2, y = 0),adjust =2)+
  geom_point(position = position_jitter(width = .15), size = .25)+
  ylab('Score')+xlab('Group')+theme_cowplot()+
  ggtitle('Figure R2: Basic Raincloud or Little Prince Plot')+
  ggsave('2basic.png', width = w, height = h)

p2
```



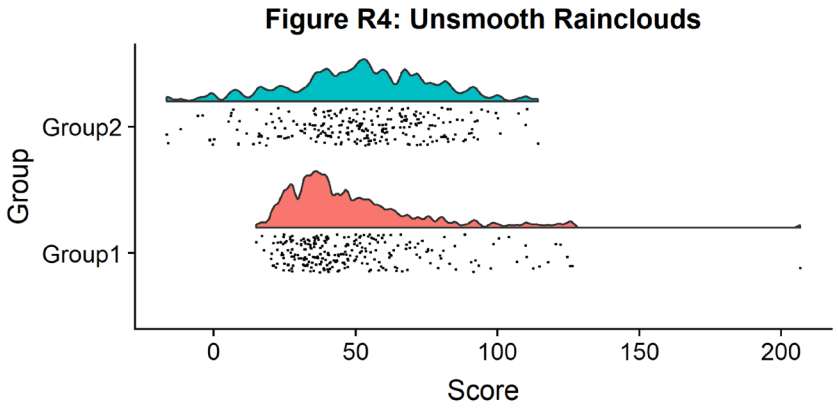
Now we can see the raw data (our 'rain'), and the overlaid probability distribution (the 'cloud'). Let's make it a bit prettier and easier to read by adding some colours. We can also use 'coordinate flip' to rotate the entire plot about the x-axis, transforming our 'little prince plots' into true rainclouds:

```
#Plot with colours and coordinate flip
p3 <- ggplot(simdat,aes(x=group,y=score, fill = group))+
  geom_flat_violin(position = position_nudge(x = .2, y = 0),adjust = 2)+
  geom_point(position = position_jitter(width = .15), size = .25)+

ylab('Score')+xlab('Group')+coord_flip()+theme_cowplot()+guides(fill = FALSE)+
  ggtitle('Figure R3: The Basic Raincloud with Colour')+
  ggsave('figs/rTutorial/3pretty.png', width = w, height = h)

p3
```



In case you want to change the smoothing kernel used to calculate the PDFs, you can do so by altering the 'adjust' flag for geom_flat_violin. For example, here we've dropped our smoothing to give a much bumpier raincloud:

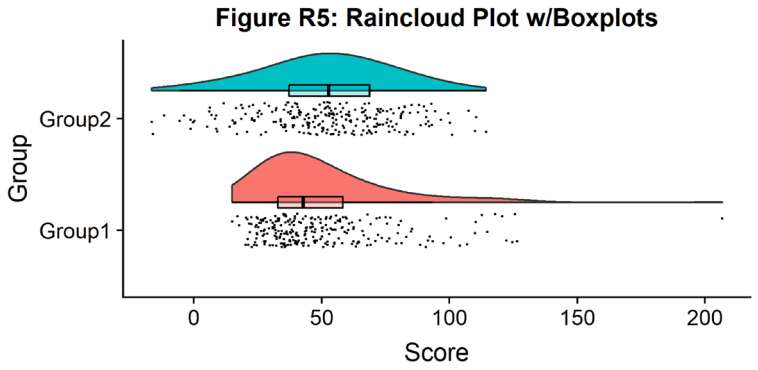
```
#Raincloud with reduced smoothing
p4 <- ggplot(simdat,aes(x=group,y=score, fill = group))+
  geom_flat_violin(position = position_nudge(x = .2, y = 0),adjust = .2)+
  geom_point(position = position_jitter(width = .15), size = .25)+

ylab('Score')+xlab('Group')+coord_flip()+theme_cowplot()+guides(fill = FALSE) +
  ggtitle('Figure R4: Unsmooth Raincloud')
  ggsave('4unsmooth.png', width = w, height = h)

  p4
```



Now we need to add something to help us easily evaluate any possible differences between our groups or conditions. To achieve this, we'll add some boxplots to complete our raincloud plots. To get the boxplots to line up however we like, we need to set our x-axis to a numeric value, so we can add a fixed offset:

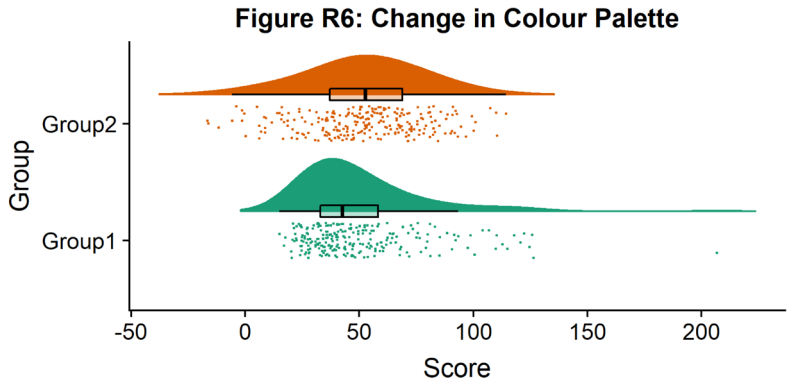
```
#Raincloud with boxplots
p5 <- ggplot(simdat,aes(x=group,y=score, fill = group))+
  geom_flat_violin(position = position_nudge(x = .25, y = 0),adjust =2)+
  geom_point(position = position_jitter(width = .15), size = .25)+
#note that here we need to set the x-variable to a numeric variable and bump it
to get the boxplots to line up with the raincloud.
   geom_boxplot(aes(x = as.numeric(group)+0.25, y = score),outlier.shape = NA,
alpha = 0.3, width = .1, colour = "BLACK") +
```

```
ylab('Score')+xlab('Group')+coord_flip()+theme_cowplot()+guides(fill = FALSE,
colour = FALSE) +
  ggtitle("Figure R5: Raincloud Plot w/Boxplots")
  ggsave('5boxplots.png', width = w, height = h)

  p5
```



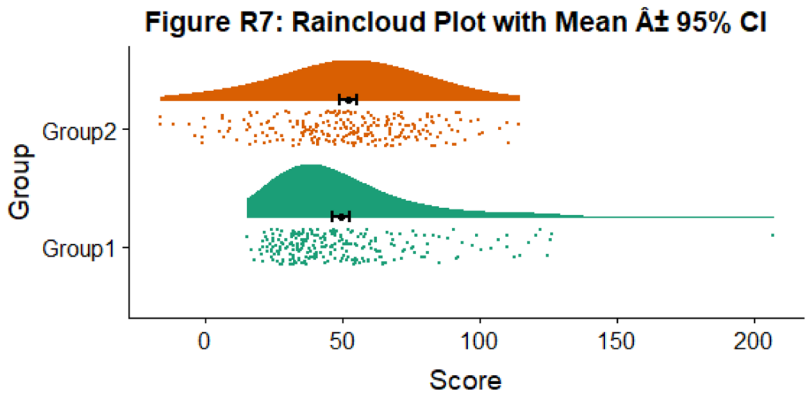
Now we'll make a few aesthetic tweaks. You may want to turn these on or off depending on your preferences. We'll take the black outline away from the plots by adding the colour = group parameter, and we'll also change colour palettes using the built-in colour brewer tool.

```
#Raincloud with boxplots
p6 <- ggplot(simdat,aes(x=group,y=score, fill = group, colour = group))+
  geom_flat_violin(position = position_nudge(x = .25, y = 0),adjust =2, trim =
FALSE)+
  geom_point(position = position_jitter(width = .15), size = .25)+
  geom_boxplot(aes(x = as.numeric(group)+0.25, y = score),outlier.shape = NA,
alpha = 0.3, width = .1, colour = "BLACK") +

ylab('Score')+xlab('Group')+coord_flip()+theme_cowplot()+guides(fill = FALSE,
colour = FALSE) +
  scale_colour_brewer(palette = "Dark2")+
  scale_fill_brewer(palette = "Dark2")+
  ggtitle("Figure R6: Change in Colour Palette")
  ggsave('6boxplots.png', width = w, height = h)

p6
```



Alternatively, you may prefer to simply plot mean or median with standard confidence intervals. Here we'll plot the mean as well as 95% confidence intervals, which we've calculated using the included SummarySE function (from https://www.rdocumentation.org/packages/Rmisc/versions/1.5/topics/summarySE), by overlaying them on of our clouds:

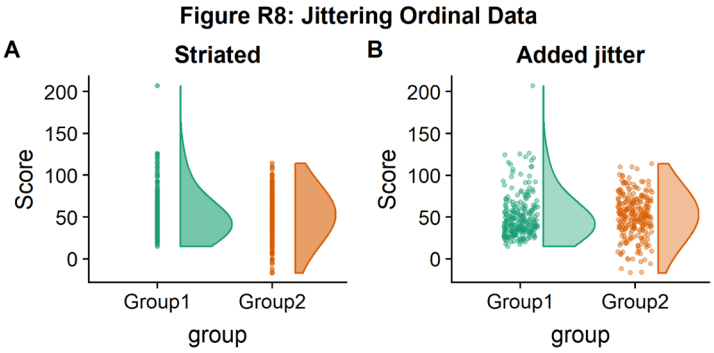
```
#Rainclouds with mean and confidence interval
p7 <- ggplot(simdat,aes(x=group,y=score, fill = group, colour = group))+
  geom_flat_violin(position = position_nudge(x = .25, y = 0),adjust =2)+
  geom_point(position = position_jitter(width = .15), size = .25)+
  geom_point(data = summary_simdat, aes(x = group, y = score_mean), position =
position_nudge(.25), colour = "BLACK")+
  geom_errorbar(data = summary_simdat, aes(x = group, y = score_mean, ymin =
score_mean-ci, ymax = score_mean+ci), position = position_nudge(.25), colour =
"BLACK", width = 0.1, size = 0.8)+

ylab('Score')+xlab('Group')+coord_flip()+theme_cowplot()+guides(fill = FALSE,
colour = FALSE) +
  scale_colour_brewer(palette = "Dark2")+
  scale_fill_brewer(palette = "Dark2")+
  ggtitle("Figure R7: Raincloud Plot with Mean Â± 95% CI")
  ggsave('7meanplot.png', width = w, height = h)

p7
```



If your data is discrete or ordinal you may need to manually add some jitter to improve the plot:

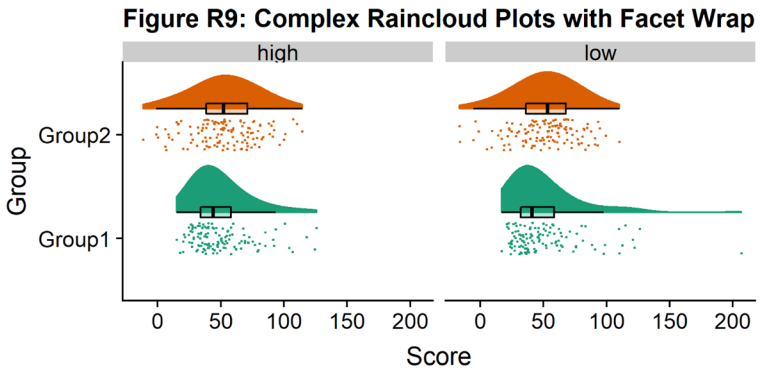
```
#Rainclouds with striated data

#Round data
simdat_round<-simdat
simdat_round$score<-round(simdat$score,0)

#Striated/grouped when no jitter applied
ap1 <- ggplot(simdat_round,aes(x=group,y=score,fill=group,col=group))+
geom_flat_violin(position = position_nudge(x = .2, y = 0), alpha = .6,adjust
=4)+geom_point(size = 1, alpha = 0.6)+ylab('Score')+scale_fill_brewer(palette
= "Dark2")+scale_colour_brewer(palette = "Dark2")+guides(fill = FALSE, col =
FALSE)+ggtitle('Striated')

#Added jitter helps
ap2 <- ggplot(simdat_round,aes(x=group,y=score,fill=group,col=group))+
geom_flat_violin(position = position_nudge(x = .2, y = 0), alpha = .4,adjust
```

```
=4)+geom_point(position=position_jitter(width = .15),size = 1, alpha =
0.4)+ylab('Score')+scale_fill_brewer(palette = "Dark2")+scale_colour_
brewer(palette = "Dark2")+guides(fill = FALSE, col = FALSE)+ggtitle('Added
jitter')

all_plot <- plot_grid(ap1, ap2, labels="AUTO")

# add title to cowplot
title <- ggdraw() +
  draw_label("Figure R8: Jittering Ordinal Data",
             fontface = 'bold')

all_plot_final <- plot_grid(title, all_plot, ncol = 1, rel_heights = c(0.1, 1))
# rel_heights values control title margins

ggsave('8allplot.png', width = w, height = h)
all_plot_final
```



Finally, in many situations you may have nested, factorial, or repeated measures data. In this case, one option is to use plot facets to group by factor, emphasizing pairwise differences between conditions or factor levels:

```
#Add additional factor/condition
simdat$gr2<-as.factor(c(rep('high',125),rep('low',125),rep('high',125),rep('low',
125)))


p9 <- ggplot(simdat,aes(x=group,y=score, fill = group, colour = group))+
  geom_flat_violin(position = position_nudge(x = .25, y = 0),adjust =2, trim =
TRUE)+
  geom_point(position = position_jitter(width = .15), size = .25)+
  geom_boxplot(aes(x = as.numeric(group)+0.25, y = score),outlier.shape = NA,
alpha = 0.3, width = .1, colour = "BLACK") +

ylab('Score')+xlab('Group')+coord_flip()+theme_cowplot()+guides(fill = FALSE,
colour = FALSE) + facet_wrap(~gr2)+
  scale_colour_brewer(palette = "Dark2")+
  scale_fill_brewer(palette = "Dark2")+
  ggtitle("Figure R9: Complex Raincloud Plots with Facet Wrap")
  ggsave('9facetplot.png', width = w, height = h)

p9
```



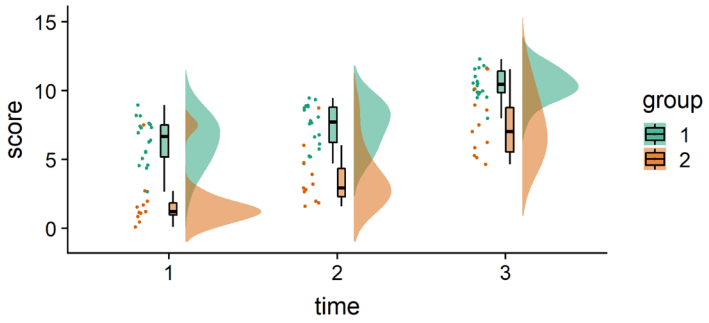
As another example, we consider some simulated repeated measures data in factorial design, where two groups are measured across three timepoints. To do so, we'll first load in some new data:

```
#load the repeated measures factorial data

rep_data <- read_csv("data/repeated_measures_data.csv",
    col_types = cols(group = col_factor(levels = c("1",
        "2")), time = col_factor(levels = c("1",
        "2", "3"))))

sumrepdat <- summarySE(rep_data, measurevar = "score",
groupvars=c("group", "time"))

head(sumrepdat)
##   group time  N score_mean score_median       sd        se        ci
## 1     1    1 18   6.362222        6.670 1.658861 0.3909972 0.8249319
## 2     1    2 18   7.468333        7.730 1.546880 0.3646032 0.7692454
## 3     1    3 18  10.482778       10.455 1.060254 0.2499043 0.5272520
## 4     2    1 11   1.847273        1.210 2.010279 0.6061219 1.3505238
## 5     2    2 11   3.684545        2.920 2.135108 0.6437594 1.4343852
## 6     2    3 11   7.358182        7.020 2.236273 0.6742616 1.5023486
```

Now, we'll plot our rainclouds with boxplots again, this time adding some dodge so we can better emphasize differences between our factors and factor levels. Note that here we need to nudge the point x-axis as a numeric valuable, as this work around does not currently work for boxplots with multiple factors:

```
# Rainclouds for repeated measures, continued
p10 <- ggplot(rep_data, aes(x = time, y = score, fill = group)) +
  geom_flat_violin(aes(fill = group),position = position_nudge(x = .1, y = 0),
adjust = 1.5, trim = FALSE, alpha = .5, colour = NA)+
  geom_point(aes(x = as.numeric(time)-.15, y = score, colour = group),position
= position_jitter(width = .05), size = 1, shape = 20)+
  geom_boxplot(aes(x = time, y = score, fill = group),outlier.shape = NA,
alpha = .5, width = .1, colour = "black")+
  scale_colour_brewer(palette = "Dark2")+
  scale_fill_brewer(palette = "Dark2")+
  ggtitle("Figure R10: Repeated Measures Factorial Rainclouds")
  ggsave('10repanvplot.png', width = w, height = h)
  #coord_flip()+
p10
```

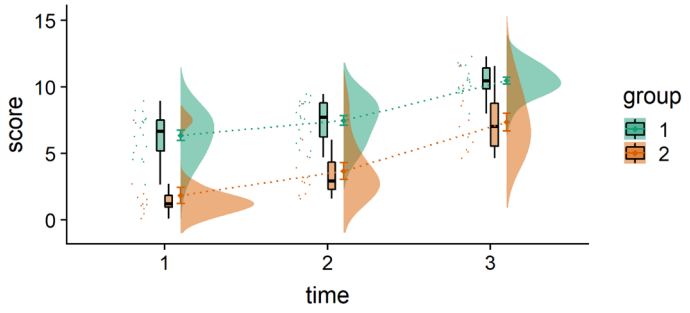
Figure R10: Repeated Measures Factorial Raincloud



Finally, you may want to add traditional line plots to emphasize factorial interactions and main effects. Here we've plotted the mean and standard error for each cell of our design and connected these with a hashed line. There are a lot of possible options though, so you'll need to decide what works best for your needs:

```
#Raincloud for repeated measures, additional plotting options

p11 <- ggplot(rep_data, aes(x = time, y = score, fill = group)) +
  geom_flat_violin(aes(fill = group),position = position_nudge(x = .1, y = 0),
adjust = 1.5, trim = FALSE, alpha = .5, colour = NA)+
  geom_point(aes(x = as.numeric(time)-.15, y = score, colour = group),position
= position_jitter(width = .05), size = .25, shape = 20)+
  geom_boxplot(aes(x = time, y = score, fill = group),outlier.shape = NA,
alpha = .5, width = .1, colour = "black")+
  geom_line(data = sumrepdat, aes(x = as.numeric(time)+.1, y = score_mean,
group = group, colour = group), linetype = 3)+
  geom_point(data = sumrepdat, aes(x = as.numeric(time)+.1, y = score_mean,
group = group, colour = group), shape = 18) +
  geom_errorbar(data = sumrepdat, aes(x = as.numeric(time)+.1, y = score_mean,
group = group, colour = group, ymin = score_mean-se, ymax = score_mean+se),
width = .05)+
  scale_colour_brewer(palette = "Dark2")+
  scale_fill_brewer(palette = "Dark2")+
  ggtitle("Figure R11: Repeated Measures - Factorial (Extended)")
  ggsave('11repanvplot2.png', width = w, height = h)
  #coord_flip()+

p11
```

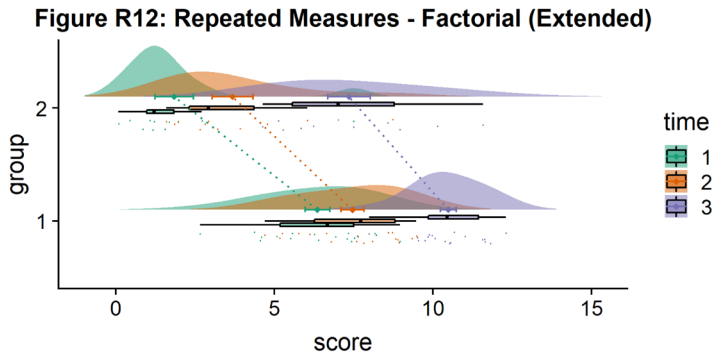
Figure R11: Repeated Measures - Factorial (Extended)



Here is the same plot, but with the grouping variable flipped:

```
#Raincloud for repeated measures, additional plotting options

p12 <- ggplot(rep_data, aes(x = group, y = score, fill = time)) +
  geom_flat_violin(aes(fill = time),position = position_nudge(x = .1, y = 0),
```

```
adjust = 1.5, trim = FALSE, alpha = .5, colour = NA)+
  geom_point(aes(x = as.numeric(group)-.15, y = score, colour = time),position
= position_jitter(width = .05), size = .25, shape = 20)+
  geom_boxplot(aes(x = group, y = score, fill = time),outlier.shape = NA,
alpha = .5, width = .1, colour = "black")+
  geom_line(data = sumrepdat, aes(x = as.numeric(group)+.1, y = score_mean,
group = time, colour = time), linetype = 3)+
  geom_point(data = sumrepdat, aes(x = as.numeric(group)+.1, y = score_mean,
group = time, colour = time), shape = 18) +
  geom_errorbar(data = sumrepdat, aes(x = as.numeric(group)+.1, y = score_
mean, group = time, colour = time, ymin = score_mean-se, ymax = score_
mean+se), width = .05)+
  scale_colour_brewer(palette = "Dark2")+
  scale_fill_brewer(palette = "Dark2")+
  ggtitle("Figure R12: Repeated Measures - Factorial (Extended)") +
  coord_flip()
  ggsave('12repanvplot3.png', width = w, height = h)

p12
```



That's it! We hope you'll be able to use this tutorial to find great illustrations for your data, and that we've given you an idea of some of the different ways you can customize your raincloud plots. Next, we'll consider how to reproduce these steps in Python and Matlab.

How to Make it Rain in Python

Python is an open source programming language (https://www.python.org) that has recently become extremely popular within data science and statistical machine learning. Our interactive Python tutorial can be found at the following URL:

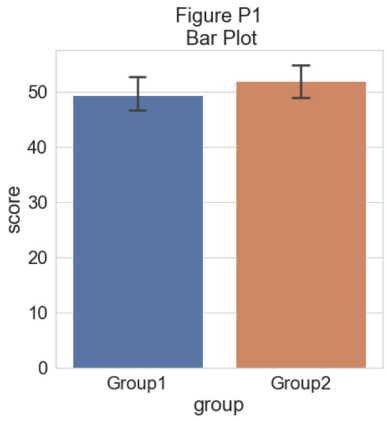
https://github.com/RainCloudPlots/RainCloudPlots/blob/master/tutorial_python/raincloud_tutorial_python.ipynb

The tutorial follows the footsteps of the R tutorial to guide you in the creation and customization of Raincloud plots. The Python implementation of Raincloud Plots is a package named PtitPrince (https://github.com/pog87/PtitPrince), written on the top of seaborn. Seaborn (https://seaborn.pydata.org) is a Python plotting library written as an extension to the Python graphic library matplotlib (https://matplotlib.org) supporting aesthetically pleasing plots and to work directly with pandas dataframes. The tutorial can be run interactively in the browser at:

https://mybinder.org/v2/gh/RainCloudPlots/RainCloudPlots/master?filepath=tutorial_python%2Fraincloud_tutorial_python.ipynb.

As first step, we will load the same dataset used before and visualize the distribution of each measure as a simple barplot with errorbars:

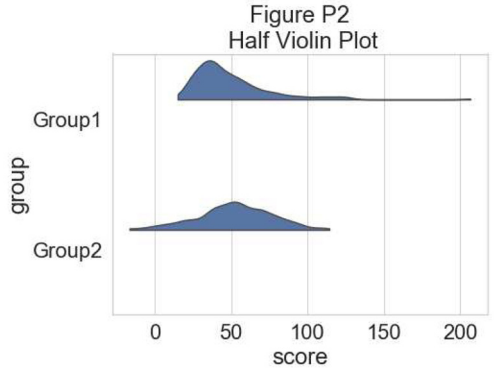
```
import pandas as pd
import ptitprince as pt
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="whitegrid",font_scale=2)
import matplotlib.collections as clt

df = pd.read_csv ("simdat.csv", sep= ",")

sns.barplot(x = "group", y = "score", data = df, capsize= .1)
```



This plot can give the reader a first idea of the dataset: which group has a larger mean value, and whether this difference is likely to be significant or not. Only the mean of each group score and the standard deviation is visualized in this plot.

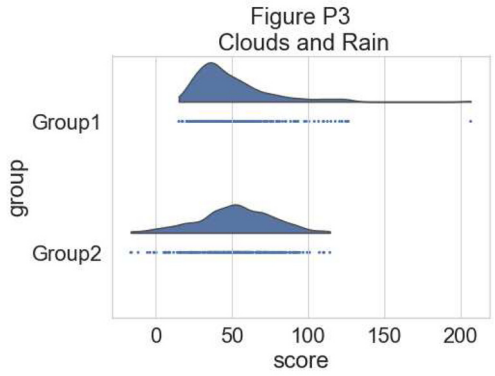
To have an idea of the distribution of our dataset we can plot a "cloud", a smoothed version of the histogram:

```
# plotting the clouds
f, ax = plt.subplots(figsize=(7, 5))
dy="group"; dx="score"; ort="h"; pal = sns.color_palette(n_colors=1)

ax=pt.half_violinplot( x = dx, y = dy, data = df, palette = pal,
     bw = .2, cut = 0.,scale = "area", width = .6, inner = None,
     orient = ort)
```

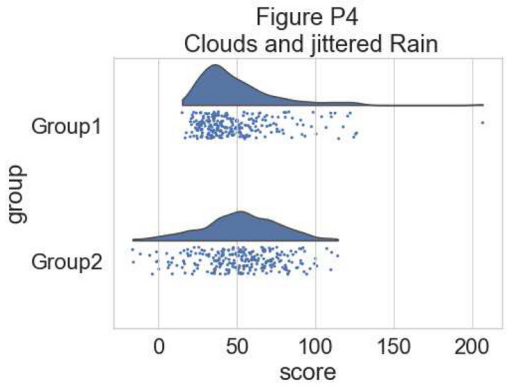


To have a more precise idea of the distribution and illustrate potential outliers or other patterns within the data, we now add the "rain", a simple monodimensional representation of the data points:

```
# adding the rain
f, ax = plt.subplots(figsize=(7, 5))
ax=pt.half_violinplot( x = dx, y = dy, data = df, palette = pal,
      bw = .2, cut = 0.,scale = "area", width = .6, inner = None,
     orient = ort)

ax=sns.stripplot( x = dx, y = dy, data = df, palette = pal,
     edgecolor = "white",size = 3, jitter = 0, zorder = 0,
     orient = ort)
```




```
# adding jitter to the rain
f, ax = plt.subplots(figsize=(7, 5))
ax=pt.half_violinplot( x = dx, y = dy, data = df, palette = pal,
     bw = .2, cut = 0.,scale = "area", width = .6, inner = None,
     orient = ort)

ax=sns.stripplot( x = dx, y = dy, data = df, palette = pal,
     edgecolor = "white",size = 3, jitter = 1, zorder = 0,
     orient = ort)
```

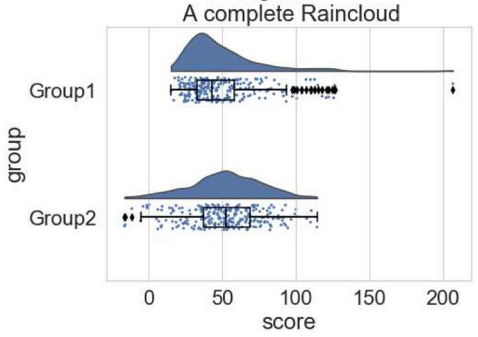


This gives a good idea of the distribution of the data points, but the median and the quartiles are not obvious, making it hard to determine statistical differences at a glance. Hence, we add an "empty" boxplot to show median, quartiles and outliers:

```
#adding the boxplot with quartiles
f, ax = plt.subplots(figsize=(7, 5))
ax=pt.half_violinplot( x = dx, y = dy, data = df, palette = pal,
```

```
       bw = .2, cut = 0.,scale = "area", width = .6, inner = None,
      orient = ort)

ax=sns.stripplot( x = dx, y = dy, data = df, palette = pal,
      edgecolor = "white", size = 3, jitter = 1, zorder = 0,
      orient = ort)

ax=sns.boxplot( x = dx, y = dy, data = df, color = "black",
      width = .15, zorder = 10, showcaps = True,
      boxprops = {'facecolor':'none', "zorder":10}, showfliers=True,
      whiskerprops = {'linewidth':2, "zorder":10},
      saturation = 1, orient = ort)
```

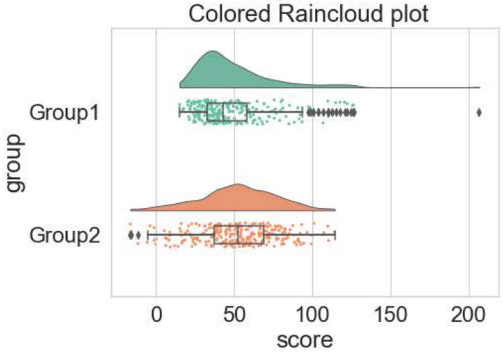
Figure P5
A complete Raincloud



Now we can set a color palette to characterize the two groups:

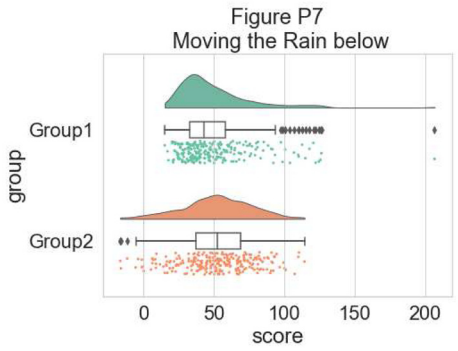
```
#adding color
pal = "Set2"
f, ax = plt.subplots(figsize=(7, 5))
ax=pt.half_violinplot( x = dx, y = dy, data = df, palette = pal,
     bw = .2, cut = 0.,scale = "area", width = .6,
     inner = None, orient = ort)

ax=sns.stripplot( x = dx, y = dy, data = df, palette = pal,
      edgecolor = "white",size = 3, jitter = 1, zorder = 0,
      orient = ort)

ax=sns.boxplot( x = dx, y = dy, data = df, color = "black",
     width = .15, zorder = 10, showcaps = True,
     boxprops = {'facecolor':'none', "zorder":10}, showfliers=True,
     whiskerprops = {'linewidth':2, "zorder":10},
     saturation = 1, orient = ort)
```

Figure P6
Colored Raincloud plot



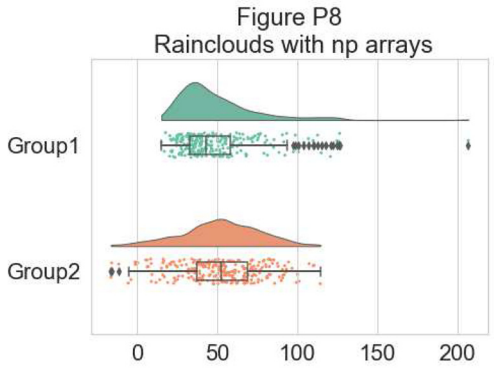
This plot is now both informative and aesthetically pleasing but written in far too many lines of code. We can use the function `pt.Raincloud` to add some automation:

```
#same thing with a single command: now x **must** be the categorical value
dx = "group"; dy = "score"; ort = "h"; pal = "Set2"; sigma = .2

ax=pt.RainCloud(x = dx, y = dy, data = df, palette = pal,
     bw = sigma,width_viol = .6, figsize = (7,5), orient = ort)
```



The 'move' parameter can be used to shift the rain below the boxplot, giving better visibility of the raw data in some instances:

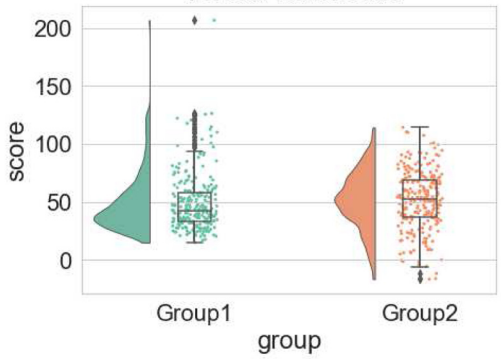
```
#moving the rain below the boxplot
dx = "group"; dy = "score"; ort = "h"; pal = "Set2"; sigma = .2

ax=pt.RainCloud(x = dx, y = dy, data = df, palette = pal,
     bw = sigma, width_viol = .6, figsize = (7,5),
     orient = ort, move = .2)
```



Further, the raincloud function works equally well with a list or numpy.array, if you prefer to use those instead of a dataframe input:

```
# Usage with a list/np.array input
dx = list(df["group"]); dy = list(df["score"])

ax=pt.RainCloud(x = dx, y = dy, palette = pal, bw = sigma,
     width_viol = .6, figsize = (7,5), orient = ort)
```

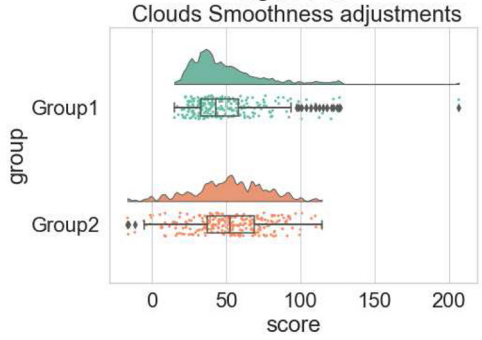
Figure P9
Vertical Raincloud



For some data, you may want to flip the orientation of the raincloud to a 'petit prince' plot. You can do this with the 'orient' flag in the pt.RainCloud Function:

```
# Changing orientation
dx="group"; dy="score"; ort="v"; pal = "Set2"; sigma = .2

ax=pt.RainCloud(x = dx, y = dy, data = df, palette = pal,
     bw = sigma,width_viol = .5, figsize = (7,5), orient = ort)
```

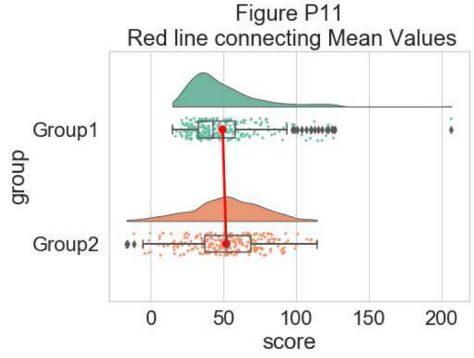
Figure P10
Clouds Smoothness adjustments



You can also change the smoothing kernel used to generate the probability distribution function of the data. To do this, you adjust the sigma parameter:

```
#changing cloud smoothness
dx="group"; dy="score"; ort="h"; pal = "Set2"; sigma = .05

ax=pt.RainCloud(x = dx, y = dy, data = df, palette = pal,
     bw = sigma,width_viol = .6, figsize = (7,5), orient = ort)
```

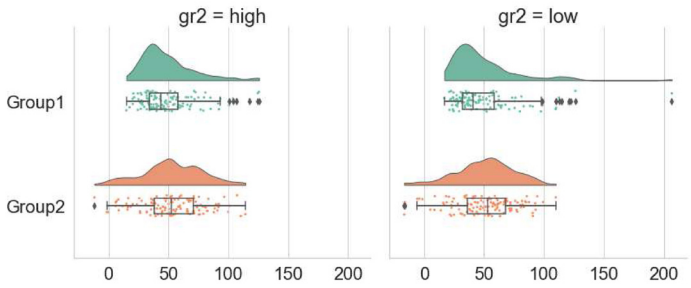
Figure P11
Red line connecting Mean Values



Finally, using the pointplot flag you can add a line connecting group mean values. This can be useful for more complex datasets, for example repeated measures or factorial data. Below we illustrate a few different approaches to plotting such data using raincloud, by changing the hue, opacity, or dodge element of the individual plots:

```
#adding a red line connecting the groups' mean value (useful for longitudinal
data)
dx="group"; dy="score"; ort="h"; pal = "Set2"; sigma = .2

ax=pt.RainCloud(x = dx, y = dy, data = df, palette = pal,
     bw = sigma, width_viol = .6, figsize = (7,5),
     orient = ort, pointplot = True)
```

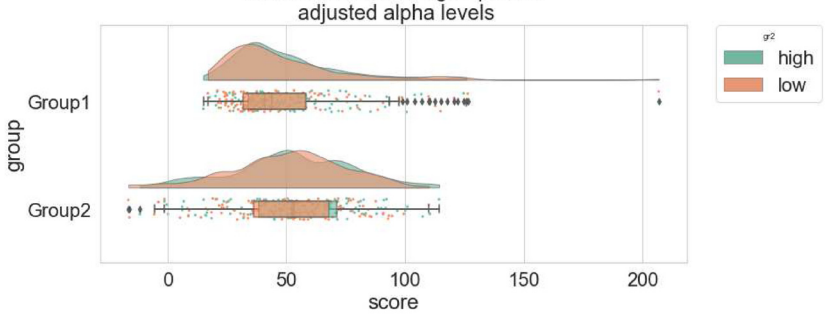
Figure P12
Raincloud in a Grid



Another flexible option is to use Facet Grids to separate different groups or factor levels, illustrated below:

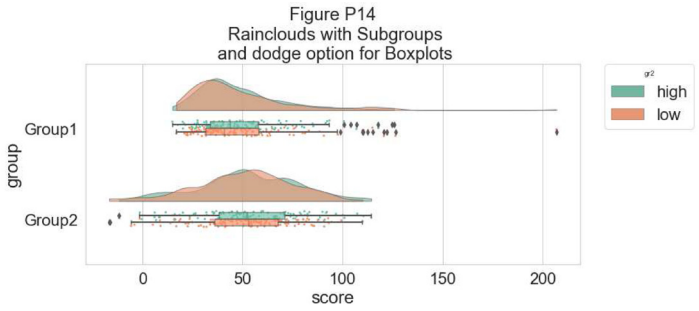
```
# Raincloud with FacetGrid
g = sns.FacetGrid(df, col = "gr2", height = 6)
g = g.map_dataframe(pt.RainCloud, x = "group", y = "score",
      data = df, orient = "h", ax = g.axes)
```

Figure P13
Raincloud with Subgroups and
adjusted alpha levels



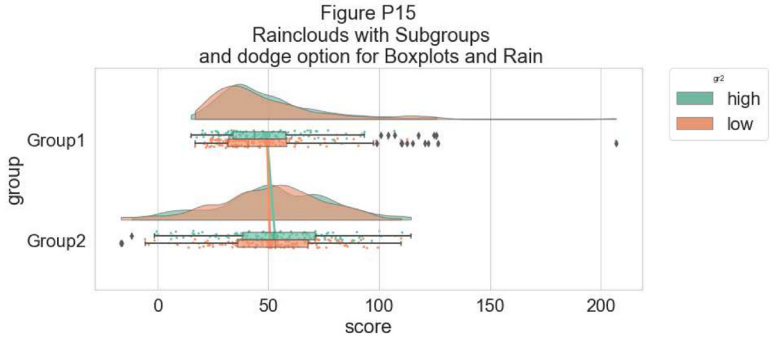
As an alternative, it is possible to use the hue input for plotting different sub-groups directly over one another, facilitating their comparison:

```
# Hue Input for Subgroups
dx="group"; dy="score"; dhue="gr2"; ort="h" pal="Set2"; sigma = .2

ax=pt.RainCloud(x = dx, y = dy, hue = dhue, data = df,
     palette = pal, bw = sigma,width_viol = .7, figsize = (12,5),
     orient = ort)
```



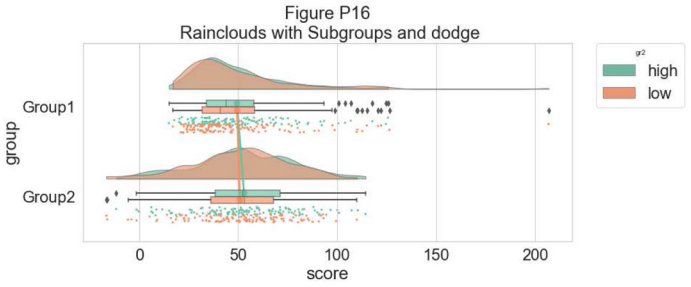
To improve the readability of this plot, we adjust the alpha-level using the associated flag (0–1 alpha intensity):

```
# Setting alpha level
ax=pt.RainCloud(x = dx, y = dy, hue = dhue, data = df,
     palette = pal, bw = sigma, width_viol = .7, figsize = (12,5),
     orient = ort , alpha = .65)
```



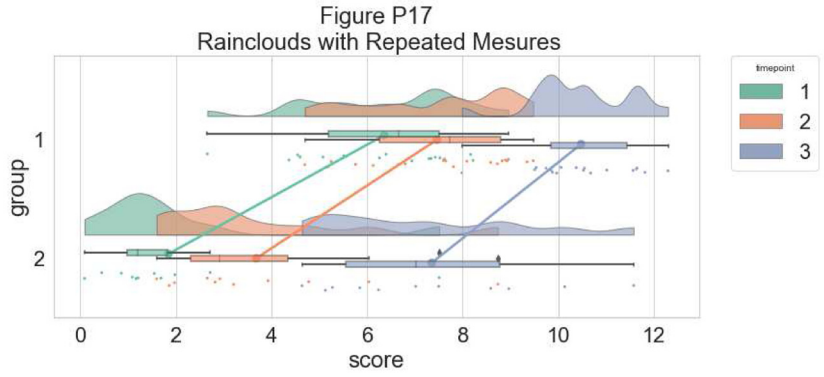
Rather than letting the two boxplots obscure one another, we can set the dodge flag to true, adding interpretability:

```
#The Dodge Flag
ax=pt.RainCloud(x = dx, y = dy, hue = dhue, data = df,
     palette = pal, bw = sigma,width_viol = .7, figsize = (12,5),
     orient = ort , alpha = .65, dodge = True)
```



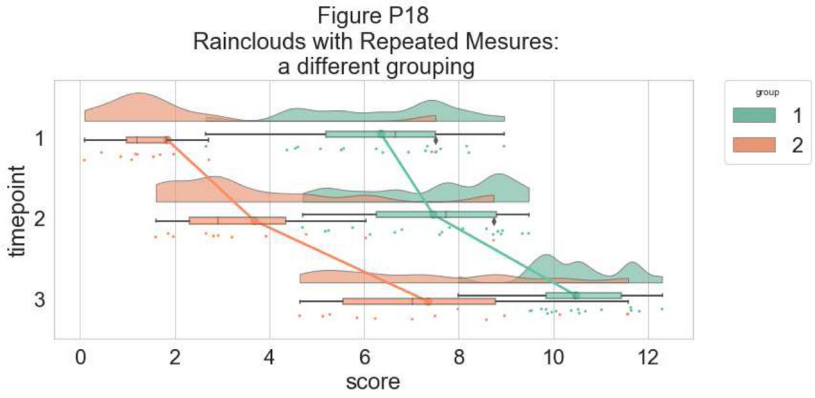
Finally, we may want to add a traditional line-plot to our graph to aid in the detection of factorial main effects and interactions. As an example, we've plotted the mean within each boxplot:

```
#same, with dodging and line
ax=pt.RainCloud(x = dx, y = dy, hue = dhue, data = df,
     palette = pal, bw = sigma, width_viol = .7,figsize = (12,5),
     orient = ort , alpha = .65, dodge = True, pointplot = True)
```



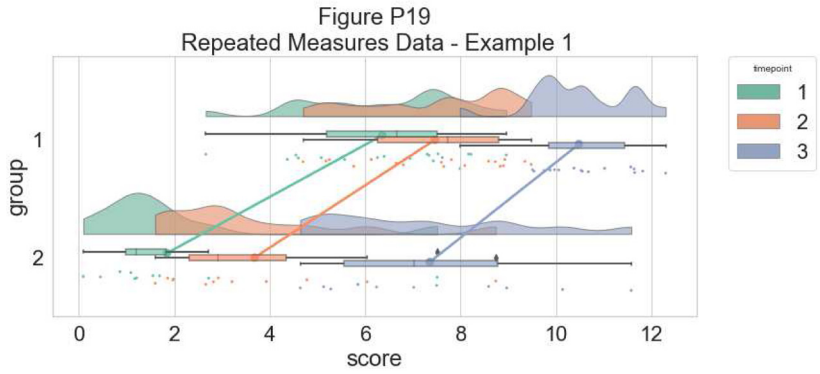
Here is the same plot, but now with the individual observations moved below the boxplots again using the 'move' parameter:

```
#moving the rain under the boxplot
ax=pt.RainCloud(x = dx, y = dy, hue = dhue, data = df,
     palette = pal, bw = sigma, width_viol = .7,figsize = (12,5),
     orient = ort , alpha = .65, dodge = True, pointplot = True,
     move = .2)
```



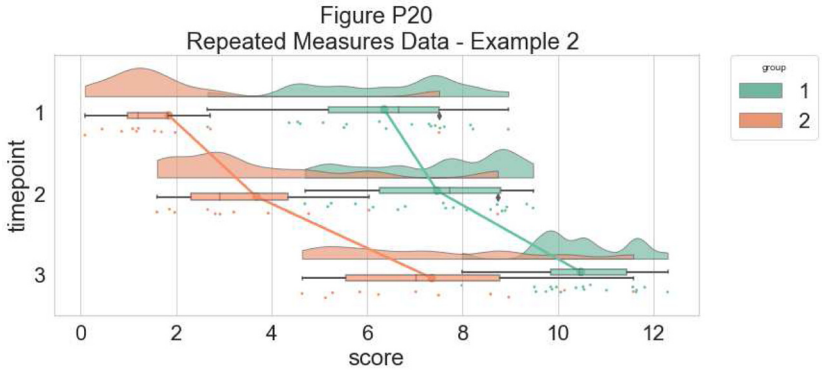
As our last example, we'll consider a complex repeated measures design with two groups and three timepoints. The goal is to illustrate our complex interactions and main-effects, while preserving the transparent nature of the raincloud plot:

```
# Load in the repeated data
df_rep = pd.read_csv ("repeated_measures_data.csv", sep= ",",
      header = None)
df_rep.columns = ["score",  "timepoint", "group"]

# Plot the repeated measures data
dx = "group"; dy="score"; dhue="timepoint"
ort="h"; pal="Set2"; sigma = .2

ax=pt.RainCloud(x = dx, y = dy, hue = dhue, data = df_rep,
     palette = pal, bw = sigma, width_viol = .7,figsize = (12,5),
     orient = ort , alpha = .65, dodge = True, pointplot = True,
     move = .2)
```



The function is flexible enough that you can flip the ordering of the factors around simply by changing which variable informs the hue parameter:

```
# Now with the group as hue
dx = "timepoint"; dy = "score"; dhue = "group"
ax=pt.RainCloud(x = dx, y = dy, hue = dhue, data = df_rep,
      palette = pal, bw = sigma, width_viol = .7, figsize = (12,5),
      orient = ort, alpha = .65, dodge = True, pointplot = True,
      move = .2)
```



That's it! Hopefully this tutorial has given you an idea of some of the different ways you can produce raincloud plots in Python. Next, we'll describe how to produce these plots in Matlab.

How to Make it Rain in Matlab

Matlab (Mathworks Inc.) is a proprietary mathematical programming language used widely in engineering, the physical sciences, and neuroscience. The code for this tutorial can be found at:

https://github.com/RainCloudPlots/RainCloudPlots/tree/master/tutorial_matlab

Here you can also find functions to create raincloud-plots (raincloud_plot.m and rm_raincloud.m), as well as a "live notebook" (raincloud_plots_tutorial.mlx) which walks the user through the customization of various raincloud plots.

First, we'll set up our path and use the colorbrewer function to define some nice colour palettes:

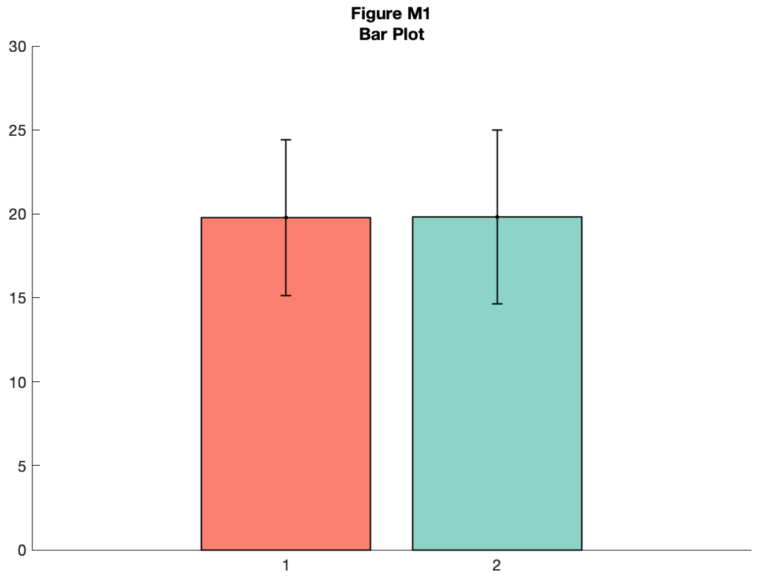
```
% set up a dynamic path
% script must be run from parent directory containing all three tutorial
% directories (i.e., the one 'above' the directory 'tutorial_matlab')

pardir = pwd;
figdir = fullfile(pardir, 'figs', 'tutorial_matlab');
if ~exist('figdir', 'dir')
    mkdir(figdir);
end

% make sure functions to generate plots are on the path
codedir = fullfile(pardir, 'tutorial_matlab');
addpath(codedir);

try
     % get nice colours from colorbrewer
     % (https://uk.mathworks.com/matlabcentral/fileexchange/34087-cbrewer---
colorbrewer-schemes-for-matlab)
     [cb] = cbrewer('qual', 'Set3', 12, 'pchip');
catch
     % if you don't have colorbrewer, accept these far more boring colours
     cb = [0.5 0.8 0.9; 1 1 0.7; 0.7 0.8 0.9; 0.8 0.5 0.4; 0.5 0.7 0.8; 1 0.8
0.5; 0.7 1 0.4; 1 0.7 1; 0.6 0.6 0.6; 0.7 0.5 0.7; 0.8 0.9 0.8; 1 1 0.4];
end

cl(1, :) = cb(4, :);
cl(2, :) = cb(1, :);

fig_position = [200 200 600 400]; % coordinates for figures
```

Now we'll generate some datapoints with similar means and standard deviations; the first is drawn from a random normal distribution and the second from a random exponential distribution. We'll plot these same data repeatedly in different ways further down:

```
n = 250;

% set a random number generator seed for reproducible results
rng(123)

d{1} = [exprnd(5, 1, n) + 15]';
d{2} = [(randn(1, n) *5) + 20]';

means = cellfun(@mean, d);
variances = cellfun(@std, d);
```

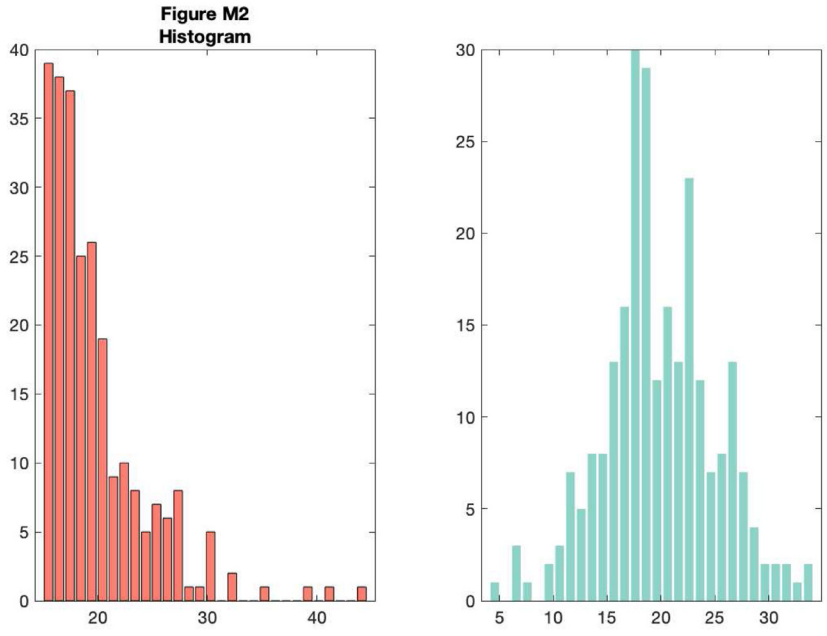
Let's create a quick bar graph of these data. This is the kind of standard visualization you see in many papers, depicting the mean of the data plus standard deviation:

```
f1 = figure('Position',fig_position); hold on;
h = bar(means, 'FaceColor', 'flat', 'LineWidth',.9);

h(1).CData(1, :) = cl(1, :);
h(1).CData(2, :) = cl(2, :);

e = errorbar(1:2, means, variances, '.k', 'LineWidth',.9);
set(gca, 'XTick', 1:2)
title('Bar Plot');

% save
print(f1, fullfile(figdir, '1bar.png'), '-dpng');
```



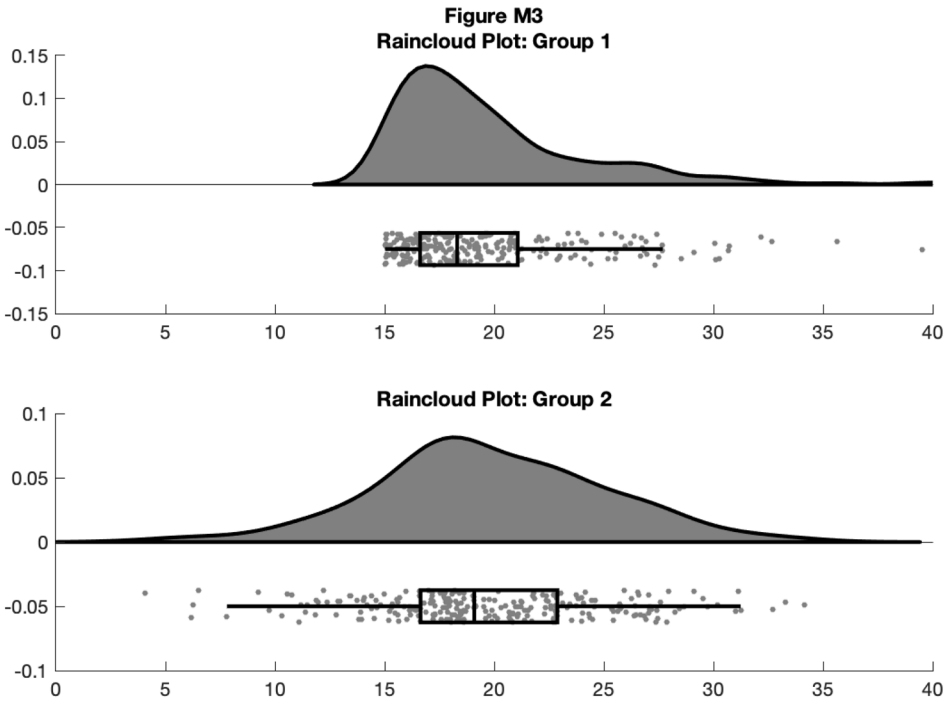
As you can see, this tells you something about the data, but a lot of really useful and important information is hidden such as the 'shape' or distribution of the data and the raw observations themselves. A histogram nicely shows some of what we're missing:

```
f2 = figure('Position', fig_position);
subplot(1, 2, 1)
[n1, x1] = hist(d{1}, 30);
bar(x1, n1, 'FaceColor', cl(1,:), 'EdgeColor', 'k');
title('Histogram')
subplot(1, 2, 2)
[n2, x2] = hist(d{2}, 30);
bar(x2, n2, 'FaceColor', cl(2,:), 'EdgeColor', 'none');

% save
print(f2, fullfile(figdir, '2hist.png'), '-dpng');
```



However, now we've lost the summary data. The raincloud plot tries to bring these elements together in one intuitive plot. You can use the 'raincloud_plot.m' function accompanying this tutorial to produce these plots in Matlab:

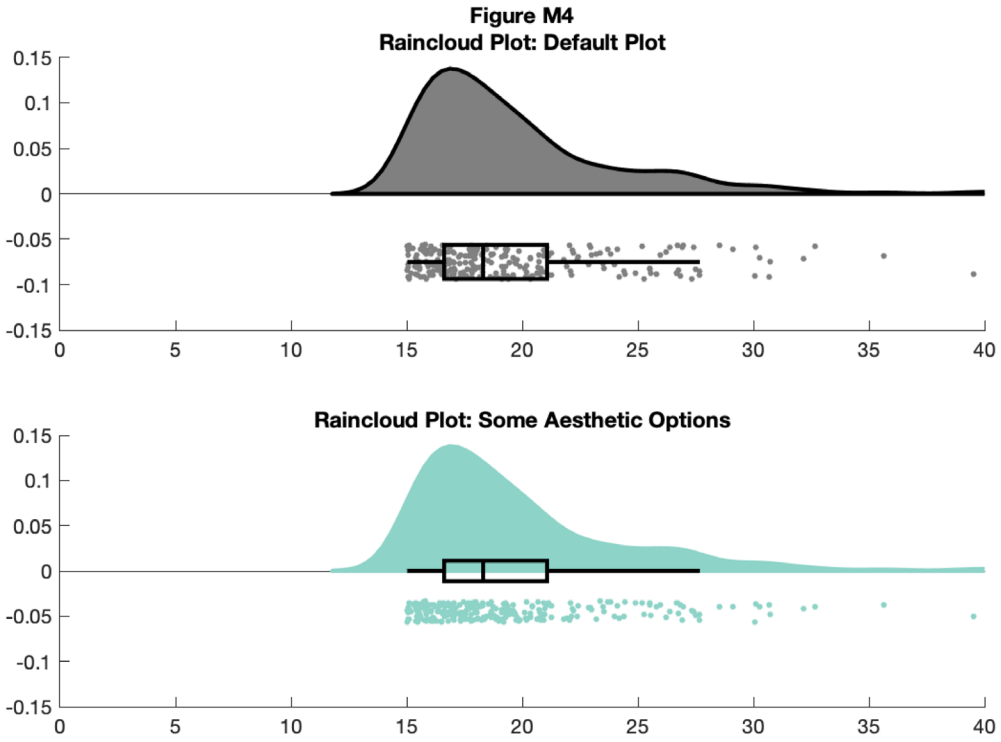
```
f3 = figure('Position', fig_position);
subplot(2, 1, 1)
h1 = raincloud_plot('d{1}, 'box_on', 1);
title('Raincloud Plot: Group 1')
set(gca,'XLim', [0 40]);
box off
subplot(2, 1, 2)
h2 = raincloud_plot(d{2}, 'box_on', 1);
title('Raincloud Plot: Group 2');
set(gca,'XLim', [0 40]);
box off


% save
print(f3, fullfile(figdir, '3Rain1.png'), '-dpng');
```



This gives us the distribution (probability density plot), summary data (box plot), and raw observations all in one place. Now we'll walk you through some of the options of the function, which you can use to change various aesthetic properties of the plot. The function only requires a vector of the data you want to plot as the input. Additionally, there are a variety of optional flags you can call to turn the boxplots on and off, to alter ('dodge') the position of the boxes and dots, and to change various aesthetics such as linewidth, colors, and so on. For example, by setting a few different flags we can create more colorful plots:

```
f4 = figure('Position', fig_position);
subplot(2, 1, 1)
h1 = raincloud_plot(d{1}, 'box_on', 1);
```

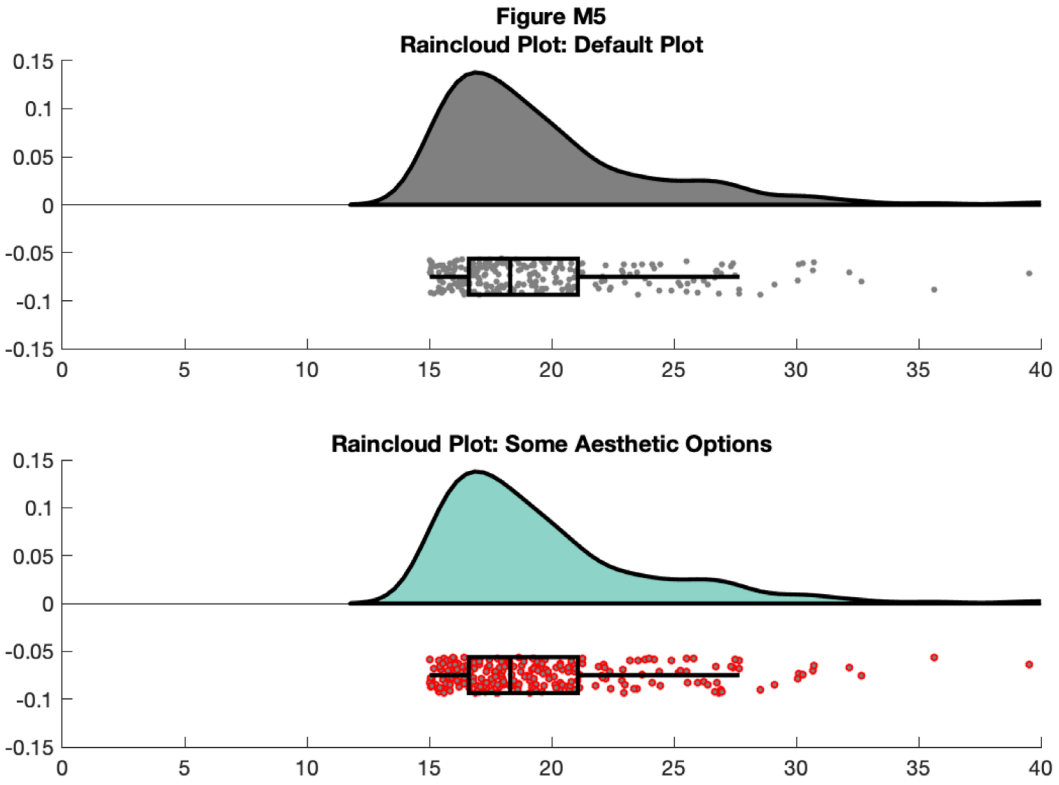
```
title('Raincloud Plot: Default Plot')
set(gca,'XLim', [0 40]);
box off
subplot(2, 1, 2)
h2 = raincloud_plot(d{1}, 'box_on', 1, 'box_dodge', 1, 'box_dodge_amount',...
0, 'dot_dodge_amount', .3, 'color', cb(1,:), 'cloud_edge_col', cb(1,:));
title('Raincloud Plot: Some Aesthetic Options');
set(gca,'XLim', [0 40]);
box off

% save
print(f4, fullfile(figdir, '4Rain2.png'), '-dpng');
```



The function returns a cell array for various figure parts, so you can also call the base function and then change things with normal 'set' commands, like so:

```
f5 = figure('Position', fig_position);
subplot(2, 1, 1)
h1 = raincloud_plot(d{1}, 'box_on', 1);
title('Raincloud Plot: Default Plot')
set(gca,'XLim', [0 40]);
box off
subplot(2, 1, 2)
h2 =  raincloud_plot(d{1}, 'box_on', 1);
title('Raincloud Plot: Some Aesthetic Options');
set(h2{1},'FaceColor', cb(1, :)) % handles 1-6 are the cloud area,
scatterpoints, and boxplot elements respectively
set(h2{2}, 'MarkerEdgeColor', 'red') %
```

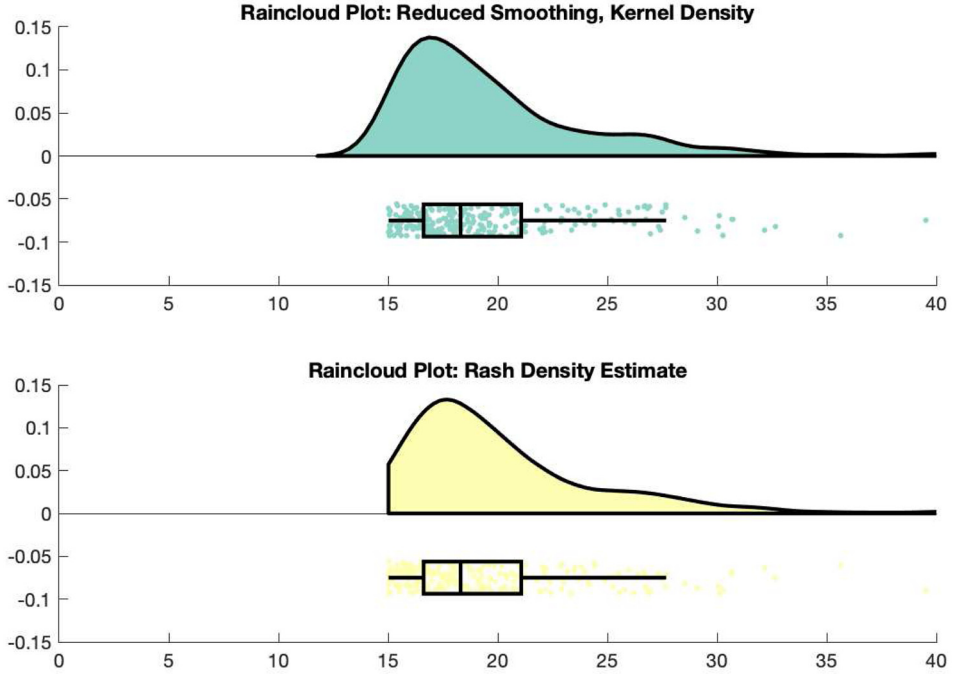
```
set(gca,'XLim', [0 40]);
box off

% save
print(f5, fullfile(figdir, '5Rain3.png'), '-dpng');
```



You can also control the smoothness of the probability density function by calling the 'bandwidth' parameter. Additionally, if you have Cyril Pernet's robust statistics toolbox on your path, you can call the 'rash' function for an alternative kernel density function:

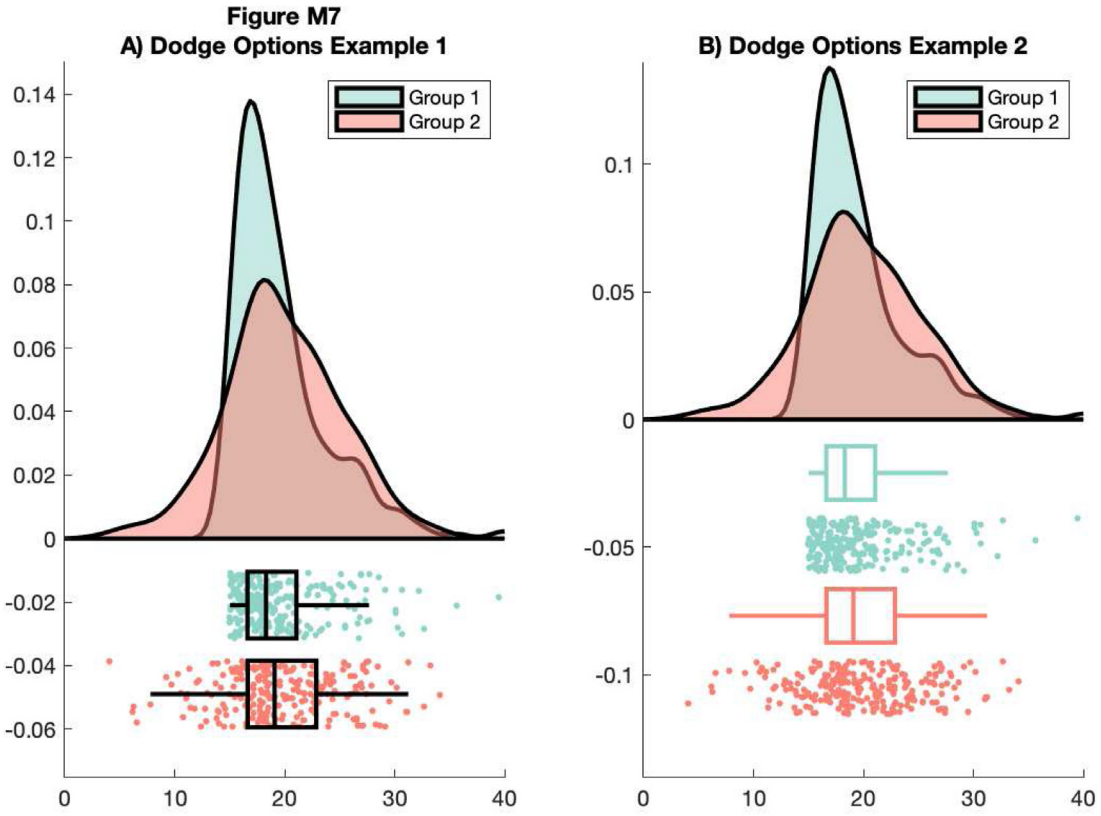
```
f6 = figure('Position', fig_position);
subplot(2, 1, 1)
h1 = raincloud_plot(d{1}, 'box_on', 1, 'color', cb(1,:), 'bandwidth', .2,
'density_type', 'ks');
title('Raincloud Plot: Reduced Smoothing, Kernel Density')
set(gca,'XLim', [0 40]);
box off
subplot(2,1,2)
h2 = raincloud_plot(d{1}, 'box_on', 1, 'color', cb(2,:), 'bandwidth', 1,
'density_type', 'rash');
title('Raincloud Plot: Rash Density Estimate')
set(gca,'XLim', [0 40]);
box off

% save
print(f6, fullfile(figdir, '6Rain4.png'), '-dpng');
```

Figure M6
Raincloud Plot: Reduced Smoothing, Kernel Density



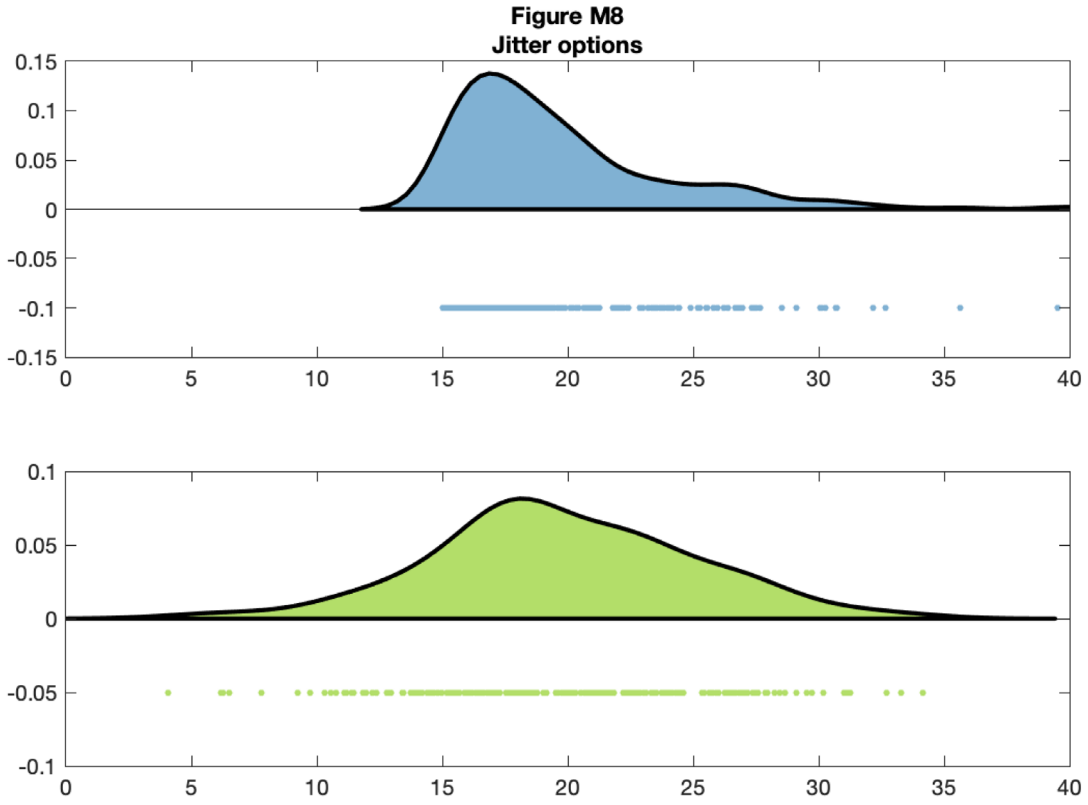
Here, we'll use the dot and box dodge options to create an overlapping set of raincloud plots, useful for group comparison. The function can be called repeatedly (e.g., from within a loop) - each iteration will overlay the previous. Note that here we're using the 'alpha' parameter to make the plot area see-through:

```
% example 1
f7 = figure('Position', fig_position);
subplot(1, 2 ,1)
h1 = raincloud_plot(d{1}, 'box_on', 1, 'color', cb(1,:), 'alpha', 0.5,...
    'box_dodge', 1, 'box_dodge_amount', .15, 'dot_dodge_amount', .15,...
    'box_col_match', 0);
h2 = raincloud_plot(d{2}, 'box_on', 1, 'color', cb(4,:), 'alpha', 0.5,...
    'box_dodge', 1, 'box_dodge_amount', .35, 'dot_dodge_amount', .35,
'box_col_match', 0);
legend([h1{1} h2{1}], {'Group 1', 'Group 2'})
title('A) Dodge Options Example 1')
set(gca,'XLim', [0 40], 'YLim', [-.075 .15]);
box off

% example 2
subplot(1, 2, 2)
h1 = raincloud_plot(d{1}, 'box_on', 1, 'color', cb(1,:), 'alpha', 0.5,...
    'box_dodge', 1, 'box_dodge_amount', .15, 'dot_dodge_amount', .35,...
    'box_col_match', 1);
h2 = raincloud_plot(d{2}, 'box_on', 1, 'color', cb(4,:), 'alpha', 0.5,...
    'box_dodge', 1, 'box_dodge_amount', .55, 'dot_dodge_amount', .75,...
    'box_col_match', 1);
legend([h1{1} h2{1}], {'Group 1', 'Group 2'})
title('B) Dodge Options Example 2')
set(gca,'XLim', [0 40]);
box off

% save
print(f7, fullfile(figdir, '7Rain5.png'), '-dpng')'
```



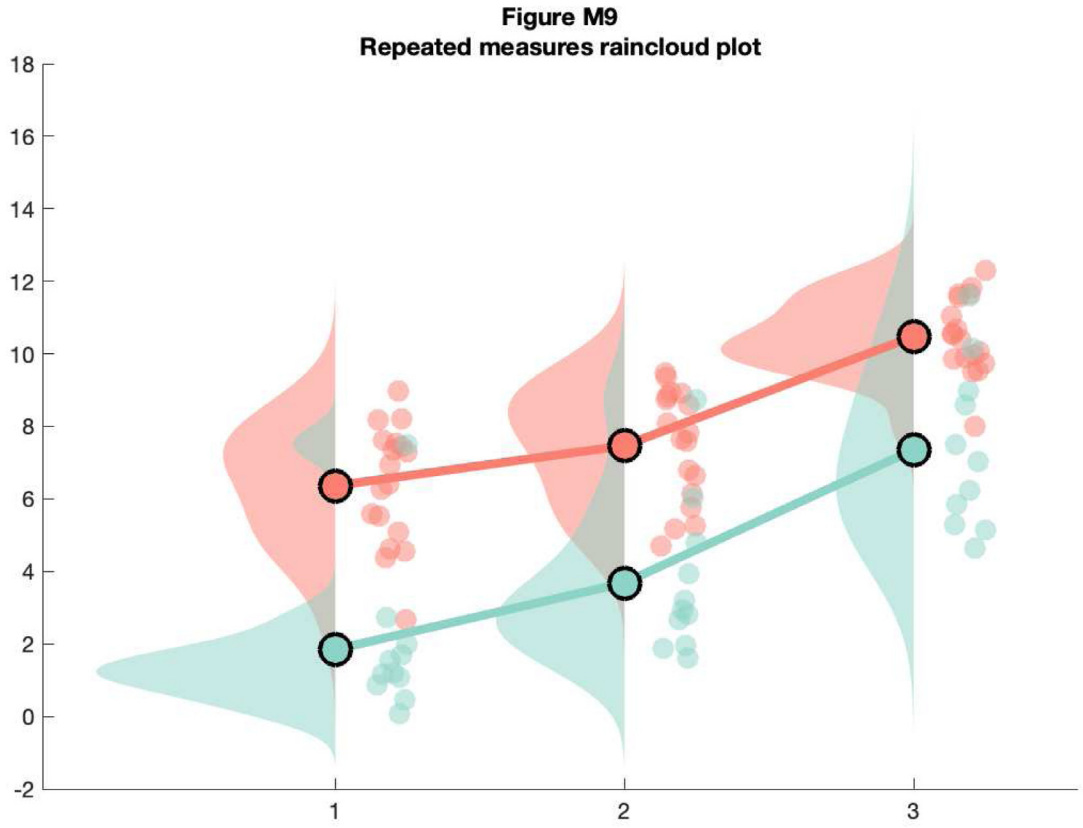
You can control the jitter and position of the 'raindrops' in the Y-plane by calling the figure handles:

```
f8 = figure('Position', fig_position);
subplot(2, 1, 1)
h1 = raincloud_plot(d{1}, 'color', cb(5,:));
set(gca,'XLim',[0 40]);
h1{2}.YData = repmat(-0.1, n, 1);

subplot(2, 1, 2)
h2 = raincloud_plot(d{2}, 'color', cb(7,:));
set(gca,'XLim',[0 40]);
h2{2}.YData = repmat(-0.05,n,1);

% save
print(f8, fullfile(figdir, '8Rain6.png'), '-dpng');
```



For the final examples, we'll consider a more complex factorial situation where we have multiple groups and observations. To illustrate this, we'll use a more complex implementation of raincloud encoded in the 'rm_raincloud.m' function.

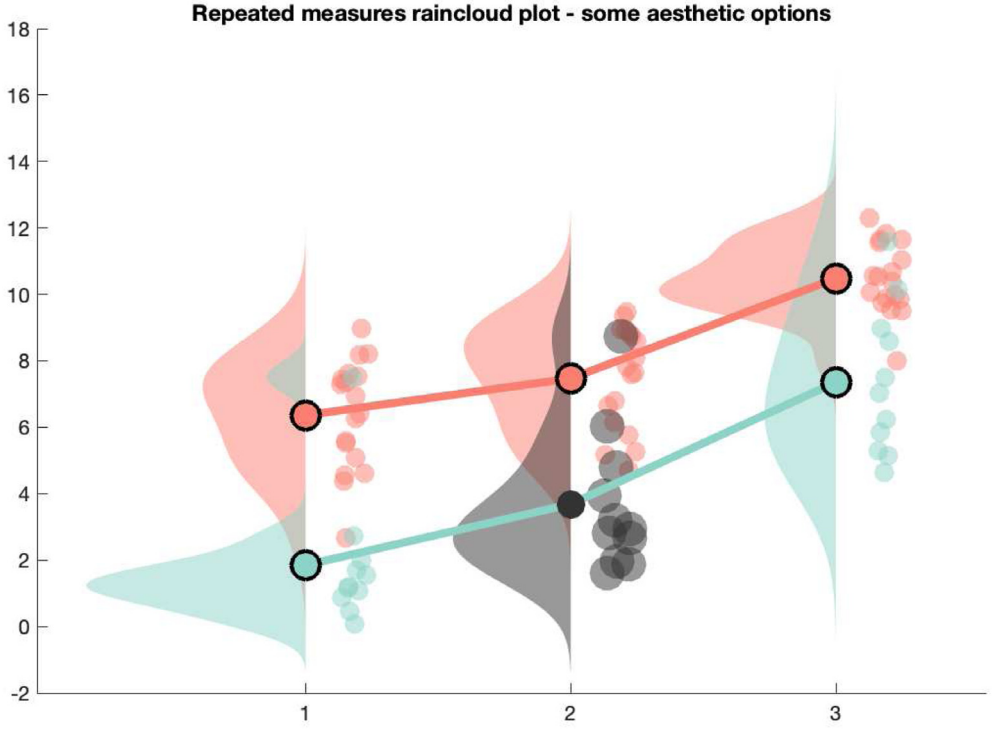
```
% grab 'repeated_measures_data.csv';
D = dlmread(fullfile(codedir, 'repeated_measures_data.csv'));

% read into cell array of the appropriate dimensions
for i = 1:3
    for j = 1:2
        data{i, j} = D(D(:, 2) == i & D(:, 3) ==j);
    end
end

% make figure
f9  = figure('Position', fig_position);
h   = rm_raincloud(data, cl);
set(gca, 'YLim', [-0.3 1.6]);
title('repeated measures raincloud plot');

% save
print(f9, fullfile(figdir, '9RmRain1.png'), '-dpng');
```



As above, 'rm_raincloud.m' returns a cell-array of handles to the various figure parts. We can add aesthetic options by calling these handles.

```
% make figure
f10 = figure('Position', fig_position);
h   = rm_raincloud(data, cl);
set(gca, 'YLim', [-0.3 1.6]);
title('repeated measures raincloud plot - some aesthetic options')

% define new colour
new_cl = [0.2 0.2 0.2];

% change one subset to new colour and alter dot size
h.p{2, 2}.FaceColor         = new_cl;
h.s{2, 2}.MarkerFaceColor   = new_cl;
h.m(2, 2).MarkerEdgeColor   = 'none';
h.m(2, 2).MarkerFaceColor   = new_cl;
h.s{2, 2}.SizeData          = 300;

% save
print(f10, fullfile(figdir, '10RmRain2`.png'), '-dpng');
```

Figure M10
Repeated measures raincloud plot - some aesthetic options



That's it! Now you should be ready to customize your Raincloud plots for a variety of different purposes. This concludes our cross-platform tutorial!

Discussion

We hope that our tutorials demonstrate the flexibility of raincloud plots for visualizing data. Raincloud plots build on a rich tradition of data graphics, enabling the user to visualize key parameters for statistical inference in a transparent an aesthetically appealing fashion. In this sense, Raincloud plots are part of a wider family of plotting tools such as beeswarms (Eklund, 2016), strip plots (Tukey, 1970), and estimation plots (Ho *et al*., 2018).

Indeed, our goal is not to argue for the superiority or novelty of raincloud plots over these and other complementary methods. Our focus is on providing a robust cross-platform tool for creating transparent plots. In general, the modularity of the raincloud plot is a strength, and we encourage the user to think carefully about the choice of individual elements (clouds, rain, & confidence intervals) depending on the particularities of their data.

It is worth mentioning that here we envision these three aspects of the raincloud plots as sub-serving particular statistical goals. In our examples, the probability distributions depicted by the split-half violin plot ('clouds') illustrate the sample variance. As such they are excellent tools for assessing how data are distributed and checking assumptions (i.e., violations of normality). Considering this, we caution against the use of clouds in this form for statistical inference at a glance, which is better served by comparing some parameter estimates in relation to their uncertainty. Users who wish to use probability distributions for inference should instead consider a more suitable approach such as estimation plots, or by plotting a smoothed histogram of bootstrapped parameter estimates, or simply by plotting raincloud with boxplots and/or confidence intervals, as we have done in our tutorial examples. The code provided with this tutorial makes it easy to implement whatever histogram function best suits the needs of the user, simply by substituting the PDF estimation function.

Additionally, at first glance it may seem redundant to plot both raw datapoints ('rain') and data distributions ('clouds'). However, we put forth that plotting both offers several advantages. First, plotting raw datapoints can enable the automated (i.e., machine-readable) recovery of data from plots even when the data underlying the plot has been lost. Second, plotting raw data can facilitate the identification of unexpected patterns

within the data, such as ordinality or outliers, which may not be readily apparent from a probability distribution or box-plot alone. As such we recommend the combination of raw data plots and smoothed distributions (however estimated) wherever possible.

In the spirit of open science and supporting each other in improving our data visualisations, we invite readers to contribute their own variations and extensions directly to our GitHub repository (https://github.com/RainCloudPlots/RainCloudPlots). Directions on how to contribute can be found in our contributing guidelines. We are particularly indebted to the Binder team (Jupyter *et al.*, 2018), part of Project Jupyter (http://jupyter.org), whose tool allows all users to explore the R and Python examples interactively from the browser.

Preprints, Pull Requests and the value of community science

This manuscript was originally published as a preprint on the Peerj platform (https://doi.org/10.7287/peerj.preprints.27137v1). The eight months since have illustrated the remarkable potential of new publishing infrastructure and landscape make the process of publishing scientific content faster, better and more collaboratively. We here outline just a few of the positives from doing so, and hope this may serve to encourage others. Firstly, posting the manuscript as preprint has vastly widened the reach. To date (March 2019) our preprint was viewed 9803 times, with 6309 downloads. However, views and downloads alone don't necessarily entail engagement. Since publication the preprint alone has already been cited 18 times. Moreover, in depth engagement has gone well beyond mere citations. Several individuals have created their own useful tutorials, summarizing our paper and asking useful questions, posted constructive criticism, discussed raincloud plots as part of various plotting alternatives, created a shiny app, wrote an accessible tutorial using native R datasets, a new package, creating various animated interactive visualisations (github here), used to illustrate the Binder format and used in more informal blogposts on e.g. superforecasting. Our codebase itself received feedback through various avenues including formal pull requests on github, comments on the preprint, twitter replies and email. In this new version of our paper we have tried our best to integrate all these suggestions and comments, which without fail have improved the usability of our code.

Social media, specifically twitter, provided the central hub where all these benefits coalesced. The paper has been tweeted at least 750 times, with an estimated reach of up to 1,500,000 total followers, and as such is the principal driver for the engagement our preprint has received. This engagement has yielded invaluable feedback, comments, and suggestions, and were even lucky enough to track down the first instance of an early precursor of the raincloud plot (Ellison, 2018). Moreover, the paper itself was inspired by a twitter discussion, and brings together co-authors who have never met in person. Together, these interactions illustrate the fundamentally two-way street of new publishing models, which facilitate access without paywalls and allow for near instantaneous improvements to ongoing work.

Conclusion

The future of data science lies in reproducible, robust methods that communicate our results to as wide of an audience as possible. We hope that raincloud plots will help you to better understand and communicate your own data-analysis. In the present paper, we've outlined some of the strengths of these plots compared to traditional methods such as bar or violin-plots. Using the attached code and tutorials, this paper opens up the raincloud plot to a wide variety of scientists in a multitude of disciplines.

Software availability

Code available from: https://github.com/RainCloudPlots/RainCloudPlots

Archived code as at time of publication: http://doi.org/10.5281/zenodo.1402959 (Allen *et al.*, 2018).

License: MIT

Grant information

MA is supported by a Lundbeckfonden Fellowship (R272-2017-4345), the AIAS-COFUND II fellowship programme that is supported by the Marie Skłodowska-Curie actions under the European Union's Horizon 2020 (Grant agreement no 754513), and the Aarhus University Research Foundation. KW is funded by the Alan Turing Institute under the EPSRC grant EP/N510129/1. RAK is supported by the Wellcome Trust (grant number 107392/Z/15/Z).

The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Acknowledgements

The authors wish to thank Jon Roiser who suggested the name "raincloud plots" and the many scientists and software engineers that wrote code upon which this tutorial builds. MA thanks Lincoln Colling for insightful discussions regarding raincloud plots and statistical inference.

References

Allen M, Poggiali D, Whitaker K, *et al.*: **RainCloudPlots tutorials and codebase (Version v1.0).** *Zenodo.* 2018.
http://www.doi.org/10.5281/zenodo.1402959

#barbarplots. 2016.
Reference Source

Bobko P, Karren R: **The Perception of Pearson Product Moment Correlations from Bivariate Scatterplots.** *Pers Psychol.* 1979; **32**(2): 313–325.
Publisher Full Text

Chambers JM: **Graphical Methods for Data Analysis.** Chapman and Hall/CRC. 2017.
Publisher Full Text

Eklund A: **beeswarm: the bee swarm plot, an alternative to stripchart**. R package version 0.2.3. 2016.
Reference Source

Ellison AM: **Exploratory data analysis and graphic display.** *Design and Analysis of Ecological Experiments.* 1993; 14–45.
Reference Source

Guess the Correlation. In *Wikipedia*. 2017.
Reference Source

Hintze JL, Nelson RD: **Violin plots: a box plot-density trace synergism.** *Am Stat.* 1998; **52**(2): 181–184.
Publisher Full Text

Ho J, Tumkaya T, Aryal S, *et al.*: **Moving beyond P values: Everyday data analysis with estimation plots.** *bioRxiv.* 2018.
Publisher Full Text

Jupyter P, Bussonnier M, Forde J, *et al.*: **Binder 2.0 - Reproducible, interactive, sharable environments for science at scale.** In F. Akici, D. Lippa, D. Niederhut, & M. Pacer (Eds.), *Proceedings of the 17th Python in Science Conference*. 2018; 113–120.
Publisher Full Text

Kampstra P: **Beanplot: A boxplot alternative for visual comparison of distributions.** *J Stat Softw.* 2008; **28**.
Publisher Full Text

Neuroconscience: **Introducing Raincloud Plots**! 2018a.
Reference Source

neuroconscience: **Introducing Raincloud Plots**! 2018b.
Reference Source

Patil I: **ggstatsplot: "ggplot2" Based Plots with Statistical Details**. CRAN. 2018.
Reference Source

Phillips N: **The pirate plot (2.0)—the RDI plotting choice of R pirates.** *R Bloggers.* 2016.
Reference Source

Piccinini P: **Boxplots vs. Barplots**. 2016.
Reference Source

Sidiropoulos N, Sohi SH, Pedersen TL, *et al.*: **SinaPlot: an enhanced chart for simple and truthful representation of single observations over multiple classes.** *J Comput Graph Stat.* 2018; **27**(3): 673–676.
Publisher Full Text

Spence ML, Dux PE, Arnold DH: **Computations underlying confidence in visual perception.** *J Exp Psychol Hum Percept Perform.* 2016; **42**(5): 671–682.
PubMed Abstract | Publisher Full Text

Team RC: **R: A language and environment for statistical computing**. 2013.

Tufte ER: **The Visual Display of Quantitative Information.** (Reprinted Ed edition). Cheshire, Conn: Graphics Press USA. 1983.
Reference Source

Tukey JW: **Exploratory Data Analysis, limited preliminary edition, three volumes.** *Reading: Addison-Wesley.* 1970; **71**: 293–316.

Weissgerber TL, Milic NM, Winham SJ, *et al.*: **Beyond bar and line graphs: time for a new data presentation paradigm.** *PLoS Biol.* 2015; **13**(4): e1002128.
PubMed Abstract | Publisher Full Text | Free Full Text

Wickham H: **A layered grammar of graphics.** *J Comput Graph Stat.* 2010; **19**(1): 3–28.
Publisher Full Text

Wickham H, Chang W: **ggplot2: An implementation of the Grammar of Graphics.** R Package Version 0.7. 2008.

Wilke C: **Ggridges: Ridgeline plots in'ggplot2'**. R Package Version 0.4.1. 2017.
Reference Source

Wilson AM, Hubel TY, Wilshin SD, *et al.*: **Biomechanics of predator-prey arms race in lion, zebra, cheetah and impala.** *Nature.* 2018; **554**(7691): 183–188.
PubMed Abstract | Publisher Full Text

xkcd: Violin Plots. (n.d.).
Reference Source

Zylberberg A, Roelfsema PR, Sigman M: **Variance misperception explains illusions of confidence in simple perceptual decisions.** *Conscious Cogn.* 2014; **27**: 246–253.
PubMed Abstract | Publisher Full Text

Open Peer Review

Current Peer Review Status: ✔ ✔

Version 1

Reviewer Report 17 April 2019

https://doi.org/10.21956/wellcomeopenres.16574.r35184

✔ **Elena Allen**
Rodin Scientific, LLC, Albuquerque, MN, USA

The authors present a software tool to facilitate the creation of more informative data visualizations. Specifically, the "raincloud" plot offers accessibility to distributional shape, statistical summaries, and individual data points. What more could you want? While the ideas underlying the raincloud aren't novel (and the authors do address violins, boxes, beans, rugs, strips, swarms, and other predecessors), the paper provides an excellent tutorial for the uninitiated who may be unfamiliar with better approaches to data visualization. It's one thing to point out the shortcomings of standard plots and decry, "we need to do better!"; it's quite another to provide a tool that gets the job done in a few lines of code. I am particularly impressed with the thoroughness of the code examples and the commitment of the authors to make their tools open and available to the largest possible audience. Moreover, the inclusion of R, Python, and Matlab will make almost everyone happy. Folks can get started with their favorite flavor right away. Well done.

I have only relatively minor comments and suggestions for revision.

1. In the Discussion, I'd like the authors to more fully address the downsides of a raincloud plot (every plot has them). Yes, you are adding information, but at what cost? Some things that come to mind: space and ink.

Space is always at a premium in publications and presentations of dense/high-dimensional datasets. When is the inclusion of a cloud, rain and summary stats justified? How might this depend on the user's **goals** in creating the visualization, rather than just the "particularities of their data". Also, does the separation between groups necessitated by the layout of the raincloud hamper a human's ability to make comparisons? For two-group comparisons, I've always found the "asymmetric violin" (e.g., Fig. 4 of Kampstra's Beanplot paper[1]) to be unmatched in it's ability to encourage comparison. How does a raincloud compare?

Likewise, it takes a good amount of ink to create clouds; if you've already displayed all the individual data points and an indication of centrality or statistical uncertainty, does the cloud give you anything new? In some cases it would seem to violate Tufte's guideline for data-ink ratio minimization. For me, the desired

complexity of the visualization and utility of different aspects is often a function of dataset size. For example, in Figure R10 the clouds are very pretty but I already knew everything I needed to know from the relatively few data points and boxplots. In fact, one could argue that there are too few points to support such kernel density estimates. A cloud might by more appropriate in a mid-size dataset (e.g., Figure R9) where one has difficultly estimating distributional shape by eye (of course, a simple beeswarm visualization addresses this limitation in full). When the dataset gets very large the raindrops (as instantiated here) become useless. The authors do "encourage the user to think carefully about the choice of individual elements", but I'd like them to go further and identify cases where aspects of the raincloud plot might be more or less useful.

2. A comment on footnote 1, regarding the widespread use of box plots: while I can fully agree that bar plots are over- and inappropriately used, the authors (or at least the founders of #barbarplots) seem to have forgotten that bar plots are an appropriate and intuitive visualization for counts, proportions, and frequencies, where there is no interesting distributional information to display and we really just care about how much or how often a thing happened. You could use a raincloud plot to display, e.g., the proportion of trials that were successful in a task, but this is just a distribution of 0's and 1's -- your cloud would be an odd one for sure, and the resulting visualization would be a non-intuitive representation of what you actually care about.

3. A comment on the criticism of violin plots (p. 4) and associated footnote 4: Have the authors considered that if they centered their cloud PDFs, they would take up just as much ink as a violin? I'm not sure "data-ink ratio" can be invoked here. Regarding "overly provocative" violin plots...I get that it's a funny comic and the authors are in on the joke. But this is a peer-reviewed tutorial article likely to be targeted at young, diverse scientists and researchers. The authors should hold themselves to a higher standard and perhaps reflect on the underlying and unintended misogyny that such a graphic perpetuates. Can violin plots look like vaginas? Sure. Can bar plots look like dicks? Yeah. Get over it and let's get back to work.

4. A few nit-nats on the tutorial so that the visualizations follow best practices:

- In R11 and R12, avoid using a dotted or hashed line to connect repeated measures - it is very difficult to distinguish those dots from the actual data points in the raindrops.
- In example M6, yellow has virtually no contrast on white, so we really can't see the raindrops. A poor choice for color encoding.
- In the code for example M3, there is an extra apostrophe before d{1} in h1 = raincloud_plot('d{1}, 'box_on', 1);
- In all the Matlab examples please **label the axes**. Axes have been purposefully labeled as "Group" and "Score", or "Time" and "Group" in R but neglected in Matlab. As someone who spent a long time characterizing how frequently authors fail to label their variables[2], it makes me cringe to see the same mistake being made in a tutorial.
- In the code for M9, I would include comments to help readers understand how/why data is being re-packaged into a cell array. For example:

```
D = dlmread(fullfile(codedir, 'repeated_measures_data.csv'));
% D is a structured as [value, time, group]
% read into cell array of the appropriate dimensions
nGroup = 2;
nTime = 3;
data = cell(nTime, nGroup);
for ii = 1:nTime
  for jj = 1:nGroup
    data{ii, jj} = D(D(:, 2) == ii & D(:, 3) == jj);
```

```
  end
end
```

References
1. Kampstra P: Beanplot: A Boxplot Alternative for Visual Comparison of Distributions. *Journal of Statistical Software*. 2008; **28** (Code Snippet 1). Publisher Full Text
2. Allen EA, Erhardt EB, Calhoun VD: Data visualization in the neurosciences: overcoming the curse of dimensionality.*Neuron*. 2012; **74** (4): 603-8 PubMed Abstract | Publisher Full Text

Is the rationale for developing the new software tool clearly explained?
Yes

Is the description of the software tool technically sound?
Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?
Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?
Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?
Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Very experienced in data visualization with Matlab expertise.

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Reviewer Report 15 April 2019

https://doi.org/10.21956/wellcomeopenres.16574.r35182

Lisa M. DeBruine
Institute of Neuroscience and Psychology, University of Glasgow, Glasgow, UK

This software tool article clearly describes a challenge (the prevalence of uninformative or misinformation plots) and presents a flexible solution with a strong logical rationale. Raincloud plots combine the advantages of box plots for inference, the advantages of distribution plots for assessing large-scale

patterns (e.g., bimodality), and the advantages of scatter plots for assessing smaller-scale patterns (e.g., ordinality). A particular strength of this tutorial is the flexibility of the approach; several modular aspects are presented to make it easy to customise plots to fit the nature of the data, while keeping the presentation consistent enough that readers will be able to easily orient themselves to slightly different styles.

I am only competent to assess the R tutorial, but it was easy to follow and very thorough. I look forward to seeing how this paper influences data visualisation in scientific research.

Minor comments

I'm not sure I agree with the "data-to-ink ratio" argument against violin plots. An alternative perspective is not that they are mirrored density plots, but *centered* density plots. You don't really save any ink with a density plot that takes up the same vertical and horizontal space as a violin plot; likewise, you can save equivalent ink by making the violin plot half it's original width, rather than halving it down the middle. A better argument is that the density plot version makes the y-axis meaningful (although your raincloud plots omit the scale on the y-axis and I doubt many people use the actual density values).

Code notes

I personally really hate code that installs anything (it's a violation of my computer). I'd make the package setup chunk default to eval = FALSE and include a comment to turn it on if they want to install the missing packages (you can even run the first part and print a message if any packages are missing). But I guess there is a balance between making code easy for novices to run and best practices for respecting the user's computer. I appreciate you taking the time to make sure installation only happens for uninstalled packages.

In the code chunk "colour_rc" you add trim=FALSE to the flat violin, but don't mention it and take it away in the next chunk.

In the chunk "striated" you change the alpha of the violin and point between ap1 and ap2, but don't explain why. Minimise irrelevant changes between steps to avoid confusing people and explain additions (the previous plot had no alpha) to avoid cargo-cult-like behaviour.

Consider making the ggplot code less dense. I've put an example of what I mean below. Apart from that, the R tutorial is really clear and useful!

Example of less dense (more readable) code:

```
p12 <- ggplot(rep_data, aes(x = group, y = score, fill = time))+
 geom_flat_violin(aes(fill = time),
                  position = position_nudge(x = .1, y = 0),
                  adjust = 1.5,
                  trim = FALSE,
                  alpha = .5,
                  colour = NA)+
 geom_point(aes(x = as.numeric(group)-.15, y = score, colour = time),
            position = position_jitter(width = .05),
            size = .25,
```

```
        shape = 20)+
 geom_boxplot(aes(x = group, y = score, fill = time),
          outlier.shape = NA,
          alpha = .5,
          width = .1,
          colour = "black")+
 geom_line(data = sumrepdat,
        aes(x = as.numeric(group)+.1,
          y = score_mean,
          group = time,
          colour = time),
        linetype = 3)+
 geom_point(data = sumrepdat,
         aes(x = as.numeric(group)+.1,
           y = score_mean,
           group = time,
           colour = time),
         shape = 18)+
 geom_errorbar(data = sumrepdat,
          aes(x = as.numeric(group)+.1,
            y = score_mean,
            group = time,
            colour = time,
            ymin = score_mean-se,
            ymax = score_mean+se),
          width = .05)+
 scale_colour_brewer(palette = "Dark2")+
 scale_fill_brewer(palette = "Dark2")+
 ggtitle("Figure 12: Repeated Measures - Factorial (Extended)")+
 coord_flip()
```

Is the rationale for developing the new software tool clearly explained?
Yes

Is the description of the software tool technically sound?
Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?
Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?
Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?
Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: I have substantial experience developing teaching materials for R stats, including ggplot.

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Comments on this article

Version 1

Reader Comment 14 Oct 2019

Hilmar Brohmer, Social Psychology, Institute of Psychology, University of Graz, Graz, Austria

Dear authors,

Thank you for these highly informative plots that hopefully - at some point - supersede conventional bar plots.

However, I have a suggestion regarding the jitter (i.e., the "rain"): I think jittered data can be deceiving because one tends to interpret the position of the dots in the figure, although the position is random in two dimensions. This could be especially problematic for extreme cases that could then look like outliers.

I might have three solutions how to treat the dots alternatively. 1) One could use stacked dot plots (Figure 2b) as "rain" in combination with a "cloud". However, because both rain and clouds might look somewhat similar, one could criticize that this plot might bear some redundance. 2) One could use overlapping dot plots similar to Figure R8 A or P3. In this kind of plot, transparency of the dots would be required so that an overlap of the dots (i.e., higher concentration of data) creates more intense / darker colors. On the downside, one cannot see easily, if there are 50 or 500 pp in the data because all the dots are on one line. 3) One could only jitter the dots in one dimension. For instance in Figure M9 (which I like most), one could jitter the rain only horizontally to some degree and keep the transparency of it. That way, data points would be vertically always on the "right" position and one would still get a good overview of the individual data.

I don't know if this idea is possible to implement easily, but it would certainly help.

Thank you and kind regards,

Hilmar Brohmer

Competing Interests: none