The hviPlotR package: Generating `plot.sas` style figures in R

John Ehrlinger
Quantitative Health Sciences
Lerner Research Institute
Cleveland Clinic

Abstract

We introduce the R package **hviPlotR**, a set of tools for creating publication quality graphics in R. The **hviPlotR** package is designed to replace the `plot.sas` macro we currently use in SAS. The package includes both R recipes for generating our standard graphics using **ggplot2** commands and a set of themes designed to format those figures for both manuscript and `PowerPoint` targets.

The goal of this package vignette is to introduce the **hviPlotR** methodology, as well as to document the best practices of creating our publication quality graphics for both manuscripts and power point presentations.

This document is included with the **hviPlotR** package as a package vignette, installed into R when the package is installed, and view able using the `vignette("hviPlotR")` command.

Keywords: publication graphics, powerpoint, ggplot2, plot.sas.

1. About this document

This document is an introduction to the R package **hviPlotR**, a set of tools for creating publication quality graphics in R. The package and this document describe the process of creating graphics in R that conform to the standards of the clinical investigations statistics group within The Heart & Vascular Institute at the Cleveland Clinic. These graphics are analogous to those generated with the `plot.sas` macro in SAS.

This document is the package vignette for the **hviPlotR** package, and as such is the primary documentation for the package. The latest version of the document can be obtained with the

```
R> vignette("hviPlotR", package = "hviPlotR")
```

The goal is to update this document as the package is updated to include all relevant changes for publication.

2. Introduction

For many years, the mainstay for generating graphics for manuscripts and presentations in the statistics group in HVI has been the `plot.sas` macro using SAS. However, recently, we

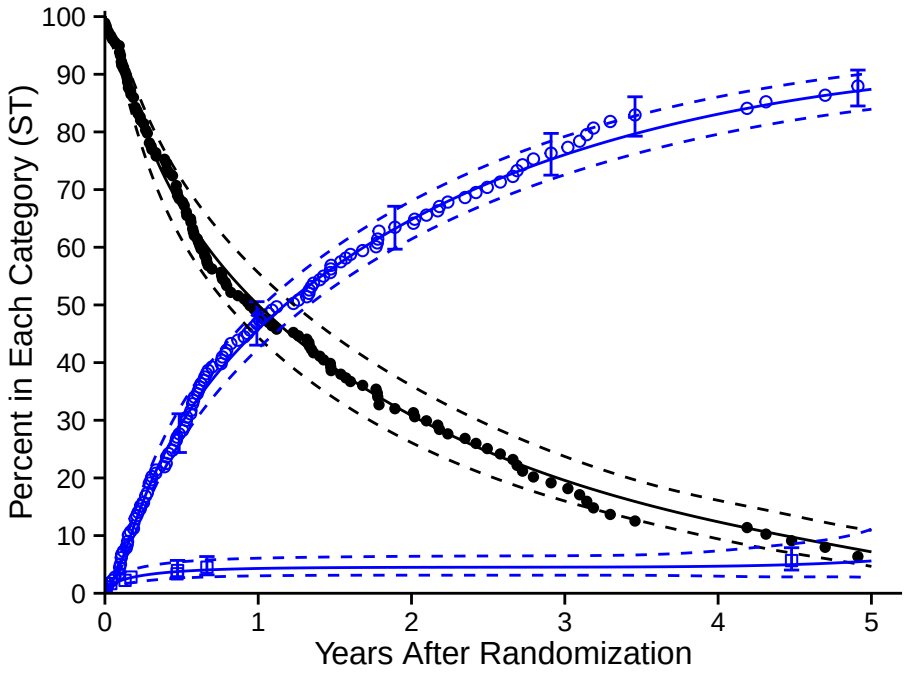


Figure 1: Demonstration figure

have had issues migrating this macro to newer versions of SAS (> 8.0) and MicroSoft Office products (> 2003).

In an effort to alleviate the versifying problems, and to standardize the generation of figures within R, we have developed the **hviPlotR** R package. The goal of the package, and this document, is simplify the creation of publication quality graphics in R. We are specifically encoding the best practices of the HVI Clinical Investigations formatting, so that our statisticians will be able to simply create graphics for publication with a minimal amount of effort.

The **hviPlotR** package also implements best practices for R graphics by leveraging the **ggplot2** package (Wickham 2009). The **ggplot2** package is an implementation of the Grammar of Graphics (Wilkinson 2005), which is a formalization of graphical concepts, and the building of graphical objects from a sequence of independent components. These components can be combined in many different ways.

The `plot.sas` macro is also an implementation of a graphics grammar. The grammar is derived from the ZETA pen plotters, which used GML (Graphics Machine Language) to control between 4 and and 8 colored pens for generating color line and point figures. Because both systems use a graphics language it is straight forward to translate commands between the two methods.

This document outlines how to generate figures using the **ggplot2** and **hviPlotR** packages. Our approach is to demonstrate the R commands to generate the same elements created with `plot.sas` commands. Section 3 gives an overview of the methodology of the `plot.sas` macro and Section 4 details how to create line and point plots with **ggplot2**. A key part of **hviPlotR** package is custom themes for figures. Once you have created your figure, Section 6 details how

```
%let STUDY=/studies/cardiac/valves/aortic/replacement/partner_publication_
    office/partner1b/mortality_5y
*************************************************************************;
* Bring in PostScript plot macro                                        ;
filename plt "!MACROS/plot.sas"; %inc plt;
filename gsasfile pipe 'lp';
*-----------------------------------------------------------------------;
*                                                                       ;
*                       P O S T S C R I P T   P L O T S
*-----------------------------------------------------------------------;
* Multiple decrement, nonparametric and parametric                      ;
filename gsasfile "&STUDY/graphs/ce.states.ST.ps";

*-----------------------------------------------------------------------;
* Create the figure here   !                                            ;
*-----------------------------------------------------------------------;
%plot(goptions gsfmode=replace, device=pscolor, gaccess=gsasfile end;
      id l="&STUDY/graphs/ce.states.ST.sas percent", end;
     labelx l="Years After Randomization", end;
     axisx order=(0 to 5 by 1), minor=none, end;
     labely l="Percent in Each Category (ST)", end;
     axisy order=(0 to 100 by 10), minor=none, end;
```

Listing 1: plot.sas commands: Figure setup.

to get the formatting correct for manuscripts or presentations. Section 7 describes functions for saving the figures to simplify the import into publication documents.

3. The `plot.sas` macro

To demonstrate the process, we first look at some example code using the `plot.sas` macro. This code is intended to generate a figure for manuscript publication and was modified to generate Figure 1.

Note the first line of the code block in Listing 1 indicates the path to the file location. The `filename` statments bring in the `plot.sas` macro, indicate how to print, and where to save the graphics file. The `plot.sas` macro call starts with the `%plot` command. The first line sets global graphic values, including the file where the figure will be saved (see Section 7). Each `plot.sas` command is terminated with the `end;` statement. We'll look at each command type individually.

The `id l=` command sets the footnote text used for manuscript figures to identify where the figure is saved (see Section 7). The `labelx` and `labely` commands set the axis label text (Section 4.2) and the `axisx` and `axisy` set the scales for each axis locating text and tics (Section 4.3).

The `tuple` command builds up graphics objects within the figure plot window. The first set of tuple commands builds up a set of three elements containing both points (Section 4.4) and errorbars (Section 4.5). Each `tuple` statement operates on a dataset indicated by the `set` command.

Symbols shapes and sizes are specified with the `symbol` and `symbsize` commands (Section 4.8).

```
/******NON-PARAMETRIC: SYMBOLS AND CONFIDENCE BARS *******/
tuple set=green, symbol=dot, symbsize=1/2, linepe=0, linecl=0,
     ebarsize=3/4, ebar=1,
     x=iv_state, y=sginit, cll=stlinit, clu=stuinit, color=black,
     end;
tuple set=green, symbol=circle, symbsize=1/2, linepe=0, linecl=0,
     ebarsize=3/4, ebar=1,
     x=iv_state, y=sgdead1, cll=stldead1, clu=studead1, color=blue,
     end;
tuple set=green, symbol=square, symbsize=1/2, linepe=0, linecl=0,
     ebarsize=3/4, ebar=1,
     x=iv_state, y=sgstrk1, cll=stlstrk1, clu=stustrk1, color=blue,
     end;
```

Listing 2: plot.sas commands: points and errorbar tuple statements.

```
/**********PARAMETRIC : SOLID LINES AND CONFIDENCE INTERVALS**********/
tuple set=all, x=years, y=noinit, cll=clinit, clu=cuinit,
     width=0.5,color=black,
     end;

tuple set=all, x=years, y=nodeath, cll=cldeath, clu=cudeath,
     width=0.5,color=blue,
     end;

tuple set=all, x=years, y=nostrk, cll=clstrk, clu=custrk,
     linecl=2, width=0.5,color=blue,
     end;
);
run;
*******************************************************************************;
```

Listing 3: plot.sas commands: lines tuple statements.

The second set of `tuple` statements build up a set of three elements containing lines and confidence intervals (Section 4.6).

The `plot.sas` macro code is closed by the ending `);` characters, and SAS is instructed to `run;` the code. Running combines building the figure by combining elements from `label`, `axis` and `tuple` statements and saving it into the file specified by the `gsasfile` variable. The resulting figure is shown in Figure 2.

Note that much of the figure formatting is mixed within the `tuple` statements using `width`, `color`, `linepe` and `linecl` commands. In the `plot.sas` macro, omitting these commands will generate a figure with the default values specified within the `device` theme (Section 6).

A similar set of `plot.sas` commands is used to create presentation graphics. Differences include the target `device` and `ftext` as well as some handling of figure labels with `value` instead of `label` commands. We also have rules for what to and not to include in presentation graphics (Section 9).

4. Generating ggplot2 graphics

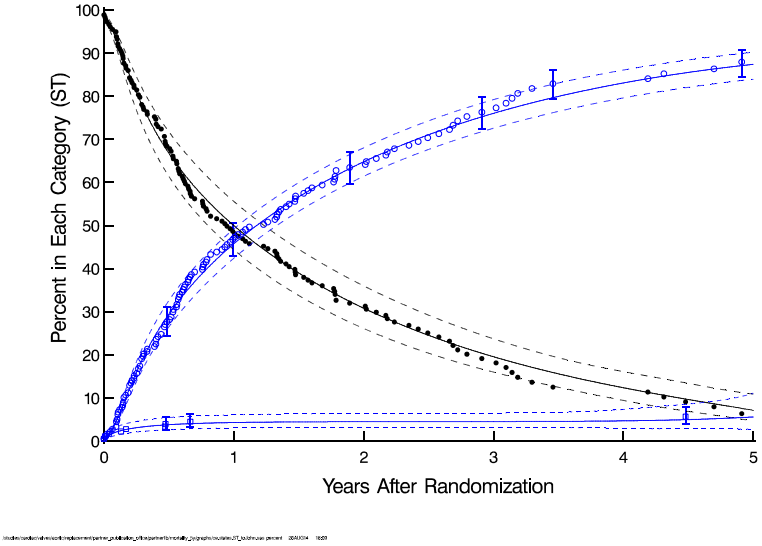


Figure 2: Manuscript figure (SAS version)

```
*-----------------------------------------------------------------------------;
*                                                                             ;
*       C G M   F I L E S   F O R   P O W E R P O I N T   S L I D E S
*-----------------------------------------------------------------------------;
* Competing risks, parametric only                                            ;
filename gsasfile "&STUDY/graphs/ce.states.ST.cgm";
%plot(goptions gsfmode=replace, device=cgmmppa, ftext=hwcgm001, end;
     axisx order=(0 to 5 by 1), minor=none, value=(height=2.4), end;
     axisy order=(0 to 100 by 20), minor=none, value=(height=2.4),
     value=(height=2.4 j=r ' ' '20' '40' '60' '80' '100'), end;
     tuple set=all, x=years, y=noinit, width=3, color=gray, end;
     tuple set=all, x=years, y=nostrk, width=3, color=red, end;
     tuple set=all, x=years, y=nodeath, width=3, color=blue, end;
);
run;
```

Listing 4: plot.sas commands: CGM instructions.

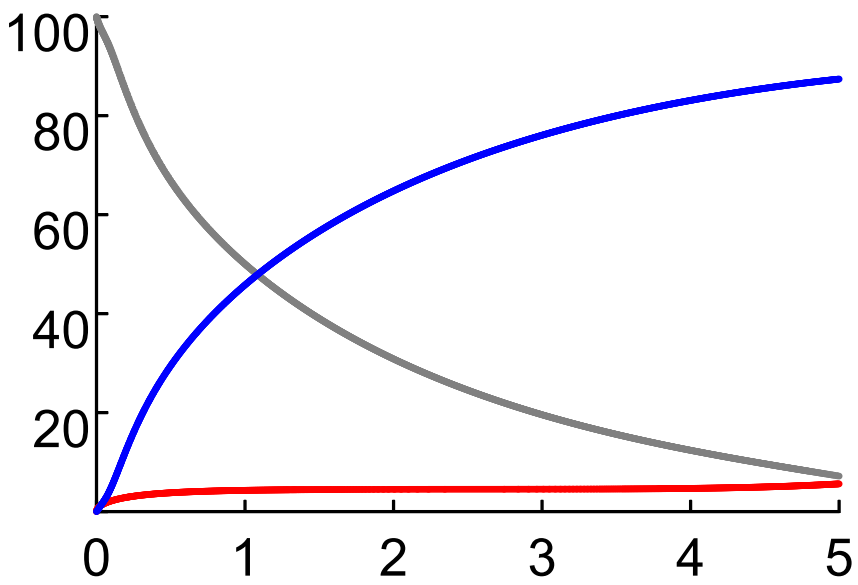


Figure 3: PowerPoint figure (SAS version)

In order to create figures similar to the `plot.sas` macro, using R, we will make extensive use of the **ggplot2** package. This will require translating from the graphics language of `plot.sas` to the graphics language of **ggplot2**.

For the remainder of this document, R code will be highlighted in grey boxes, as shown below. We will refer to these blocks as *code chunks*. You can run each code chunk individuallym, using copy/paste into an interactive R session, or a stand alone R script.

This tutorial requires the **hviPlotR** package for data and themes we will be discussing. You can load it with the following commands:

```
# Install the latest hviPlotR package.
#
# The devtools package is installed on all our
# jjnb-gen servers as well as other R instances.
library(devtools)

# To get the latest version.
install_github("ehrlinger/hviPlotR")
```

4.1. Initialize the figure

Refering back to the SAS code chunks in Section 3, the first section of the code sets the current working directory, and does some house keeping, including loading the `plot.sas` macro. Similarly, to get started in R, we first load the required libraries: **ggplot2** for graphics, and **hviPlotR** for themes. The following code chunk also sets the initial default theme to a generic black and white format, and brings in a pair of example datasets.

```
# load required libraries
library(ggplot2)   # Plotting environment
library(hviPlotR)  # CCF HVI plotting functionality

theme_set(theme_bw()) # A reasonable default plotting theme

# Load the example datasets
data(parametric, package="hviPlotR")
data(nonparametric, package="hviPlotR")
```

One advantage of **ggplot2** is that figures can be built up in successive statements. This tutorial will make extensive use of this to demonstrate the process. Starting in this code chunk, we will save the intermediate objects in the `ccf_plot` variable. Here we simply create an empty **ggplot2** figure that we will be adding to as we work through the commands in the `plot.sas` macro. Note that we include the `%plot()` command in the comment above the equivalent **ggplot2** command.

```
## To reproduce the plot.sas function, line by line.
###-------------
## There are SAS options we will not use here.
#
#  %plot(goptions gsfmode=replace, device=pscolor, gaccess=gsasfile end;
ccf_plot <- ggplot()
```

4.2. Labels

The next section of the SAS code in Section 3 sets the x and y axis titles, as well as the location of the majoe axis tick marks. We will splot this up for our R code. **ggplot2** uses a `labs` function to set the axis labels.

```
###-------------
## Labels are a single command, scales control the axis
#
#    labelx l="Years After Randomization", end;
#    labely l="Percent in Each Category (ST)", end;
ccf_plot <- ccf_plot +
  labs(x="Years After Randomization",
       y="Percent in Each Category (ST)")
```

The `labs` function can also be used to set the plot `title` and legend titles. We will not cover that functionality here, details are available in Wickham (2009) or through the Internet.

4.3. Scales

Axis ticks are controled with the `scale` functions. **ggplot2** has many different `scale` functions. These functions will work on one axis at a time, so for a typical continous axis, we refer to the `scale_x_continuous` or `scale_y_continuous` functions. Major axis are controlled using the `breaks` argument. This code uses a sequence of numbers to set the location of major tick marks (`seq(0,5,1)`). One mark for every year starting at 0, and ending at 5. Minor tick marks are automatically generated, but can also be specified using a `minor_breaks=` argument. You could also specifiy the breaks using a vector of values (`c(0,1,2,3,4,5)`), as well as relabel the ticks manually using a `labels=` argument.

Note that the `scale_` functions do not restrict the figure viewport at all. They are simply used to setup the axis tick marks. You can specify that the y-axis ticks are only from 0 to 50, and the figure would have a blank from 50 to the limits of the data. We discuss controlling the figure viewport in Section 4.10.

```
###-------------
## Labels are a single command, scales control the axis
#
#      axisx order=(0 to 5 by 1), minor=none, end;
#      axisy order=(0 to 100 by 10), minor=none, end;
ccf_plot <- ccf_plot +
  scale_x_continuous(breaks=seq(0,5,1))+
  scale_y_continuous(breaks=seq(0,100,10))
```

4.4. Points

Up to this point, we have only created and *decorated* the plot object stored in the `ccf_plot` variable. Showing the figure (`show`) or saving the figure would result in an error, since we have not added any data to the object, or described how we want it displayed.

The fundamental statement of the `plot.sas` macro is the `tuple` statement. The first `tuple` statement we see in the example code sets the *data* set (`set=green`), the symbol *shape* (`symbol=dot`), *size* (`symbsize=1/2`) and *color* (`color=black`). It turns off lines so only points will be shown (`linepe=0, linecl=0,`). It also handles error bars (`ebarsize=3/4, ebar=1`), which will be discuss in Section 4.5. The last line tells the macro about the point placement (`x=iv_state, y=sginit, cll=stlinit, clu=stuinit`) for the points (x, y) and upper (`clu`) and lower (`cll`) error bar limits.

The `geom_` set of functions in **ggplot2** is the functional equivalent to the `tuple` statement. The difference is the user specifies the graphical element desired using separate function calls. So points are plotting using the `geom_point` function, lines are generated with the `geom_line` (Section 4.6) and error bars are generated with the `geom_errorbar` function (Section 4.5).

Each of these functions takes a `data` argument, and an aesthetica function (`aes()`) is used to describe point within the graph using variables within the data set. The following code chunk demonstrates this use plotting the `iv_state` variable on the x-axis and the `sginit` variable along the y-axis. The variables are defined in the `nonparametric` data set we loaded in the setup code chunk in Section 4.

```
###-------------
## /******NON-PARAMETRIC: SYMBOLS AND CONFIDENCE BARS *******/
##
## Each tuple statement corresponds to one or more geom_ statements
#     tuple set=green, symbol=dot, symbsize=1/2, linepe=0, linecl=0,
#       ebarsize=3/4, ebar=1,
#       x=iv_state, y=sginit, cll=stlinit, clu=stuinit, color=black, end;

ccf_plot <- ccf_plot +
  geom_point(data=nonparametric, aes(x=iv_state, y=sginit))

show(ccf_plot)
```

Once we have added data to the `ggplot` object, we can display the figure as shown in Figure 4. Until know the figure has been manipulated by sequentially adding function calls to the

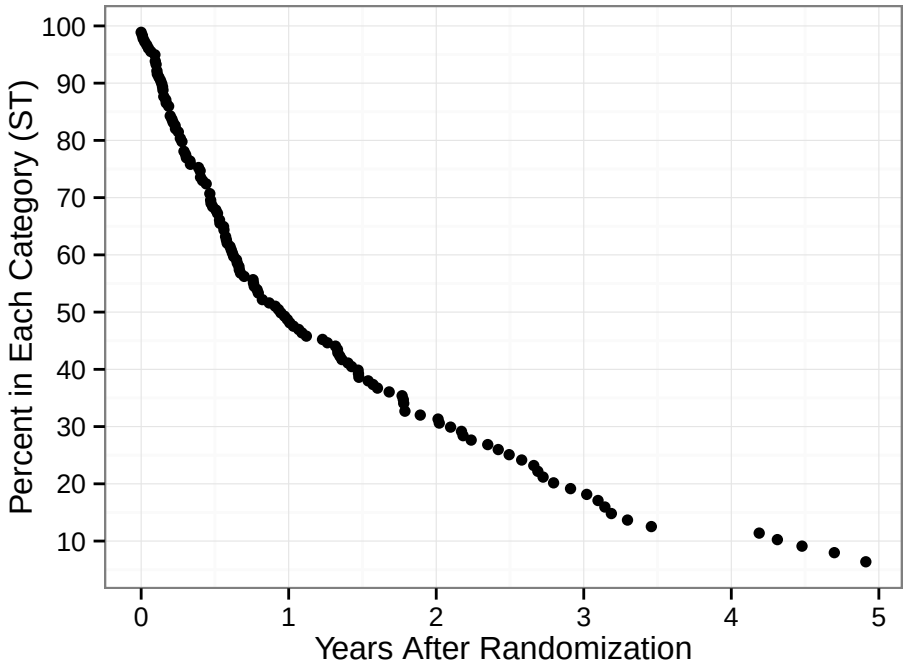


Figure 4: Point Plot

`ccf_plot` object. To display the figure you can either use the `show()` function, or simply use the object name at the command line.

Note that we have used the default *shape*, *size* and *color* for this figure. These can be manipulated by adding arguments to the `geom_` functions, outside of the `aes()` function, as we will demonstrate in the following sections.

4.5. ErrorBars

Instead of using a single function to set points, lines and error bars, **ggplot2** uses individual function calls to control these elements. The `geom_errorbar` function takes the same arguments as the other `geom_` functions. However, since an errorbar is defined with upper and lower limits, we need to supply an `ymax` and `ymin` argument to the graphic aesthetic function. This code chunk plots both points, and error bars for the next two data series, the `sgdead1` variable with errorbars running from `stldead1` to `studead1` and `sgstrk1` variable with errorbars running from `stlstrk1` to `stustrk1`. As we see in Figure 5, both series were added in `color="blue"`, with different point shapes (`shape=1` and `shape=0`). We manipulated the error bar size with the `width` argument

```
#     tuple set=green, symbol=circle, symbsize=1/2, linepe=0, linecl=0,
#       ebarsize=3/4, ebar=1,
#       x=iv_state, y=sgdead1, cll=stldead1, clu=studead1, color=blue, end;
ccf_plot <- ccf_plot +
  geom_point(data=nonparametric, aes(x=iv_state, y=sgdead1),color="blue",shape=1) +
  geom_errorbar(data=nonparametric, aes(x=iv_state, ymin=stldead1, ymax=studead1),
                color="blue", width=.1)
```

```
#      tuple set=green, symbol=square, symbsize=1/2, linepe=0, linecl=0,
#       ebarsize=3/4, ebar=1,
#       x=iv_state, y=sgstrk1, cll=stlstrk1, clu=stustrk1, color=blue, end;
ccf_plot <- ccf_plot +
  geom_point(data=nonparametric, aes(x=iv_state, y=sgstrk1),color="blue",shape=0) +
  geom_errorbar(data=nonparametric, aes(x=iv_state, ymin=stlstrk1, ymax=stustrk1),
                color="blue", width=.1)

show(ccf_plot)
```

```
Warning:  Removed 7 rows containing missing values (geom_point).
Warning:  Removed 117 rows containing missing values (geom_point).
```

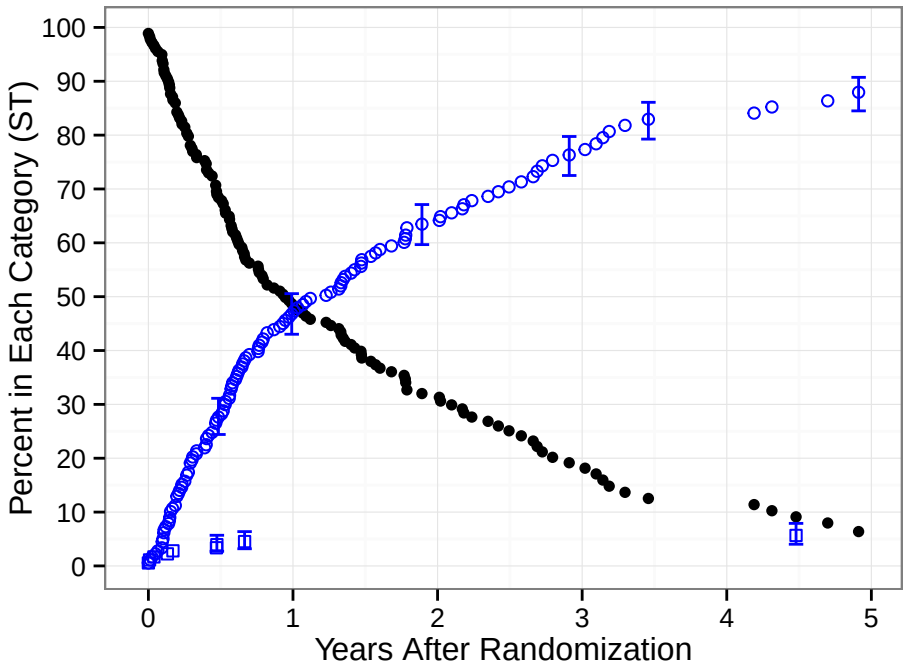


Figure 5: Error Bar Plot

Note that the `x` variable is the same for all three data series and the associated error bars. Also, since we do not want an error bar at every data point, a large number points have the upper and lower error bar variables set to `NA`. This is the same behavior as the `plot.sas` macro. **ggplot2** does print warnings when we attempt to plot a series with missing values. We typically suppress those warnings, but left them here for illustration purposes only.

4.6. Lines

Similar to points and error bars, the `geom_line` function is used to plot lines. We use the `linetype` argument to specify the line styles. We do have to generate a seperate `geom_line` function call for each limit of the confidence limit, since it is constructed of two lines (the upper and lower confidence limit). Alternatively, we can use the `geom_ribbon` to generate

the confidence band using a shaded region and only a single call. The aesthetic argument for `geom_ribbon` takes a `ymax` and `ymin` argument just as the `geom_errorbar` function.

```
# /**********PARAMETRIC : SOLID LINES AND CONFIDENCE INTERVALS**********/
# tuple set=all, x=years, y=noinit, cll=clinit, clu=cuinit,
# width=0.5,color=black, end;

ccf_plot <- ccf_plot+
  geom_line(data=parametric, aes(x=years, y=noinit))+
  geom_line(data=parametric, aes(x=years, y=clinit), linetype="dashed")+
  geom_line(data=parametric, aes(x=years, y=cuinit), linetype="dashed")
#
# tuple set=all, x=years, y=nodeath, cll=cldeath, clu=cudeath,
# width=0.5,color=blue, end;
ccf_plot <- ccf_plot+
  geom_line(data=parametric, aes(x=years, y=nodeath), color="blue")+
  geom_line(data=parametric, aes(x=years, y=cldeath), linetype="dashed", color="blue")+
  geom_line(data=parametric, aes(x=years, y=cudeath), linetype="dashed", color="blue")
#
# tuple set=all, x=years, y=nostrk, cll=clstrk, clu=custrk,
# linecl=2, width=0.5,color=blue, end;
ccf_plot <- ccf_plot+
  geom_line(data=parametric, aes(x=years, y=nostrk), color="blue")+
  geom_line(data=parametric, aes(x=years, y=clstrk), linetype="dashed", color="blue")+
  geom_line(data=parametric, aes(x=years, y=custrk), linetype="dashed", color="blue")

show(ccf_plot)
```

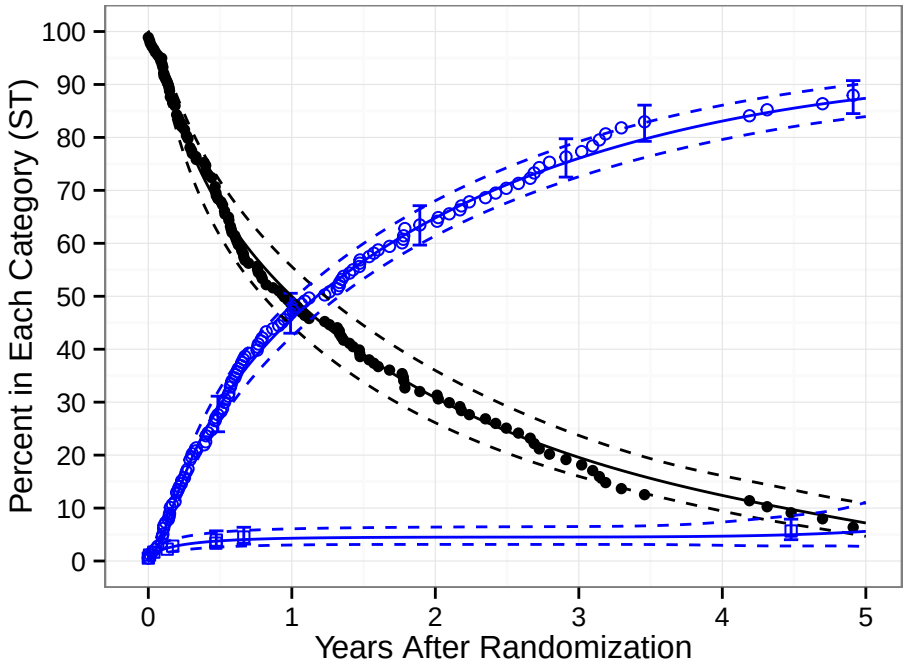


Figure 6: Line Plot with confidence bands

This time, we used the `parametric` data set in the `data` argument, as we want to use a different set of points for generating these lines.

4.7. Linetypes

The `linetype` argument takes a named string as a value, to set the different line styles. The set of styles is shown in Figure 7.

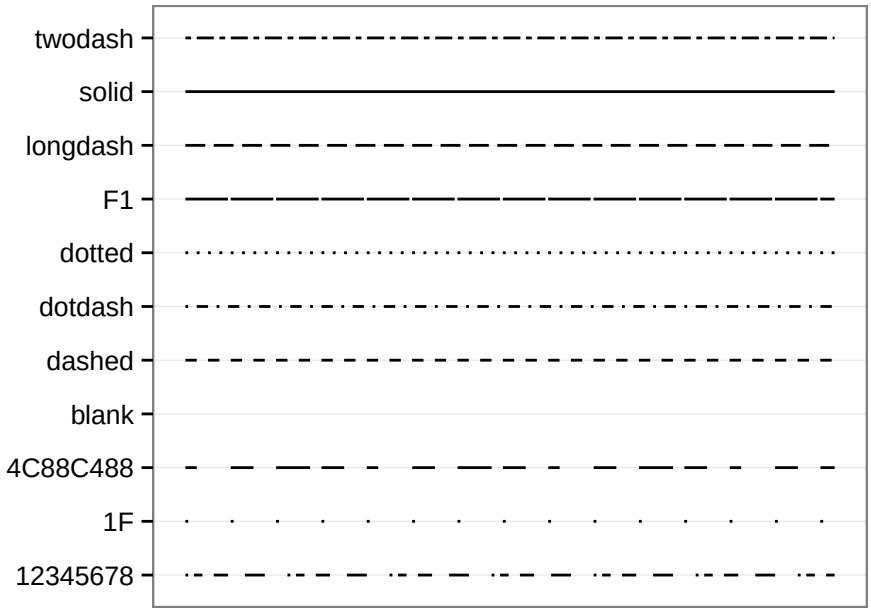


Figure 7: ggplot2 linetype table

4.8. Shapes

The `shape` argument takes numeric arguments. Though not user freindly, it is at least consistent. Figure 8 shows the catalog of shapes with corresponding numeric argument constructed using the ones place along the x-axis, and tens along the y-axis. So the filled dot, default point shape shown in black in Figure 6 is shape 20.

4.9. Colors

You can specify colors in R by numeric index, name (as we have done), hexadecimal, or RGB specification. For example `col=1` and `col="white"` are equivalent. The chart in Figure 9 was produced with code developed by Glynn (2005). See his R Color Chart website for all the details you would ever need about using colors in R.

Color theory encompasses a multitude of definitions, concepts and design applications - enough to fill several encyclopedias. However, there are three basic categories of color theory that are logical and useful : The color wheel, color harmony, and the context of how colors are used.

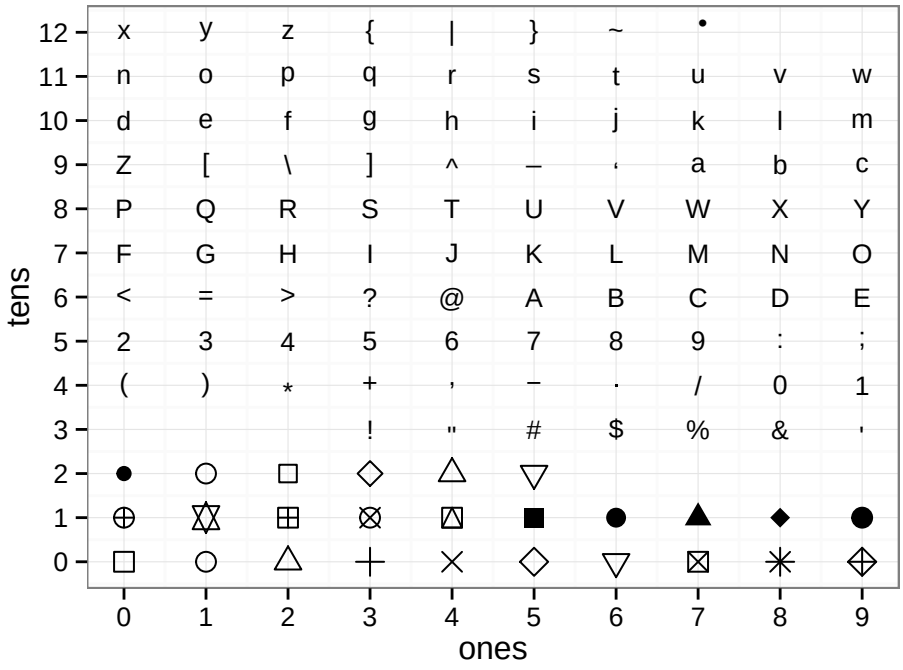


Figure 8: ggplot2 shape table

ColorBrewer (Harrower and Brewer 2003) is an online tool (`http://colorbrewer2.org/`) designed to help people select good color schemes for maps and other graphics. We encourage the use of ColorBrewer as a good, safe introduction to selecting colors based on theoretically good practices.

The **RColorBrewer** package citepNeuwirth:2011 simplifies the selection of ColorBrewer colors into R. We have used **RColorBrewer** to get a list of colors, and assign colors manually to specific variable values using the **ggplot2** `aes()` mechanism. The ColorBrewer palletes have also been built into the **ggplot2** `scale_` functions in the `scale_color_brewer` function. We have made extensive use of the `palette="Set1"` color palette in figures we have generated. There are also a series of other `scale_color_` functions in **ggplot2** to aid the user in selecting good color schemes for many different settings.

4.10. Global Commands

```
# Special commands to force origin to 0,0
ccf_plot <- ccf_plot +
  coord_cartesian(xlim=c(0,5.2), ylim=c(0,101))

show(ccf_plot)
```

5. PowerPoint Figures

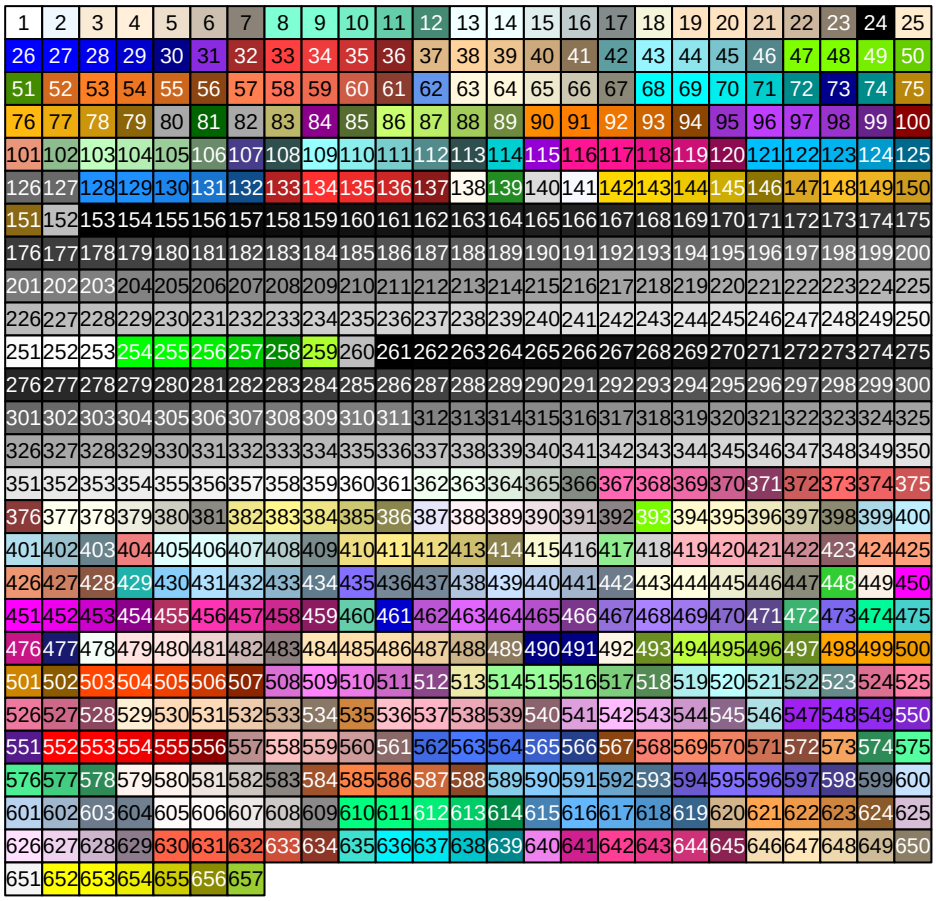


Figure 9: R colors

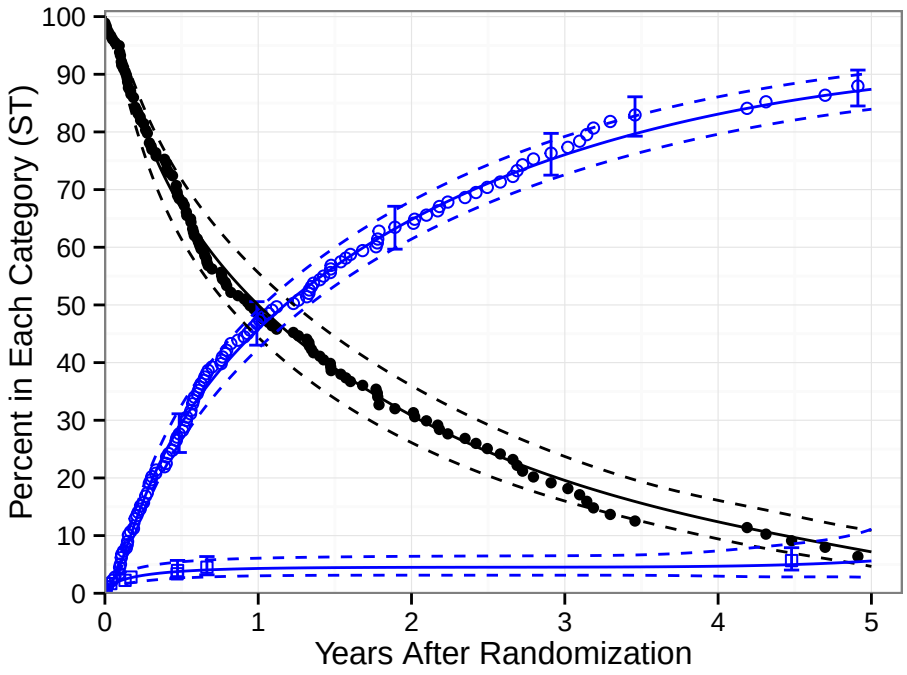


Figure 10: Adjusting the viewport

As a second example, we recreate a figure that was created for PowerPoint with the `plot.sas` macro. In most cases, we do not include points when generating presentation figures, so this figure was generated with only `geom_line` function calls. We also show how the figure can be created in a single set of function calls.

```
# %plot(goptions gsfmode=replace, device=cgmmppa, ftext=hwcgm001, end;
# axisx order=(0 to 5 by 1), minor=none, value=(height=2.4), end;
# axisy order=(0 to 100 by 20), minor=none, value=(height=2.4),
# value=(height=2.4 j=r ' ' '20' '40' '60' '80' '100'), end;
# tuple set=all, x=years, y=noinit, width=3, color=gray, end;
# tuple set=all, x=years, y=nostrk, width=3, color=red, end;
# tuple set=all, x=years, y=nodeath, width=3, color=blue, end;
# );
ccf_pptPlot <- ggplot()+
  scale_x_continuous(breaks=seq(0,5,1))+
  scale_y_continuous(breaks=seq(0,100,20))+
  geom_line(data=parametric, aes(x=years, y=noinit), color="grey", size=1.5)+
  geom_line(data=parametric, aes(x=years, y=nostrk), color="red", size=1.5)+
  geom_line(data=parametric, aes(x=years, y=nodeath), color="blue", size=1.5)

show(ccf_pptPlot)
```

6. ggplot2 themes for publication

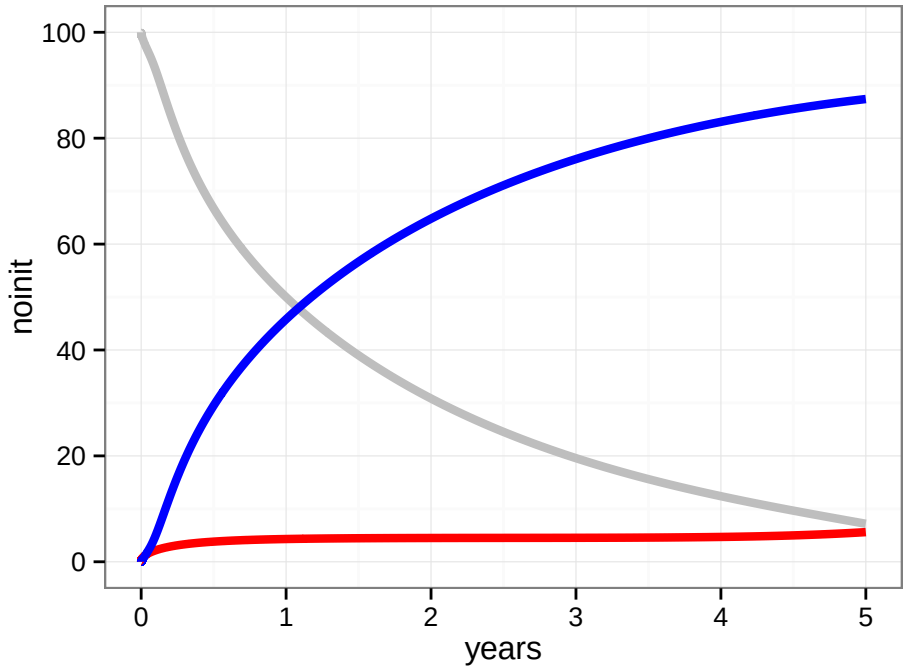


Figure 11: PowerPoint Figures

6.1. Theme for Manuscripts

```
# Set the theme for manuscripts,
theme_set(theme_man())

# show the figure.
ccf_plot

# Reset the theme to the resonable default used previously.
theme_set(theme_bw())
```

6.2. Theme for Presentations

```
# Update the PowerPoint Figure to include the PPT Theme, and remove axis labels.
# Axis labels will be added manually in powerpoint.
ccf_pptPlot <- ccf_pptPlot+
  labs(x="",y="")+
  theme_ppt()

# Show the figure... the theme statement is used so the axis tick marks and values
# are visible in this document.
ccf_pptPlot +
  theme(plot.background = element_rect(fill='blue', colour='blue'))
```

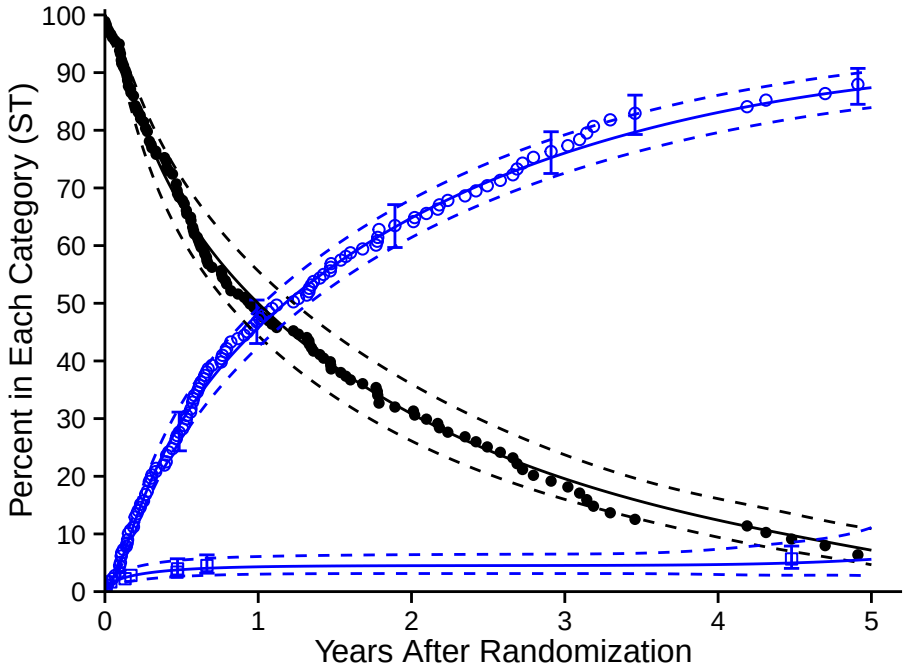


Figure 12: Theme for Manuscripts

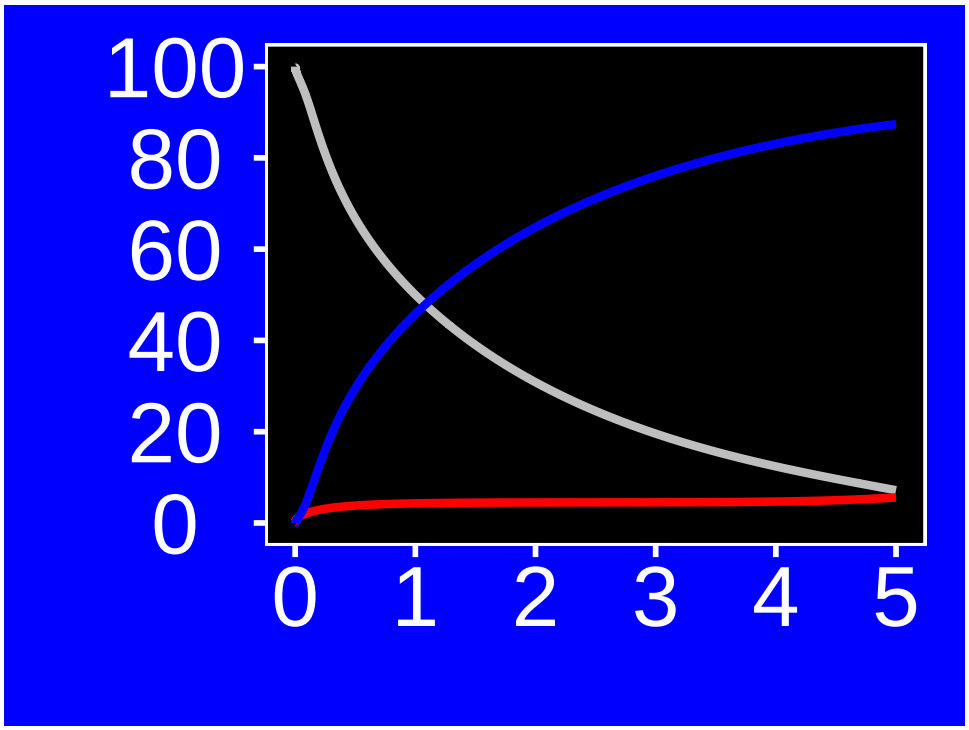


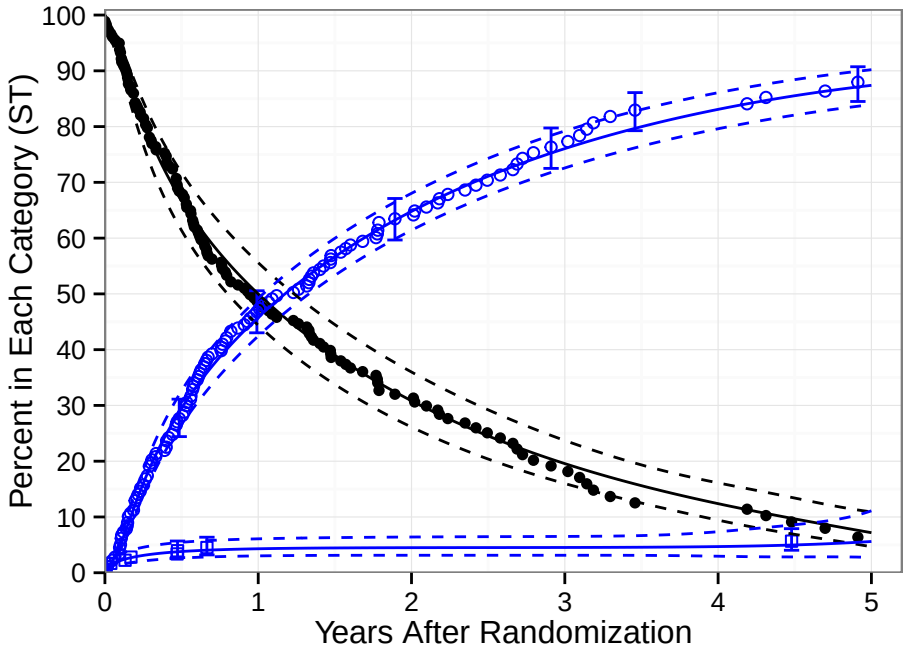
Figure 13: Theme for Presentations

7. Saving Publication graphics

7.1. Manuscript graphics

```
library(gridExtra)

ccf_savePlot <- arrangeGrob(ccf_plot,
                            sub = textGrob(getwd(),
                                           x = 0, hjust = -.01, vjust=1,
                                           gp = gpar(fontface = "italic", fontsize = 10)))
ccf_savePlot
```



/Users/ehrlini/Dropbox/Research/hviPlotR/vignettes

ggsav

7.2. PowerPoint graphics

We use the **ReporteRs** package (Gohel 2014) to insert vector based figures from R into PowerPoint documents. The latest version of the **ReporteRs** package is available from http://davidgohel.github.io/ReporteRs/. We install this package as we installed the **hviPlotR** package.

```
# Install the latest ReporteRs package.
#
# The devtools package is installed on all our
# jjnb-gen servers as well as other R instances.
library(devtools)

# To get the latest version.
install_github("davidgohel/ReporteRs")
```

Basically, the package works by opening a saved PowerPoint Presentation, and inserting new slides containing graphs or tables into the document. The resulting document is then saved to a new presentation. We then pass this presentation to our collaborators, who then copy and paste the **ggplot2** slides into their own presentations.

The **ggplot2** graphics that are inserted into the presentation are converted into an editable vector based format. When the document is edited in PowerPoint, graphical components like points, lines, text can be easily modified to match the presenters style.

The following code block is an R recipe for saving the `ccf_pptPlot` created in Section 6.2.

```
library(ReporteRs)
# Create a powerPoint document using ../inst/RDPresentation.pptx
# as a template document.
doc = pptx(template=paste("../inst/RDPresentation.pptx", sep=""))

# Here we define powerpoint document filename to write
# the presentation. This will be overwritten
pptx.file = paste("RDExample.pptx", sep="")

##--------
# For each graph, addSlide. The graphs require the
# âĂIJTitle and ContentâĂ▪ template.
doc = addSlide( doc, "Title and Content" )

# Place a title
doc = addTitle( doc, "Treatment Difference" )

# Now add the graph into the powerPoint doc
doc = addPlot( doc=doc, fun=print,
               x=ccf_pptPlot+theme_ppt() ,
               editable = TRUE,
               offx=.75, offy=1.1, width=8, height=6)
##--------
## IF you want to add more, just repeat between the ##-------- comments
##--------

# write the output powerpoint doc.
# This will not overwrite an open document, since open PPT files are locked.
writeDoc( doc, pptx.file )
```

The only modification possibly require for this recipe may be moving the insertion point (`offx` and `offy` arguments) and/or size (`width` and `height`) of the figure in the `addPlot()` function call.

8. Generating other figure types

8.1. Bar Charts

8.2. Histograms

8.3. Additional Figure Types

9. Graphics rules to live by

10. Conclusions

In this article, we present some functions in the **hviPlotR** package for R

References

Glynn EF (2005). "R Color Chart." http://research.stowers-institute.org/efg/R/Color/Chart/index.htm. Accessed: 2014-09-16.

Gohel D (2014). *ReporteRs: Microsoft Word, Microsoft Powerpoint and HTML documents generation from R*. R package version 0.6.1, URL http://davidgohel.github.io/ReporteRs/index.html,http://groups.google.com/group/reporters-package.

Harrower M, Brewer CA (2003). "ColorBrewer.org: An Online Tool for Selecting Colour Schemes for Maps." *The Cartographic Journal*, pp. 27–37. doi:10.1179/000870403235002042. URL http://colorbrewer2.org/.

Wickham H (2009). *ggplot2: elegant graphics for data analysis*. Springer New York. ISBN 978-0-387-98140-6.

Wilkinson L (2005). *The Grammar of Graphics (Statistics and Computing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA. ISBN 0387245448.

Affiliation:

John Ehrlinger
Quantitative Health Sciences
Lerner Research Institute
Cleveland Clinic
9500 Euclid Ave
Cleveland, Ohio 44195
E-mail: john.ehrlinger@gmail.com
URL: http://www.lerner.ccf.org/qhs/people/ehrlinj/
URL: https://github.com/ehrlinger/hviPlotR