Customizing Graphs Using R

Scott Richter

2018 QMS Summer Workshop Series

1 Base R Plotting

The data:

In November and December of 2003, the New Haven Fire Department administered oral and written exams for promotion to Lieutenant and Captain. Under the contract between the City of New Haven and the firefighter's union, the written exam received a weight of 60% and oral exam received a weight of 40%. Applicants with a total score of 70% or above pass the exam and become eligible for promotion. A total of 118 firefighters took the exam. Among them, 77 took the Lieutenant exam, and 41 took the Captain exam. During the time of the exams, there were 8 Lieutenant and 7 Captain positions available. The City Charter of New Haven specifies that when "g" promotions are made, the Department must select them from the top "g+2" scorers. Consequently, top 10 Lieutenant scorers and top 9 Captain scorers are eligible for potential promotion. The City of New Haven decided not to certify the exam and promoted no one, because an insufficient number of minorities would receive a promotion to an existing position. Ricci and other test-takers who would be considered for promotion had the city certified the exam sued the city for reverse discrimination. The District Court decided that the plaintiffs did not have a viable disparate impact claim. The appeals court confirmed the district court's ruling in Feb, 2008. On June 29, 2009, the Supreme Court decided that the City's failure to certify the tests was a violation of Title VII of the Civil Rights Act of 1964.

Of interest is examining differences in average test scores among the white, black and Hispanic test-takers.

```
ricci=read.csv(file="C:/Users/sjricht2/Box Sync/Consulting/Workshops/R Worksh
op II/Module_2/Data/RicciData.csv", header=T)
str(ricci)
```

```
## 'data.frame':    118 obs. of  5 variables:
##  $ Race    : Factor w/ 3 levels "B","H","W": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Position: Factor w/ 2 levels "Captain","Lieutenant": 1 1 1 1 1 1 1 1 2
2 ...
##  $ Oral    : num  82.4 68.6 71 52.4 67.6 ...
##  $ Written : int  70 74 70 77 56 50 53 49 76 86 ...
##  $ Combine : num  75 71.8 70.4 67.2 60.6 ...
```

We start by generating summary statistics by race.

```
tapply(ricci$Oral,
       ricci$Race,
       FUN=mean)

##        B        H        W
## 63.95222 57.03696 69.01765

tapply(ricci$Oral,
       ricci$Race,
       FUN=sd)

##        B        H        W
## 10.97465 10.99093 12.03918

tapply(ricci$Oral,
       ricci$Race,
       FUN=length)

##  B  H  W
## 27 23 68
```

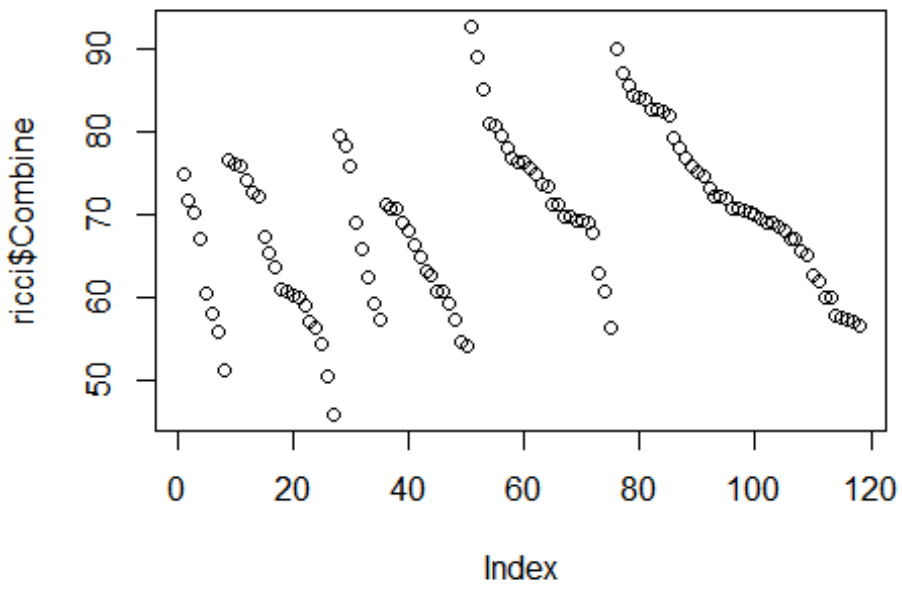
However, the **ddplyr** function in the *plyr* package allows multiple functions be specified and organizes the results conveniently into a table

```
library(plyr)
ddply(ricci,
      c("Race"),
      summarise,
      N = length(Oral),
      mean = mean(Oral),
      sd = sd(Oral),
      se = sd/sqrt(N))

##   Race  N     mean       sd       se
## 1    B 27 63.95222 10.97465 2.112072
## 2    H 23 57.03696 10.99093 2.291767
## 3    W 68 69.01765 12.03918 1.459965
```

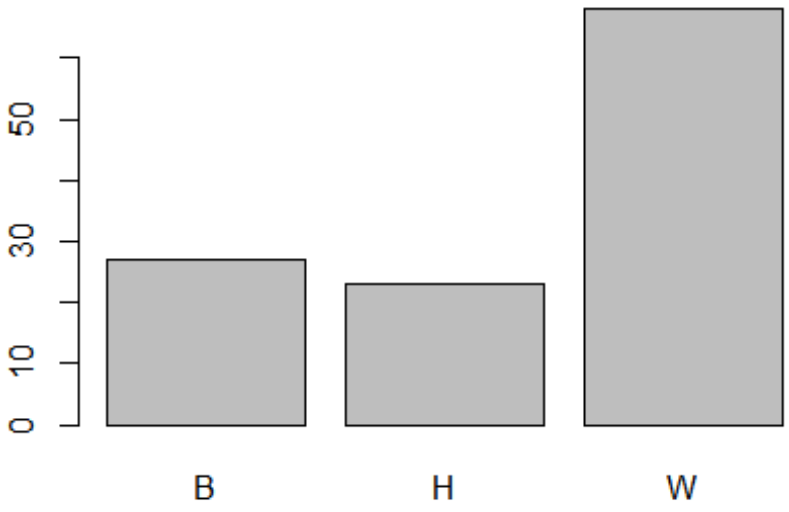
1.1 Basic plotting

plot is a general purpose plotting function. Supplied a single variable, it will produce a plot of the data values in the order they appear in the data frame or vector if numeric, or as a frequency bar chart if the variable is a factor.

```
plot(ricci$Combine)
```

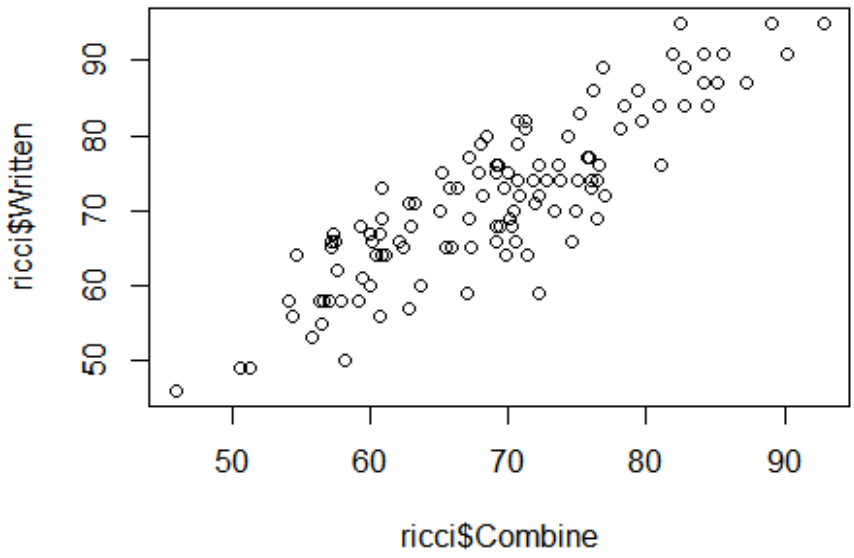



```
plot(ricci$Race)
```

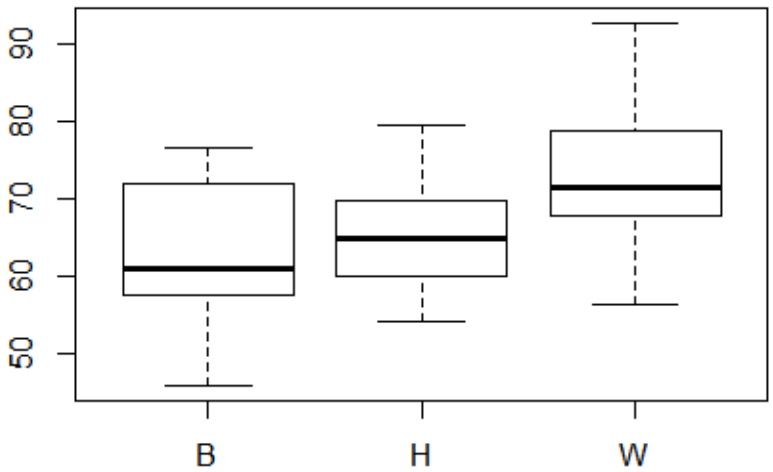


Supplied with two variables, it will produce a scatterplot if both variables are numeric or side-by-side boxplots if one of the variables is a factor.

```
plot(ricci$Combine, ricci$Written)
```

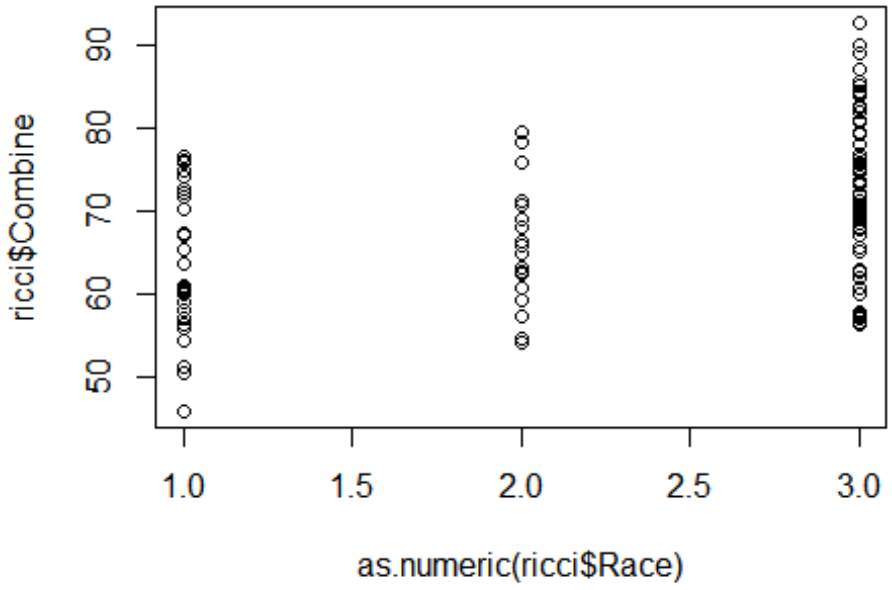



```
plot(ricci$Race,ricci$Combine)
```



The **as.numeric** fuction can be used to "coerce" the factor into a numeric variable, in which case side-by-side dotplots can be produced.

```
plot(as.numeric(ricci$Race),ricci$Combine)
```



Documentation for **plot** shows only a handful of arguments

plot {graphics} R Documentation

Generic X-Y Plotting

Description

Generic function for plotting of R objects. For more details about the graphical parameter arguments, see `par`.

For simple scatter plots, `plot.default` will be used. However, there are `plot` methods for many Robjects, including `function`s, `data.frame`s, `density` objects, etc. Use `methods(plot)` and the documentation for these.

Usage

```
plot(x, y, ...)
```

Arguments

x
: the coordinates of points in the plot. Alternatively, a single plotting structure, function or *any* **R** *object with a* `plot` *method* can be provided.

y
: the y coordinates of points in the plot, *optional* if x is an appropriate structure.

...
: Arguments to be passed to methods, such as graphical parameters (see `par`). Many methods will accept the following arguments:

  `type`
  : what type of plot should be drawn. Possible types are

    - `"p"` for **p**oints,
    - `"l"` for **l**ines,
    - `"b"` for **b**oth,
    - `"c"` for the lines part alone of `"b"`,
    - `"o"` for both '**o**verplotted',
    - `"h"` for '**h**istogram' like (or 'high-density') vertical lines,
    - `"s"` for stair **s**teps,
    - `"S"` for other **s**teps, see 'Details' below,
    - `"n"` for no plotting.

    All other `type`s give a warning or an error; using, e.g., `type = "punkte"` being equivalent to `type = "p"` for S compatibility. Note that some methods, e.g.`plot.factor`, do not accept this.

  `main`
  : an overall title for the plot: see `title`.

  `sub`
  : a sub title for the plot: see `title`.

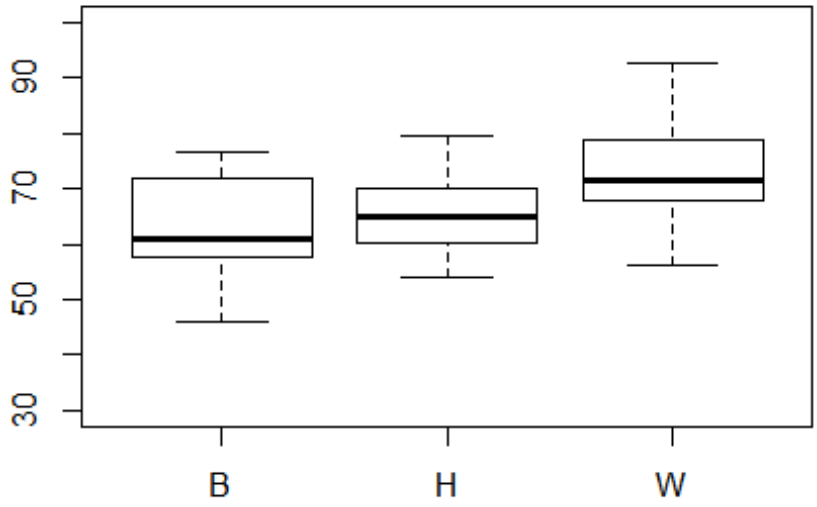
  `xlab`
  : a title for the x axis: see `title`.

  `ylab`
  : a title for the y axis: see `title`.

  `asp`
  : the *y/x* aspect ratio, see `plot.window`.

However, there are many other arguments that can be specified. For example, the y-axis limits can be specified:

```
plot(ricci$Race,ricci$Combine,
     ylim=c(30,100))
```



The **par** graphics function can be used to set many plotting parameters:

(https://www.rdocumentation.org/packages/graphics/versions/3.4.3/topics/par)

Here is a nice brief overview: (https://www.statmethods.net/advgraphs/parameters.html)

We will discuss more plotting options in the next section.

1.2 Generating a Means Plot

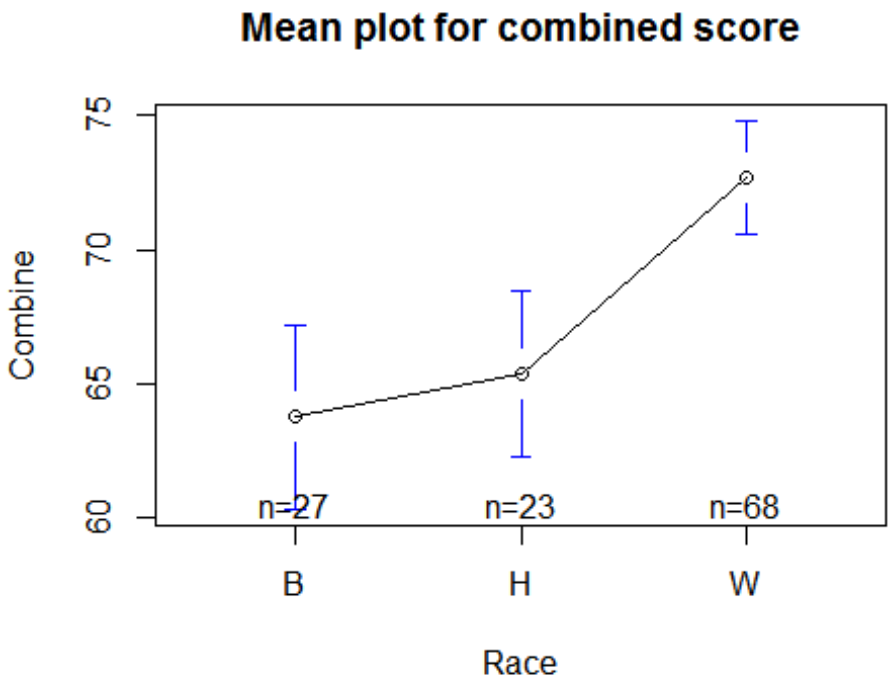
We will investigate generating and customizing a means plot. We will first generate a single plot for the Combined score by Race, and then three plots, one for each score. We will use the **plotmeans** function in the *gplots* package.

*Syntax: plotmeans(formula, data=NULL, subset, na.action, bars=TRUE, p=0.95, minsd=0, minbar, maxbar, xlab=names(mf)[2], ylab=names(mf)[1], mean.labels=FALSE, ci.label=FALSE, n.label=TRUE, text.n.label="n=", digits=getOption("digits"), col="black", barwidth=1, barcol="blue", connect=TRUE, ccol=col, legends=names(means), xaxt, use.t=TRUE, lwd=par("lwd"), ...)

We start with a basic plot of the means by race for the combined score, then add a title.

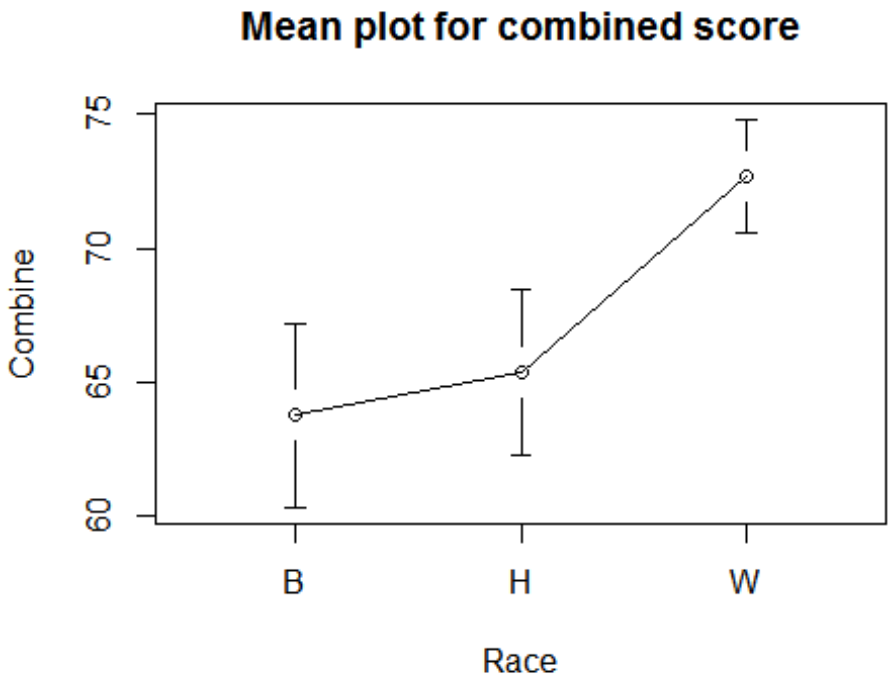
```
library(gplots)
```

```
##
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':
##
##     lowess

plotmeans(Combine ~ Race, data=ricci)

title("Mean plot for combined score")
```



Notice that the CI bars are blue, which in grayscale will appear lighter than the black lines connecting the means. We can add the *barcol="black"* option in **plotmeans** to change this. Also, we may not want the sample sizes printed on the plot, and thus also add *n.label=FALSE*.

```
plotmeans(Combine~ Race, data=ricci,
          barcol="black",
          n.label=FALSE)

title("Mean plot for combined score")
```



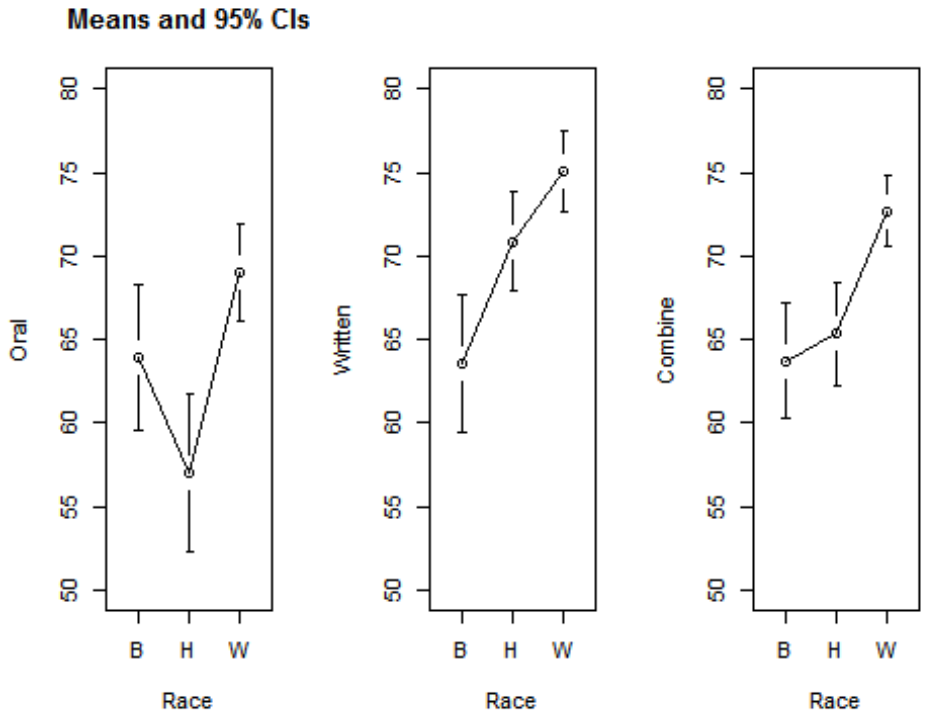
Now we want to create means plot for all three scores separately and place them side-by-side in a panel plot.

We illustrate three new options below:

- *mfrow=c(1,3)* creates a 1x3 template in which to place results;
- *ylim=c(50,80)* specifies the y-axis limits.
- *main=* places a title at the top of the plot
- *mfrow=c(1,1)* sets the results template back to "normal"

```
par(mfrow=c(1,3))

plotmeans(Oral~Race,
          data=ricci,
          ylim=c(50,80),
          barcol="black",
          n.label=FALSE,
          main='Means and 95% CIs')

plotmeans(Written~Race,
          data=ricci,
          ylim=c(50,80),
```

```
          barcol="black",
          n.label=FALSE)

plotmeans(Combine~ Race,
          data=ricci,
          ylim=c(50,80),
          barcol="black",
          n.label=FALSE)
```

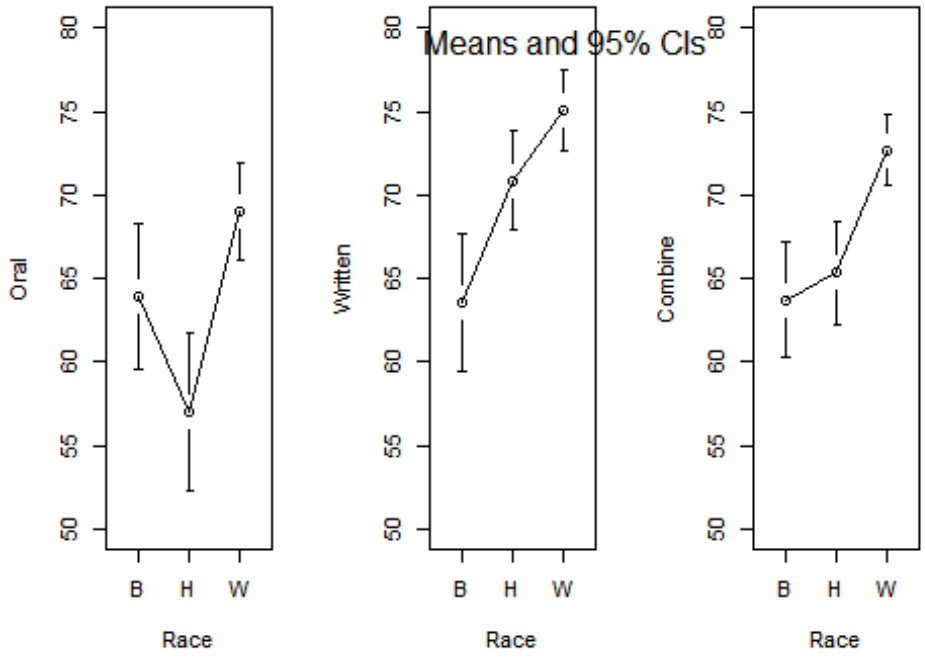



```
par(mfrow=c(1,1))
```

Notice that the title appeared only above the first plot. In order to have the plot heading centered over the entire plot, we redraw without the *main* option and use the *mtext* option instead.

```
par(mfrow=c(1,3))

plotmeans(Oral~ Race,
          data=ricci,
          ylim=c(50,80),
          barcol="black",
          n.label=FALSE)
```

```
plotmeans(Written~ Race,
          data=ricci,
          ylim=c(50,80),
          barcol="black",
          n.label=FALSE)

plotmeans(Combine~ Race,
          data=ricci,
          ylim=c(50,80),
          barcol="black",
          n.label=FALSE)

par(mfrow=c(1,1))

mtext("Means and 95% CIs")
```



The title doesn't look quite as we would like. Let's investigate options for the **mtext** function.

Write Text into the Margins of a Plot

Description

Text is written in one of the four margins of the current figure region or one of the outer margins of the device region.

Usage

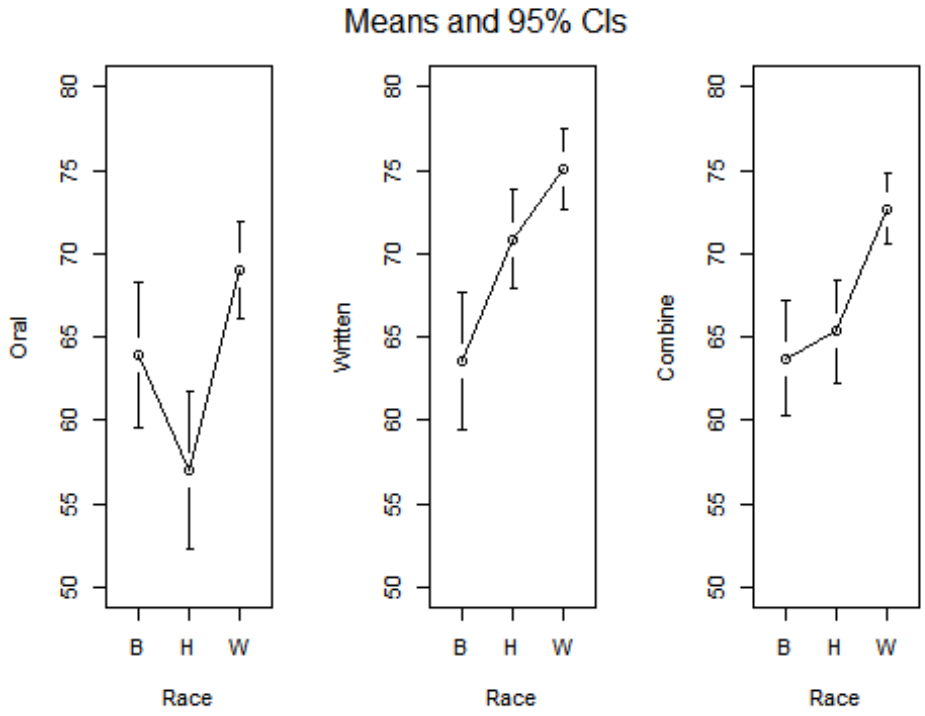
```
mtext(text, side = 3, line = 0, outer = FALSE, at = NA,
      adj = NA, padj = NA, cex = NA, col = NA, font = NA, ...)
```

Arguments

`text`	a character or expression vector specifying the *text* to be written. Other objects are coerced by `as.graphicsAnnot`.
`side`	on which side of the plot (1=bottom, 2=left, 3=top, 4=right).
`line`	on which MARgin line, starting at 0 counting outwards.
`outer`	use outer margins if available.

Thus, it appears that we can fix the heading by specifying a combination of *side, outer* and *line* functions.

```
par(mfrow=c(1,3))

plotmeans(Oral~ Race, data=ricci,
          ylim=c(50,80),
          barcol="black",
          n.label=FALSE)

plotmeans(Written~ Race, data=ricci,
          ylim=c(50,80),
          barcol="black",
          n.label=FALSE)

plotmeans(Combine~ Race, data=ricci,
          ylim=c(50,80),
          barcol="black",
          n.label=FALSE)

par(mfrow=c(1,1))

mtext("Means and 95% CIs", side = 3, outer = T,line = -2)
```



1.3 Further Customization

In this section we will discuss how to customize the plot to satisfy some requirements specifed by the journal *Science*.

Journal plot requirements

Resolution.

- Raster line art should have a minimum resolution of 600 dots per inch (dpi) and, preferably, should have a resolution of 1200 dpi.
- Figures should be sized to fit on single 8.5" × 11" sheets.

Figure layout and scaling.

- Panels should be set close to each other, and common axis labels should not be repeated.
- Avoid using y-axis labels on the right that repeat those on the left.
- Use solid symbols for plotting data if possible.

- Line widths should be legible upon reduction (minimum of 0.5 pt at the final reduced size).

Typefaces and labels

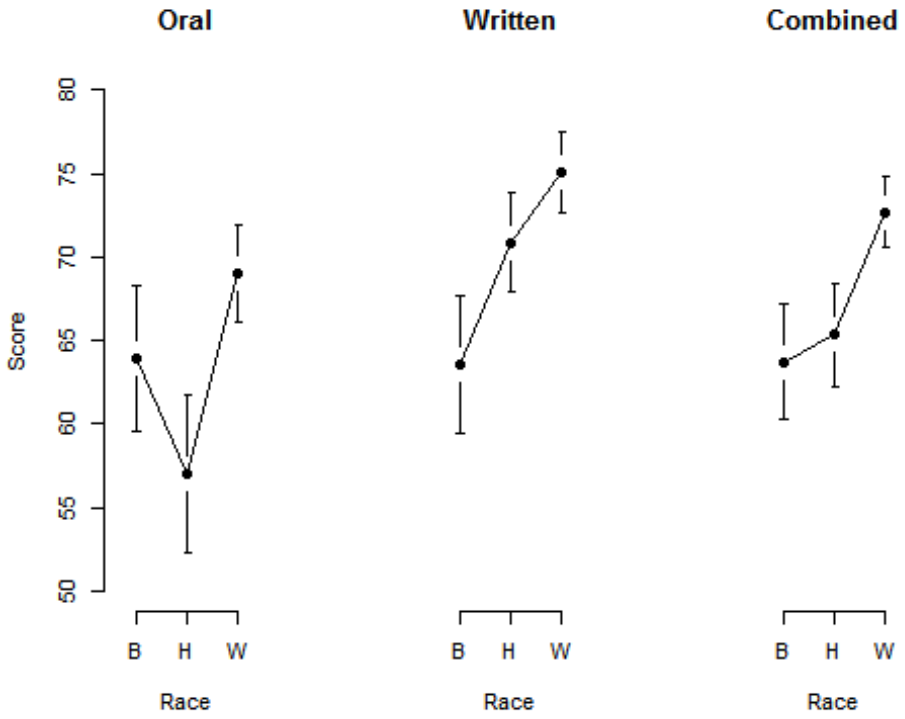
- Use a sans-serif font whenever possible (we prefer Helvetica).

In the following we add general plotting options to

- increase the line thickness (*lwd=1.5*)
- draw solid circles instead of open (*pch=19*)
- omit the box around each plot (*bty='n'*)
- maximize the plotting area (*pty='m'*)
- print a label under each x-axis (*xlab='Race'*)
- print labels on the y-axes (*ylab='Score', "", ""*)

```
par(mfrow=c(1,3))

plotmeans(Oral~Race, data=ricci,
                     ylim=c(50,80),
                     barcol="black",
                     n.label=FALSE,
                     lwd=1.5,
                     pch=19,
                     xlab="Race",
                     ylab="Score",
                     main='Oral',
                     bty='n',
                     pty='m')

plotmeans(Written~Race, data=ricci,
                        ylim=c(50,80),
                        barcol="black",
                        n.label=FALSE,
                        lwd=1.5,
                        pch=19,
                        xlab="Race",
                        ylab="",
                        main='Written',
                        yaxt='n',
                        bty='n',
                        pty='m')

plotmeans(Combine~Race, data=ricci,
                        ylim=c(50,80),
```

```
                    barcol="black",
                    n.label=FALSE,
                    lwd=1.5,
                    pch=19,
                    xlab="Race",
                    ylab="",
                    main='Combined',
                    yaxt='n',
                    bty='n',
                    pty='m')
```




```
par(mfrow=c(1,1))
```

BMP, JPEG, PNG and TIFF image files can be created using a set of related functions. PDF graphics files can also be created.

These functions all start *graphics devices*, which do not return an object. The **dev.off()** function shuts down the graphics device and allows the image to be created and saved. The plot is not displayed, but instead is saved in the working directory, which we specify using the **setwd** function.

We illustrate the syntax of the first set of graphics devices using the **tiff** function:

```
tiff(filename = "Rplot%03d.tif",
     width = 480, height = 480, units = "px", pointsize = 12,
     compression = c("none", "rle", "lzw", "jpeg", "zip",
     "lzw+p", "zip+p"), bg = "white", res = NA, family = "",
     restoreConsole = TRUE, type = c("windows", "cairo"),
     antialias)
```

The options below specify size 8.5x11 inches and a resolution of 1200 dpi. (We also specify *family = sans* to use a sans-serif font, a requirement we will meet later)

```
setwd("C:/Users/sjricht2/Box Sync/Consulting/Workshops/R Workshop II/Module_2
/revised version/")

tiff("Plot1.panel.tiff",
     width = 8.5,
     height = 11,
     units='in',
     res=1200,
     family="sans")

par(mfrow=c(1,3))

plotmeans(Oral~Race, data=ricci,
                     ylim=c(50,80),
                     barcol="black",
                     n.label=FALSE,
                     lwd=1.5,
                     pch=19,
                     xlab="Race",
                     ylab="Score",
                     main='Oral',
                     bty='n',
                     pty='m')

plotmeans(Written~Race, data=ricci,
                        ylim=c(50,80),
                        barcol="black",
                        n.label=FALSE,
                        lwd=1.5,
                        pch=19,
                        xlab="Race",
                        ylab="",
                        main='Written',
                        yaxt='n',
                        bty='n',
                        pty='m')

plotmeans(Combine~Race, data=ricci,
```

```
                        ylim=c(50,80),
                        barcol="black",
                        n.label=FALSE,
                        lwd=1.5,
                        pch=19,
                        xlab="Race",
                        ylab="",
                        main='Combined',
                        yaxt='n',
                        bty='n',
                        pty='m')

par(mfrow=c(1,1))

dev.off()

## png
##   2
```

Next, we illustrate with the **pdf** function. PDF is often preferred for graphics due to the flexibility in display and printing, as well as high resolution. Another advantage for our application is that the *sans* family in **pdf** is Helvetica.

```
pdf(file = "Plot1.panel.pdf",
    width = 8.5,
    height = 11,
    family = "sans")

par(mfrow=c(1,3))

plotmeans(Oral~Race, data=ricci,
                     ylim=c(50,80),
                     barcol="black",
                     n.label=FALSE,
                     lwd=1.5,
                     pch=19,
                     xlab="Race",
                     ylab="Score",
                     main='Oral',
                     bty='n',
                     pty='m')

plotmeans(Written~Race, data=ricci,
                        ylim=c(50,80),
                        barcol="black",
                        n.label=FALSE,
                        lwd=1.5,
                        pch=19,
```

```
                         xlab="Race",
                         ylab="",
                         main='Written',
                         yaxt='n',
                         bty='n',
                         pty='m')

plotmeans(Combine~Race, data=ricci,
                         ylim=c(50,80),
                         barcol="black",
                         n.label=FALSE,
                         lwd=1.5,
                         pch=19,
                         xlab="Race",
                         ylab="",
                         main='Combined',
                         yaxt='n',
                         bty='n',
                         pty='m')

dev.off()

## png
##   2

par(mfrow=c(1,1))
```

2 ggplot2 package: brief introduction

From ggplot2.org: "ggplot2 is a plotting system for R, based on the grammar of graphics, which tries to take the good parts of base and lattice graphics and none of the bad parts. It takes care of many of the fiddly details that make plotting a hassle (like drawing legends) as well as providing a powerful model of graphics that makes it easy to produce complex multi-layered graphics."

```
#install.packages("ggplot2")
library(ggplot2)
```

We will need to calculate confidence intervals separately and supply to the *ggplot* function.

The *Rmisc* package is handy for creating a data frame containing the means and margins of errors. Here we create objects containing means and margins of error.

```
install.packages("Rmisc")
```

```
## Installing package into 'C:/Users/sjricht2/Documents/R/win-library/3.4'
## (as 'lib' is unspecified)
```

```
## package 'Rmisc' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\sjricht2\AppData\Local\Temp\RtmpIJbAyM\downloaded_packages
```

```
library(Rmisc)
```

```
## Loading required package: lattice
```

```
Summary.CI1 <- summarySE(data=ricci,
                      measurevar="Oral",
                      groupvars="Race",
                      na.rm=FALSE,
                      conf.interval=.95)
Summary.CI1
```

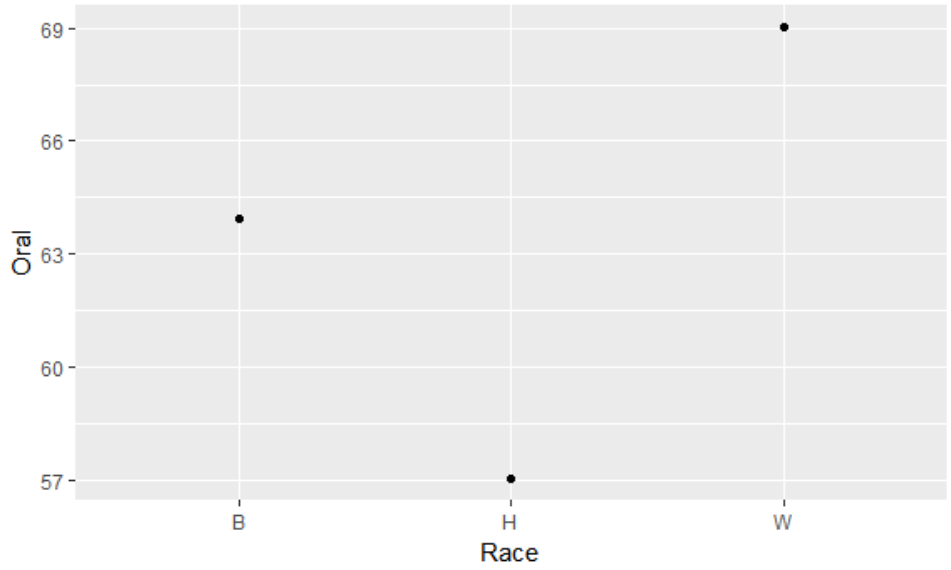
```
##   Race  N     Oral       sd       se       ci
## 1    B 27 63.95222 10.97465 2.112072 4.341427
## 2    H 23 57.03696 10.99093 2.291767 4.752834
## 3    W 68 69.01765 12.03918 1.459965 2.914102
```

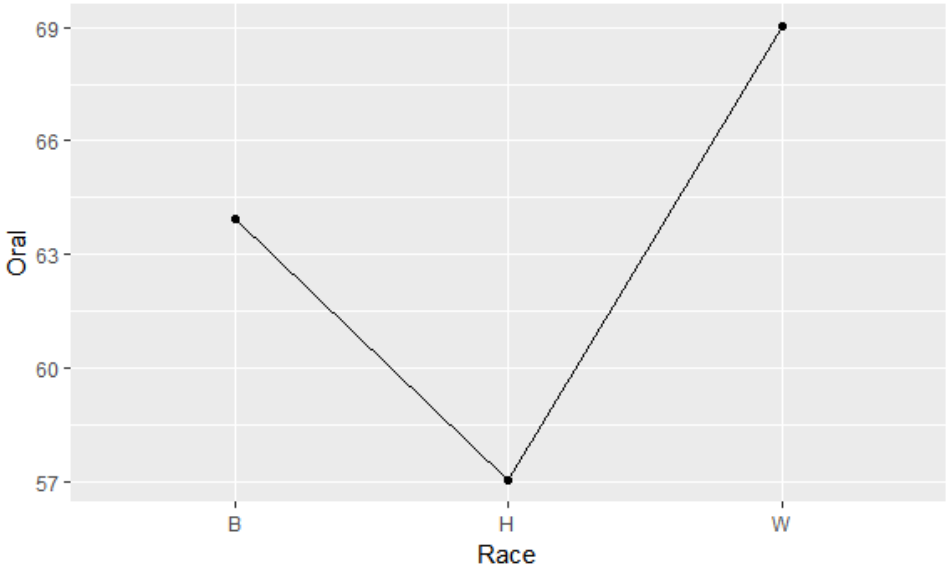
```
Summary.CI2 <- summarySE(data=ricci,
                     measurevar="Written",
                     groupvars="Race",
                     na.rm=FALSE,
                     conf.interval=.95)

Summary.CI3 <- summarySE(data=ricci,
                     measurevar="Combine",
                     groupvars="Race",
                     na.rm=FALSE,
                     conf.interval=.95)
```

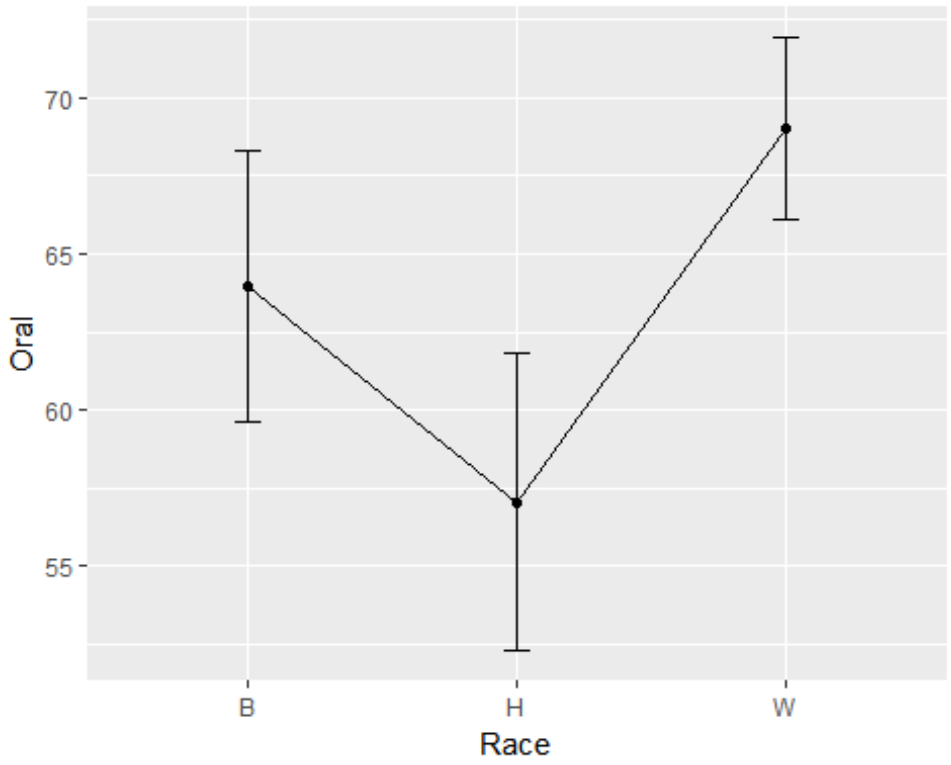
Next, we investigate a few different aspects of the **ggplot** function.

- *geom_point()*
- *geom_line()*
- *geom_errorbar()*
- *aes()*

```
ggplot(Summary.CI1,
                aes(x=Race,
                    y=Oral,
                    group=2)) + geom_point()

ggplot(Summary.CI1,
                aes(x=Race,
                    y=Oral,
                    group=2)) + geom_point() +
                    geom_line()

plot1.g <- ggplot(Summary.CI1,
                aes(x=Race,
                    y=Oral,
                    group=2)) + geom_point() +
                    geom_line() +
          geom_errorbar(aes(ymin=Oral-ci, ymax=Oral+ci),
                        width=.1)
plot1.g
```



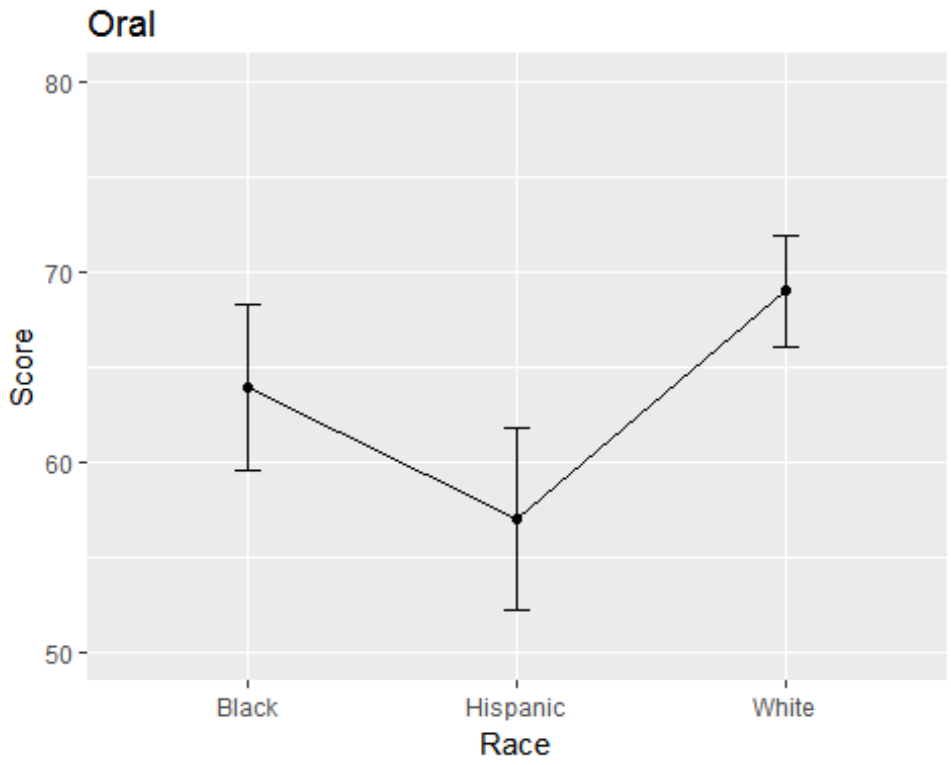




We want to format the plots as before, with "Score" as the label on the y-axis of the first plot, and the variable ("Oral", "Written" or "Combined") at the top of each plot.

```
plot1b.g <-ggplot(Summary.CI1,
           aes(x=Race, y=Oral, group=2)) +
           geom_point() + geom_line() + geom_errorbar(aes(ymin=Oral-ci, ymax=
Oral+ci),
```

```
              width=.1) +
scale_x_discrete(name="Race",labels=c("Black","Hispanic","White")) + labs(y="
Score") + ggtitle("Oral") + ylim(c(50,80))

plot1b.g
```



The default is to place the title left-justified

```
plot1b.g <- plot1b.g + theme(plot.title = element_text(hjust = 0.5))

plot1b.g
```

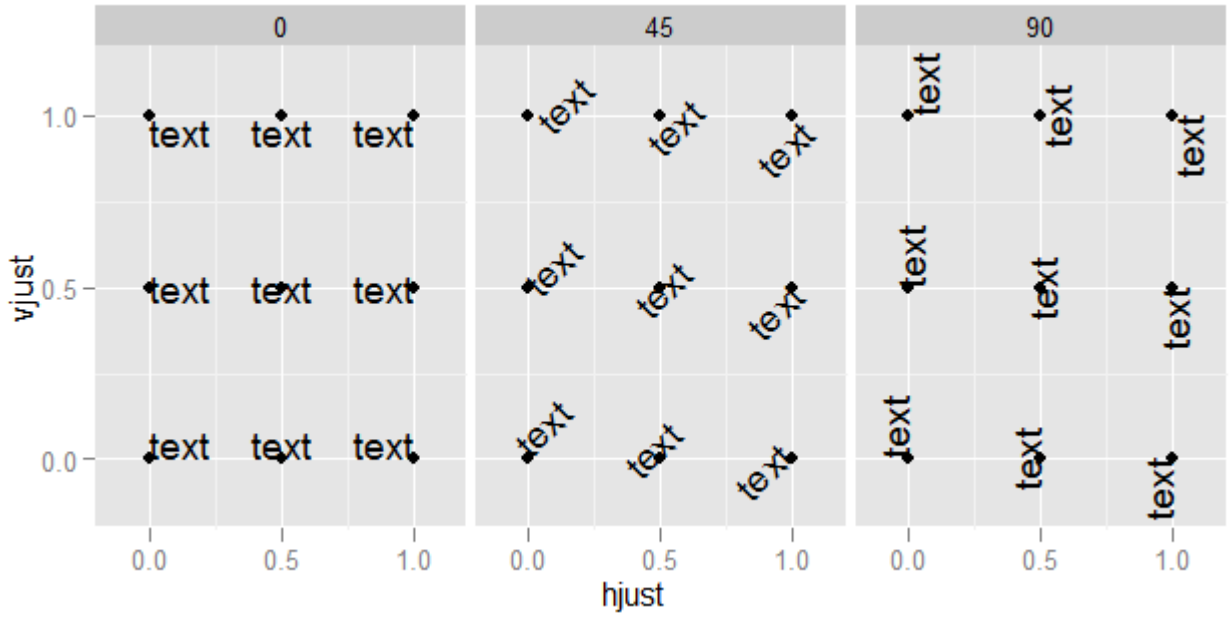
The value of `hjust` and `vjust` are only defined between 0 and 1:

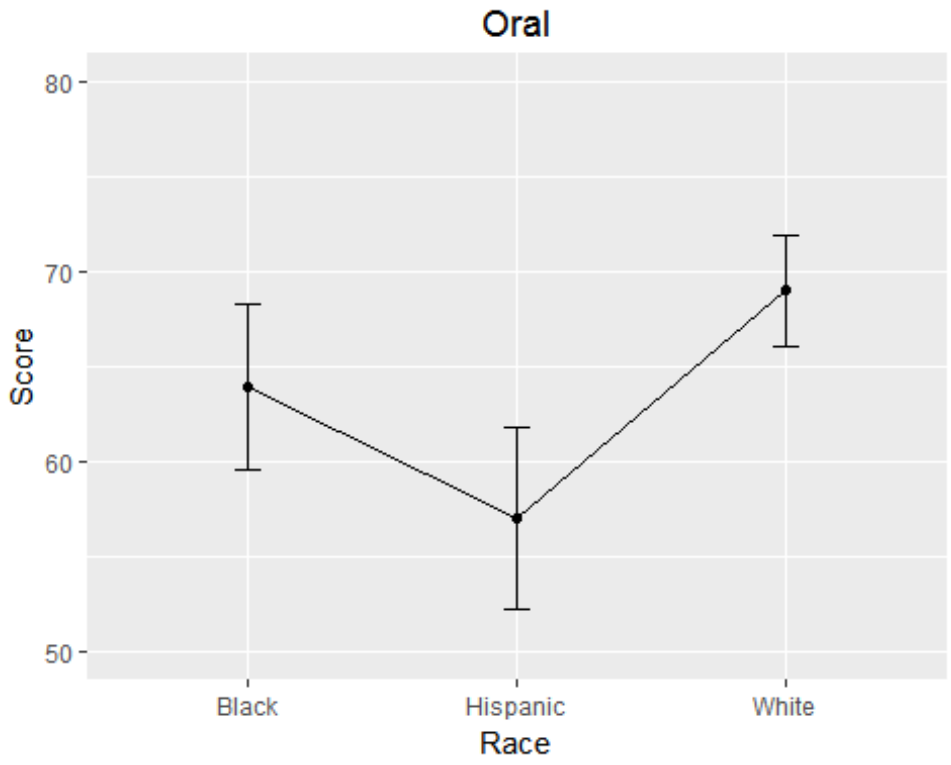
- 0 means left-justified
- 1 means right-justified

Source: ggplot2, Hadley Wickham, page 196

`hjust` controls horizontal justification and `vjust` controls vertical justification.
An example should make this clear:

```
td <- expand.grid(
   hjust=c(0, 0.5, 1),
   vjust=c(0, 0.5, 1),
   angle=c(0, 45, 90),
   text="text"
)

ggplot(td, aes(x=hjust, y=vjust)) +
   geom_point() +
   geom_text(aes(label=text, angle=angle, hjust=hjust, vjust=vjust)) +
   facet_grid(~angle) +
   scale_x_continuous(breaks=c(0, 0.5, 1), expand=c(0, 0.2)) +
   scale_y_continuous(breaks=c(0, 0.5, 1), expand=c(0, 0.2))
```

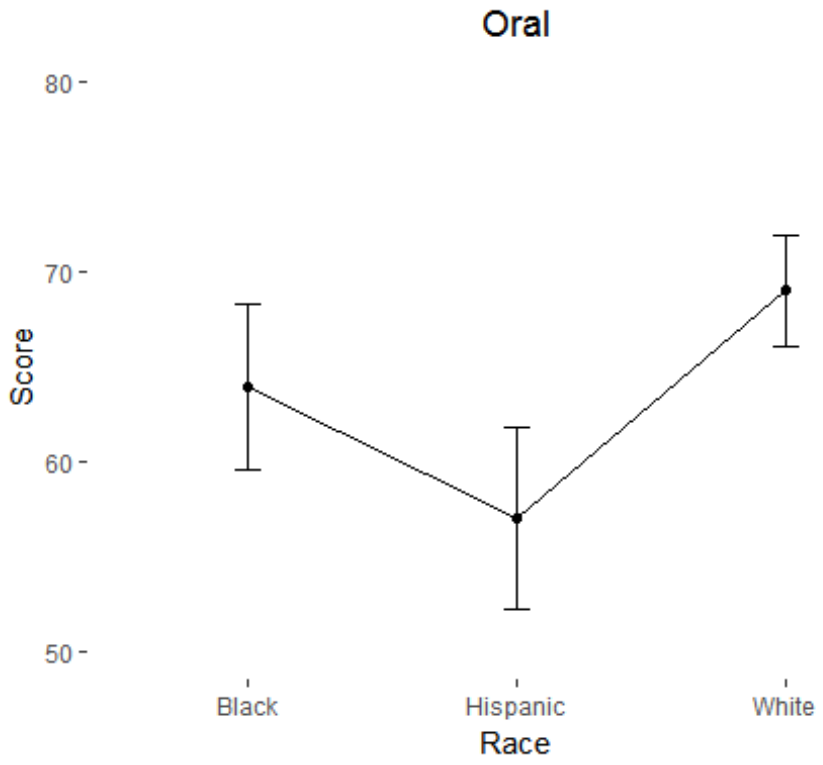




This introduces a theme: (http://ggplot2.tidyverse.org/reference/theme.html)

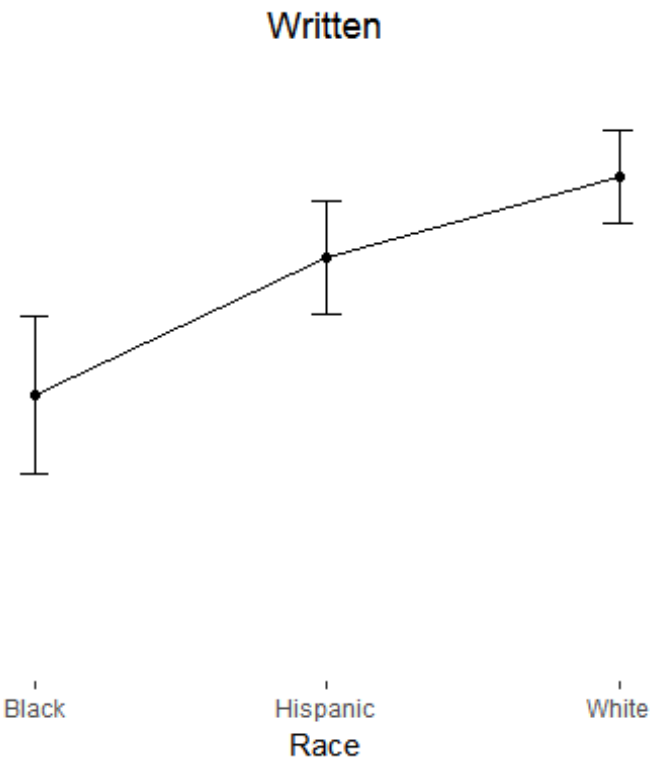
Finally, we also might like to remove the shading and gridlines.

```
plot1b.g <- plot1b.g + theme(panel.grid.minor = element_blank(),
panel.background = element_blank())

plot1b.g
```

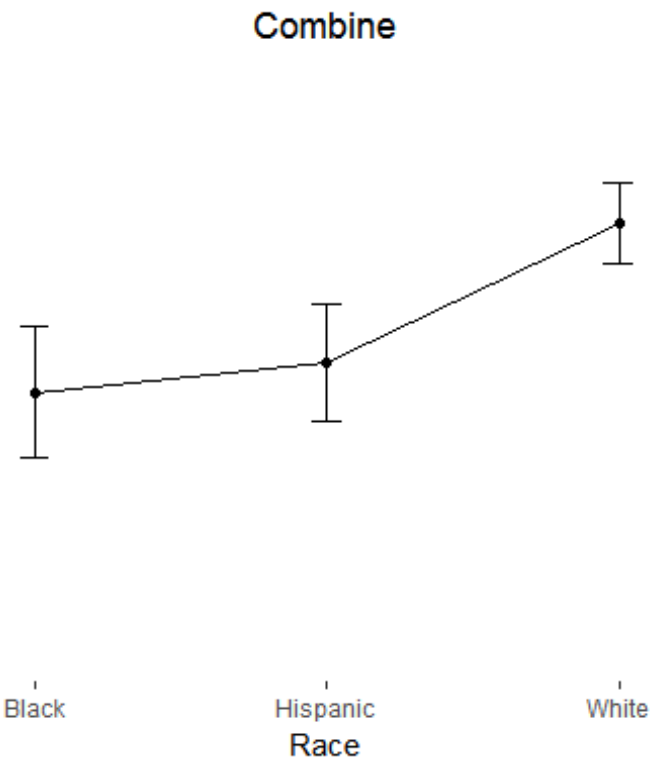


We repeat the above steps for plots 2 and 3. However, we want to omit the y-axis for each. This can be done by specifying several other theme elements.

```
plot2b.g <-ggplot(Summary.CI2,
           aes(x=Race, y=Written, group=2)) +
           geom_point() + geom_line() + geom_errorbar(aes(ymin=Written-ci,
                                                          ymax=Written+ci),
              width=.1) +
scale_x_discrete(name="Race",labels=c("Black","Hispanic","White")) + ggtitle(
"Written") + ylim(c(50,80))

plot2b.g <- plot2b.g + theme(plot.title = element_text(hjust = 0.5))

plot2b.g <- plot2b.g + theme(panel.grid.minor = element_blank(),
panel.background = element_blank())

plot2b.g <- plot2b.g +
  theme(axis.ticks.y=element_blank(),
        axis.text.y=element_blank(),
        axis.title.y=element_blank())

plot2b.g
```

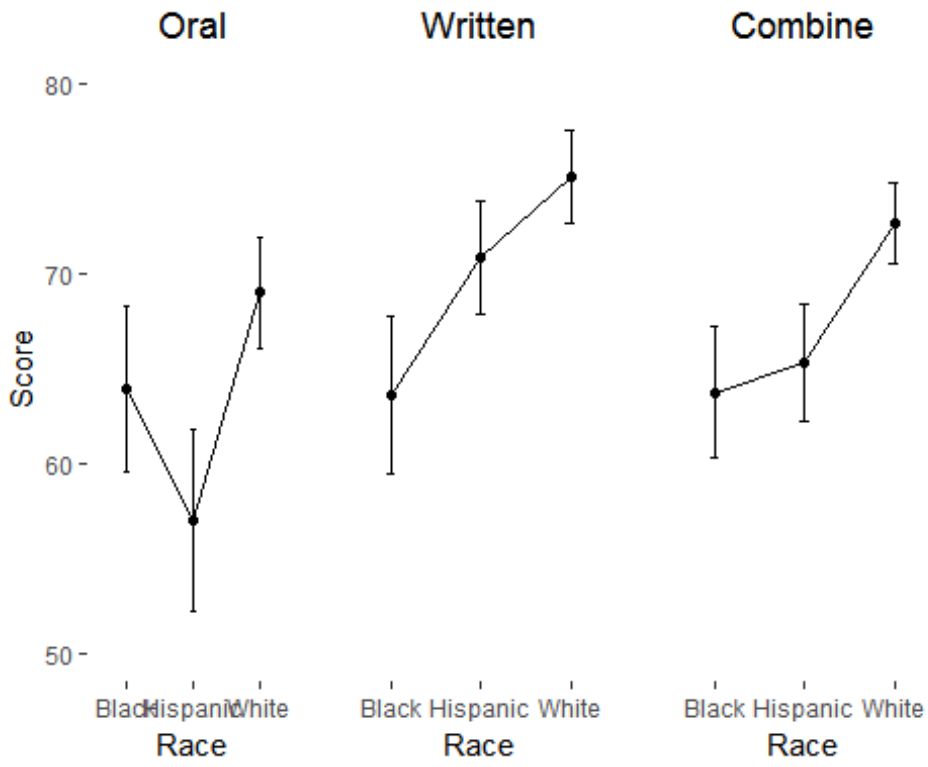



```
plot3b.g <-ggplot(Summary.CI3,
           aes(x=Race, y=Combine, group=2)) +
           geom_point() + geom_line() + geom_errorbar(aes(ymin=Combine-ci, ym
ax=Combine+ci),
              width=.1) +
scale_x_discrete(name="Race",labels=c("Black","Hispanic","White")) + ggtitle(
"Combine") + ylim(c(50,80))

plot3b.g <- plot3b.g + theme(plot.title = element_text(hjust = 0.5))

plot3b.g <- plot3b.g + theme(panel.grid.minor = element_blank(),
panel.background = element_blank())

plot3b.g <- plot3b.g +
  theme(axis.ticks.y=element_blank(),
        axis.text.y=element_blank(),
        axis.title.y=element_blank())

plot3b.g
```



To create a panel plot, we can try the **gridarrange** function in the *gridExtra* package

```
install.packages("gridExtra")

## Installing package into 'C:/Users/sjricht2/Documents/R/win-library/3.4'
## (as 'lib' is unspecified)

## package 'gridExtra' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\sjricht2\AppData\Local\Temp\RtmpIJbAyM\downloaded_packages

library(gridExtra)

combinedplots<-grid.arrange(plot1b.g, plot2b.g, plot3b.g, ncol=3)
```



This seemed to work. One slight issue is that when the y-axis ticks and labels were removed from the second and third plots, the plot area expanded to cover that space, making those plots wider than the first.

Another option: “Wrap” the individual plots according to the values of another variable. This requires reformatting the data.

Recall the original summary data:

```
Summary.CI1
```

```
##   Race  N     Oral       sd       se       ci
## 1    B 27 63.95222 10.97465 2.112072 4.341427
## 2    H 23 57.03696 10.99093 2.291767 4.752834
## 3    W 68 69.01765 12.03918 1.459965 2.914102
```

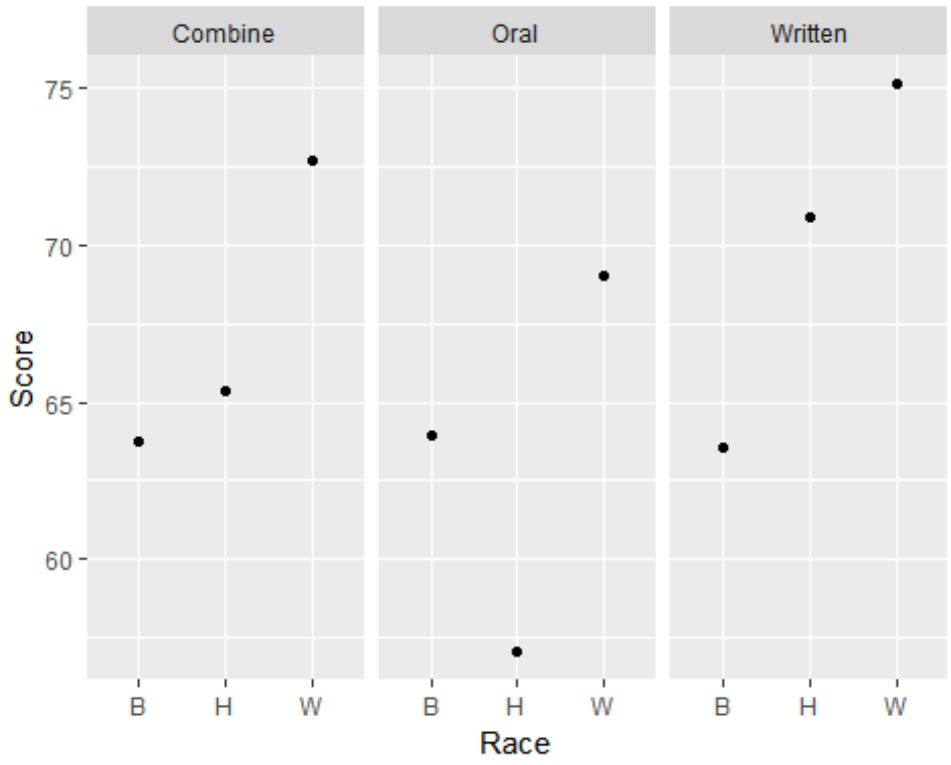
The reformatted data will look like this:

```
summary=read.csv(file="C:/Users/sjricht2/Box Sync/Consulting/Workshops/R Work
shop II/Module_2/Data/Summary.csv", header=T)
summary
```

```
##   Race  N    Part  Mean   ci
## 1    B 27    Oral 63.95 4.34
## 2    H 23    Oral 57.04 4.75
## 3    W 68    Oral 69.02 2.91
## 4    B 27 Written 63.59 4.12
## 5    H 23 Written 70.87 2.96
## 6    W 68 Written 75.12 2.43
## 7    B 27 Combine 63.74 3.46
## 8    H 23 Combine 65.34 3.09
## 9    W 68 Combine 72.68 2.14
```

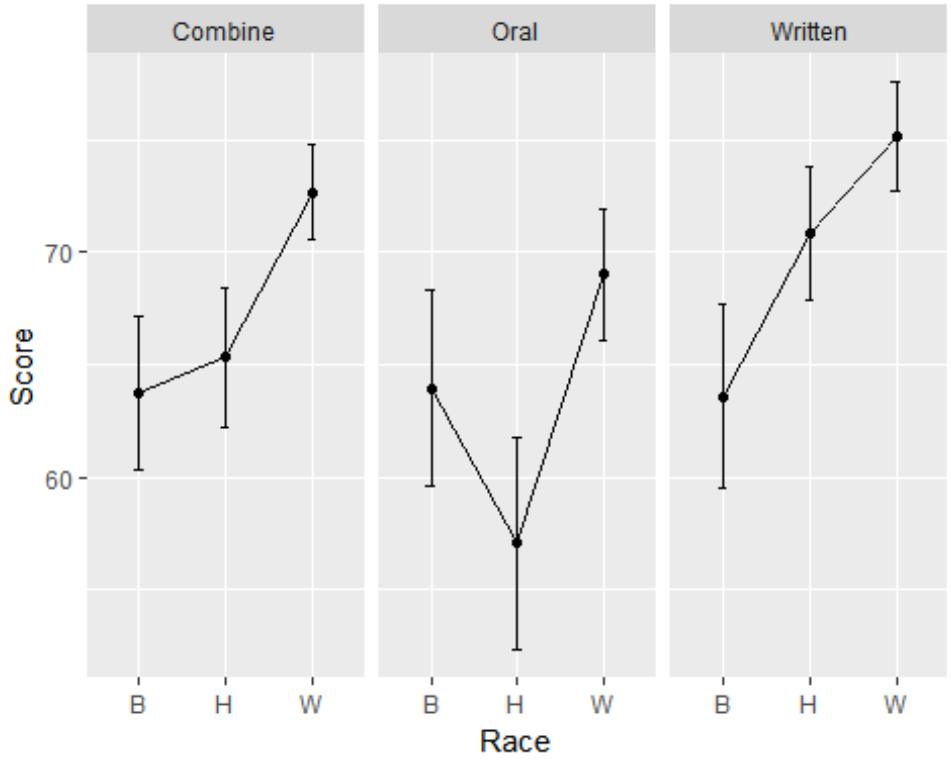
```
panel.facet1 <- ggplot(summary, aes(Race, Mean))  +
  geom_point() +
  geom_line() +
 ylab("Score") + facet_wrap(~ Part)
panel.facet1
```

```
## geom_path: Each group consists of only one observation. Do you need to
## adjust the group aesthetic?
## geom_path: Each group consists of only one observation. Do you need to
## adjust the group aesthetic?
## geom_path: Each group consists of only one observation. Do you need to
## adjust the group aesthetic?
```



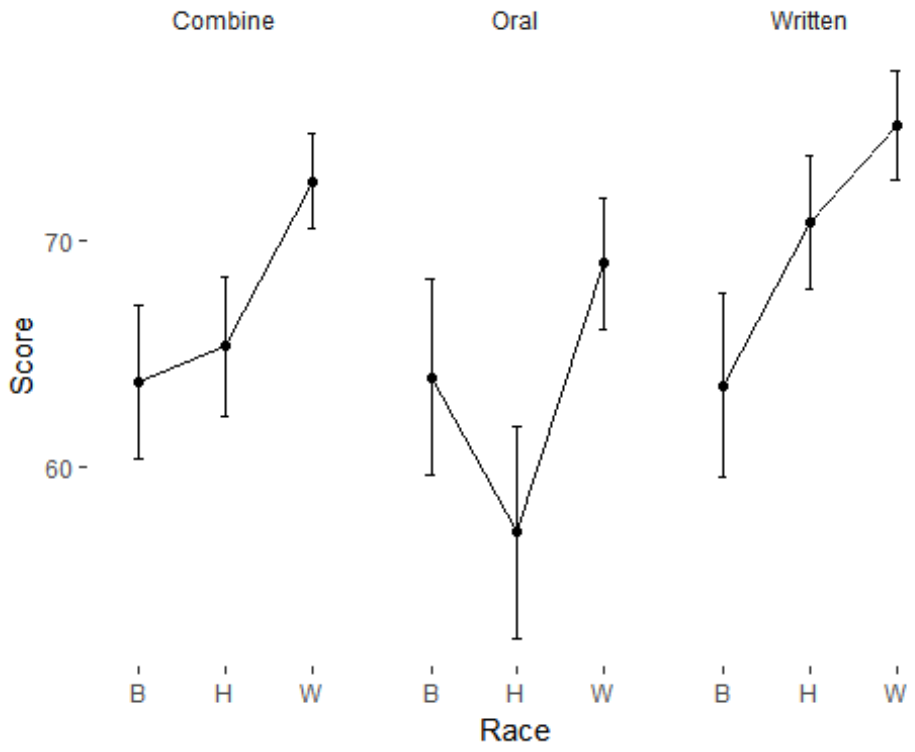
The means are not connected. This is because the "means" are considered single observations in the new data set. We can fix this by setting the group aesthetic to 1.

```
panel.facet2 <- ggplot(summary, aes(Race, Mean, group=1))  +
  geom_point() +
  geom_line() +
  geom_errorbar(aes(ymin=Mean-ci, ymax=Mean+ci),width=.1) + ylab("Score") + f
acet_wrap(~ Part)
panel.facet2
```



We again may want to remove the shading.

```
panel.facet3 <- panel.facet2 + theme(
panel.grid.minor = element_blank(),
panel.background = element_blank(),
strip.background = element_blank())
panel.facet3
```



Finally, we create and save a pdf image file. (Can be either be a device function (e.g. png), or one of “eps”, “ps”, “tex” (pictex), “pdf”, “jpeg”, “tiff”, “png”, “bmp”, “svg” or “wmf” (windows only).)

```
ggsave("Combined",
       plot=panel.facet3,
       device = "pdf",
       width=8.5, height=11, units="in",
       dpi=1200)

unlink("Combined.pdf")
```