PSYC 6780: Lab Tutorial 3

Chris M. Fiacconi

Data Visualization with *ggplot2*

Within the *tidyverse* collection of packages, there is a fantastic and powerful graphics package called *ggplot2*. In this tutorial, we will learn how to use the functions within the *ggplot2* package to visualize data, and make beautiful, publication-ready figures.

A key concept when using *ggplot2* to make graphs is *layering*. The whole premise of *ggplot2* is that you can add new features/elements to a figure by adding additional layers that contain different geometric elements, or *geoms*. This feature of *ggplot2* allows incredible flexibility in making figures, and can be used to effectively and efficiently communicate the key aspects of your data. Let's start with some basics.

Scatterplots

First, let's load in the *ggplot2* package by calling *tidyverse*:

```r
library(tidyverse)
```

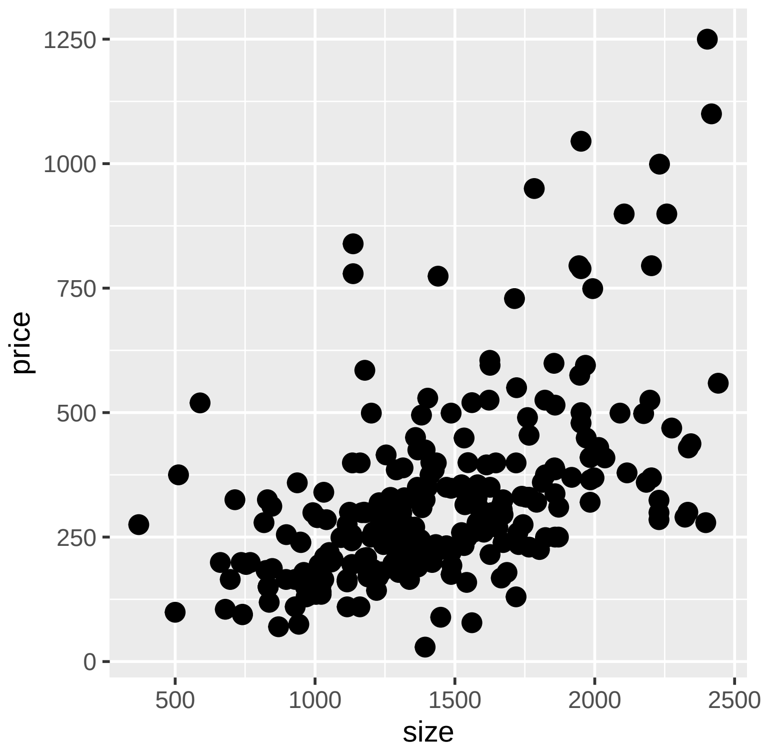
```
## Warning: package 'ggplot2' was built under R version 3.5.2
```

Now, let's load in the data file, *realestate.txt* so that we can make a scatterplot. This data file contains information on the square footage and listing prices of different houses on the market.

```r
listings<-read.table("realestate.txt",header=T)
head(listings, n=10)
```

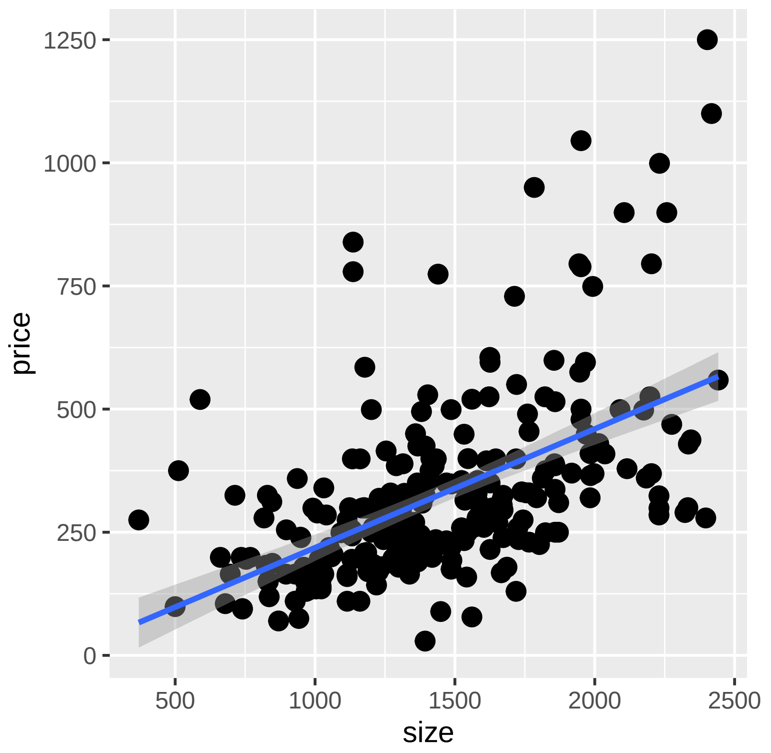
```
##       mls           location price bedrooms bathrooms     size
## 1  132842      Arroyo Grande 795.0        3         3 2202.730
## 5  136282 Santa Maria-Orcutt 109.9        3         1 1160.358
## 6  136431             Oceano 324.9        3         3 1672.254
## 7  137036 Santa Maria-Orcutt 192.9        4         2 1489.235
## 8  137090 Santa Maria-Orcutt 215.0        3         2 1347.093
## 10 137570         Atascadero 319.0        3         2 1229.107
## 11 138053 Santa Maria-Orcutt 350.0        3         2 1625.803
## 12 138730 Santa Maria-Orcutt 249.0        3         2 1300.642
## 13 139291      Arroyo Grande 299.0        2         2 1167.791
## 14 139427 Santa Maria-Orcutt 235.9        3         2 1300.642
```

```r
# Make scatterplot
ggplot(listings,aes(x=size,y=price)) +
  geom_point(size=3) # Must add a geom to ggplot command in order to make figure appear!
```

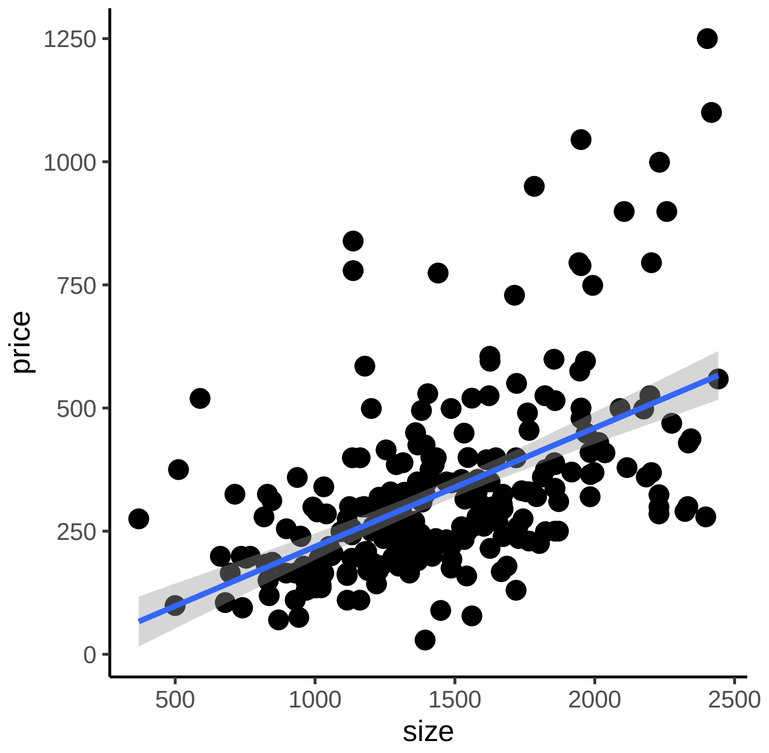


We can easily add a line of best fit along with its 95% CI using the **geom_smooth** command:

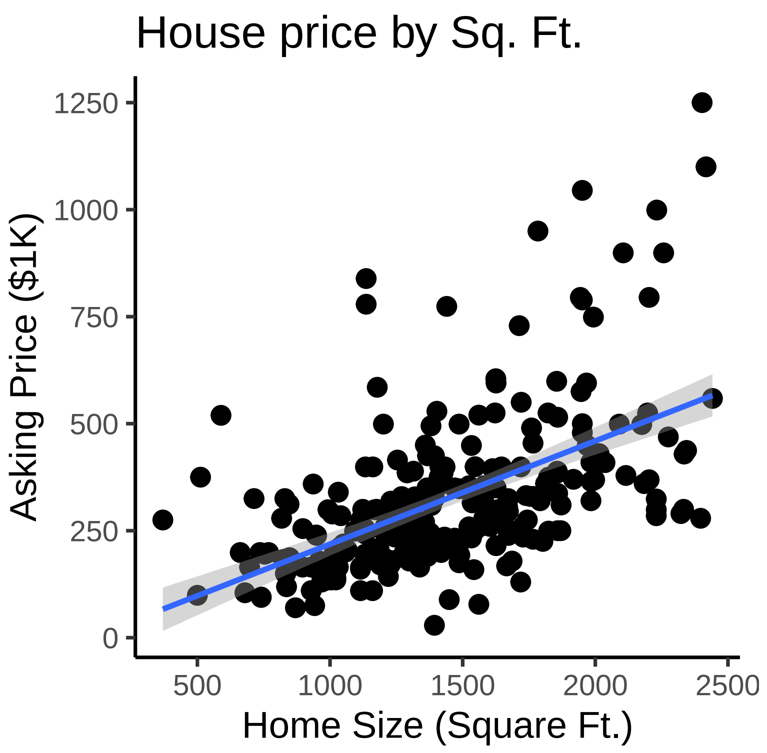
```
ggplot(listings,aes(x=size,y=price)) +
  geom_point(size=3) +
  geom_smooth(method="lm")
```



```r
# Same plot but change background (theme)
ggplot(listings,aes(x=size,y=price)) +
  geom_point(size=3) +
  geom_smooth(method="lm") +
  theme_classic()
```



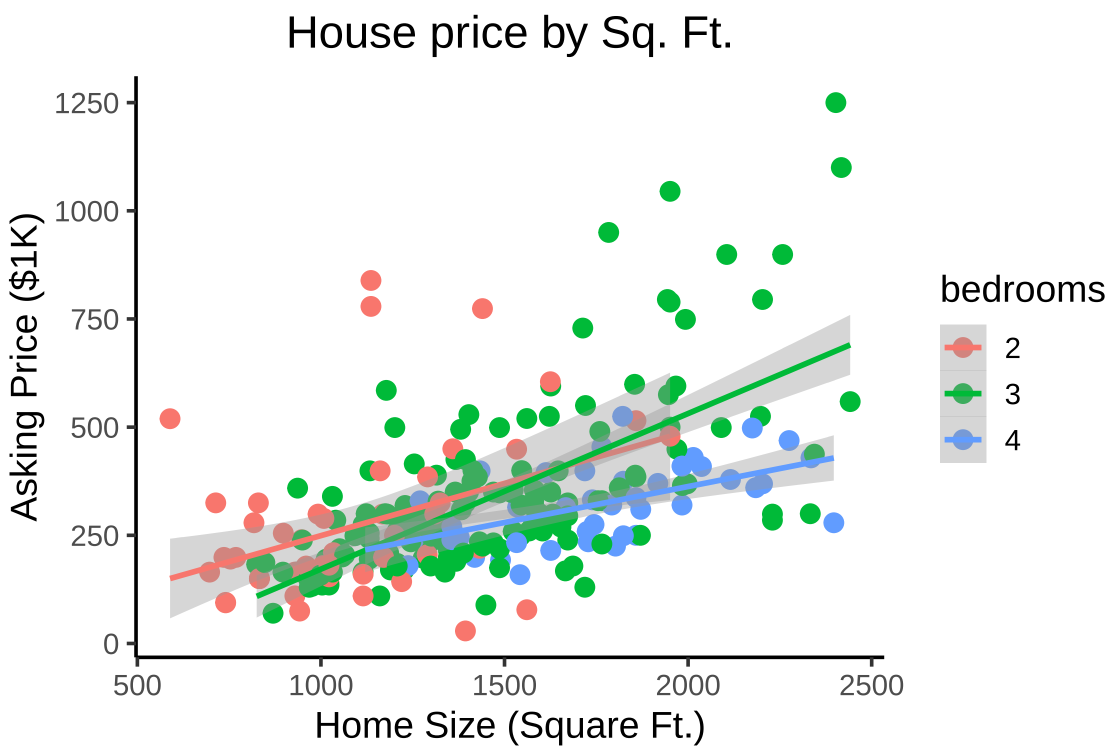
```r
# Same plot but change axis labels and make numbers larger
ggplot(listings,aes(x=size,y=price)) +
  geom_point(size=3) +
  geom_smooth(method="lm") +
  theme_classic(base_size=14) +
  labs(title="House price by Sq. Ft.",x ="Home Size (Square Ft.)",y="Asking Price ($1K)")
```



Splitting Data & Plotting Multiple Variables in the Same Scatterplot

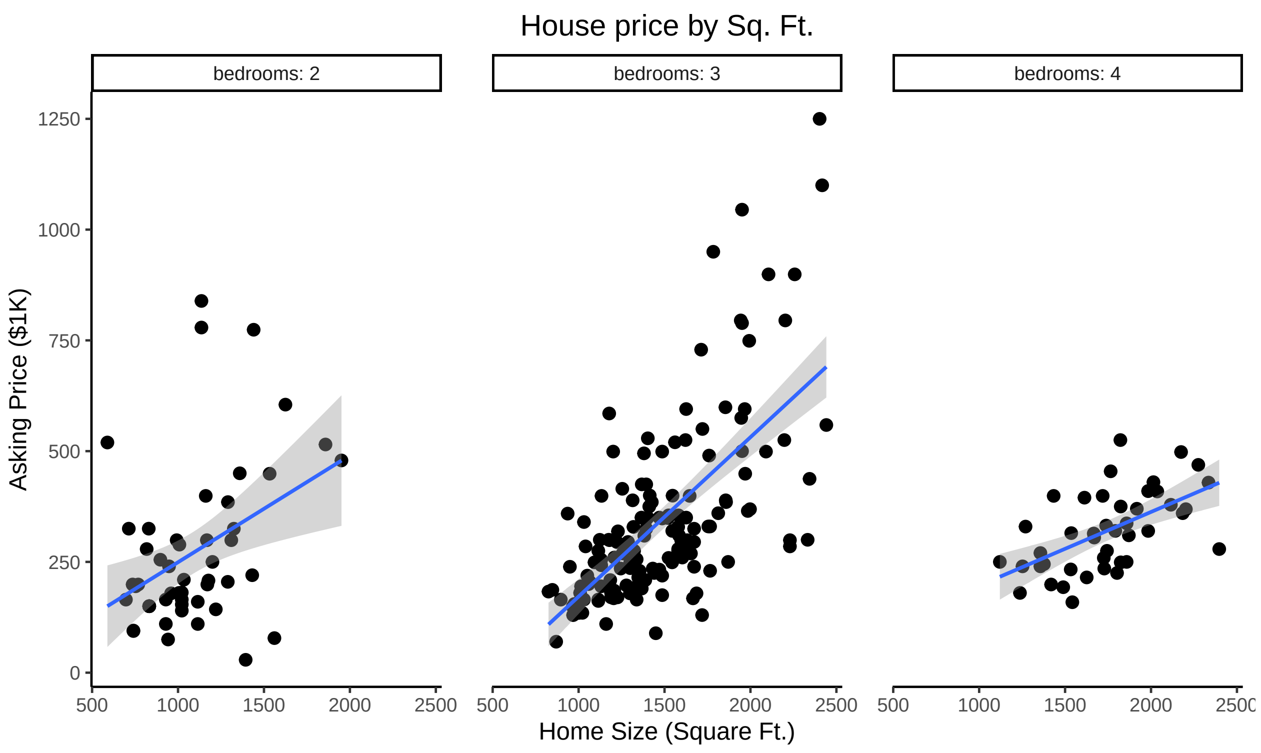
Now, what if we want to separate our data points by number of bedrooms? That is, is the relationship between home size and asking price different depending on how many bedrooms are in the home? We can easily do this by mapping the *bedrooms* variable in our aesthetic mappings. However, before we do this, we need to convert the *bedrooms* variable from a continuous-variable to a discrete-variable (i.e., factor)

```
listings$bedrooms<-as.factor(listings$bedrooms)

# Get listings for only 2, 3, 4 bedroom homes
listings_subset<-listings %>% filter(bedrooms %in% c(2,3,4))

ggplot(listings_subset,aes(x=size,y=price,color=bedrooms)) +
  geom_point(size=3) +
  geom_smooth(method="lm") +
  theme_classic(base_size=14) +
  labs(title="House price by Sq. Ft.",x ="Home Size (Square Ft.)",y="Asking Price ($1K)") +
  theme(plot.title = element_text(hjust = 0.5))
```



We could also separate the data based on bedrooms by creating separate plots for each bedroom value. To do this we can use the **facet_grid** and/or **facet_wrap** commands:

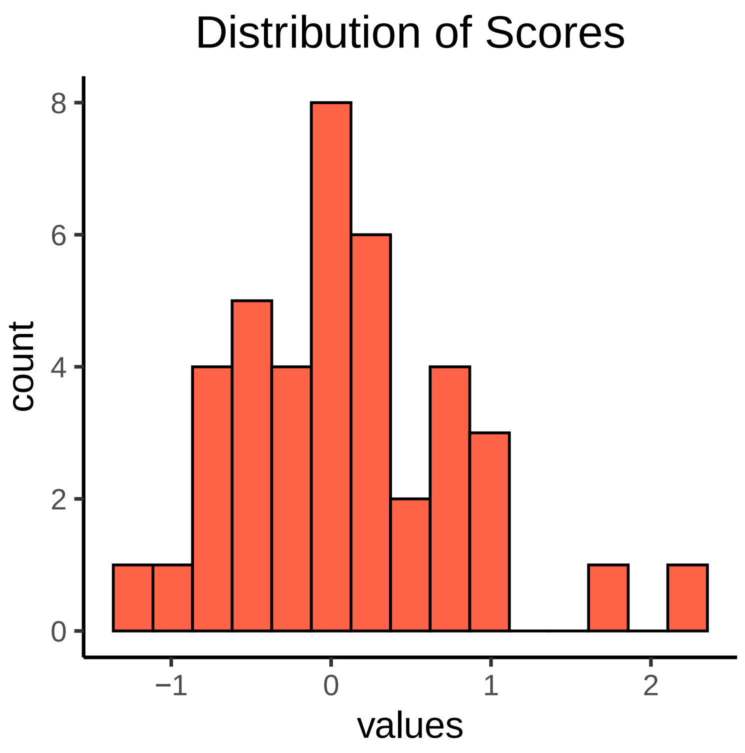
```
# Another way to separate the data by no. of bedrooms is to create
# separate plots for each bedroom value using facet_grid
library(grid)

ggplot(listings_subset,aes(x=size,y=price)) +
  geom_point(size=3) +
  geom_smooth(method="lm") +
  theme_classic(base_size=14) +
  labs(title="House price by Sq. Ft.",x ="Home Size (Square Ft.)",y="Asking Price ($1K)") +
  facet_grid(~bedrooms,labeller=label_both) +
  theme(panel.spacing = unit(2, "lines"),plot.title = element_text(hjust = 0.5))
```



Histograms

Another informative way to visualize data is by using histograms, which are useful for looking at the distribution of values of a variable. Like scatterplots, histograms are very simple to create using *ggplot2*.

```
# Create a variable with random numbers
set.seed(100)

# Draw random numbers from standard normal distribution
some_numbers<-data.frame(values=rnorm(n=40,mean=0,sd=1))

# Make Histogram
ggplot(some_numbers,aes(x=values)) +
  geom_histogram(fill="tomato",color="black",bins=15) +
  theme_classic(base_size=14) +
  ggtitle("Distribution of Scores") +
  theme(plot.title = element_text(hjust = 0.5))
```

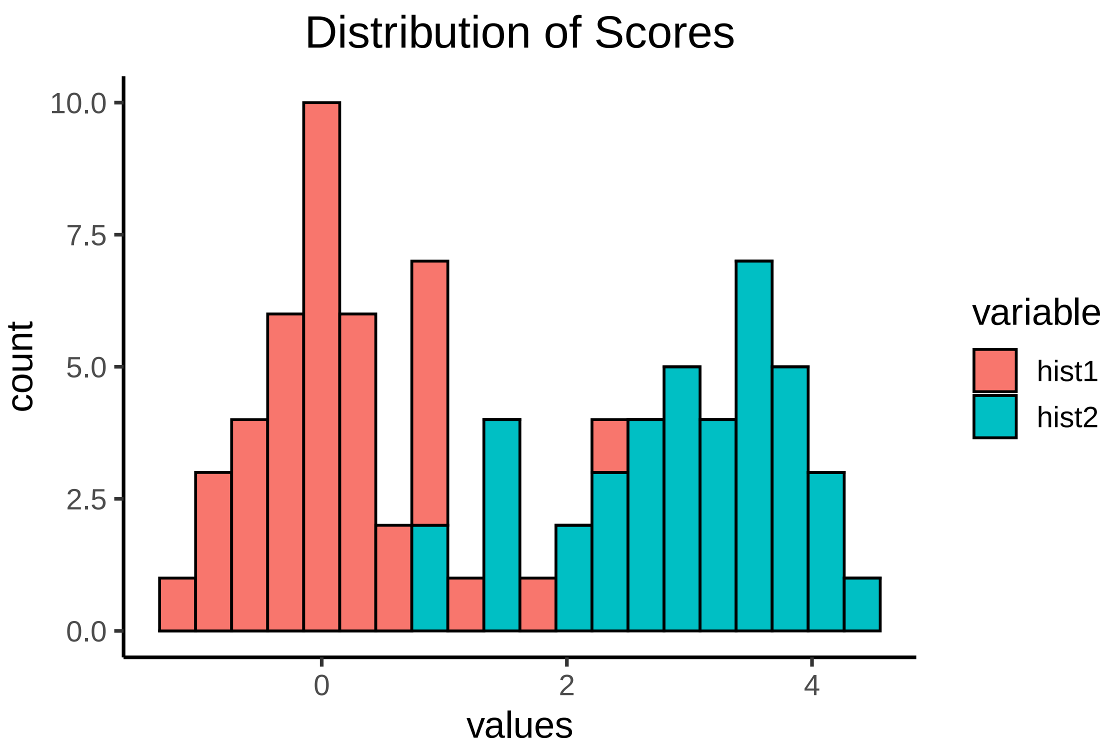


Plotting Multiple Histograms on Same Plot

```
# Draw random numbers from normal distribution with mean = 3, sd = 1
set.seed(101)
some_numbers2<-rnorm(n=40,mean=3,sd=1)

# Combine some_numbers and some_numbers2 into one data frame
all_numbers<-data.frame(variable=rep(c("hist1","hist2"),times=1,each=40),
                        values=c(some_numbers$values,some_numbers2))

str(all_numbers)
```

```
## 'data.frame':    80 obs. of  2 variables:
##  $ variable: Factor w/ 2 levels "hist1","hist2": 1 1 1 1 1 1 1 1 1 1 ...
##  $ values  : num  -0.5022 0.1315 -0.0789 0.8868 0.117 ...
```

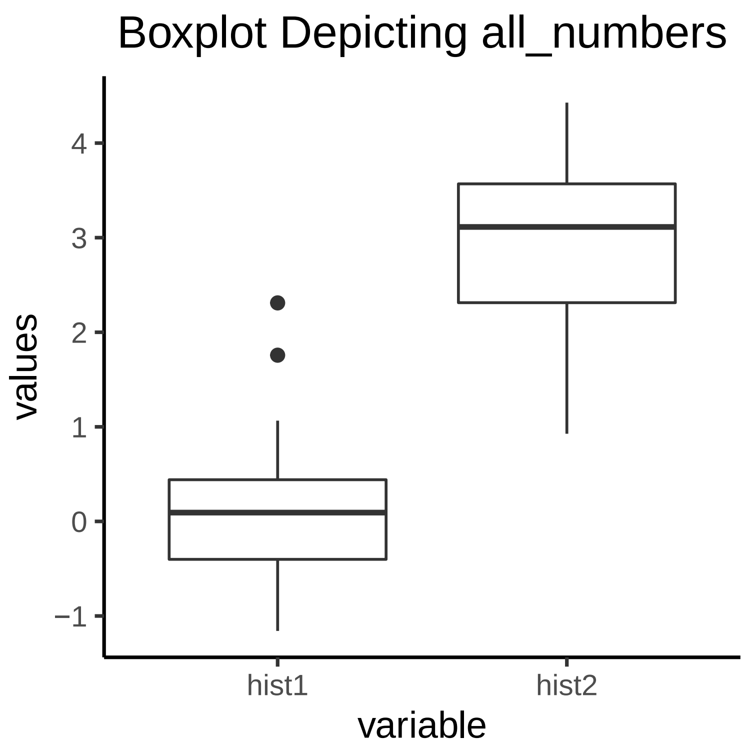
```
# Make Histogram showing both variables
ggplot(all_numbers,aes(x=values,fill=variable)) +
  geom_histogram(color="black",bins=20) +
  theme_classic(base_size=14) +
  ggtitle("Distribution of Scores") +
  theme(plot.title = element_text(hjust = 0.5))
```



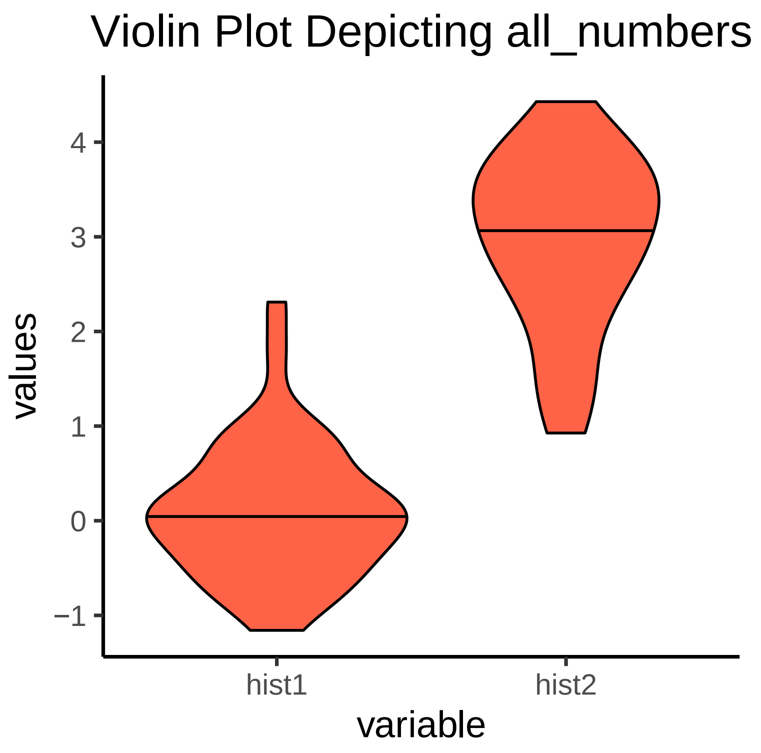
Boxplots and Violin Plots

Another way to visualize individual data points in a way that illustrates how they are distributed is by using boxplots and violin plots. Most of you are now familiar with boxplots. Violin plots are similar by they more explicitly illustrate the shape of the distribution. We'll use the *all_numbers* data frame again to make some boxplots and violin plots.

```r
# Make boxplot with geom_boxplot
ggplot(all_numbers,aes(x=variable,y=values)) +
  geom_boxplot(outlier.size=2,outlier.shape=19) +
  theme_classic(base_size=14) +
  ggtitle("Boxplot Depicting all_numbers") +
  theme(plot.title = element_text(hjust = 0.5))
```



```r
# Make violin plot with geom_violin
ggplot(all_numbers,aes(x=variable,y=values)) +
  geom_violin(fill="tomato",color="black",draw_quantiles=.5) +
  theme_classic(base_size=14) +
  ggtitle("Violin Plot Depicting all_numbers") +
  theme(plot.title = element_text(hjust = 0.5))
```



Barplots

One of the most common forms of data visualization in experimental psychology is a bar graph. Bar graphs depict condition means and typically an error bar that captures either the standard error associated with each mean or the corresponding 95% CI. In addition, it is often informative to depict the data from individual subjects in a barplot so that others can verify that condition means are not unduly influenced by a few outliers.

Between-Subjects Designs

Let's start with an example using a 2 × 3 factorial between-subjects design.

```r
bp_data<-read.table(file="Ch7T5.txt",header=TRUE)
str(bp_data)
```

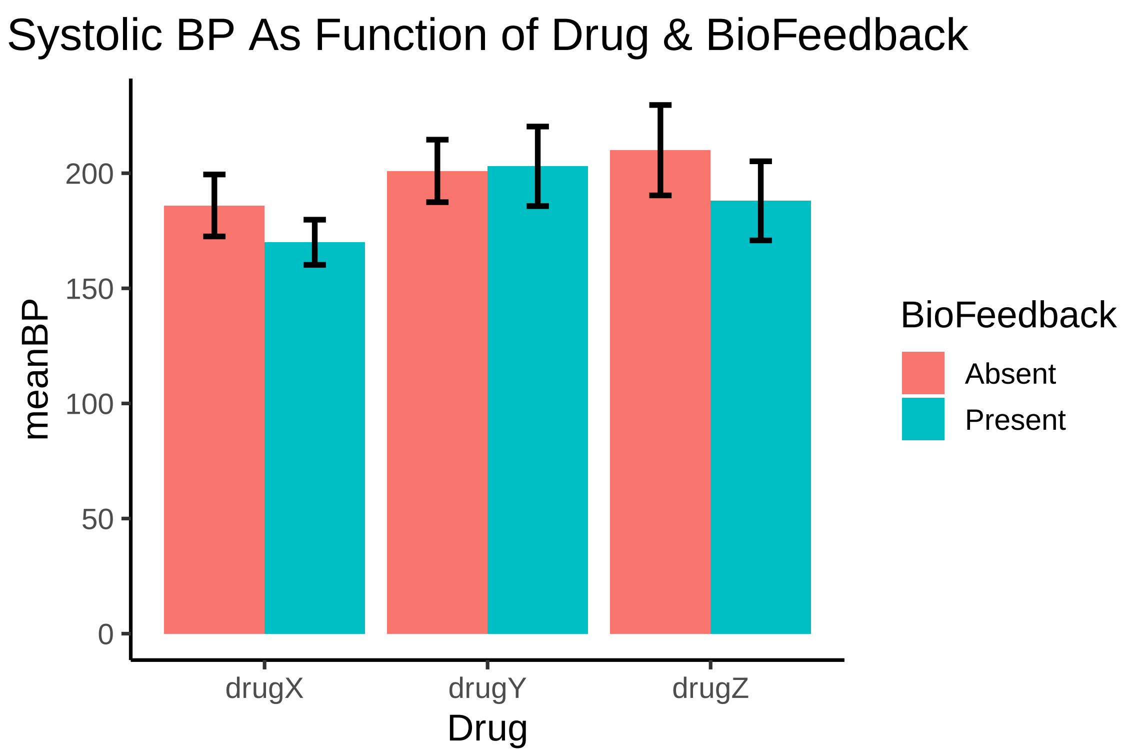
```
## 'data.frame':    30 obs. of  4 variables:
##  $ Score      : int  170 175 165 180 160 186 194 201 215 219 ...
##  $ BioFeedback: Factor w/ 2 levels "Absent","Present": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Drug       : Factor w/ 3 levels "drugX","drugY",..: 1 1 1 1 1 2 2 2 2 2 ...
##  $ Subj       : int  1 2 3 4 5 6 7 8 9 10 ...
```

```r
# Get overall condition means and SEM
bp_means<-bp_data %>%
  group_by(Drug,BioFeedback) %>%
  summarize(meanBP=mean(Score),SEM=sd(Score)/sqrt(n()),n=n())

print(bp_means)
```

```
## # A tibble: 6 x 5
## # Groups:   Drug [?]
##   Drug  BioFeedback meanBP   SEM     n
##   <fct> <fct>        <dbl> <dbl> <int>
## 1 drugX Absent         186  4.85     5
## 2 drugX Present        170  3.54     5
## 3 drugY Absent         201  4.89     5
## 4 drugY Present        203  6.22     5
## 5 drugZ Absent         210  7.07     5
## 6 drugZ Present        188  6.19     5
```

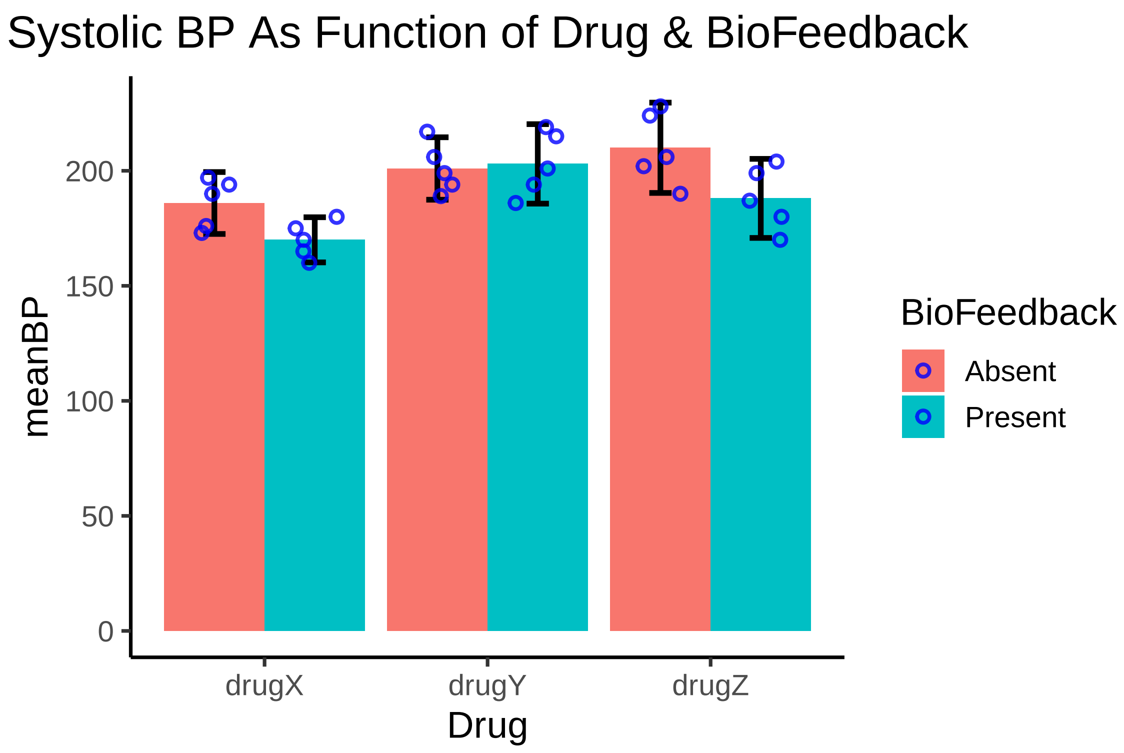
```r
# Calculate 95% CIs - first get critical t-value
t_crit<-qt(p=.975,df=4)

bp_means_withCI<-bp_means %>% # Add CI column to bp_means
  mutate(CI=SEM*t_crit)

# Make barplot with geom_bar
bp_bar<-ggplot(bp_means_withCI,aes(x=Drug,y=meanBP,fill=BioFeedback)) +
  geom_bar(stat="identity",position="dodge") +
  theme_classic(base_size=14) +
  ggtitle("Systolic BP As Function of Drug & BioFeedback") +
  theme(plot.title = element_text(hjust = 0.5))
```

```r
# Add error bars
bp_bar +
  geom_errorbar(aes(ymin=meanBP-CI,ymax=meanBP+CI),size=1,width=0.2,position=position_dodge(0.9))
```



In addition to error bars, which give an indication of how reliable each estimate of the population mean is, we can also add points to illustrate the individual subject data.

```r
# Add individual subject points
bp_bar +
  geom_errorbar(aes(ymin=meanBP-CI,ymax=meanBP+CI),size=1,width=0.2,position=position_dodge(0.9)) +
  geom_jitter(data=bp_data,aes(x=Drug,y=Score,fill=BioFeedback),size=1.5,shape=1,color="blue",
              stroke=1,alpha=.8,position=position_jitterdodge(dodge.width=0.9,jitter.width=.4))
```



Within-Subject Designs: One-Way

The process of visualizing data from within-subject designs is similar to that of between-subject designs. In essence, the goal is to show the mean score in each condition or cell of the design along with some indicator of error variability. Plotting the condition means in a within-subject design would proceed in an identical fashion to a between-subjects design, but depicting meaningful estimates of error variability is slightly more complicated (see Morey, 2009). The calculation of the SEM and 95% CIs diverge from those calculations used in a between-subject design because we can ignore variation that is due to differences between individual subjects. As a consequence, our error bars shrink reflecting the increase in power that is characteristic of within-subject designs.

In addition, plotting the individuals subject points is also slightly more complicated because each indivdual appears in each condition. Therefore, we need lines to connect each subject across conditions.

Let's look at an example of a one-way within-subjects (or repeated-measures) design to show how we can implement these modifications. The data can be found in the *Ch11T5.txt* file.

```
baby_data<-read.table(file="Ch11T5.txt",header=T)
str(baby_data)
```

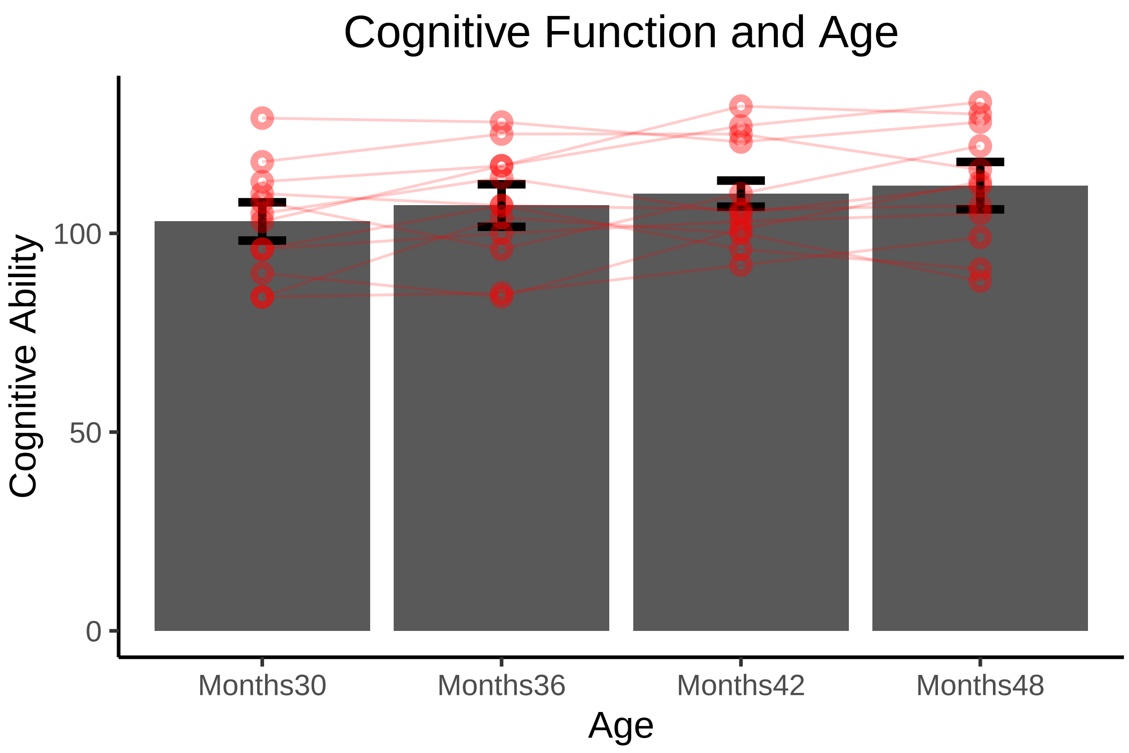
```
## 'data.frame':    48 obs. of  3 variables:
##  $ Age    : Factor w/ 4 levels "Months30","Months36",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Score  : int  108 103 96 84 118 110 129 90 84 96 ...
##  $ Subject: int  1 2 3 4 5 6 7 8 9 10 ...
```

```
# install Rmisc package for computing within-subject 95% CIs (Morey, 2009)
# install.packages("Rmisc") - do this only once
baby_means<-Rmisc::summarySEwithin(data=baby_data,measurevar="Score",betweenvars=NULL,
                                   withinvars="Age",idvar="Subject") # Note the quotations
```

```r
print(baby_means)
```

```
##        Age  N Score       sd       se       ci
## 1 Months30 12   103 7.579878 2.188122 4.816025
## 2 Months36 12   107 8.413553 2.428784 5.345717
## 3 Months42 12   110 5.163978 1.490712 3.281035
## 4 Months48 12   112 9.393744 2.711740 5.968500
```

```r
# Make barplot
ggplot(baby_means,aes(x=Age,y=Score)) +
  geom_bar(stat="identity") +
  labs(y="Cognitive Ability") +
  theme_classic(base_size=14) +
  geom_errorbar(aes(ymin=Score-ci,ymax=Score+ci),width=0.2,size=1.5) +
  geom_point(data=baby_data,aes(x=Age,y=Score),size=1.5,shape=1,color="red",
             stroke=2,alpha=.4) +
  geom_line(data=baby_data,aes(x=Age,y=Score,group=Subject),color="red",alpha=.2) +
  ggtitle("Cognitive Function and Age") +
  theme(plot.title = element_text(hjust = 0.5))
```



Within-Subject Designs: Two-Way

Plotting the results from a two-way within-subjects design is very similar to plotting the results from a one-way within-subjects design. However, when using the **summarySEwithin** command, we need to specify that we have multiple within-subject factors.

Let's do an example using the data set in the *Ch12T1.txt* file.

```r
angle_noise<-read.table(file="Ch12T1.txt",header=T)
str(angle_noise) # Note that data are in WIDE format - need to convert to LONG format first
```

```
## 'data.frame':    10 obs. of  6 variables:
##  $ Absent0 : int  420 420 480 420 540 360 480 480 540 480
##  $ Absent4 : int  420 480 480 540 660 420 480 600 600 420
##  $ Absent8 : int  480 480 540 540 540 360 600 660 540 540
##  $ Present0: int  480 360 660 480 480 360 540 540 480 540
##  $ Present4: int  600 480 780 780 660 480 720 720 720 660
##  $ Present8: int  780 600 780 900 720 540 840 900 780 780
```

```r
# Convert to LONG format
angle_noise$Subject<-as.factor(1:nrow(angle_noise)) # add Subject ID column to angle_noise data
angle_long<-gather(data=angle_noise,key=Noise.Angle,value=RT,-Subject)
angle_long<-separate(data=angle_long,col=Noise.Angle,into=c("Noise","Angle"),sep=-1)
head(angle_long,n=10)
```

```
##    Subject  Noise Angle  RT
## 1        1 Absent     0 420
## 2        2 Absent     0 420
## 3        3 Absent     0 480
## 4        4 Absent     0 420
## 5        5 Absent     0 540
## 6        6 Absent     0 360
## 7        7 Absent     0 480
## 8        8 Absent     0 480
## 9        9 Absent     0 540
## 10      10 Absent     0 480
```

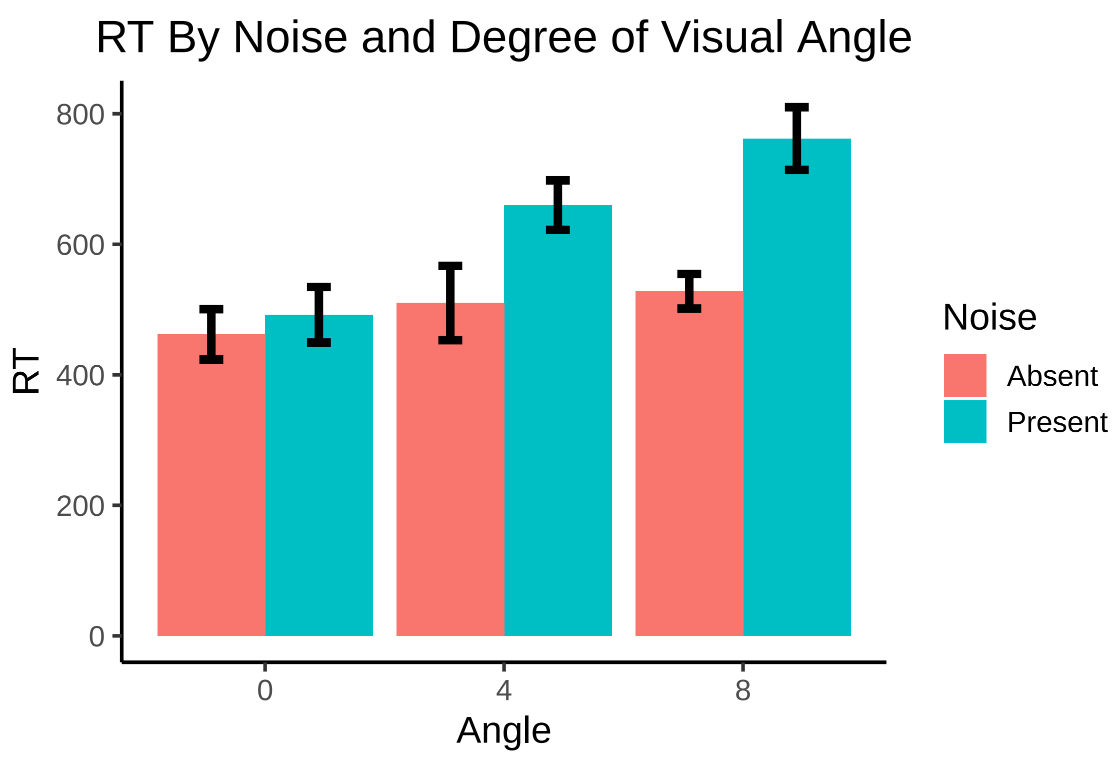
```r
# Get descriptive statistics
angle_means<-Rmisc::summarySEwithin(data=angle_long,measurevar="RT",betweenvars=NULL,
                                   withinvars=c("Noise","Angle"),idvar="Subject") # Note the quotations
```

```
## Automatically converting the following non-factors to factors: Noise, Angle
```

```r
print(angle_means)
```

```
##     Noise Angle  N  RT       sd       se       ci
## 1  Absent     0 10 462 53.92588 17.05286 38.57625
## 2  Absent     4 10 510 79.57387 25.16347 56.92371
## 3  Absent     8 10 528 37.04052 11.71324 26.49719
## 4 Present     0 10 492 59.56509 18.83614 42.61030
## 5 Present     4 10 660 53.02829 16.76902 37.93416
## 6 Present     8 10 762 67.14164 21.23205 48.03024
```

```r
# Make barplot
ggplot(data=angle_means,aes(x=Angle,y=RT,fill=Noise)) +
  geom_bar(stat="identity",position="dodge") +
  theme_classic(base_size=14) +
  geom_errorbar(aes(ymin=RT-ci,ymax=RT+ci),width=0.2,size=1.5,position=position_dodge(0.9)) +
  ggtitle("RT By Noise and Degree of Visual Angle") +
  theme(plot.title = element_text(hjust = 0.5))
```



For data that come from designs with multiple factors, it is often best *not* to include individual subject lines, as the figure can become messy and too busy looking.

Designs with Within- and Between-Subject Factors

```
age_angle<-read.table(file="Ch12T15.txt",header=TRUE)
head(age_angle,n=10) # Data in long format already
```

```
##    Subject   Age Angle  RT
## 1        1 Young     0 450
## 2        2 Young     0 390
## 3        3 Young     0 570
## 4        4 Young     0 450
## 5        5 Young     0 510
## 6        6 Young     0 360
## 7        7 Young     0 510
## 8        8 Young     0 510
## 9        9 Young     0 510
## 10      10 Young     0 510
```

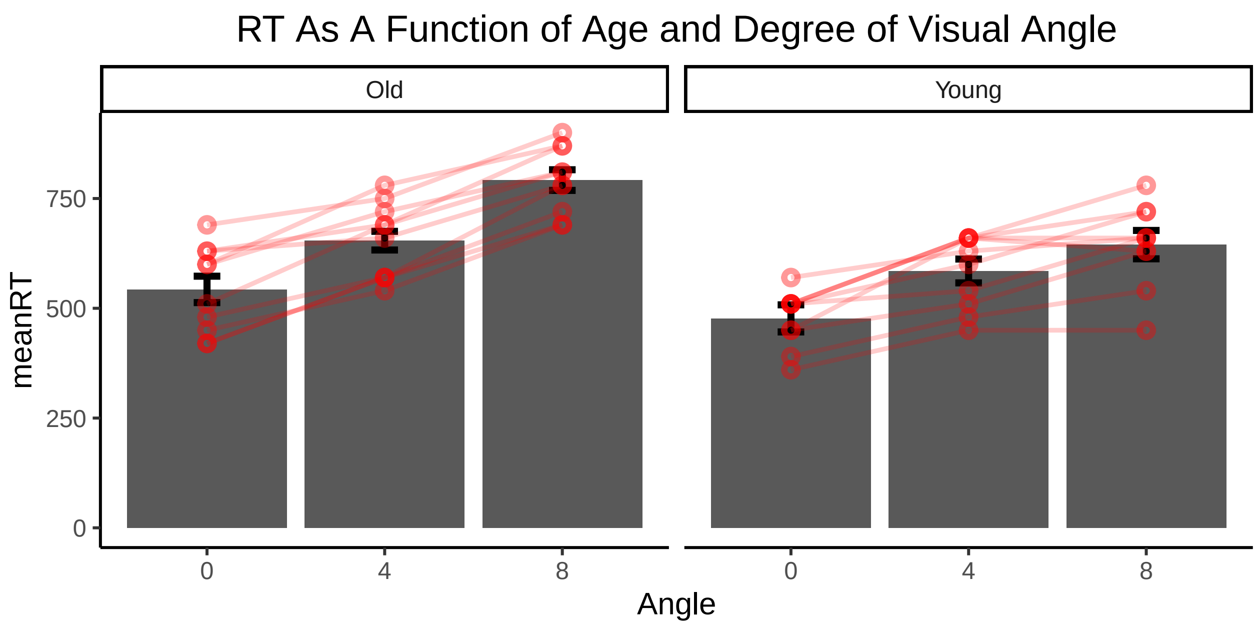
```
age_angle$Angle<-as.factor(age_angle$Angle) # Convert Angle to factor

# Get descriptive statistics using pipe - these means are for plotting
age_angle_means<-age_angle %>%
  group_by(Age,Angle) %>%
  summarize(meanRT=mean(RT),n=n())

print(age_angle_means)
```

```
## # A tibble: 6 x 4
## # Groups:   Age [?]
##   Age   Angle meanRT     n
##   <fct> <fct>  <dbl> <int>
## 1 Old   0        543    10
## 2 Old   4        654    10
## 3 Old   8        792    10
## 4 Young 0        477    10
## 5 Young 4        585    10
## 6 Young 8        645    10
```

```
# Use summarySEwithin to get corrected 95% CIs for within variable

age_angle_CIs<-Rmisc::summarySEwithin(data=age_angle,measurevar="RT",betweenvars="Age",
                                   withinvars="Angle",idvar="Subject") # Note the quotations

print(age_angle_CIs)
```

```
##     Age Angle  N  RT       sd        se       ci
## 1   Old     0 10 496 42.03173 13.291601 30.06769
## 2   Old     4 10 607 29.69287  9.389711 21.24100
## 3   Old     8 10 745 32.88870 10.400321 23.52716
## 4 Young     0 10 524 43.12772 13.638182 30.85171
## 5 Young     4 10 632 37.94733 12.000000 27.14589
## 6 Young     8 10 692 45.16636 14.282857 32.31007
```

```
# Attach ci column to age_angle_means
age_angle_means<-age_angle_means %>%
  ungroup() %>%
  mutate(CI=age_angle_CIs$ci)

# Make barplot with individual subject lines for within variable
ggplot(data=age_angle_means,aes(x=Angle,y=meanRT)) +
  facet_grid(~Age) +
  geom_bar(stat="identity",position="dodge") +
  geom_errorbar(aes(ymin=meanRT-CI,ymax=meanRT+CI),position=position_dodge(0.9),size=1.5,width=0.15) +
  geom_point(data=age_angle,aes(x=Angle,y=RT),size=1.5,shape=1,color="red",
             stroke=2,alpha=.4,position=position_dodge(0.9)) +
  geom_line(data=age_angle,aes(x=Angle,y=RT,group=Subject),lwd=1,color="red",alpha=0.2) +
  theme_classic(base_size=14) +
  ggtitle("RT As A Function of Age and Degree of Visual Angle") +
  theme(plot.title = element_text(hjust = 0.5))
```



Line Graphs

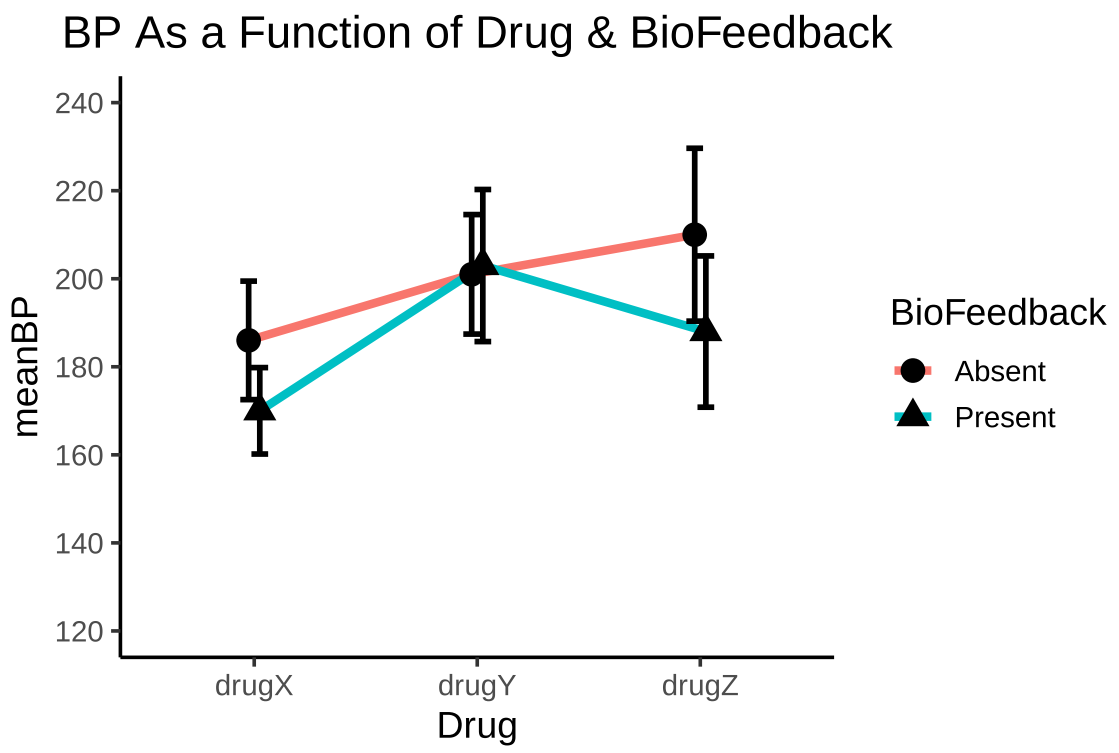
Another common form of data visualization used in figures are line graphs. As you might expect by now, to make a line graph, you need to use the **geom_line** command within *ggplot2*. Let's re-make the above bar graphs in line graph form starting with the between-subjects design example above.

Between-Subject Designs

```
print(bp_means_withCI)
```

```
## # A tibble: 6 x 6
## # Groups:   Drug [3]
##   Drug  BioFeedback meanBP   SEM     n    CI
##   <fct> <fct>        <dbl> <dbl> <int> <dbl>
## 1 drugX Absent         186  4.85     5 13.5
## 2 drugX Present        170  3.54     5  9.82
## 3 drugY Absent         201  4.89     5 13.6
## 4 drugY Present        203  6.22     5 17.3
## 5 drugZ Absent         210  7.07     5 19.6
## 6 drugZ Present        188  6.19     5 17.2
```

```
# Convert BP barplot from above into line graph
ggplot(bp_means_withCI,aes(x=Drug,y=meanBP)) +
  geom_line(aes(group=BioFeedback,color=BioFeedback),lwd=1.5,position=position_dodge(width=0.1)) +
  geom_point(aes(shape=BioFeedback),size=4,position=position_dodge(width=0.1)) +
  geom_errorbar(aes(ymin=meanBP-CI,ymax=meanBP+CI,group=BioFeedback),width=0.15,size=1,
                position=position_dodge(width=0.1)) +
  scale_y_continuous(limits=c(120,240),breaks=seq(120,240,20)) +
  theme_classic(base_size=14) +
  ggtitle("BP As a Function of Drug & BioFeedback") +
  theme(plot.title = element_text(hjust = 0.5))
```



It is probably cleaner visually to leave off the individual data points when using a line graph to illustrate the results of a between-subjects design.

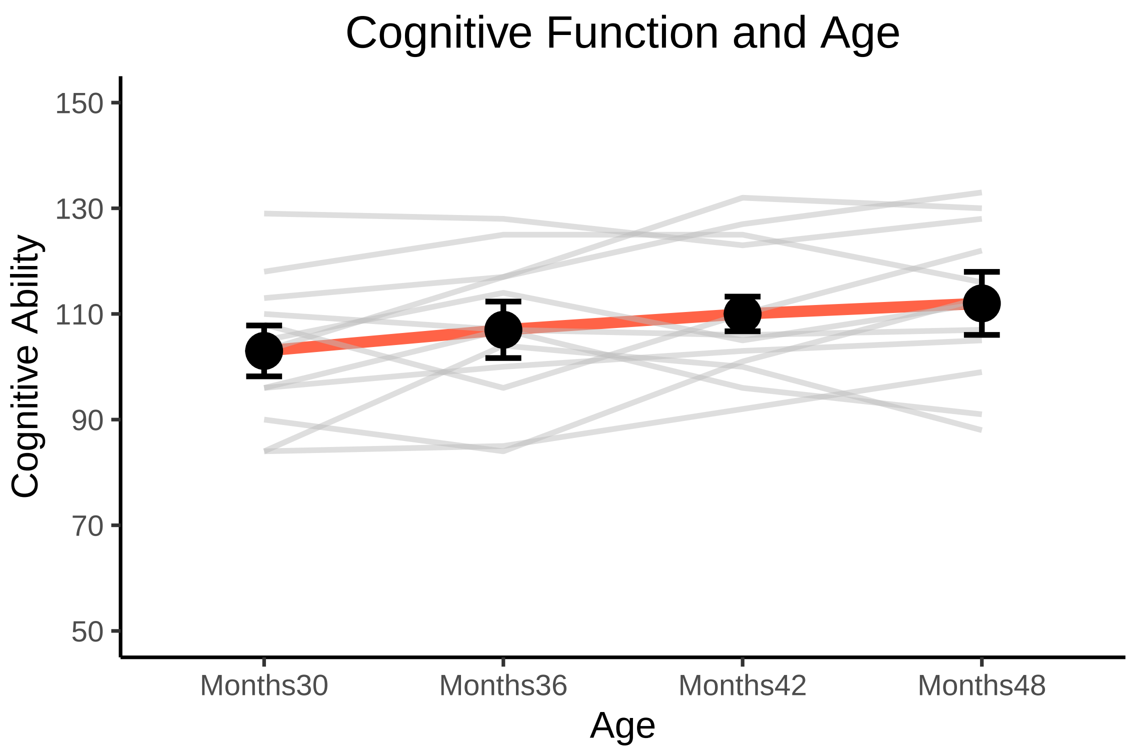
Within-Subject Designs: One-Way

Now let's convert the bar graph created from the *baby_data* data set into a line graph.

```
print(baby_means)
```

```
##        Age  N Score       sd       se       ci
## 1 Months30 12   103 7.579878 2.188122 4.816025
## 2 Months36 12   107 8.413553 2.428784 5.345717
## 3 Months42 12   110 5.163978 1.490712 3.281035
## 4 Months48 12   112 9.393744 2.711740 5.968500
```

```
# Convert bp barplot from above into line graph
ggplot(baby_means,aes(x=Age,y=Score)) +
  geom_line(aes(group=1),lwd=2,color="tomato") + # Note that group = 1
  geom_line(data=baby_data,aes(x=Age,y=Score,group=Subject),color="gray",lwd=1,alpha=0.5) +
  geom_point(size=6,shape=19) +
  geom_errorbar(aes(ymin=Score-ci,ymax=Score+ci),width=0.15,size=1) +
  scale_y_continuous(limits=c(50,150),breaks=seq(50,150,20)) +
  theme_classic(base_size=14) +
  labs(y="Cognitive Ability") +
  ggtitle("Cognitive Function and Age") +
  theme(plot.title = element_text(hjust = 0.5))
```



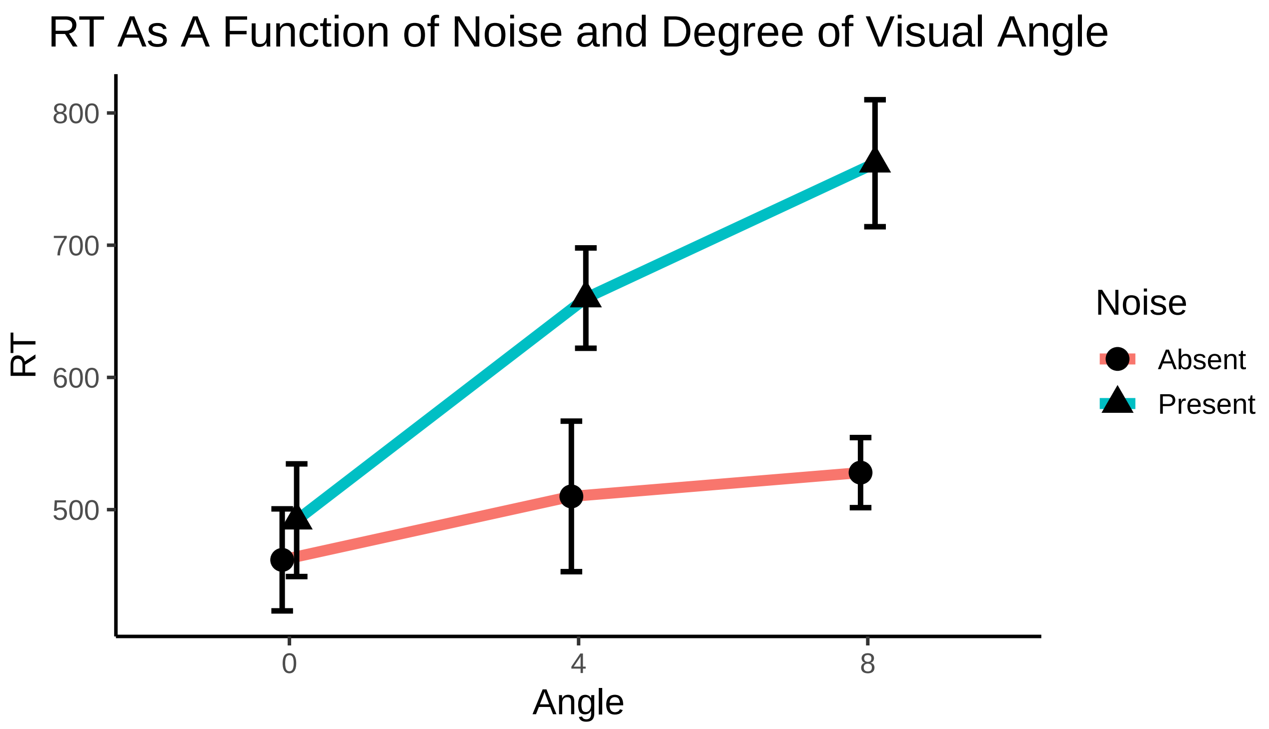
Within-Subject Designs: Two-Way

Now let's convert the bar graph created from the *angle_noise* data set into a line graph.

```
print(angle_means)
```

```
##     Noise Angle  N  RT       sd       se       ci
## 1  Absent     0 10 462 53.92588 17.05286 38.57625
## 2  Absent     4 10 510 79.57387 25.16347 56.92371
## 3  Absent     8 10 528 37.04052 11.71324 26.49719
## 4 Present     0 10 492 59.56509 18.83614 42.61030
## 5 Present     4 10 660 53.02829 16.76902 37.93416
## 6 Present     8 10 762 67.14164 21.23205 48.03024
```

```
ggplot(data=angle_means,aes(x=Angle,y=RT)) +
  geom_line(aes(group=Noise,color=Noise),lwd=2,position=position_dodge(0.1)) +
  geom_point(size=4,aes(shape=Noise),position=position_dodge(0.1)) +
  geom_errorbar(aes(ymin=RT-ci,ymax=RT+ci,group=Noise),width=0.15,size=1,
                position=position_dodge(0.1)) +
  theme_classic(base_size=14) +
  ggtitle("RT As A Function of Noise and Degree of Visual Angle") +
  theme(plot.title = element_text(hjust = 0.5))
```



Saving Plots and Creating Figures

R contains a suite of built-in functions that allow you to create and save many different kinds of picture file types:

```
bmp()
jpeg()
png()
tiff()
```

Within these functions, we can control the size of the plot we wish to create, the unit that defines the size of the image (e.g., pixels), as well as the resolution of the image in dpi (dots per inch). To illustrate how to use these functions, let's create a *.tiff* file in which we save one of the plots we created in the preceeding examples.

```
# Open new file and turn on graphics device
tiff(filename="my_figure.tiff",width=2000,height=1200,units="px",res=300)

  ggplot(data=age_angle_means,aes(x=Angle,y=meanRT)) +
  facet_grid(~Age) +
  geom_bar(stat="identity",position="dodge") +
  geom_errorbar(aes(ymin=meanRT-CI,ymax=meanRT+CI),position=position_dodge(0.9),size=1.5,width=0.15) +
  geom_point(data=age_angle,aes(x=Angle,y=RT),size=1.5,shape=1,color="red",
             stroke=2,alpha=.4,position=position_dodge(0.9)) +
  geom_line(data=age_angle,aes(x=Angle,y=RT,group=Subject),lwd=1,color="red",alpha=0.2) +
  theme_classic(base_size=14) +
  ggtitle("RT As A Function of Age and Degree of Visual Angle") +
  theme(plot.title = element_text(hjust = 0.5))

dev.off() # Turn off the graphic device that writes the image to .tiff file.
```