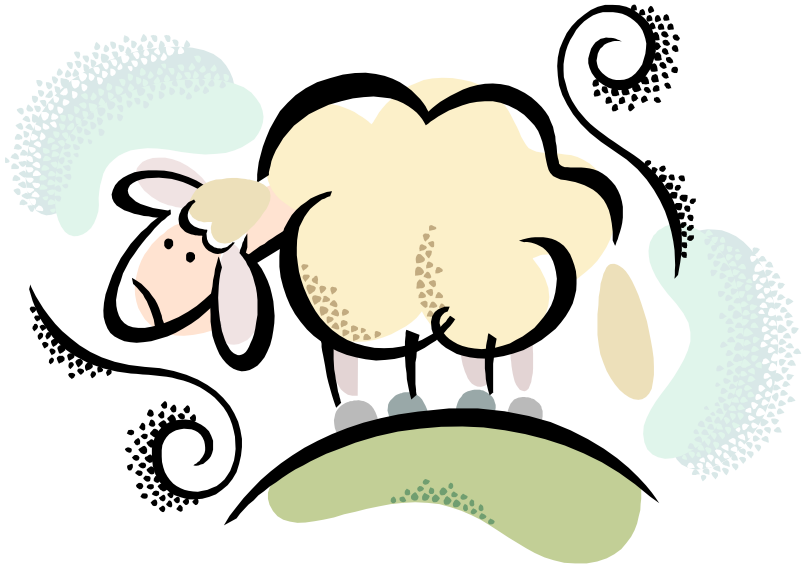


Graphics I

Legal Stuff

This presentation as well as other presentations and documents found on the sheepsqueezers.com website may contain quoted material from outside sources such as books, articles and websites. It is our intention to diligently reference all outside sources. Occasionally, though, a reference may be missed. No copyright infringement whatsoever is intended, and all outside source materials are copyright of their respective author(s).

R Lecture Series

Non-Programming Introduction	*Programming I*	*Programming II*
Graphics I		*Advanced Topics*

Charting Our Course

- Goal of this Presentation
- Overview of Graphics in R
- Graphics using the `graphics` Package
- Graphics using the `ggplot2` Package
- Graphics using the `lattice` Package
- Using Graphics Devices to Save Your Graphs
- Appendix

Goal of this Presentation

The goal of this presentation is to teach you the basics of R graphics using three of its freely available graphics packages: `graphics`, `ggplot2` and `lattice`. The `grid` package is used by several packages to aid in graphing and is not necessarily used alone to create graphs on a day-to-day basis by itself; thus, we will not discuss it in this presentation.

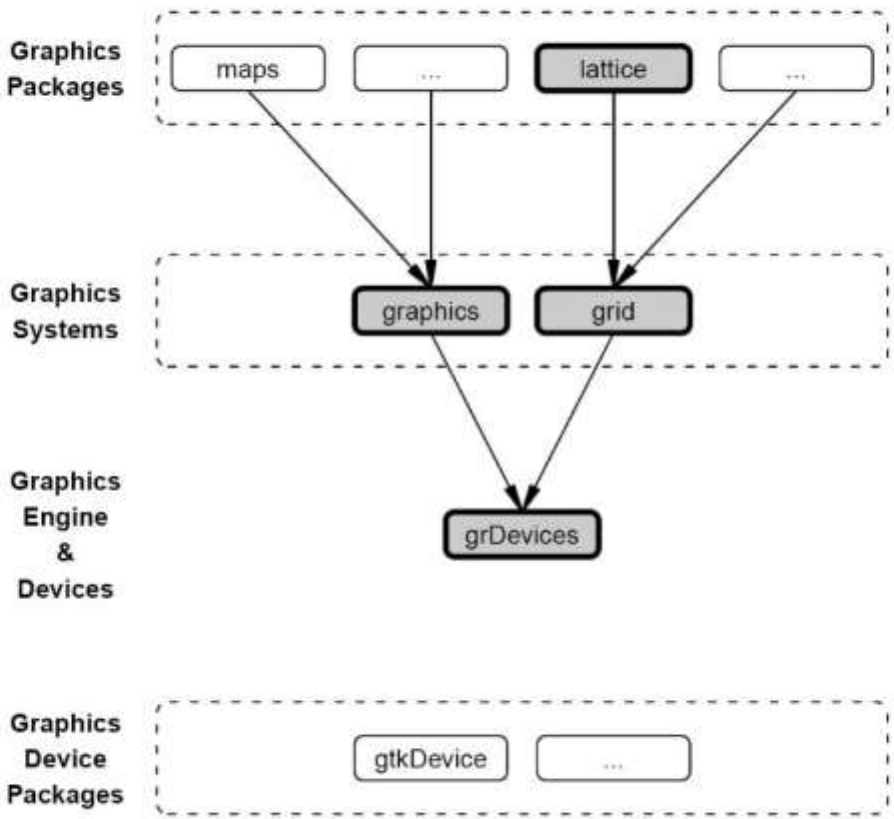
Note that many other packages are available on the CRAN website (cran.r-project.org) that also create graphs. For example, the `corrgram` package that was introduced in the Programming I presentation creates graphs of correlations via the `corrgram()` function. These additional graphing capabilities will not be discussed in this presentation, but please be aware of them.

Note that the `grDevices` package allows you to save graphs to any one of several formats such as PDF and Postscript files, to bitmap files, jpg files, etc. While this is not used to perform graphing, per se, it is used to save graphs to disk, so we will be discussing this package towards the end of the lecture.

Overview of Graphics in R

Overview of Graphics in R

The diagram below shows you how some of R's graphing packages are related.



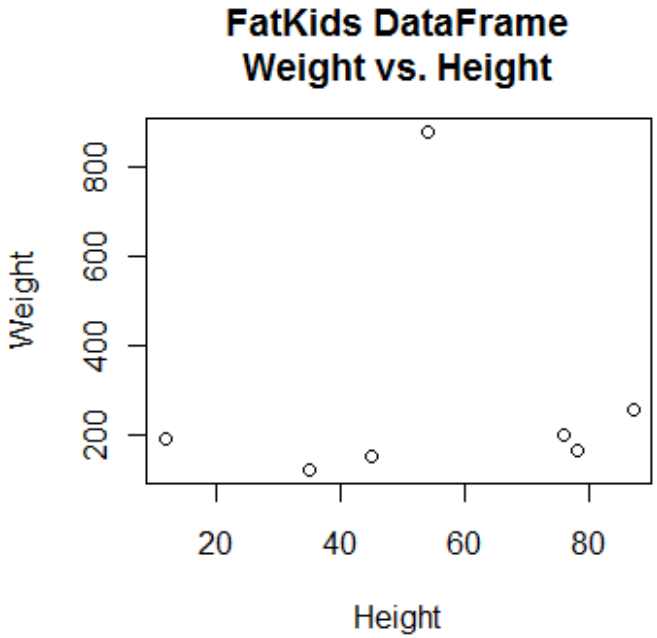
Overview of Graphics in R

Since SAS/Graph is the only graphing package available in SAS, if you've ever used it, you've gotten used to its syntax and can make reasonable graphs fairly quickly. Unfortunately, each one of the graphing packages in R has its own syntax and even the names of some of the graphing functions are different.

Briefly, when using the `graphics` package, you use what is called a *high-level* plotting function on the R command line, and then add additional *low-level* functions one after the other each on a separate R command line. For example, using the FatKids dataset, let's plot the Weight on the Y-Axis and the Height on the X-Axis, use the high-level function called `plot()`:

```
> plot(dfFatKids[,c("Height","Weight")],main="FatKids DataFrame\nWeight vs. Height")
```

The resulting plot is displayed →



Overview of Graphics in R

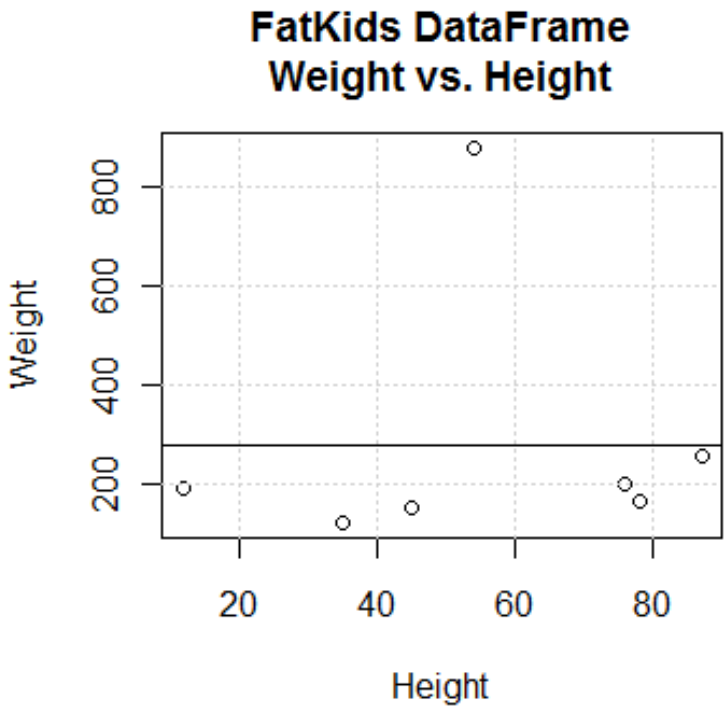
While the graph is still displayed in the window, you can add additional low-level graphing functions such as grid lines and a horizontal line through the Y-Axis at the average Weight:

```
> plot(dfFatKids[,c("Height","Weight")],main="FatKids DataFrame\nWeight vs. Height")
> grid()  # add grid lines…sweet!
> abline(mean(dfFatKids$Weight),0) #add horizontal axis at avg weight
```

The resulting graph looks like this:

Note that as long as there has not been another high-level plotting function issued at the command line, you can issue additional low-level plotting functions to modify the initial graph.

Note that if you are issuing these commands at the command line and the graphing window is being displayed, you can use *File…Copy to the clipboard* functionality to get a copy of the graph and then paste it in a document.



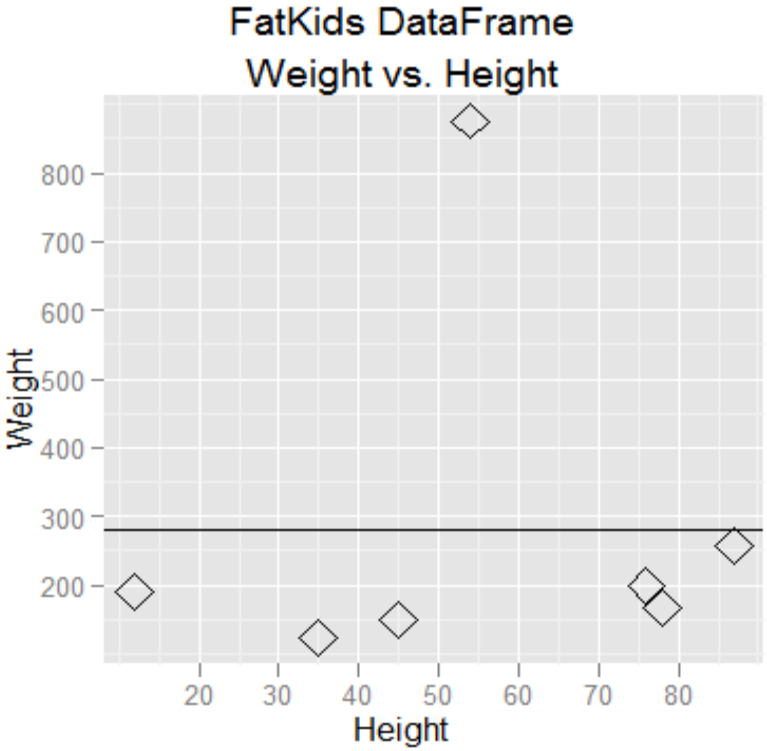
Overview of Graphics in R

Next, the `ggplot2` package uses a series of plus-sign delimited commands to create its graphs. Let's create the same graph using the function `ggplot()`:

```
> p <- ggplot(dfFatKids,aes(x=Height,y=Weight))
> p +
    geom_point(shape=23, size=5.0) +
    geom_hline(aes(yintercept=mean(dfFatKids$Weight))) +
    opts(title = "FatKids DataFrame\nWeight vs. Height")
```

Take note that the variable p is created with the "high-level" plotting function `ggplot()`, then the additional low-level functions are *added* to that.

Note that `ggplot()` alone does not actual create a graph when using the `ggplot2` package. You need the addition of at least `geom_point()` (or other similar function) to draw the points on the graph. Note also that the `ggplot2` package comes with the `qplot()` function to make creating graphs a lot easier, but it is limited.

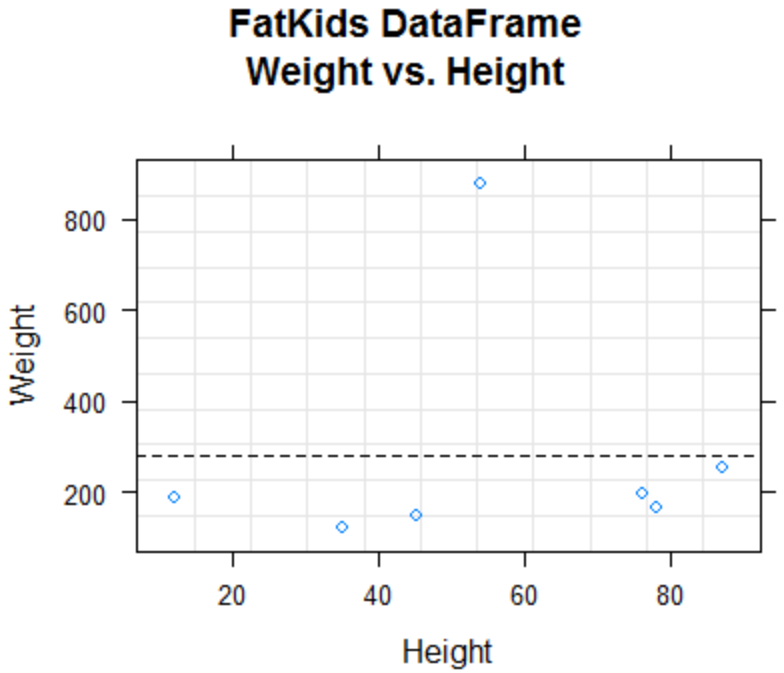


Overview of Graphics in R

Finally, let's use the `lattice` package to graph the Height vs. Weight in the FatKids dataset:

```
xyplot(Weight~Height,
       data=dfFatKids,
       xlab="Height",
       ylab="Weight",
       main="FatKids DataFrame\nWeight vs. Height",
       panel=function(x,y) {
        panel.grid(h=10,v=10)
        panel.abline(h=mean(dfFatKids$Weight),v=0,lty="dashed")
        panel.xyplot(x,y)
       }
)
```

As you see above, the `lattice` package function `xyplot()` requires an additional panel function containing calls to the `panel.grid()` function to draw the grid; `panel.abline()` to draw the horizontal line; and, finally, `panel.xyplot()` to draw the points on the graph. Normally, you won't need to modify the panel function and its default is usually fine.



Overview of Graphics in R

Now, you may be wondering why you should learn three packages. Wouldn't learning just one package suffice? Each package can create some graphs the others can't, so when you're creating graphs, you may need to create one graph from one package, but three from another package.

I have placed a description of each function available in each of the three packages in the appendix of this presentation.

Note that this is definitely not the *last word* in graphing with R and with these three packages. Please purchase the books recommended, read the R documentation and view the related websites.

Graphics using the `graphics` Package

Graphics Using the graphics Package

In this section, we will learn about the graphics that can be produced by the graphics package. Below are the high-level graphing functions available in the graphics package:

- assocplot – Association Plots – Produce a Cohen-Friendly association plot indicating deviations from independence of rows and columns in a 2-dimensional contingency table using Pearson's Chi-Squared statistic.
- barplot – Bar Plots – Creates a bar plot with vertical and horizontal bars.
- boxplot – Box Plots – Produces box-and-whisker plot(s) of given (grouped) values.
- cdplot – Conditional Density Plots – Computes and plots conditional densities describing how the conditional distribution of a categorical variable y changes over a numerical variable x.
- contour – Display Contours – Create a contour plot, or add contour lines to an existing plot.
- coplot – Conditioning Plots – Produces two variants of the conditioning plots.
- curve – Draw Function Plots – Draws a curve corresponding to the given function.
- dotchart – Cleveland Dot Plots – Draws a Cleveland Dot Plot.
- filled.contour – Level (Contour) Plots – Produces a contour plot with the areas between the contours filled in solid colors.
- fourfoldplot – Fourfold Plots – Create a fourfold display of a 2 by 2 by k contingency table on the current graphics device, allowing for the visual inspection of the association between two dichotomous variables in one or several populations (strata).
- hist – Histograms – Computes a histogram.
- matplot – Plot Columns of Matrices – Plot the columns of one matrix against the columns of another matrix (first against first, second against second, etc.).
- mosaicplot – Mosaic Plots – Plots a mosaic plot on the current graphics device.
- pairs – Scatterplot Matrices – A matrix of scatterplots is produced for each pair of numeric variables in the input data.
- persp – Perspecitive Plots – Draws perspective plots of surfaces over the x-y plane.
- pie – Pie Charts – Draws a Pie Chart
- plot – X-Y Plotting – Generic function for plotting of R objects.
- smoothScatter – Scatterplots with Smoothed Densities Color Representation – Produces a smoothed color density representation of the scatterplot, obtained through a kernel density estimate.

Graphics Using the `graphics` Package

- spineplot – Spline Plots and Spinograms – Spine plots are a special case of mosaic plots, and can be seen as a generalization of stacked (or highlighted) bar plots. Spinograms are an extension of histograms.
- stars – Star (Spider/Radar) Plots and Segment Diagrams – Draws star plots or segment diagrams of a multivariate data set.
- stem – Stem-and-Leaf Plots – Produces a stem-and-leaf plot of the given values.
- stripchart – 1-Dimensional Scatter Plots – stripchart produces one dimensional scatter plots (or dot plots) of the given data. These plots are a good alternative to boxplots when sample sizes are small.
- sunflowerplot – Sunflower Scatter Plot – Multiple points are plotted as "sunflowers" with multiple leaves ("petals") such that overplotting is visualized instead of accidental and invisible.

There are two functions you can use to locate the points when your graph has been displayed:

- identify – Identify Points in a Scatter Plot – Reads the position of the graphics pointer when the left mouse button is pressed; it then searches the data set for the closest point to the mouse pointer and returns the row number from the data set.
- locator – Graphical Input – Reads the position of the graphics cursor when the left mouse button is pressed.

Additional low-level functions are used to annotate the graph:

- abline – Add Straight Lines to a Plot – Adds one or more straight lines through the current plot.
- arrows – Add Arrows to a Plot – draws arrows between pairs of points.
- axis – Add an Axis to a Plot – Adds an axis to the current plot allowing the specification of the side, position, labels, and other options.
- box – Draw a box around a plot – Draws a box around the current plot in the given color and linetype.
- grid – Add Grid to a Plot – Adds an nx by ny rectangular grid to an existing plot.
- legend – Add Legends to a Plot – Can be used to add legends to plots.

Graphics Using the `graphics` Package

Additional low-level functions are used to annotate the graph (continued):

- `lines` – Adds Connected Line Segments to a Plot – Joins Points with Line Segments
- `mtext` – Write Text into the Margins of a Plot – Text is written in one of the four margins of the current figure region or one of the outer margins of the device region.
- `points` – Add Points to a Plot – Draws a sequence of points, centered, at the specified coordinates.
- `polygon` – Polygon Drawing – Draws a polygon whose vertices are provided.
- `rect` – Draw One or More Rectangles – Draws a rectangle (or sequence of rectangles) with the given coordinates, fill and border colors.
- `rug` – Add a Rug to a Plot – Adds a rug representation (1-d plot) of the data to a plot.
- `segments` – Add Line Segments to a Plot – Draw line segments between pairs of points.
- `symbols` – Draw Symbols on a Plot – This function draws symbols (circles, squares, stars, etc.) on a plot.
- `text` – Add Text to a Plot – Draws a string at the coordinates provided.
- `title` – Plot Annotations – Used to add labels to a plot such as title , subtitle, x-axis label, y-axis label, etc.

Finally, similar to SAS's `goptions` statement, you can use the `par()` function to view or set the current graphics parameters:

- `par()`

While we won't show you all of the graph types listed above, we will show you some of them.

Graphics Using the `graphics` Package

Recall the Fat Kids data frame from the previous lectures. We've expanded upon it so that there is a Height and Weight entry for each month of 2008 and 2009:

```
> dfFatKidsHist
  FirstName DateOfMeas Height Weight Gender Group
1    ALBERT 2009-01-01     25    110      M  Grp1
2    ALBERT 2009-02-01     26    110      M  Grp1
3    ALBERT 2009-03-01     27    110      M  Grp1
4    ALBERT 2009-04-01     28    110      M  Grp1
5    ALBERT 2009-05-01     30    120      M  Grp1
6    ALBERT 2009-06-01     32    120      M  Grp1 …
```

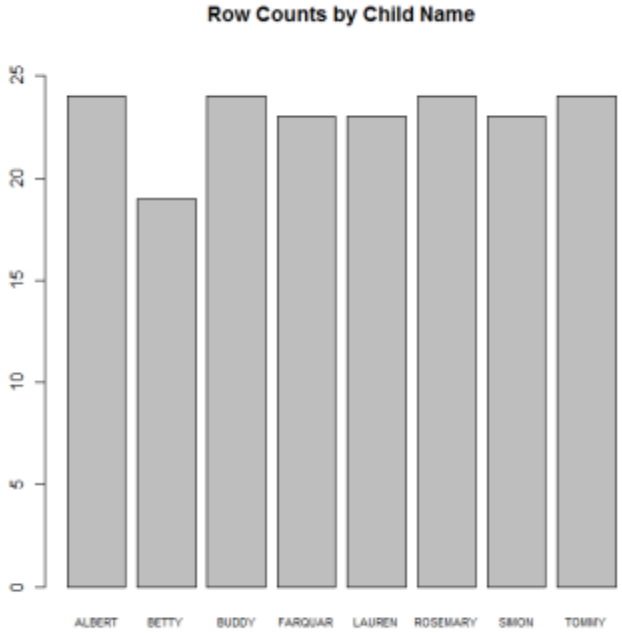
Let's produce a bar chart of the number of entries for each child. Note that the `barplot()` function expects the data to be summarized and stored in a vector or matrix whose columns will be represented as bars in the bar chart. Also, the results of the `table()` function seem to work as well:

```
> tFatKidsHist_FirstName_Counts <- table(dfFatKidsHist$FirstName)
> tFatKidsHist_FirstName_Counts
 ALBERT    BETTY    BUDDY  FARQUAR   LAUREN ROSEMARY    SIMON    TOMMY
     24       19       24       23       23       24       23       24
```

To produce the barplot, we use the following command:

Graphics Using the `graphics` Package

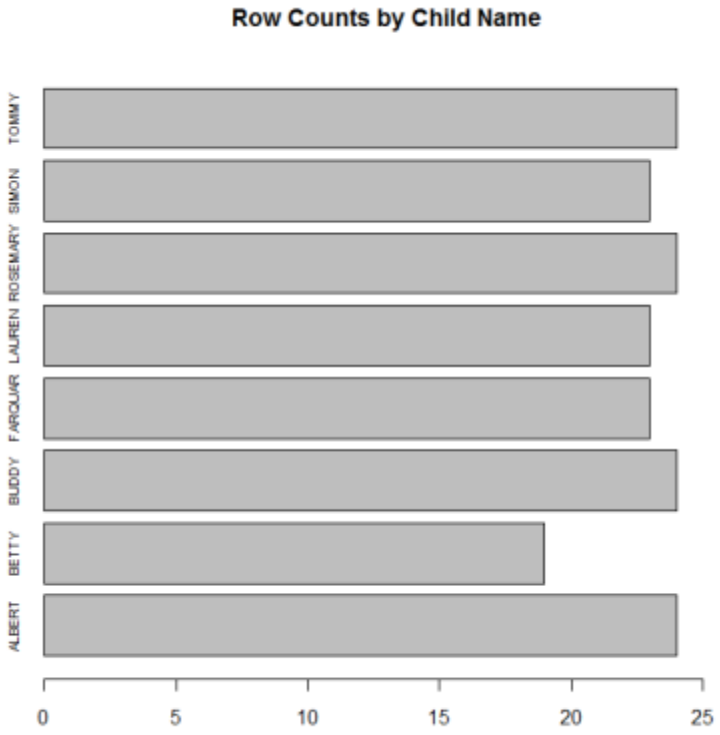

```
> barplot(tFatKidsHist_FirstName_Counts,cex.names=.70,main="Row Counts by Child Name",ylim=c(0,26))
```



The main parameter represents the graph title, the cex.names parameter is a value used to size the text. The ylim parameter gives a start and end value for the y-axis values. Note how the columns of the `table()` function are represented on the chart. We can also create a horizontal bar chart:

```
> barplot(tFatKidsHist_FirstName_Counts,cex.names=.70,
          main="Row Counts by Child Name",horiz=TRUE,xlim=c(0,26))
```

Graphics Using the `graphics` Package



Next, let's try to produce the row counts by year by gender. Given that we have two years, let's summarize the data to the year by gender. Here is the code to summarize the data down to the year by gender:

```
> tFatKidsHist_FirstName_Counts_by_Year <-
                              table(format(dfFatKidsHist$DateOfMeas,"%Y"),dfFatKidsHist$Gender)
> tFatKidsHist_FirstName_Counts_by_Year
      F  M
2008 31 58
2009 35 60
```

Graphics Using the `graphics` Package

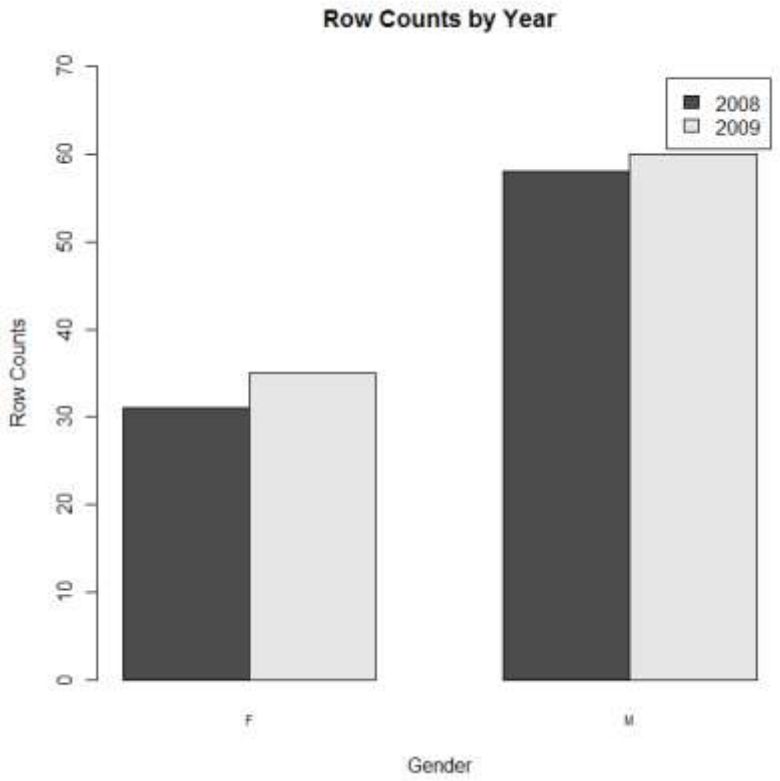
Here is the code to produce the bar chart:

```
> barplot(tFatKidsHist_FirstName_Counts_by_Year,cex.names=.7,
          main="Row Counts by Year",ylim=c(0,70),legend=TRUE,beside=TRUE,
          xlab="Gender",ylab="Row Counts")
```

As you see above, the legend parameter lets R know that you want to produce a legend. The beside option tells the `barplot()` function to graph the bars side-by-side. Again, I have modified the ylim parameter to allow enough room to have the legend show comfortably on the graph.

Also note that xlab labels the x-axis and ylab labels the y-axis.

Take note that the year values on the previous page are row names and not part of any column!

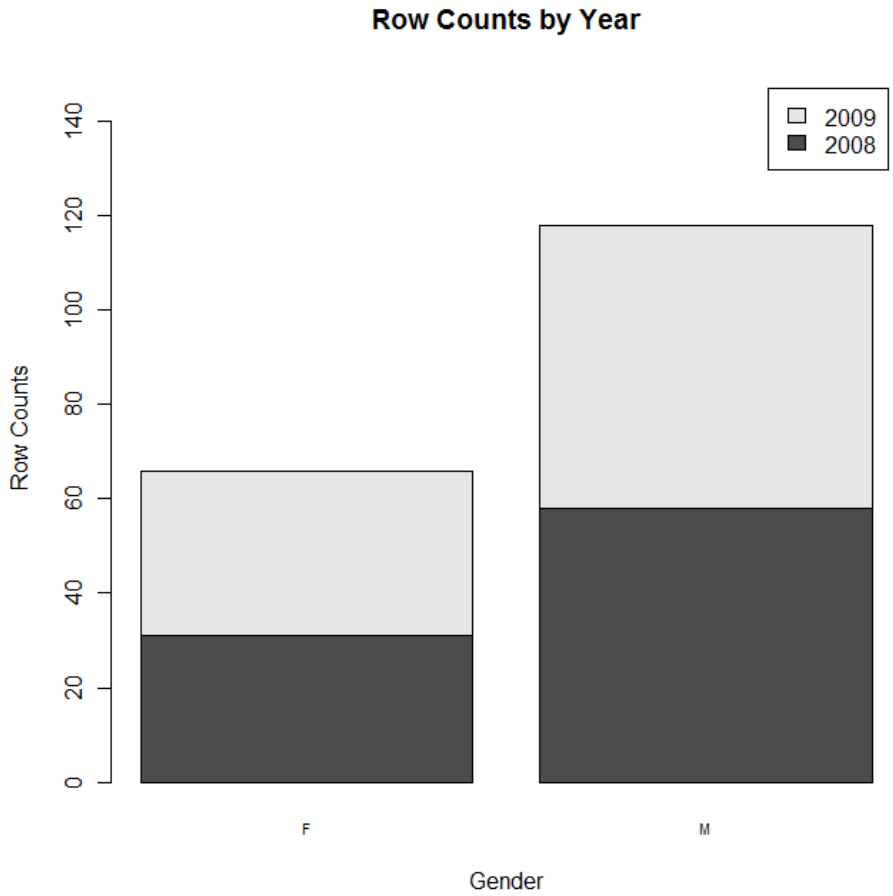


Graphics Using the `graphics` Package

If you want to have stacked bar charts, use this code:

```
> barplot(tFatKidsHist_FirstName_Counts_by_Year,cex.names=.7,
          main="Row Counts by Year",ylim=c(0,150),legend=TRUE,beside=FALSE,
          xlab="Gender",ylab="Row Counts")
```

As you see above, the beside parameter is set to FALSE indicating that stacked bars are desired:



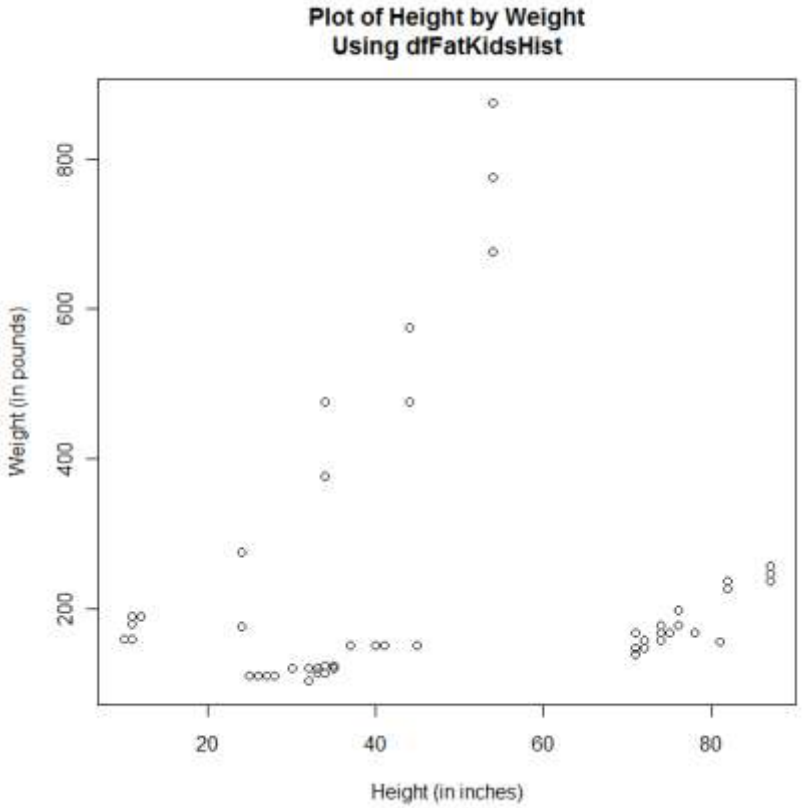
Graphics Using the `graphics` Package

You can plot points on a graph using the `plot()` function. This function, unlike the `barplot()` function, does not need to have the data summarized:

```
> plot(dfFatKidsHist$Height,dfFatKidsHist$Weight,type="p",
       main="Plot of Height by Weight\nUsing dfFatKidsHist",
       xlab="Height (in inches)",ylab="Weight (in pounds)")
```

As you see, the x-axis the first parameter and the y-axis is the second. The type parameter is a "p" for points. It can also be "l" for connected lines, "b" for points and lines, "h" for vertical spiked lines, etc.

Now, I know that the data has a lot of repeated Height/Weight values that will be plotted on top of each other on the graph. These overlayed points will not be shown on the graph, as you can see to the right. In order to see whether several points are on top of each other, you can either use the `jitter()` function to move the Height (say) values slightly left or right of the actual value, or use the sunflowerplot.

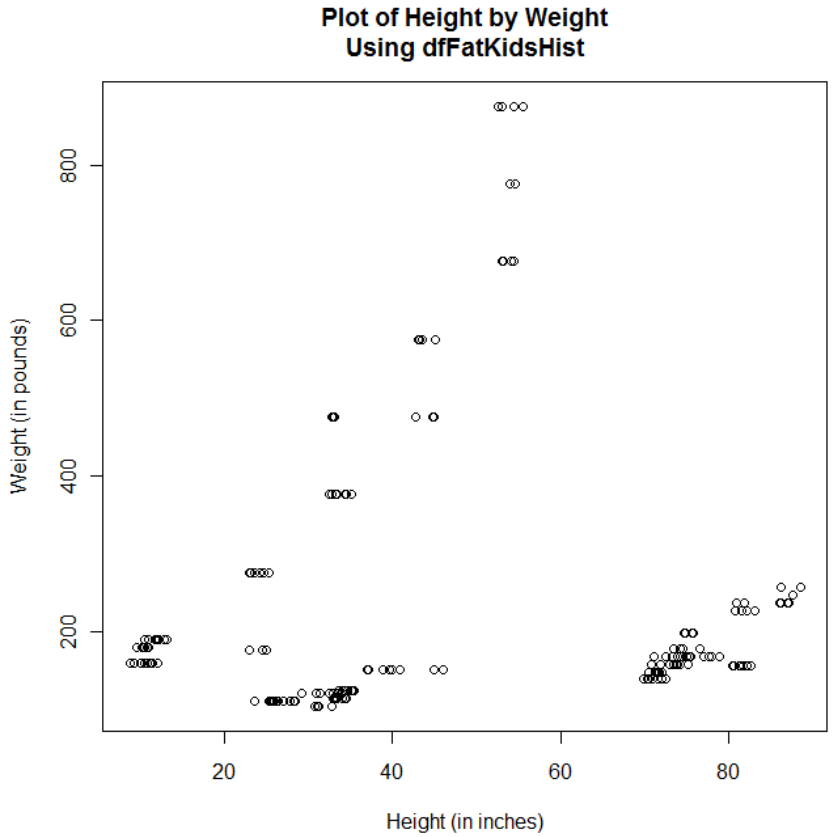


Graphics Using the `graphics` Package

Here is the modified code to jitter the Height:

```
> plot(jitter(dfFatKidsHist$Height,amount=0),dfFatKidsHist$Weight,type="p",
       main="Plot of Height by Weight\nUsing dfFatKidsHist",
       xlab="Height (in inches)",ylab="Weight (in pounds)")
```

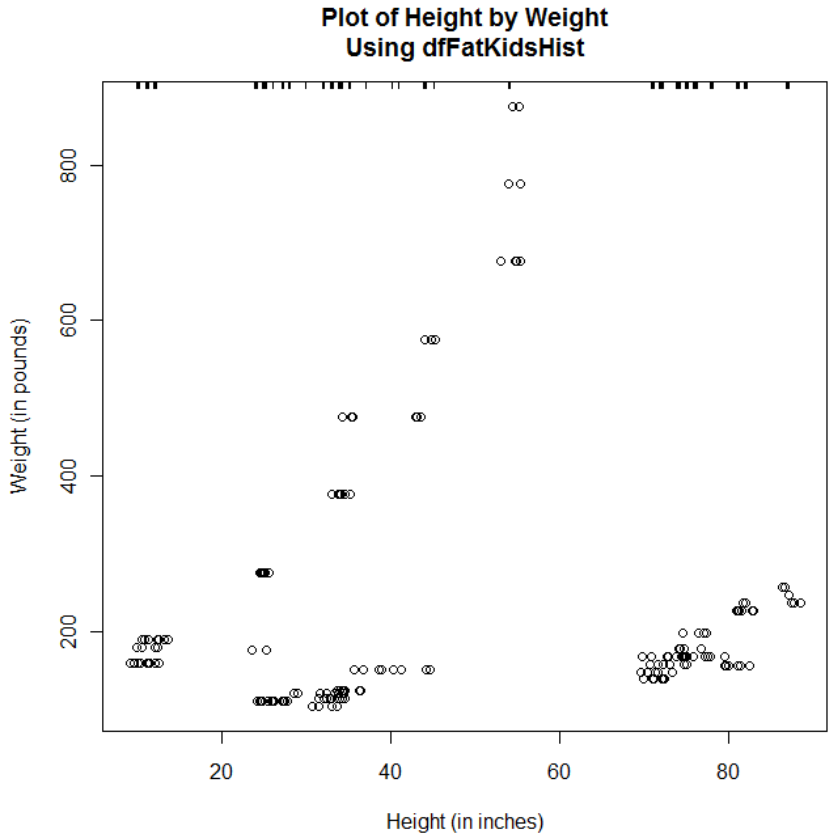
As you see, the `jitter()` function takes a variable as well as the amount to jitter. As you see, amount is set to zero, so the `jitter()` function will compute a nice jitter itself. If it seems that you need glasses when looking at this graph, it's because the Height's have been slightly lowered or raised so that they are no longer exactly on top of each other.



Graphics Using the `graphics` Package

Instead of jittering your data, another nice feature is to add a rug to the plot using the `rug()` low-level function. A rug is a one-dimensional representation of the density of points. The more points that are on top of each other, the darker the rug. At this point, you can decide if you want to jitter the plot itself since the rug helps display the overlap:

```
plot(jitter(dfFatKidsHist$Height,amount=0),dfFatKidsHist$Weight,type="p",
       main="Plot of Height by Weight\nUsing dfFatKidsHist",
       xlab="Height (in inches)",ylab="Weight (in pounds)")
rug(jitter(dfFatKidsHist$Height),ticksize=.01,side=3)
```

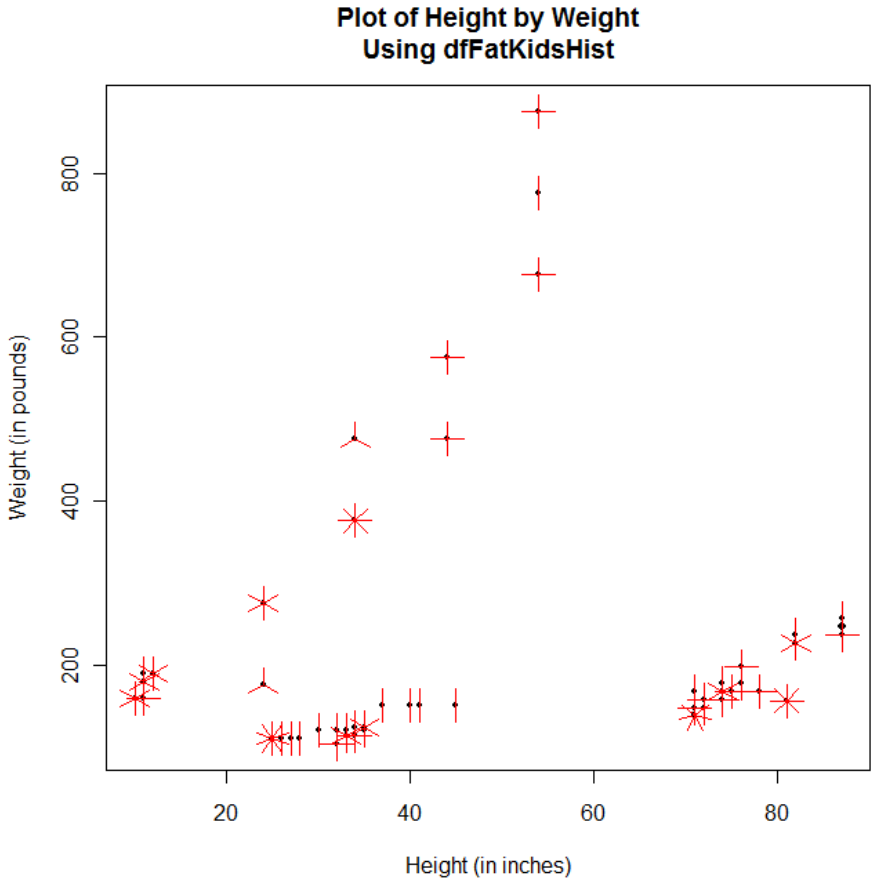


Graphics Using the `graphics` Package

Here is the code to create a sunflower plot:

```
> sunflowerplot(dfFatKidsHist$Height,dfFatKidsHist$Weight,
                main="Plot of Height by Weight\nUsing dfFatKidsHist",
                xlab="Height (in inches)",ylab="Weight (in pounds)")
```

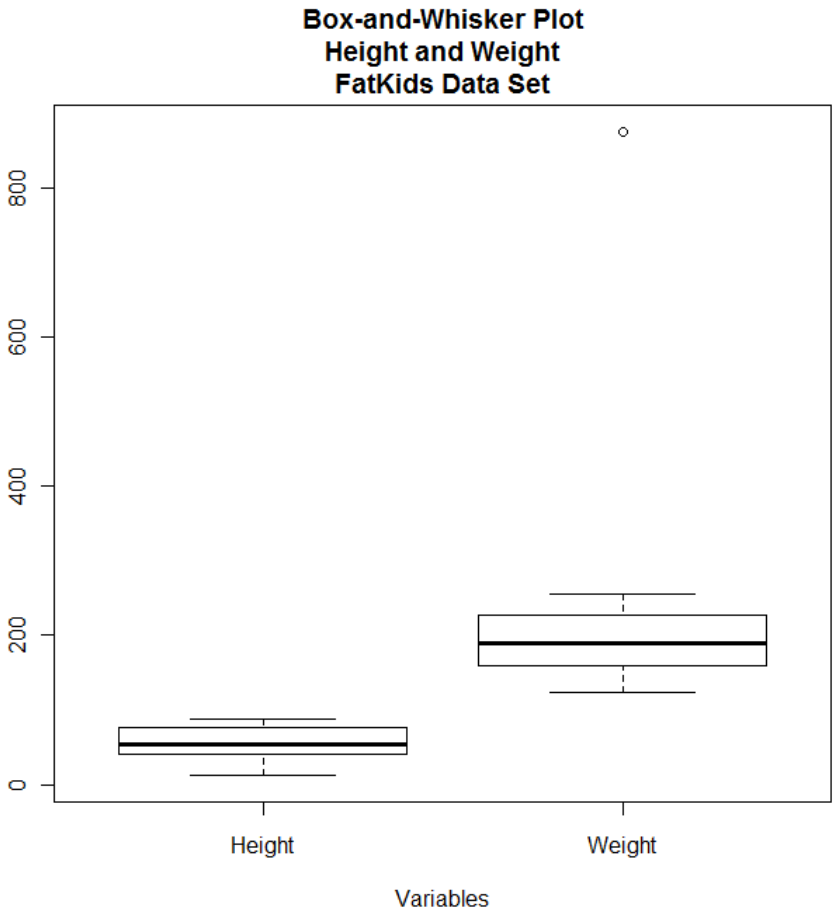
This is very similar to the `plot()` function except that where there are two points on top of each other, two red lines are drawn. When there are three points on top of each other, you will see three red lines, and so on.



Graphics Using the `graphics` Package

Who doesn't love a good box-and-whisker plot, eh? We can use the `boxplot()` function to created a box plot. Here, we create a box-and-whisker plot of the Height and Weight from the dfFatKids data frame:

```
> boxplot(dfFatKids[,c("Height","Weight")],
          main="Box-and-Whisker Plot\nHeight and Weight\nFatKids Data Set",
          xlab="Variables")
```

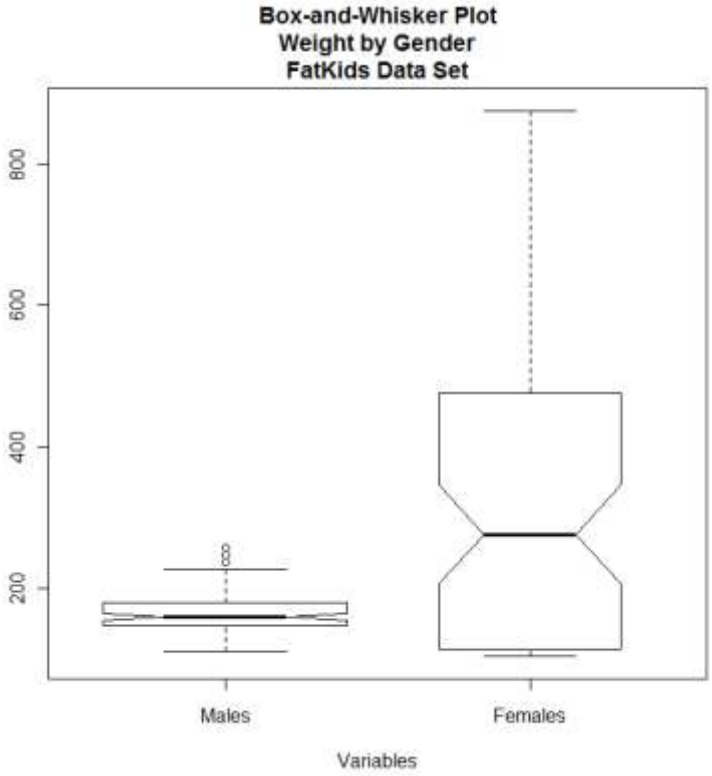


There are several additional nice features of the `boxplot()` function. One is that you can request a "notch" indicating that if two notches do not overlap that there is evidence that the medians are different. The notch=TRUE option allows for this. If you would like the width of the boxes to be such that they represent the number of observations, you can use varwidth=TRUE option.

Graphics Using the `graphics` Package

Let's re-create out box-and-whisker plot using both notch and varwidth. In order for the comparison to make sense, let's compare the Weight of the Males vs. Females in the dfFatKids data frame:

```
dfFatKids_Weight_Males <- dfFatKidsHist[dfFatKidsHist$Gender=="M",c("Weight")]
dfFatKids_Weight_Females <- dfFatKidsHist[dfFatKidsHist$Gender=="F",c("Weight")]
length(dfFatKids_Weight_Females) <-
                max(length(dfFatKids_Weight_Males),length(dfFatKids_Weight_Females))
dfFatKids_Weight_Both <- cbind(dfFatKids_Weight_Males,dfFatKids_Weight_Females)
colnames(dfFatKids_Weight_Both) <- c("Males","Females")
boxplot(dfFatKids_Weight_Both,
        main="Box-and-Whisker Plot\nWeight by Gender\nFatKids Data Set",
        xlab="Variables",
        notch=TRUE,
        varwidth=TRUE)
```



Graphics Using the `graphics` Package

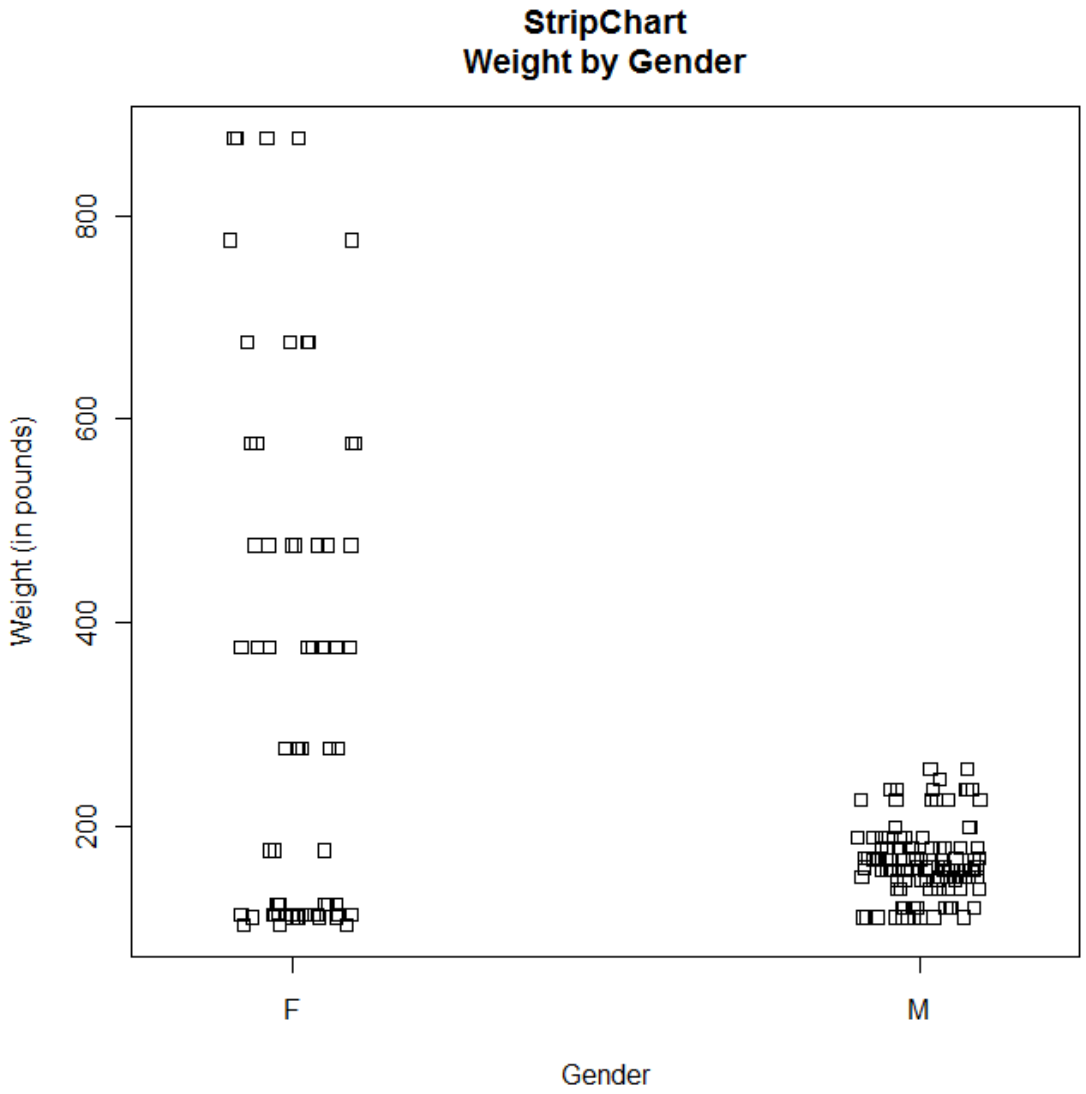
As an alternative to a boxplot for data with a small sample size, you can use the stripchart. Unlike the boxplot, you don't have to fiddle with the data to get a Male vs. Female comparison chart:

```
> stripchart(Weight ~ Gender,
             main="StripChart\nWeight by Gender",
             xlab="Gender",
             ylab="Weight (in pounds)",
             vertical=TRUE,
             data=dfFatKidsHist)
```

One nice option is to set method="jitter" so that the points don't fall on such a straight line (see next slide).

Graphics Using the `graphics` Package

```
> stripchart(Weight ~ Gender,
             main="StripChart\nWeight by Gender",
             xlab="Gender",
             ylab="Weight (in pounds)",
             vertical=TRUE,
             data=dfFatKidsHist,
             method="jitter")
```



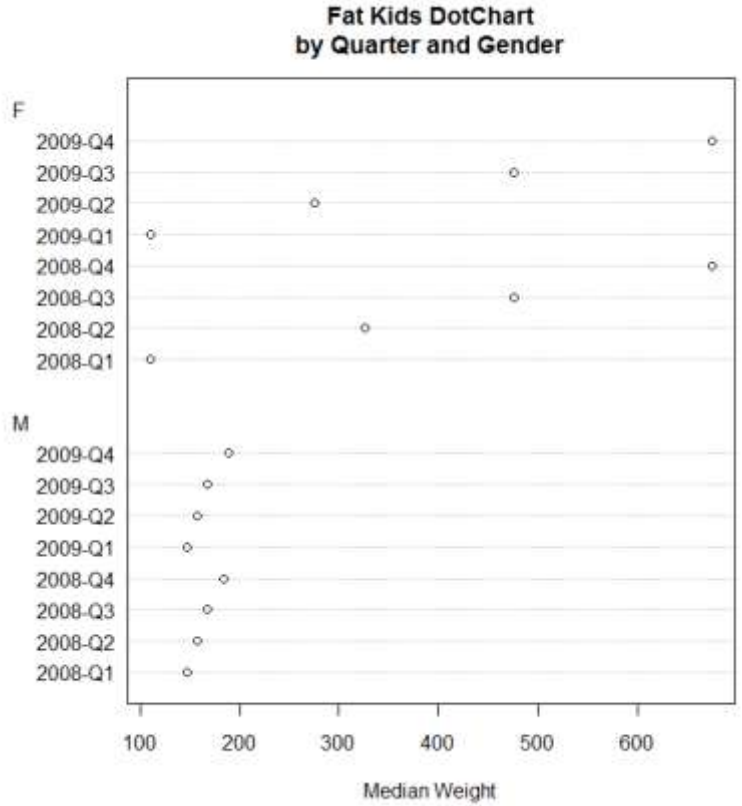
Graphics Using the `graphics` Package

Another nice way of looking at data is by using a dot chart. This is similar to a bar chart, but with dots. Ha-ha! First, let's create the median Weight by quarter, then produce the dot chart using the `dotchart()` function:

```
dfFatKidsHist$MeasQtr <-
              paste(format(dfFatKidsHist$DateOfMeas,"%Y"),quarters(dfFatKidsHist$DateOfMeas),sep="-")
mFatKidsHist_QtrSex_Median <-
              tapply(dfFatKidsHist$Weight,dfFatKidsHist[,c("MeasQtr","Gender")],median)
dotchart(mFatKidsHist_QtrSex_Median,
         main="Fat Kids DotChart\nby Quarter and Gender",
         xlab="Median Weight")
```

Here is what `mFatKidsHist_QtrSex_Median` looks like:

```
         Gender
MeasQtr     F     M
  2008-Q1 110 147.0
  2008-Q2 326 157.0
  2008-Q3 476 168.0
  2008-Q4 676 183.5
  2009-Q1 110 147.0
  2009-Q2 276 157.0
  2009-Q3 476 168.0
  2009-Q4 676 189.0
```



Graphics Using the `graphics` Package

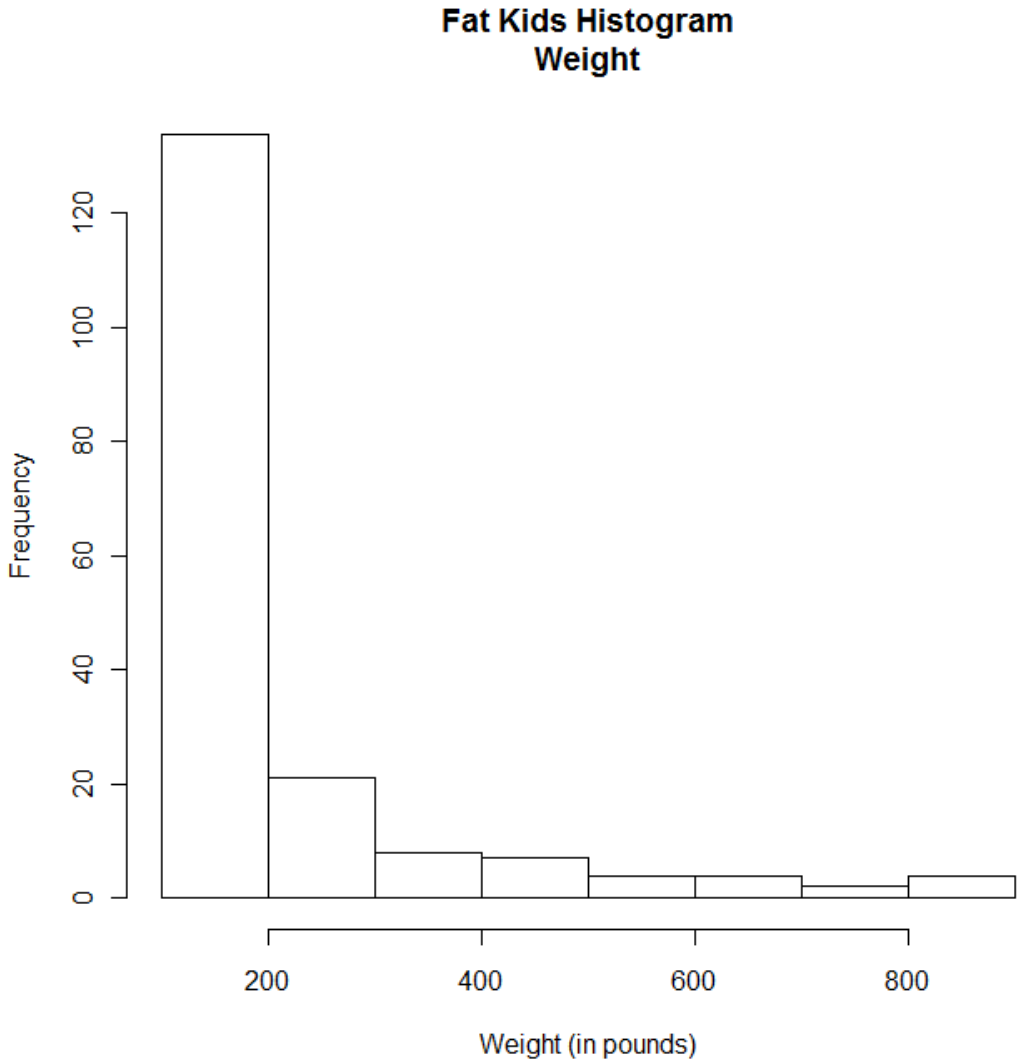
If you're into kickin' your statistics *old school*, maybe a stem-and-leaf plot is more your style:

```
> stem(dfFatKidsHist$Height)

  The decimal point is 1 digit(s) to the right of the |

  1 | 000000111111111111222222
  1 |
  2 | 444444444
  2 | 5555555555667788
  3 | 00222222333333334444444444444444
  3 | 5555555577
  4 | 001144444444
  4 | 55
  5 | 4444444444
  5 |
  6 |
  6 |
  7 | 11111111111112222224444444444
  7 | 55556666668888
  8 | 1111111122222222
  8 | 7777777
```

Graphics Using the `graphics` Package

Speaking of old, the lowly Histogram is old, but not quite dead:

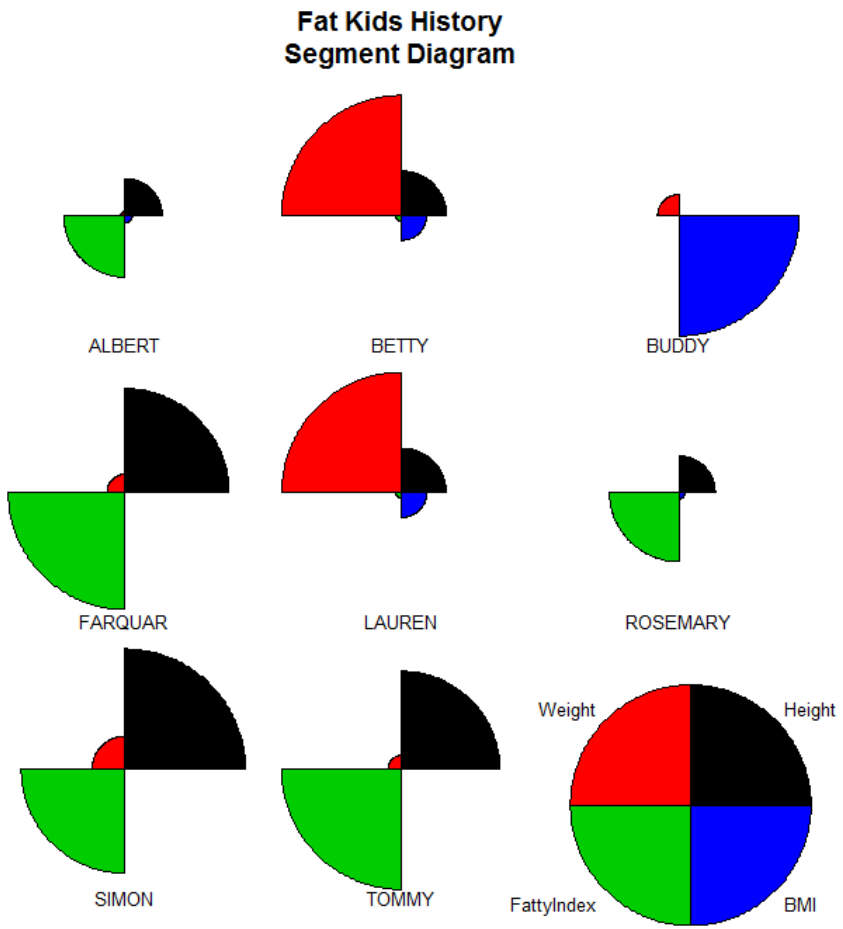
```
> hist(dfFatKidsHist$Weight,
       main="Fat Kids Histogram\nWeight",
       xlab="Weight (in pounds)")
```



Graphics Using the `graphics` Package

Speaking of new, the `stars()` function can produce a very nice segment chart when multiple numeric variables are being displayed. Let's compute the mean of the Height, Weight, and compute the FattyIndex and Body Mass Index to be used on the chart:

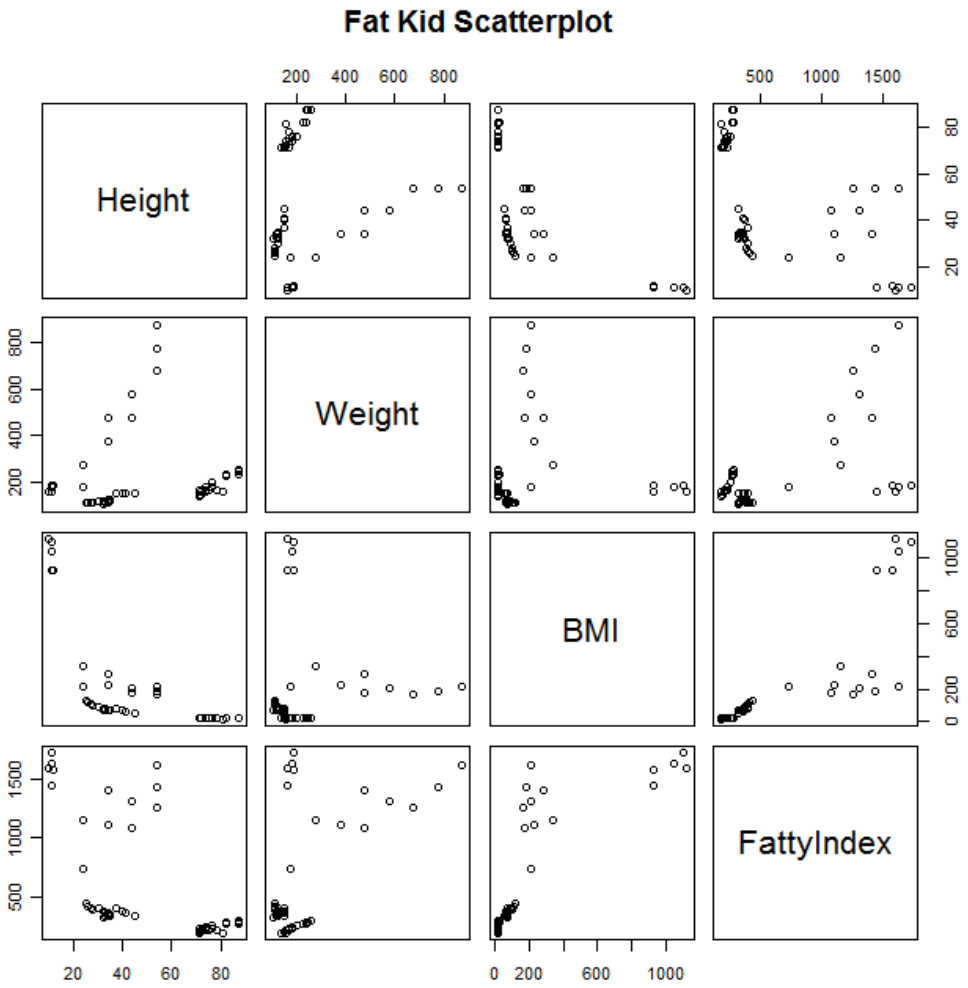
```
> z <- aggregate(dfFatKidsHist[,c("Height","Weight")],
                 list(FirstName = dfFatKidsHist$FirstName),
                 mean)
> z$FattyIndex <- z$Height/z$Weight
> z$BMI <- 703*z$Height/z$Weight^2
> z
  FirstName   Height   Weight FattyIndex       BMI
1    ALBERT 33.25000 126.6667 0.26250000 1.4568750
2     BETTY 37.78947 453.2632 0.08337204 0.1293080
3     BUDDY 11.00000 174.0000 0.06321839 0.2554168
4   FARQUAR 73.34783 165.8261 0.44231778 1.8751537
5    LAUREN 37.56522 452.8696 0.08294931 0.1287641
6  ROSEMARY 32.91667 114.4167 0.28769119 1.7676350
7     SIMON 83.17391 207.7391 0.40037673 1.3548956
8     TOMMY 69.75000 153.9167 0.45316730 2.0697993
> stars(z[, 2:5],
        key.loc=c(7, 2),
        main="Fat Kids History\nSegment Diagram",
        draw.segments=TRUE,
        labels=z$FirstName)
```



Graphics Using the `graphics` Package

Another way to view multiple variables at once is with a scatterplot matrix using the `pairs()` function:

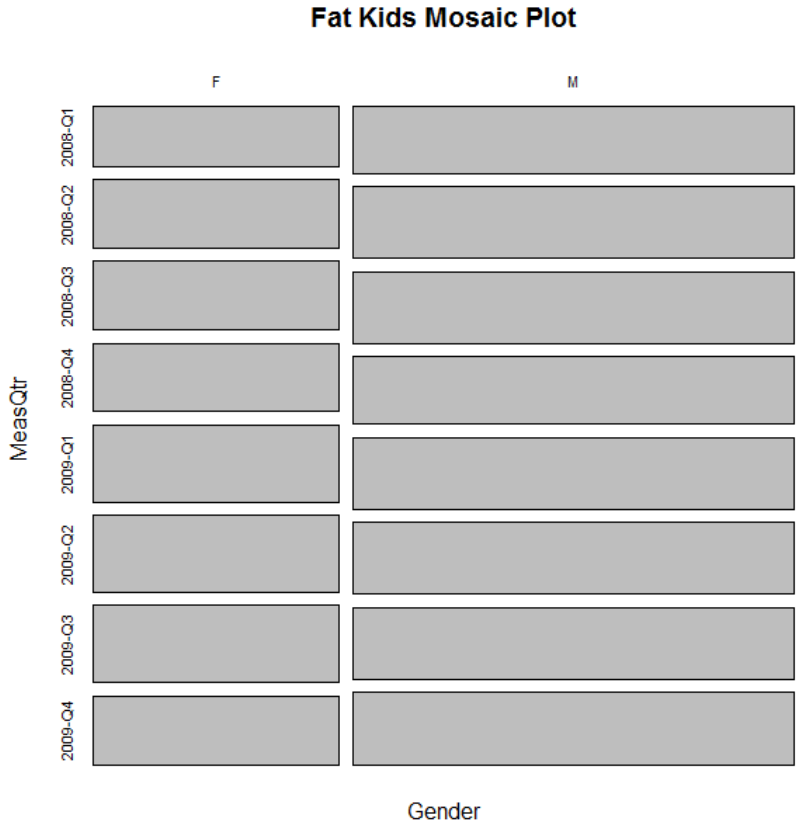
```
> dfFatKidsHist$BMI <- with(dfFatKidsHist,703*Weight/Height^2)
> dfFatKidsHist$FattyIndex <- with(dfFatKidsHist,100*Weight/Height)
> pairs(dfFatKidsHist[,c(3,4,8,9)], main = "Fat Kid Scatterplot")
```



Graphics Using the `graphics` Package

You can use a mosaic plot to show the relative differences with row counts. Use the `table()` function to get the frequency counts:

```
> t2 <- table(dfFatKidsHist[,c("Gender","MeasQtr")])
> t2
       MeasQtr
Gender 2008-Q1 2008-Q2 2008-Q3 2008-Q4 2009-Q1 2009-Q2 2009-Q3 2009-Q4
     F       7       8       8       8       9       9       9       8
     M      14      15      15      14      15      15      15      15
> mosaicplot(t2,main="Fat Kids Mosaic Plot")
```

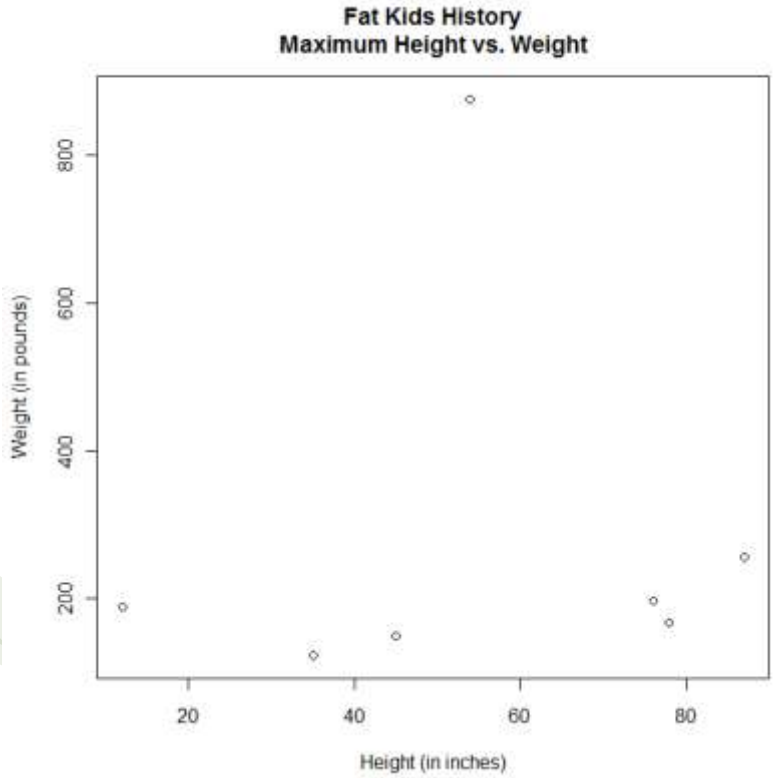


Graphics Using the `graphics` Package

As alluded to in the overview, you can add additional *bling* to your graphics using the low-level functions such as `grid()` and `abline()`. Below, let's use our Fat Kids history data and determine the maximum weight for each child. Then, let's determine the maximum weight and which child it belongs to. Next, we will plot this data and make a note as to who is the fattest fatty:

```
> dfFatKidsHist_Maximum_Weight <-
                aggregate(dfFatKidsHist[,3:4],list(FirstName=dfFatKidsHist$FirstName),max)
> dfFatKidsHist_Maximum_Weight
  FirstName Height Weight
1    ALBERT     45    150
2     BUDDY     12    189
3   FARQUAR     76    198
4    LAUREN     54    876
5  ROSEMARY     35    123
6     SIMON     87    256
7     TOMMY     78    167
>
> dfFatKidsHist_Overall_MaxWt <-
             with(dfFatKidsHist_Maximum_Weight,dfFatKidsHist_Maximum_Weight[Weight==max(Weight),])
> dfFatKidsHist_Overall_MaxWt
  FirstName Height Weight
4    LAUREN     54    876
>
> plot(dfFatKidsHist_Maximum_Weight$Height,dfFatKidsHist_Maximum_Weight$Weight,
       main="Fat Kids History\nMaximum Height vs. Weight",
       xlab="Height (in inches)",
       ylab="Weight (in pounds)")
```

Graphics Using the `graphics` Package

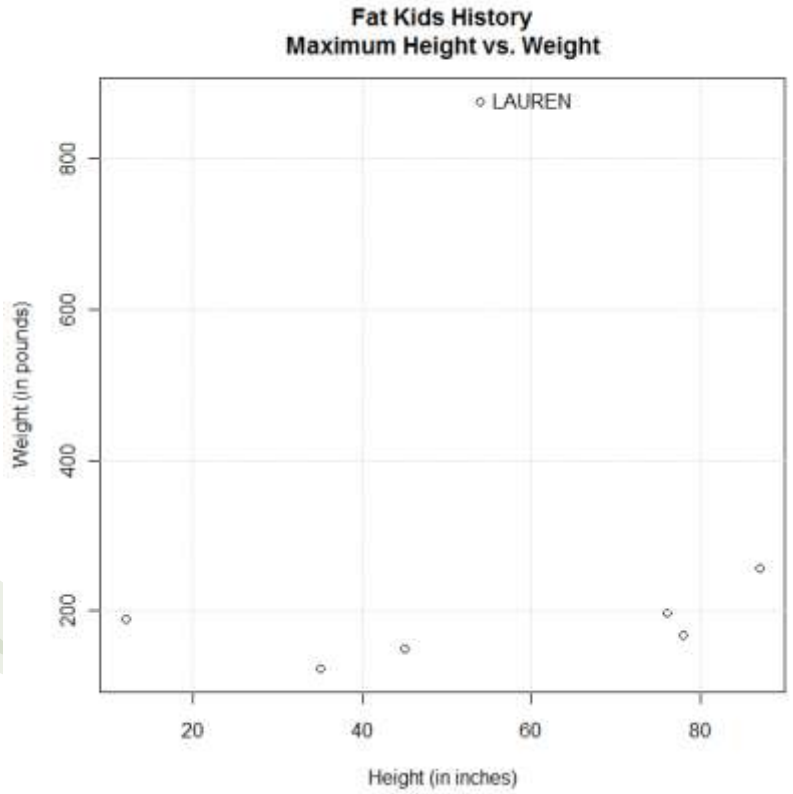


As you see above, there is one point above the rest. Based on our data, it's Lauren with a best-in-class weight of 876 pounds. Let's place a grid on the graph as well as label Lauren's point on the graph:

```
> grid()
> text(x=dfFatKidsHist_Overall_MaxWt$Height,
       y=dfFatKidsHist_Overall_MaxWt$Weight,
       label=dfFatKidsHist_Overall_MaxWt$FirstName,
       pos=4)
```

pos is the position of the text, where 1=below, 2=left, 3=above, 4=right.

Graphics Using the `graphics` Package



Just for fun, let's place a linear regression line on the graph using the linear regression function `lm()` as well as the `abline()` function shown in the overview:

```
> lmFatKidsHist_Maximum_Weight <- lm(Weight ~ Height, data=dfFatKidsHist_Maximum_Weight)
> lmFatKidsHist_Maximum_Weight

Call:
lm(formula = Weight ~ Height, data = dfFatKidsHist_Maximum_Weight)

Coefficients:
(Intercept)       Height
   245.8936       0.6143
```

Graphics Using the `graphics` Package

Here is the summary of the linear regression:

```
> summary(lmFatKidsHist_Maximum_Weight)

Call:
lm(formula = Weight ~ Height, data = dfFatKidsHist_Maximum_Weight)

Residuals:
      1       3       4       5       6       7       8
-123.54  -64.27  -94.58  596.93 -144.40  -43.34 -126.81

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 245.8936   267.7686   0.918    0.401
Height        0.6143     4.4158   0.139    0.895

Residual standard error: 291 on 5 degrees of freedom
Multiple R-squared: 0.003856,   Adjusted R-squared: -0.1954
F-statistic: 0.01935 on 1 and 5 DF,  p-value: 0.8948
```

We can gain access to the coefficients using the coef vector:

```
> lmFatKidsHist_Maximum_Weight$coef
(Intercept)      Height
245.8936324   0.6143271
> lmFatKidsHist_Maximum_Weight$coef[1] # Can use the list [[ ]] notation as well!
(Intercept)
 245.8936324
> lmFatKidsHist_Maximum_Weight$coef[2]
   Height
0.6143271
>
```

Graphics Using the `graphics` Package

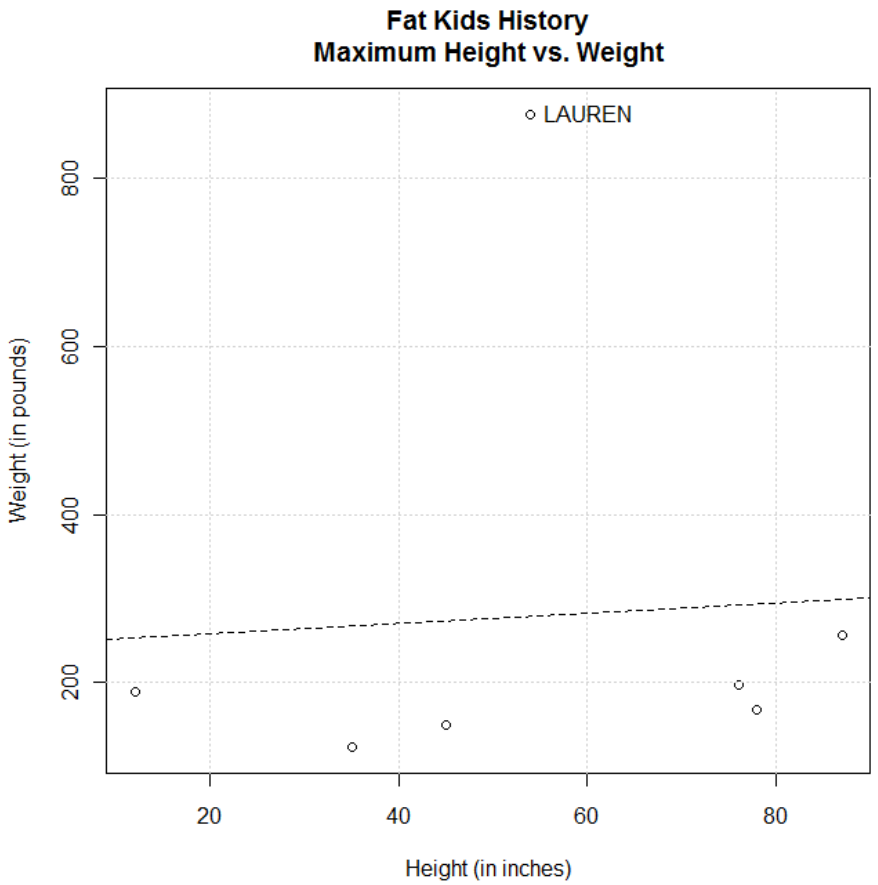
Let's add the regression line to the graph using the coef vector from the linear regression (represented as a list, by the way):

```
> abline(coef=lmFatKidsHist_Maximum_Weight$coef,lty=2)
```

Note that `abline()` has a parameter that takes the coefficients vector!!

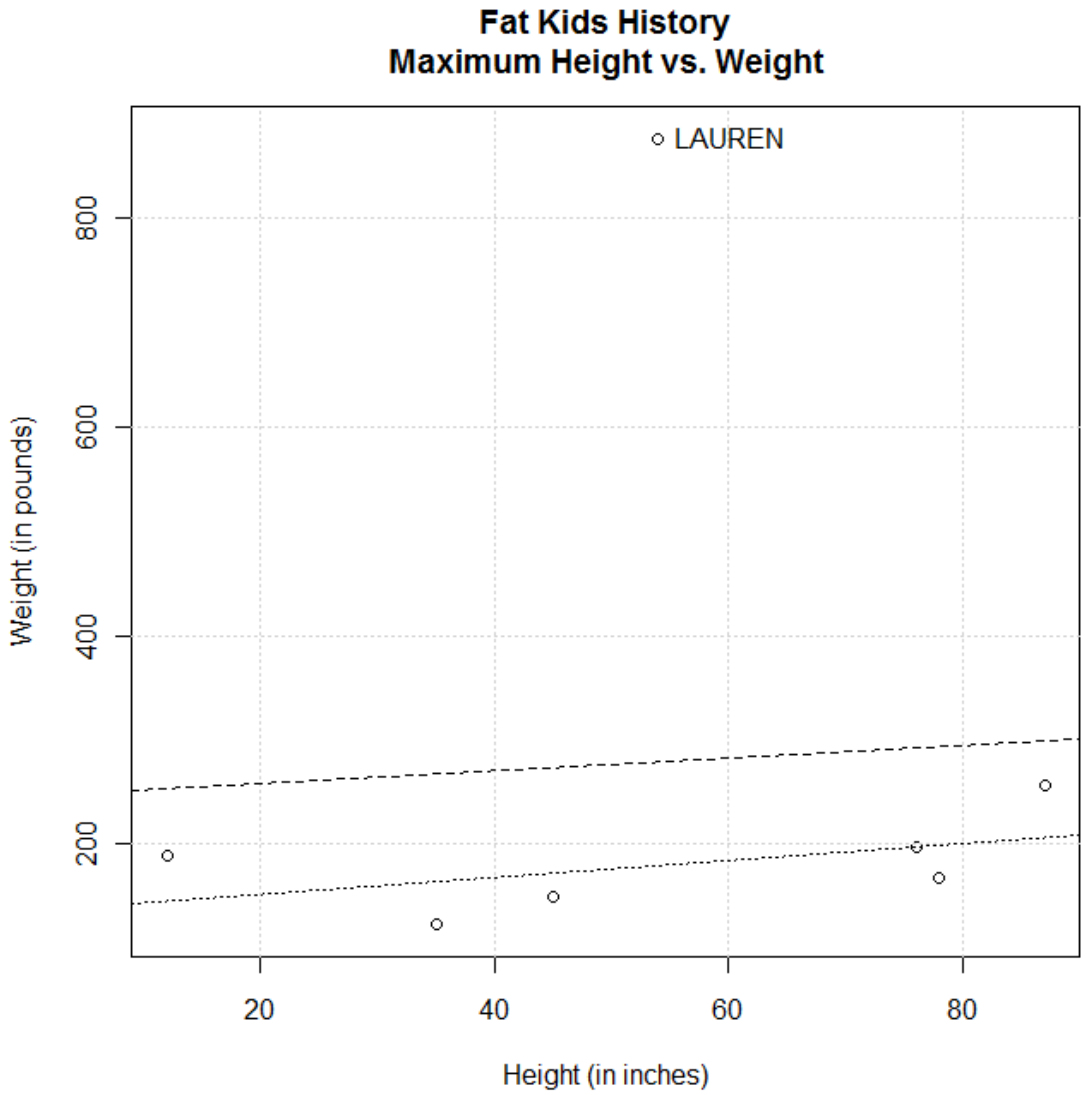
Note that lty=2 is the type of line you want drawn on the graph. In this case, lty=2 is a dashed line. See ?par for more lty line types.

Just for fun, let's see what a line would look like if we removed Lauren from the regression and replotted the line.



Graphics Using the `graphics` Package

```
> lmFatKidsHist_Maximum_Weight_NoOutlier <-
          lm(Weight ~ Height, data=dfFatKidsHist_Maximum_Weight,subset=(FirstName!="LAUREN"))
> abline(coef=lmFatKidsHist_Maximum_Weight_NoOutlier$coef,lty=3)
```

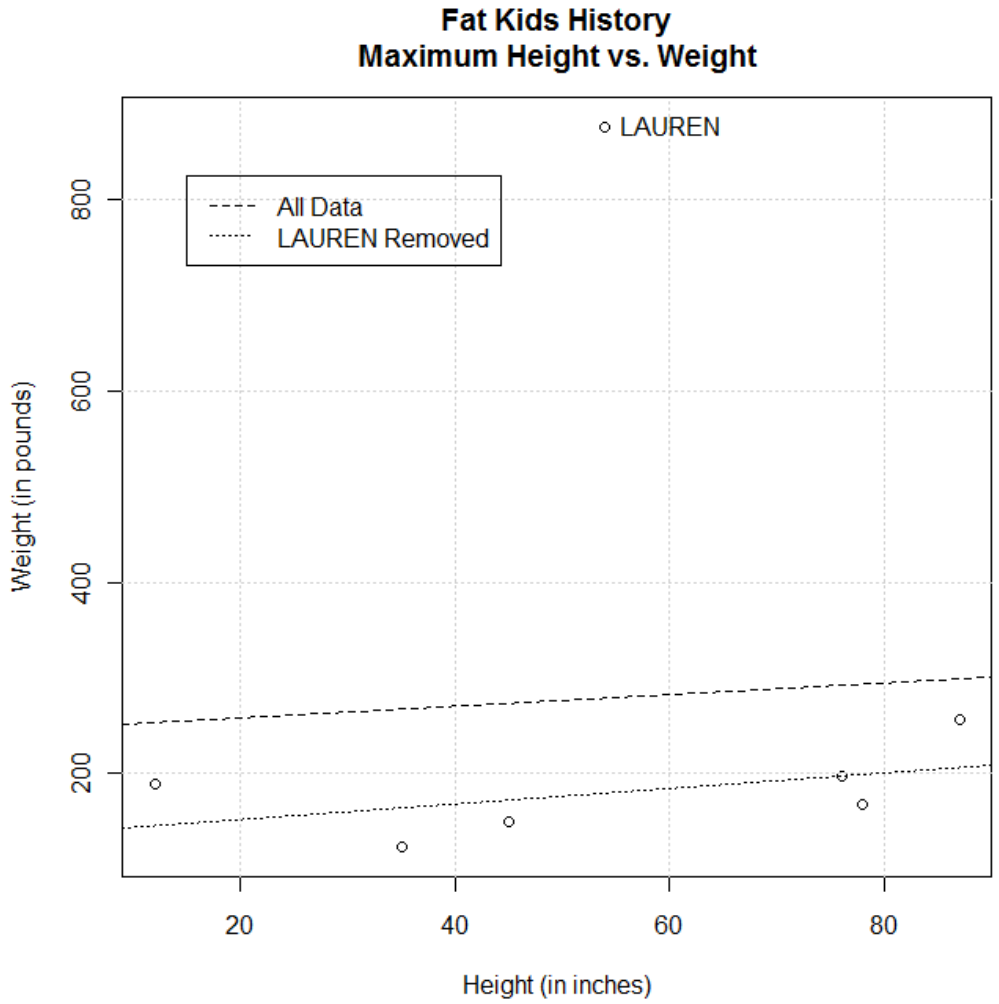



Graphics Using the `graphics` Package

Finally, let's add a legend to the graph:

```
> legend(15,825,legend=c("All Data","LAUREN Removed"),lty=c(2,3))
```

The first parameter is the x-coordinate (Height) and second parameter is the y-coordinate (Weight). The third parameter is a **vector** of labels to appear in the legend, and the fourth parameter is a vector of corresponding line types used.



Graphics using the `ggplot2` Package

Graphics Using the `ggplot2` Package

As mentioned in the overview, creating graphs using the `ggplot2` package entails creating a series of low-level functions added to the high-level plotting function using the plus-sign. Note that since this idea of adding functions can be a bit intimidating at first, the `ggplot2` package also has the `qplot()` (or, *quick plot*) function which allows you to quickly and easily create a graph. We'll start off with how to use the `qplot()` function then move onto the adding functions idea.

The `ggplot2` package implements what is know as the *Grammar of Graphics* and combines the advantages of the base graphics and lattice graphics.

Note that this package does not allow for three-dimensional graphics.

Don't forget to issue the `library(ggplot2)` code before attempting to use ggplot2 graphics! If you don't have `ggplot2` installed, you can install it by issuing `install.packages("ggplot2",dep=T)` at the R command line.

Graphics Using the ggplot2 Package

The syntax for quick plot is:

```
qplot(x, y = NULL, z=NULL, ..., data, facets = . ~ ., margins=FALSE, geom = "auto",
      stat=list(NULL), position=list(NULL), xlim = c(NA, NA), ylim = c(NA, NA), log = "",
      main = NULL, xlab = deparse(substitute(x)), ylab = deparse(substitute(y)), asp = NA)
```

Arguments

x - x values
y - y values
z - z values
... - other arguments passed on to the geom functions
data - data frame to use (optional)
facets - faceting formula to use margins whether or not margins will be displayed
geom - geom to use (can be a vector of multiple names)
stat - statistic to use (can be a vector of multiple names)
position - position adjustment to use (can be a vector of multiple names)
xlim - limits for x axis (aesthetics to range of data)
ylim - limits for y axis (aesthetics to range of data)
log - which variables to log transform ("x", "y", or "xy")
main - character vector or expression for plot title
xlab - character vector or expression for x axis label
ylab - character vector or expression for y axis label
asp - the y/x aspect ratio

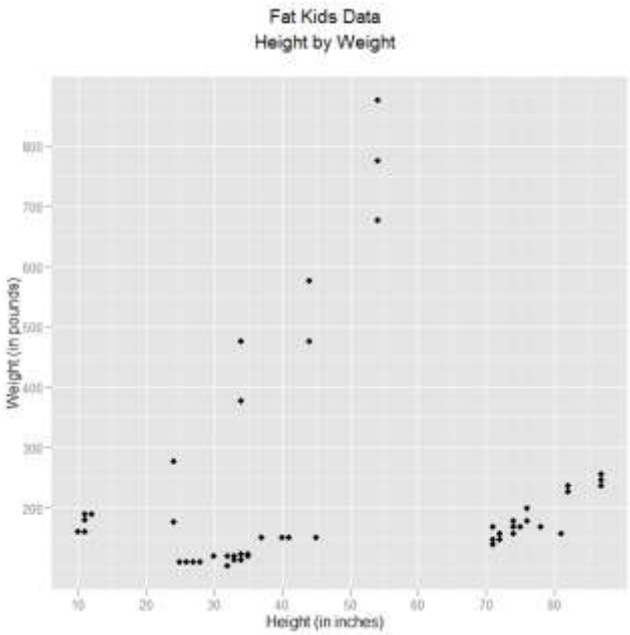
Note that the geom parameter is set to auto. This means that `qplot()` will attempt to figure out the type of graph you want produced based on the x and y values. For example:

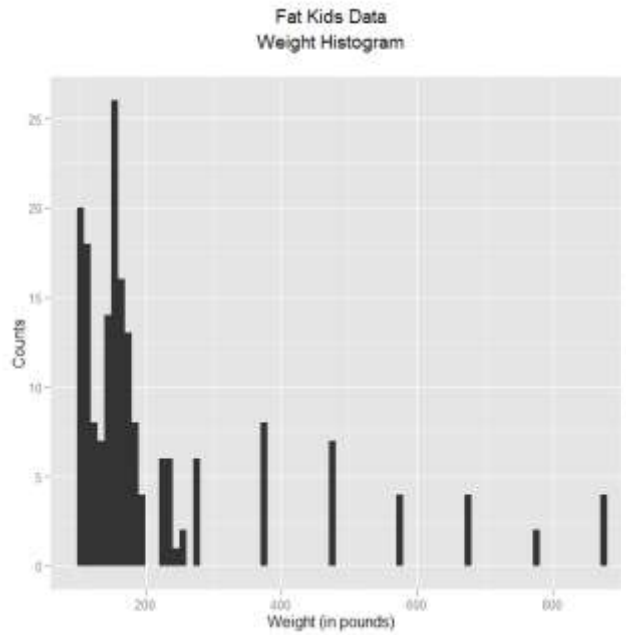
1. Given an x value and no y value, a histogram of x will be produced.
2. Given an x and y value, a scatterplot will be produced.
3. Given a y value and no x value, the x-axis will be forced to the numbers 1, 2, 3, … and a scatterplot will be produced.

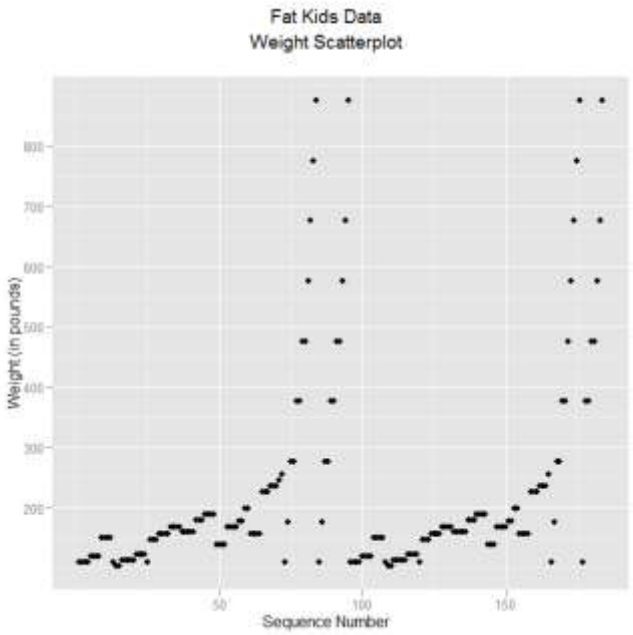
Graphics Using the ggplot2 Package

Let's show examples of these:

```
qplot(x=Height,
      y=Weight,
      data=dfFatKidsHist,
      main="Fat Kids Data\nHeight by Weight\n",xlab="Height (in inches)",ylab="Weight (in pounds)")

qplot(x=Weight,
      data=dfFatKidsHist,binwidth=10,
      main="Fat Kids Data\nWeight Histogram\n",xlab="Weight (in pounds)",ylab="Counts")

qplot(y=Weight,
      data=dfFatKidsHist,
      main="Fat Kids Data\nWeight Scatterplot\n",ylab="Weight (in pounds)",xlab="Sequence Number")
```



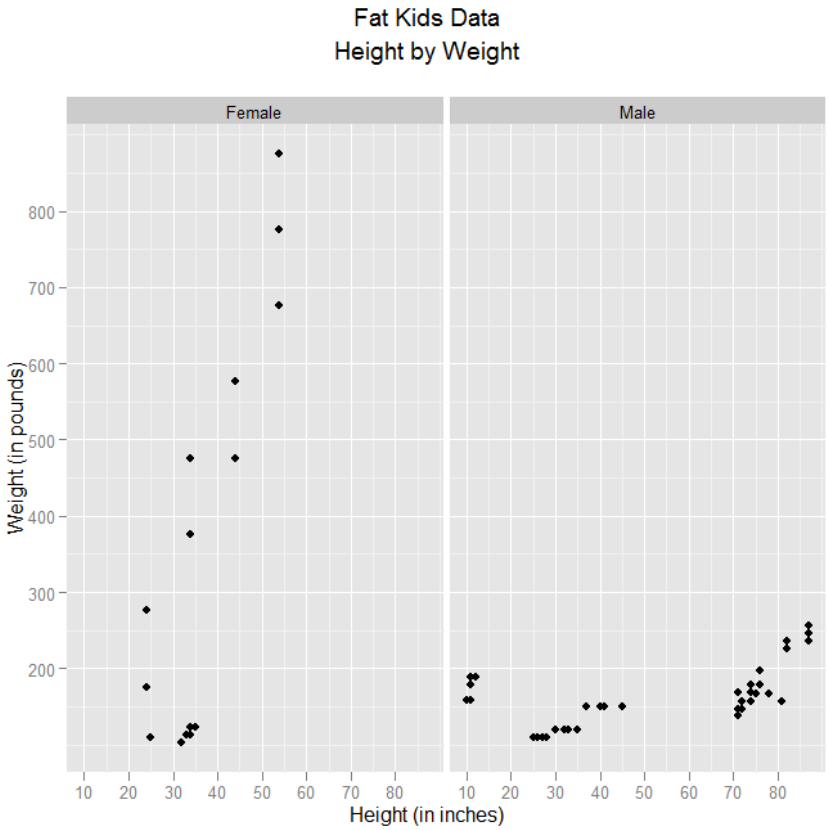




Graphics Using the `ggplot2` Package

Another nice feature is the ability to use *faceting* in order to produce several graphs by a given categorical variable such as Gender, Region, etc. For example, let's produce a scatterplot of the Height and Weight by Gender:

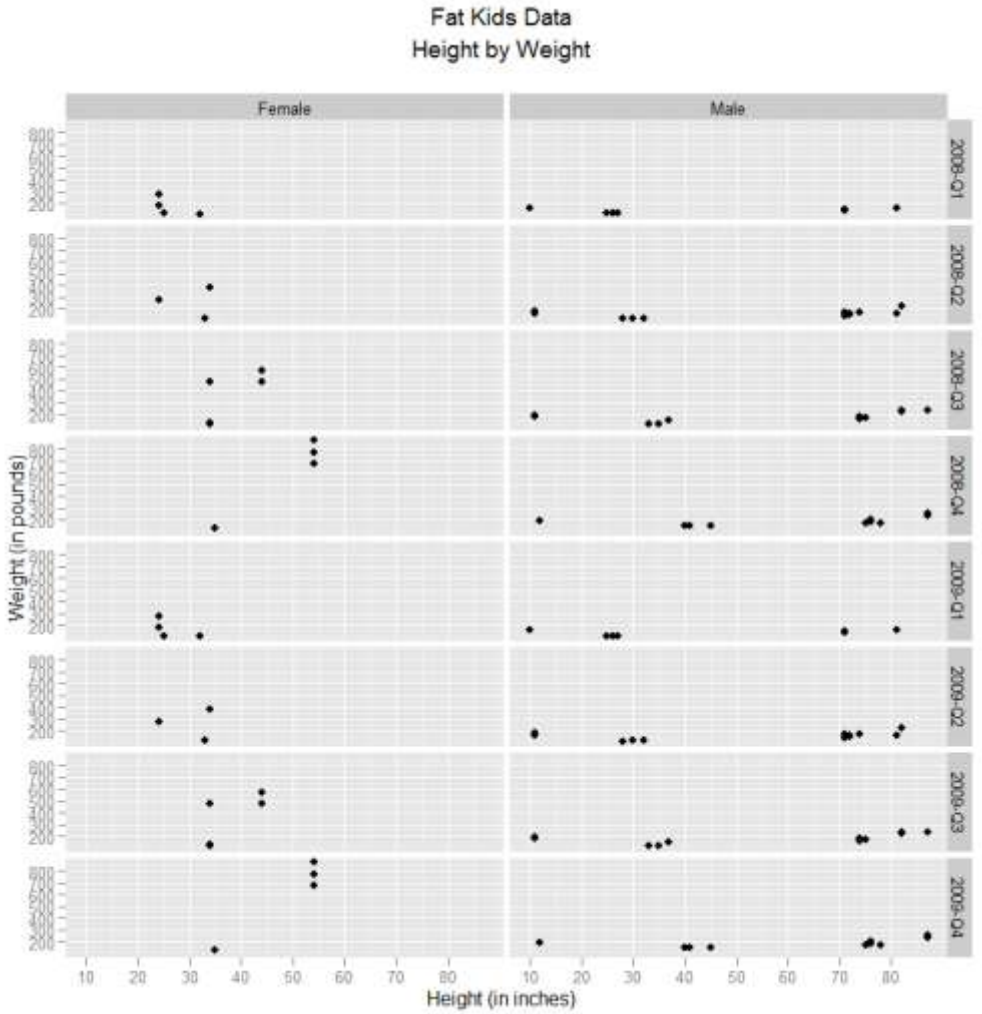
```
> levels(dfFatKidsHist$Gender) <- c("Female","Male")
> qplot(x=Height,
        y=Weight,
        data=dfFatKidsHist,
        facets=. ~ Gender,
        main="Fat Kids Data\nHeight by Weight\n",xlab="Height (in inches)",ylab="Weight (in pounds)")
```



Graphics Using the `ggplot2` Package

We can also add another break point in the facets parameter:

```
> qplot(x=Height,
        y=Weight,
        data=dfFatKidsHist,
        facets=MeasQtr ~ Gender,
        main="Fat Kids Data\nHeight by Weight\n",xlab="Height (in inches)",ylab="Weight (in pounds)")
```



Graphics Using the ggplot2 Package

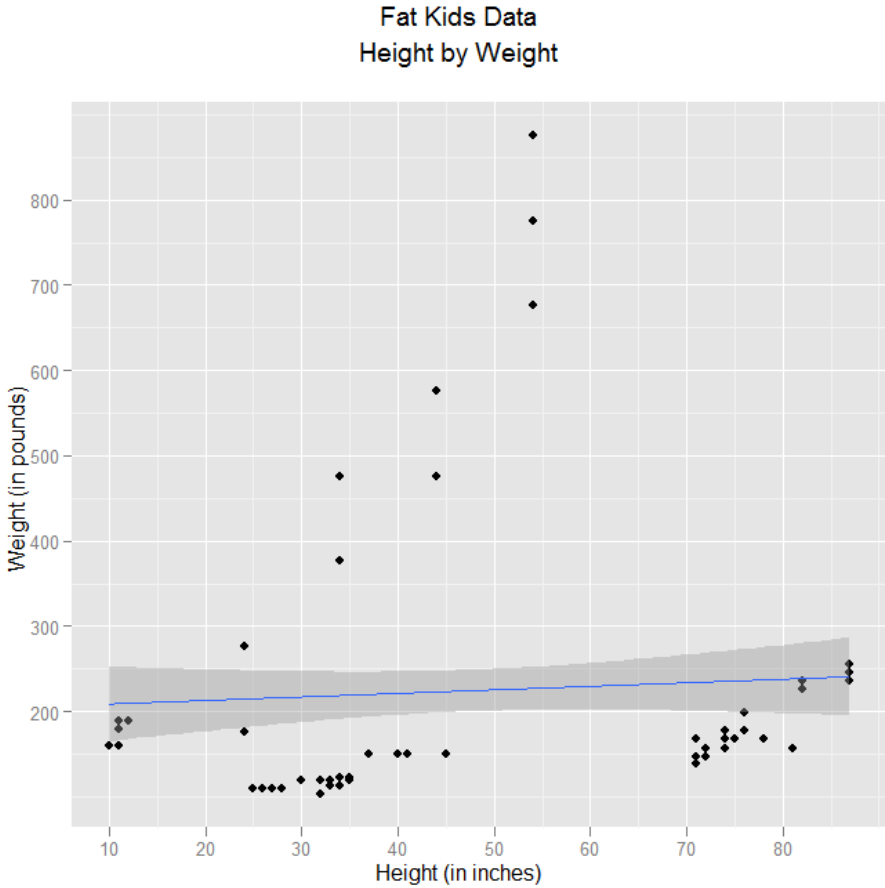
You're not limited to these 3 graph types. By modifying the geom parameter, you can produce many types of graphs. The geom parameter is set to auto, but you can override this parameter with a vector of one or more of the following:

```
abline - line, specified by slope and intercept
area - area plots
bar - bars, rectangles with bases on y-axis
blank - blank draws nothing
boxplot - box-and-whisker plot
contour - display contours of 3D surface in 2D
crossbar - hollow bar with middle indicated by horizontal line
density - display a smooth density estimator
density_2d - contours from a 2D density estimate
errorbar - error bars
histogram - histogram
hline/vline - horizontal/vertical line
interval - base for all interval (range) geoms
jitter - points, jittered to reduce overplotting
line - connect observations in order of x values
linerange - an interval represented by a vertical line
path - connect observations in original order
point - points, as for a scatterplot
pointrange - an interval represented by a vertical line, with a point in the middle
polygon - a filled path
quantile - add quantile lines from a quantile regression
ribbon - ribbons, y range with continuous x values
rug - marginal rug plots
segment - single line segments
smooth - add a smoothed condition mean
step - connect observations with stairs
text - text annoations
tile - tile plot
```

Graphics Using the `ggplot2` Package

Now, based on this information, you can add additional *bling* to your graphs. For example, let's add a linear regression line to our scatterplot:

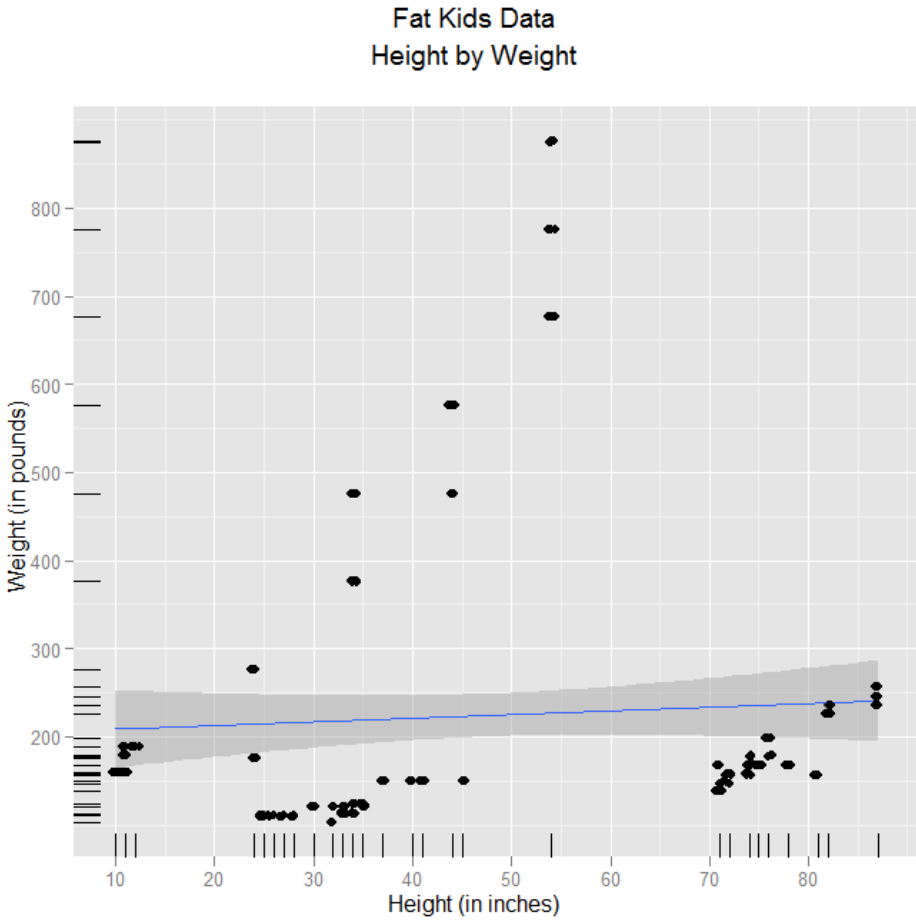
```
qplot(x=Height,
      y=Weight,
      data=dfFatKidsHist,
      geom=c("point","smooth"),
      method="lm",
      formula=y ~ x,
      main="Fat Kids Data\nHeight by Weight\n",xlab="Height (in inches)",ylab="Weight (in pounds)")
```



Graphics Using the ggplot2 Package

We can continue on and add additional geoms to our plot:

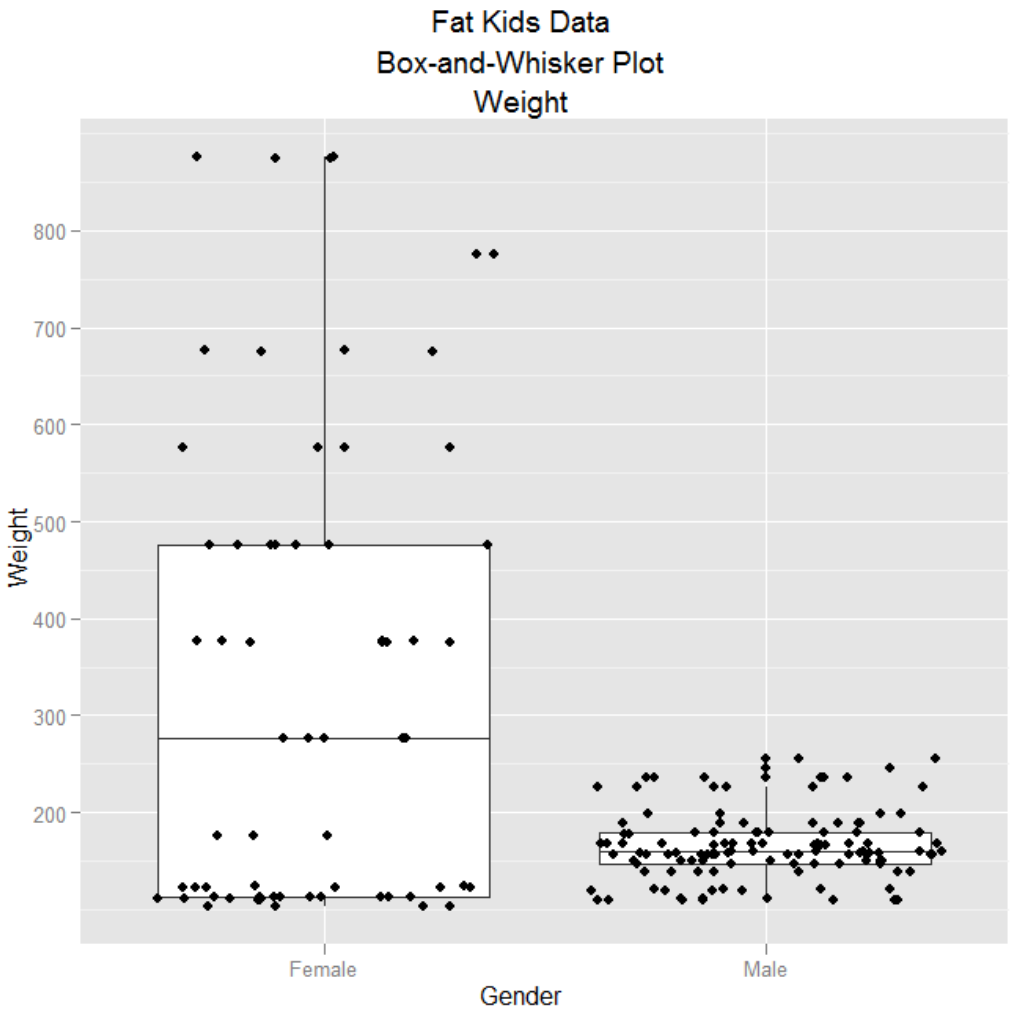
```
qplot(x=Height,
      y=Weight,
      data=dfFatKidsHist,
      geom=c("point","smooth","rug","jitter"),
      method="lm",
      formula=y ~ x,
      main="Fat Kids Data\nHeight by Weight\n",xlab="Height (in inches)",ylab="Weight (in pounds)")
```



Graphics Using the ggplot2 Package

Let's produce a box-and-whisker plot of the Weight with jittering:

```
qplot(x=Gender,
      y=Weight,
      data=dfFatKidsHist,
      geom=c("boxplot","jitter"),
      main="Fat Kids Data\nBox-and-Whisker Plot\nWeight")
```



Graphics Using the `ggplot2` Package

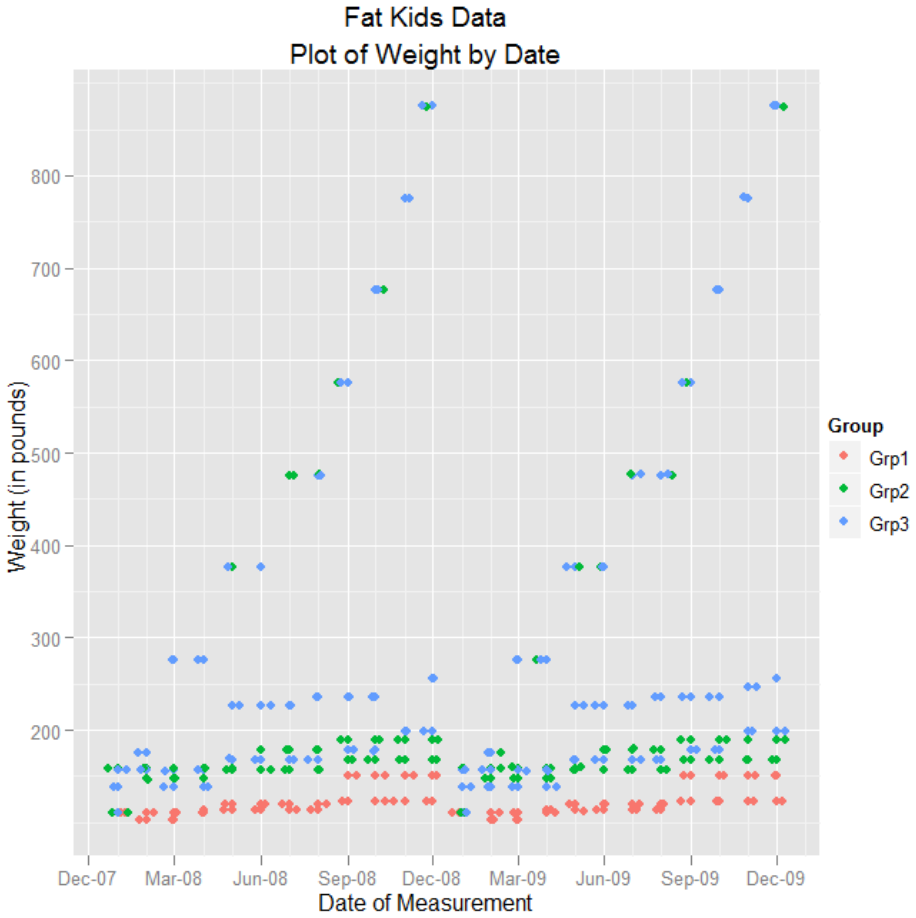
Let's produce another scatterplot, by with the Date of Measurement on the x-axis and the Weight on the Y-Axis. Also, let's break it up by Group:

```
qplot(x=DateOfMeas,
      y=Weight,
      data=dfFatKidsHist,
      facets=Group~.,
      main="Fat Kids Data\nPlot of Weight by Date",xlab="Date of Measurement",ylab="Weight (in pounds)")
```

Graphics Using the ggplot2 Package

Referring to the previous slide, rather than breaking up the graph into three separate sections, maybe we can use color to show who is in what group:

```
qplot(x=DateOfMeas,
      y=Weight,
      data=dfFatKidsHist,
      geom=c("point","jitter"),
      color=Group,
      main="Fat Kids Data\nPlot of Weight by Date",xlab="Date of Measurement",ylab="Weight (in pounds)")
```

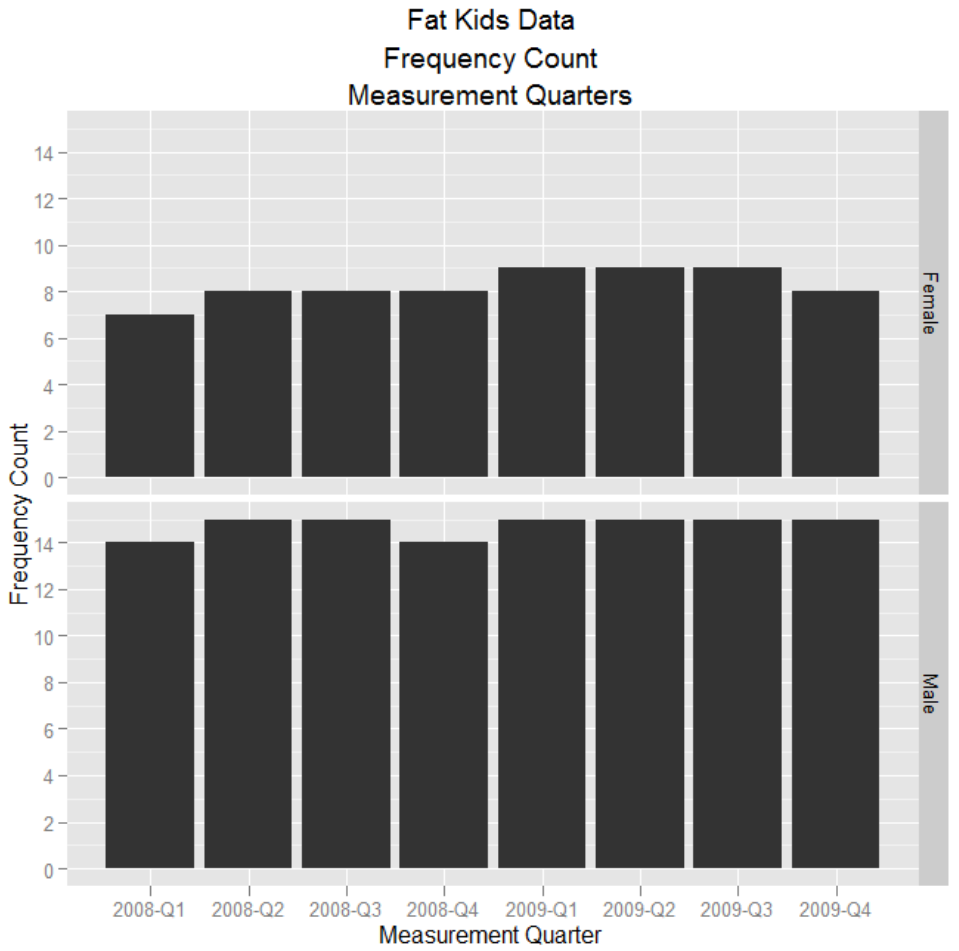


Graphics Using the ggplot2 Package

Now, let's create a box plot of the frequency counts by the measurement quarter variable:

```
qplot(x=MeasQtr,
      data=dfFatKidsHist,
      facets=Gender~.,
      main="Fat Kids Data\nFrequency Count\nMeasurement Quarters",
      xlab="Measurement Quarter",ylab="Frequency Count")
```

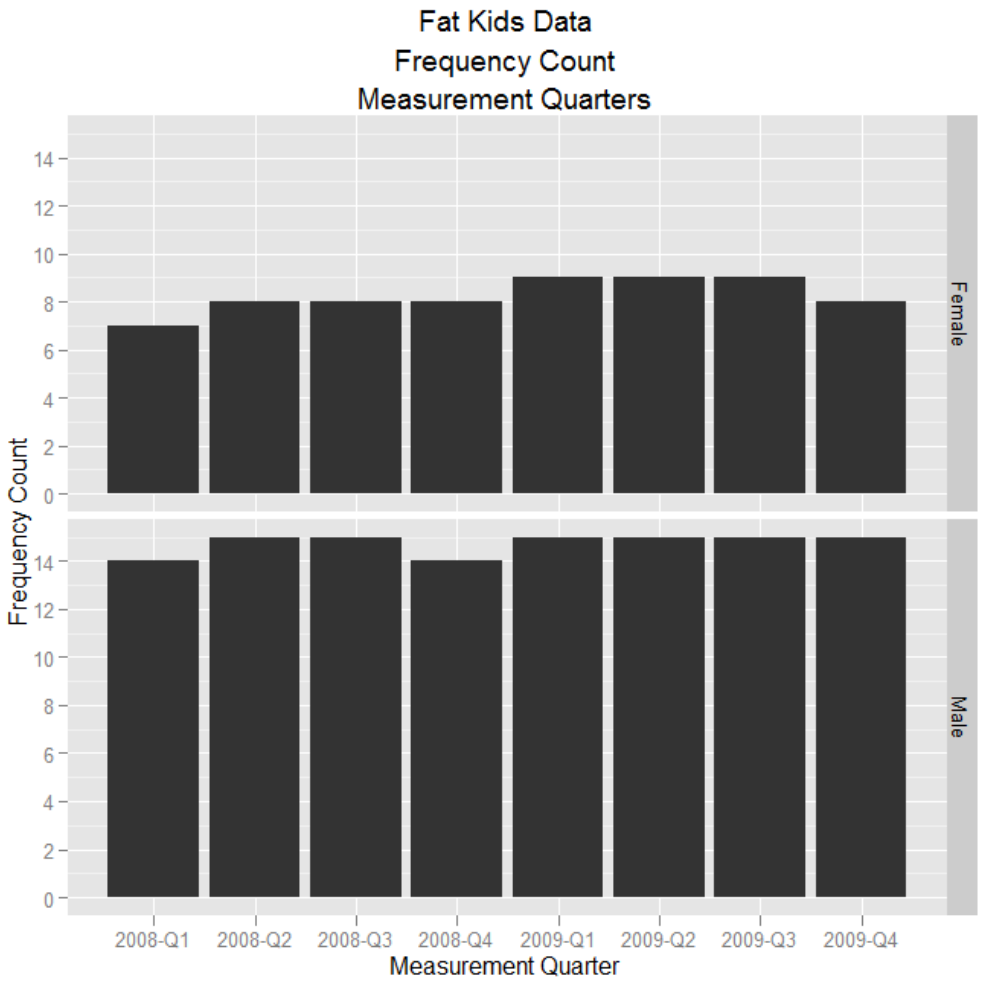
You may be wondering how `qplot()` knew to count the number of rows by the measurement quarter. As mentioned above, by default when the parameter x is defined and y is not, a histogram is produced. This is equivalent to specifying geom="bar". What we have not mentioned is that there is a default statistic defined for each geom. You can override the statistic. For `geom="bar"`, the parameter stat is defined as bin: `stat="bin"`.



Graphics Using the ggplot2 Package

Just to prove it to you:

```
qplot(x=MeasQtr,
      data=dfFatKidsHist,
      facets=Gender~.,
      geom="bar",stat="bin",
      main="Fat Kids Data\nFrequency Count\nMeasurement Quarters",
      xlab="Measurement Quarter",ylab="Frequency Count")
```



Graphics Using the ggplot2 Package

Here is a list of statistics in ggplot2:

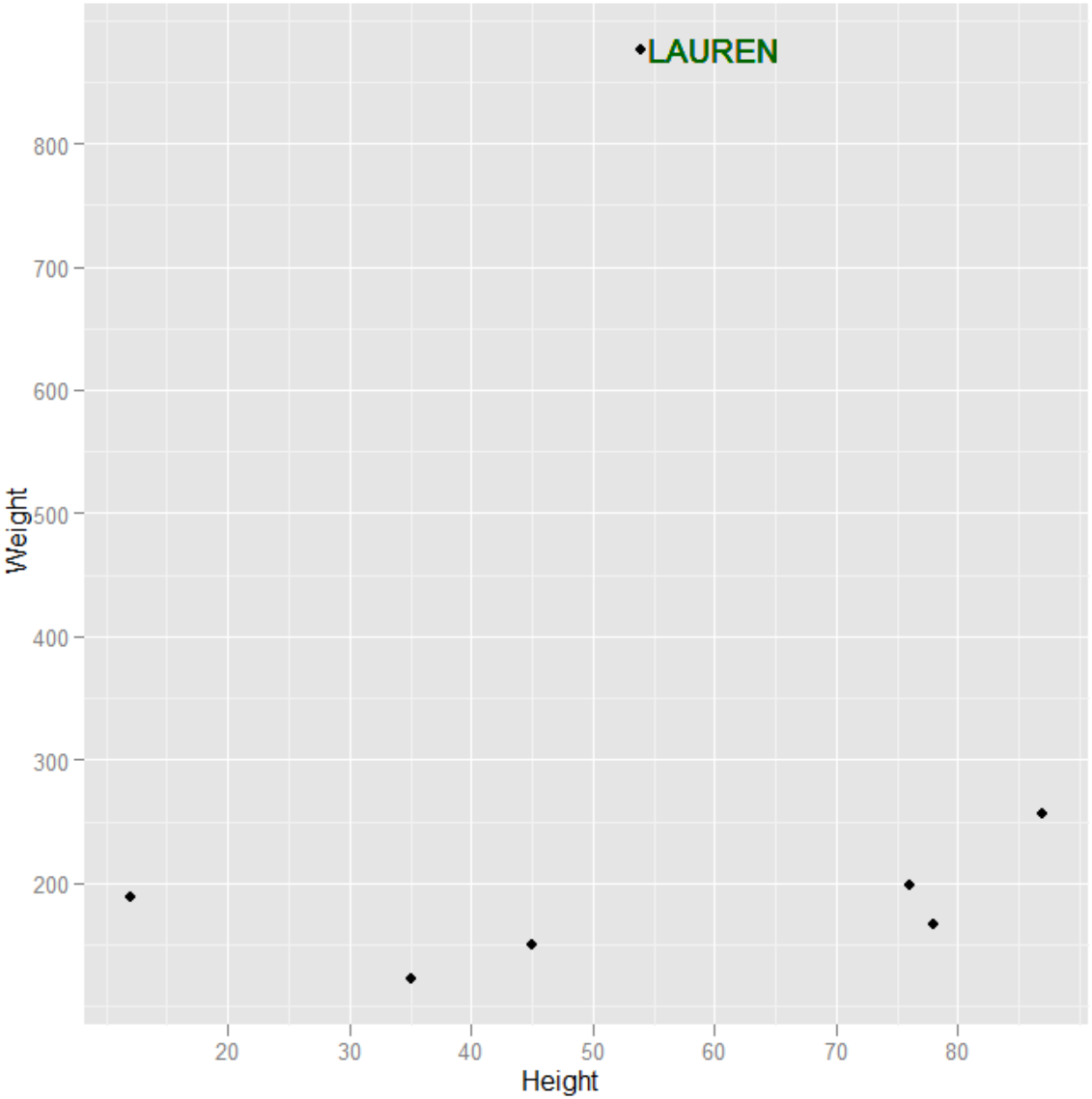
```
bin - bin data
boxplot - calculate components of box-and-whisker plot
contour - contours of 3D data
density - density estimation, 1D
density_2d - density estimation, 2D
function - superimpose a function
identity - don't transform data
qq - calculation for quantile-quantile plot
quantile - continuous quantiles
smooth - add a smoother
spoke - convert angle and radius to xend and yend
step - create stair steps
sum - sum unique values
summary - summarize y values at every unique x
unique - remove duplicates
```

Now, at this point, we can begin to *add* additional information to a qplot using the plus-sign. We will briefly show this concept, but will then dive into the full power of ggplot2. Let's add a textual annotation to a graph:

```
qplot(x=Height,y=Weight,data=dfFatKids)
 + geom_text(aes(x=60,y=876,label="LAUREN"),color="darkgreen")
```

For each of the geometries *ggg* mentioned on slide #48, there is a corresponding function geom_*ggg* you can use with the plus-sign. But, unlike those *ggg* geoms mentioned on slide #48, you have more options if you use the geom_*ggg* functions.

Graphics Using the `ggplot2` Package



Graphics Using the ggplot2 Package

The geom_text() function allows you to add text to a plot. In order to specify the location and text, we use the aes() function to add an aesthetics, such as location and label, to the graph. We then changed the color of the text to dark green.

Now, as mentioned, each of the geoms mentioned earlier with respect to qplot() has a corresponding individual geom function that allows you greater flexibility and control. Here is a list of the geom functions: geom_abline(), geom_area(), geom_bar(), geom_bin2d(), geom_blank(), geom_boxplot(), geom_contour(), geom_crossbar(), geom_density(), geom_density2d(), geom_errorbar(), geom_errorbarh(), geom_freqpoly(), geom_hex(), geom_histogram(), geom_hline(), geom_jitter(), geom_line(), geom_linerange(), geom_path(), geom_point(), geom_pointrange(), geom_polygon(), geom_quantile(), geom_rect(), geom_ribbon(), geom_rug(), geom_segment(), geom_smooth(), geom_step(), geom_text(), geom_tile(), geom_vline().

I won't outline all of these functions, but will show you a few. Now, in order to use these geom functions, you have to create a ggplot2 object first using the ggplot() function. This object does not actual produce a graph, but prepares you to use the geom functions listed above which are responsible for creating a graph. Here is how you create a ggplot2 object:

```
> gFatKids1 <- ggplot(dfFatKids,aes(x=Height,y=Weight))
```

Take note that the aes() function here provides the variables that will be used by the geom functions later. Let's produce a scatterplot using geom_point():

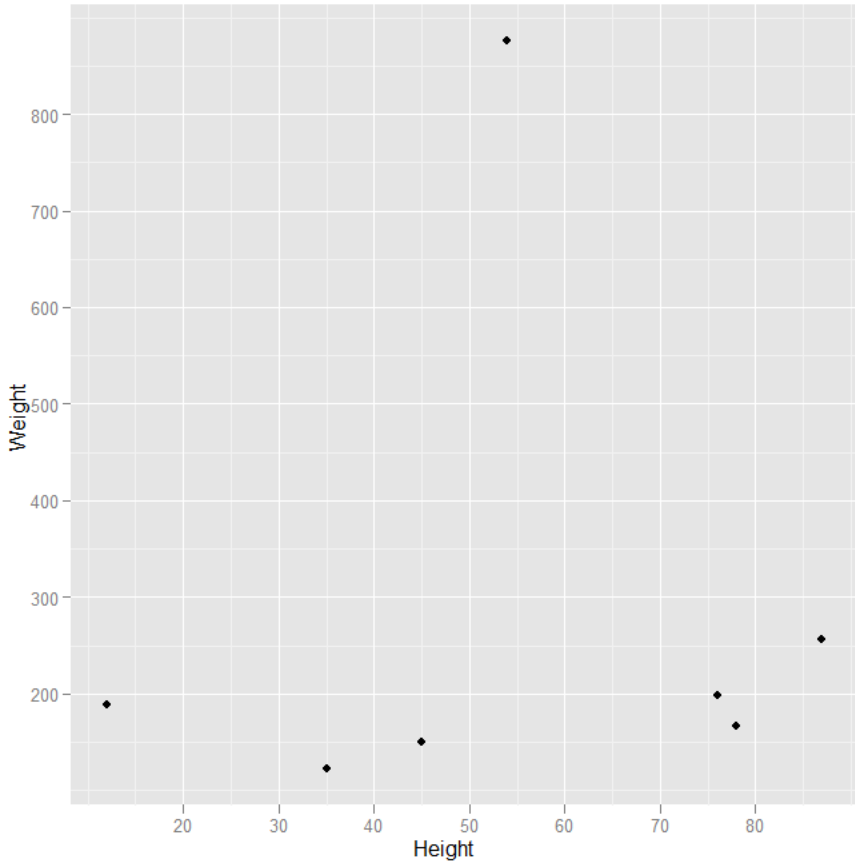
Graphics Using the ggplot2 Package

```
> gFatKids1 <- ggplot(dfFatKids,aes(x=Height,y=Weight))
> gFatKids1 + geom_point()
```


As you see below, the graph produced is the very familiar Height by Weight graph. It's the `geom_point()` function which is responsible for producing the graph while the `ggplot()` function was responsible for indicating the data frame as well as the variables to be used. This idea of *adding* a function, such as `geom_point()`, to the object gFatKids1 is called *layering*. You can add several layers to this graph by *adding* additional functions to it. For example, let's add a horizontal line at Lauren's weight:

```
> gFatKids1 <- ggplot(dfFatKids,aes(x=Height,y=Weight))
> gFatKids1 +
  geom_point() +
  geom_hline(aes(yintercept=876))
```

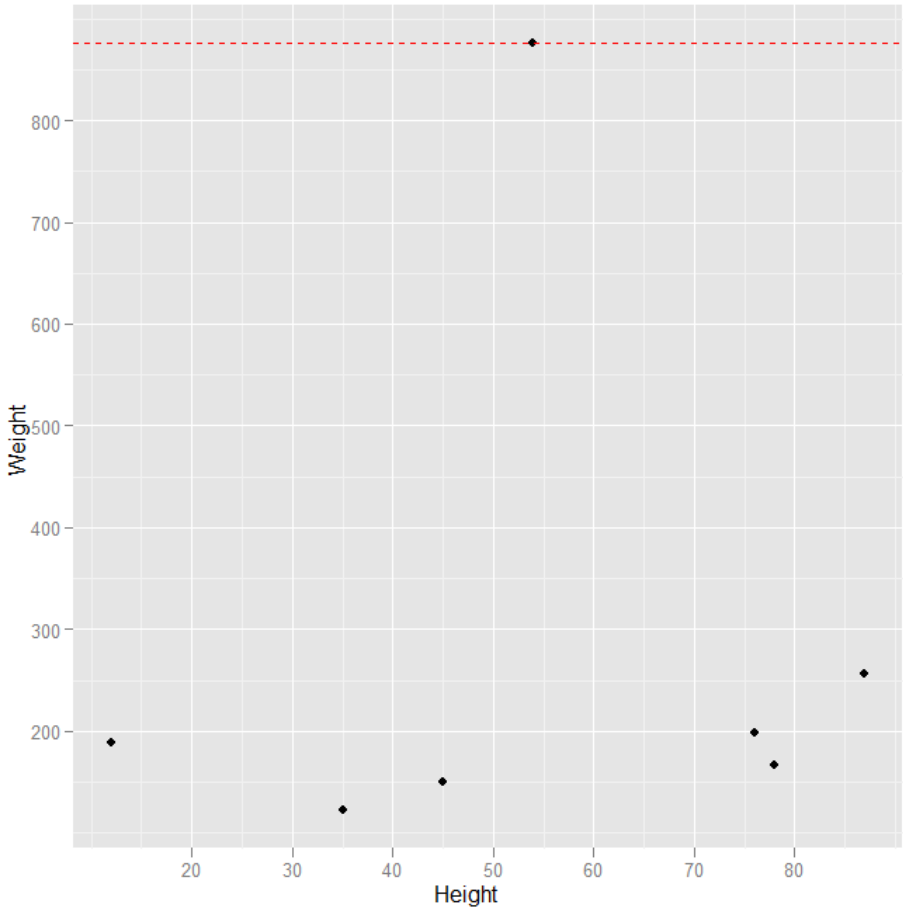
See next slide:



Graphics Using the ggplot2 Package

As you see below, there is a horizontal line at Lauren's best-in-class weight of 876 pounds. If you look up geom_hline in the R Help, you will find that, for this particular function, this function not only takes a yintercept, but also takes the color, linetype and size of the line as parameters. Let's pimp out our graph by making the line dashed and colored red:

```
gFatKids1 <- ggplot(dfFatKids,aes(x=Height,y=Weight))
gFatKids1 + geom_point() + geom_hline(aes(yintercept=876),color="red",linetype=2,size=0.5)
```

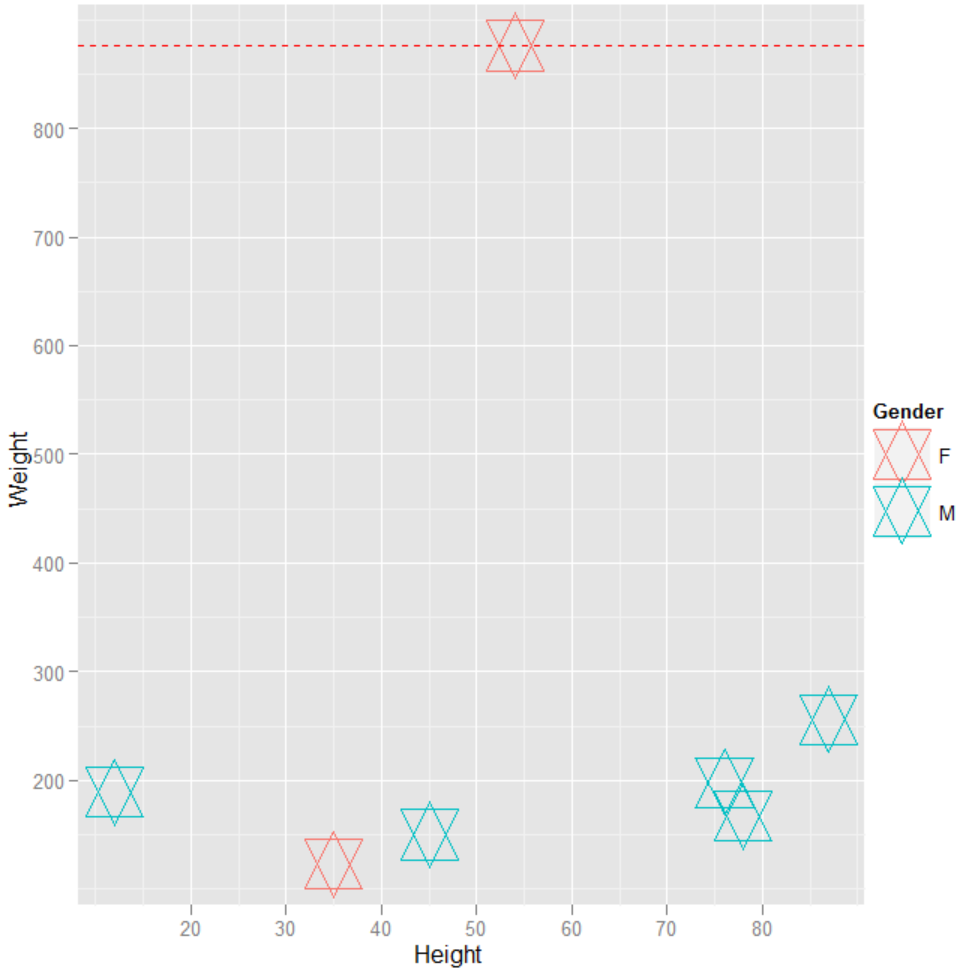


Graphics Using the `ggplot2` Package

Now, let's take a look at the `geom_point()` function. We can specify what color each point can be by using the color parameter within the `aes()` function. Let's color each point based on Gender, change its shape as well as its size:

```
gFatKids1 <- ggplot(dfFatKidsGender,aes(x=Height,y=Weight))
gFatKids1 + geom_point(aes(color=Gender),shape=11,size=10) +
            geom_hline(aes(yintercept=876),color="red",linetype=2,size=0.5)
```

As you can see, our Fat Kids are in Hebrew school…which explains the "guilt" weight…tee-hee…

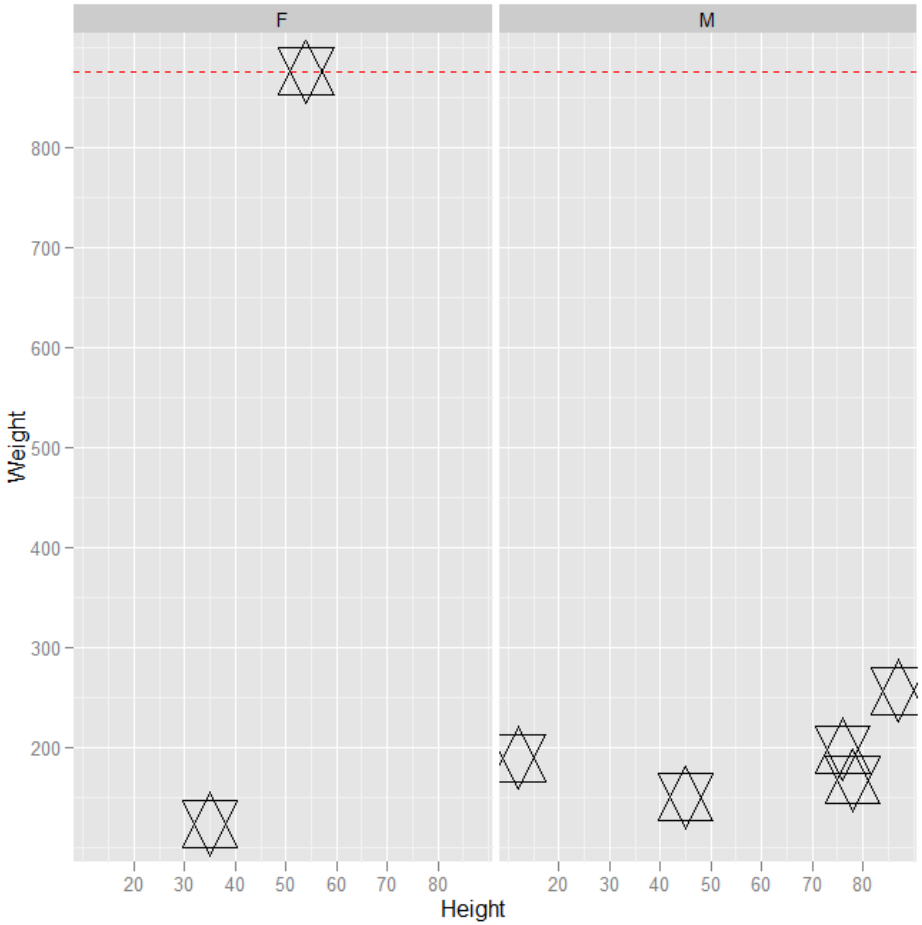


Graphics Using the ggplot2 Package

Recall that we created facets when using the `qplot()` function earlier. We can also use facets with the new plus-sign oriented code as well. In order to use facets, you must use the `facet_grid()` function:

```
gFatKids1 <- ggplot(dfFatKidsGender,aes(x=Height,y=Weight))
gFatKids1 + geom_point(shape=11,size=10) +
            geom_hline(aes(yintercept=876),color="red",linetype=2,size=0.5) +
            facet_grid(facets = . ~ Gender)
```

Notice that we have two facets, one for the Males and one for the Females. But, there seems to be a problem with the horizontal line, doesn't there? The line is set to 876 because that is the maximum weight for the Females. That's not true for the Males, though, it's 256. How can we create a separate horizontal line for the Females as well as the Males?



Graphics Using the ggplot2 Package

In order to do that, we need to create a data frame containing two fields:

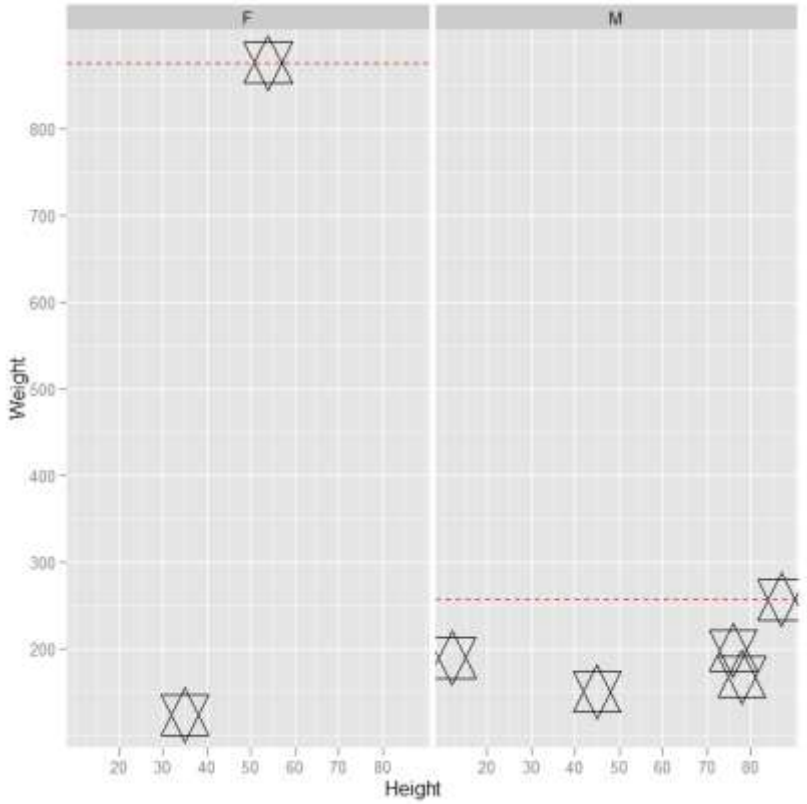
LineHeight = indicates the yintercept values (can be a vector of values)

Gender = indicates the Gender (can be a vector of values, should be a factor)

We use that data frame in the geom_hline() function:

```
dfGenderPlotData <- data.frame(Gender=as.factor(c("F","M")),LineHeight=c(876,256))
gFatKids1 <- ggplot(dfFatKidsGender,aes(x=Height,y=Weight))
gFatKids1 + geom_point(shape=11,size=10) +
            facet_grid(facets = . ~ Gender) +
            geom_hline(aes(yintercept=LineHeight),data=dfGenderPlotData,color="red",linetype=2,size=0.5)
```

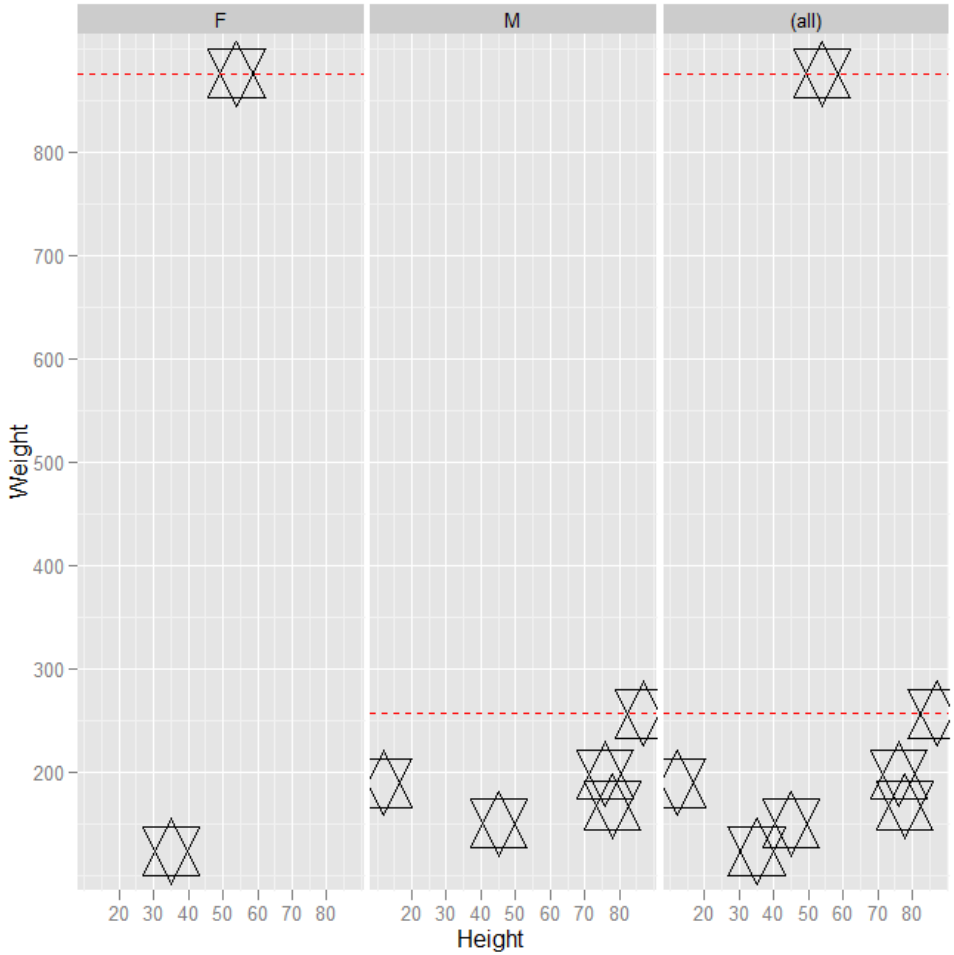
Besides the the facet_grid() function, you can use the facet_wrap() function if you just want a bunch of panels that don't exactly belong in a grid layout. See the manual under ?facet_wrap.



Graphics Using the ggplot2 Package

Now, if you would like to add a total column next to the Female and Male facets, use the `margins=TRUE` option on the `facet_grid()` function:

```
dfGenderPlotData <- data.frame(Gender=as.factor(c("F","M")),LineHeight=c(876,256))
gFatKids1 <- ggplot(dfFatKidsGender,aes(x=Height,y=Weight))
gFatKids1 + geom_point(shape=11,size=10) +
            facet_grid(facets = . ~ Gender,margins=TRUE) +
            geom_hline(aes(yintercept=LineHeight),data=dfGenderPlotData,color="red",linetype=2,size=0.5)
```

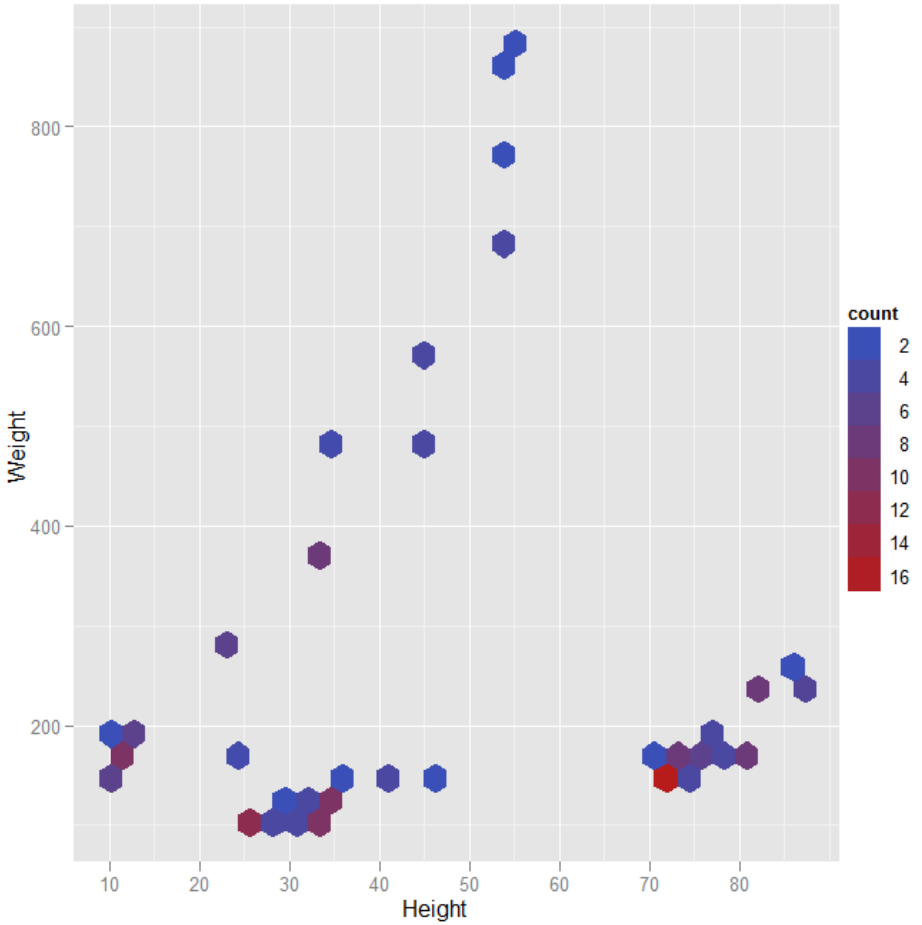


Graphics Using the ggplot2 Package

Rather than focusing more on the graphs we've seen, such as histograms and box-and-whisker plots, let's take a look at some more interesting plots. For example, instead of plotting points on a graph, we can plot hexagrams and color them to indicate the number of points within each hexagram. To do this, we use the `geom_hex()` function along with the `stat_binhex()` function:

```
gFatKidsHist1 <- ggplot(dfFatKidsHist,aes(x=Height,y=Weight))
gFatKidsHist1 + stat_binhex() + geom_hex()
```

By default, there are 30 bins, but you can change that using the `stat_binhex()` function: `stat_binhex(bins=30)`.



Graphics Using the ggplot2 Package

So far, we've seen the `ggplot()` function used, as well as several geom_* functions used and a stat_* function used. To reiterate, the `ggplot()` function sets up the graph, but does not produce a graph itself. One of the geom_* functions produces the graph. For example, the `geom_point()` function produced a point graph. The stat_* functions are used to modify the type of statistics that are being computed. For example, when we used the `geom_hex()` function, we can use the `stat_binhex()` function to change the default binning from 30 to something else.

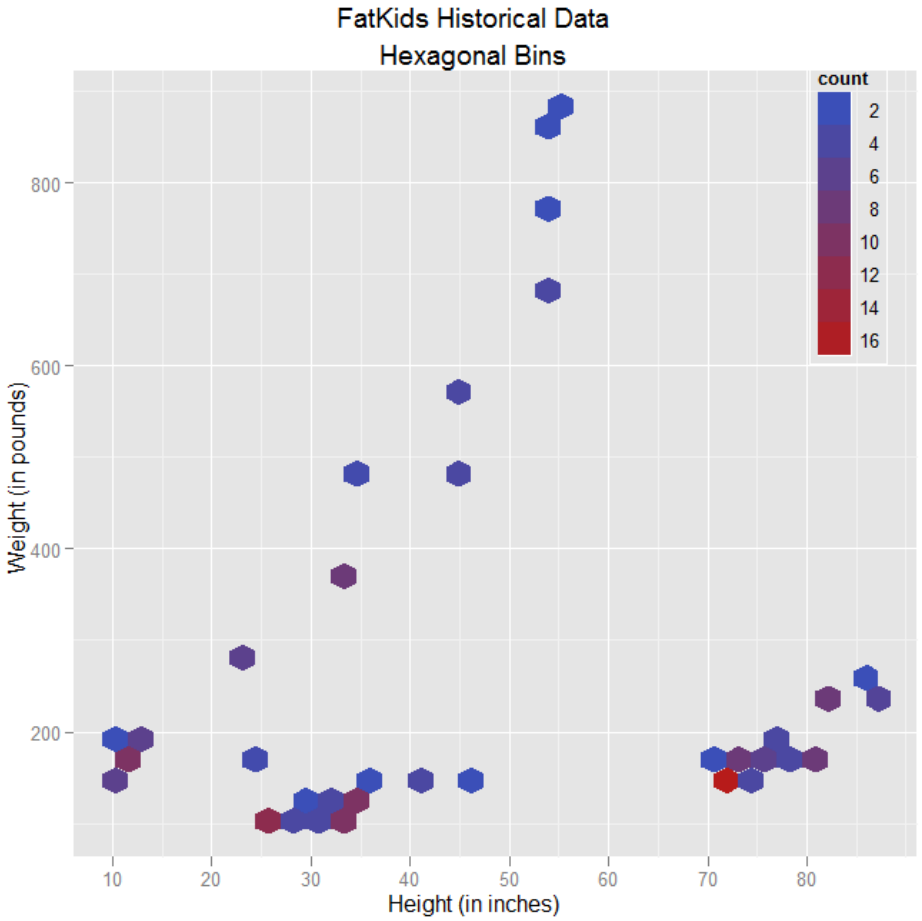
There are several other functions you can use to modify your graphs. As we saw in the overview, we used the `opts()` function to place a title on the graph. You can find a list of options (along with examples) on this webpage: http://github.com/hadley/ggplot2/wiki/+opts()-List. We can also change the x- and y-axis text using the `xlab()`, `ylab()` or `labs()` function.

Using the example on the previous slide, let's add a title as well as more descriptive text on the x- and y-axes and let's move the legend to the upper-right corner of the graph:

```
gFatKidsHist1 <- ggplot(dfFatKidsHist,aes(x=Height,y=Weight))
gFatKidsHist1 +
  stat_binhex() +
  geom_hex() +
  labs(x="Height (in inches)",y="Weight (in pounds)") +
  opts(title="FatKids Historical Data\nHexagonal Bins",legend.position=c(.90,.75))
```

Graphics Using the `ggplot2` Package

The graph appears like this:



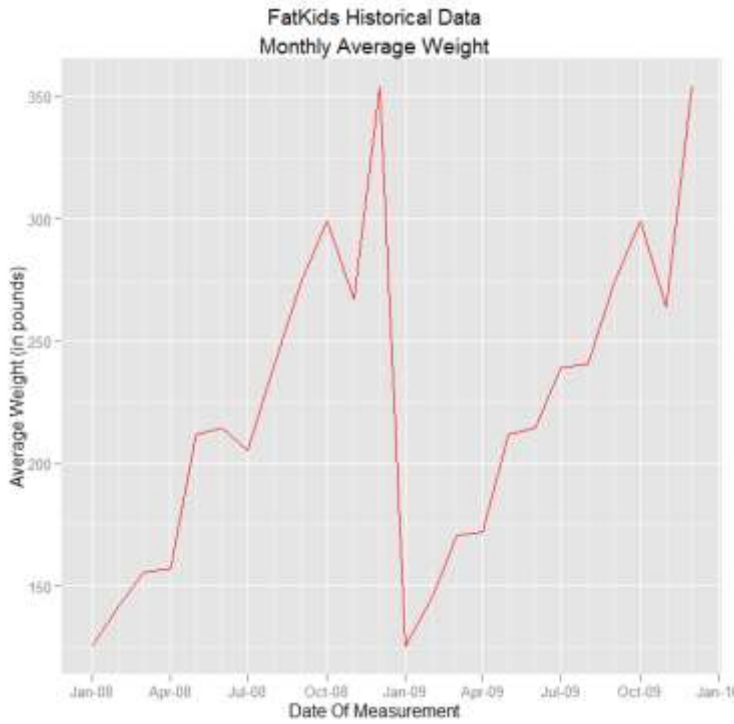
Next, let's produce a graph of the historical Fat Kid data by month computing average Weight:

Graphics Using the ggplot2 Package

Here is the code:

```
dfFatKidsHist_AverageWeight_by_Month <-
                              with(dfFatKidsHist,aggregate(Weight,list(DateOfMeas=DateOfMeas),mean))
colnames(dfFatKidsHist_AverageWeight_by_Month)[2] <- "AverageWeight"
   DateOfMeas AverageWeight
1  2008-01-01      125.3750
2  2008-02-01      141.8333
…
23 2009-11-01      264.1429
24 2009-12-01      354.2500

gFatKidsHist2 <- ggplot(dfFatKidsHist_AverageWeight_by_Month,aes(x=DateOfMeas,y=AverageWeight))
gFatKidsHist2 + geom_line(color="red") +
  labs(x="Date Of Measurement",y="Average Weight (in pounds)") +
  opts(title="FatKids Historical Data\nMonthly Average Weight")
```

The graph appears on the right. As you see, the x-axis shows only a few dates and the text is horizontal. First, let's try to show all of the months at an angle of 45 degrees:



Graphics Using the ggplot2 Package

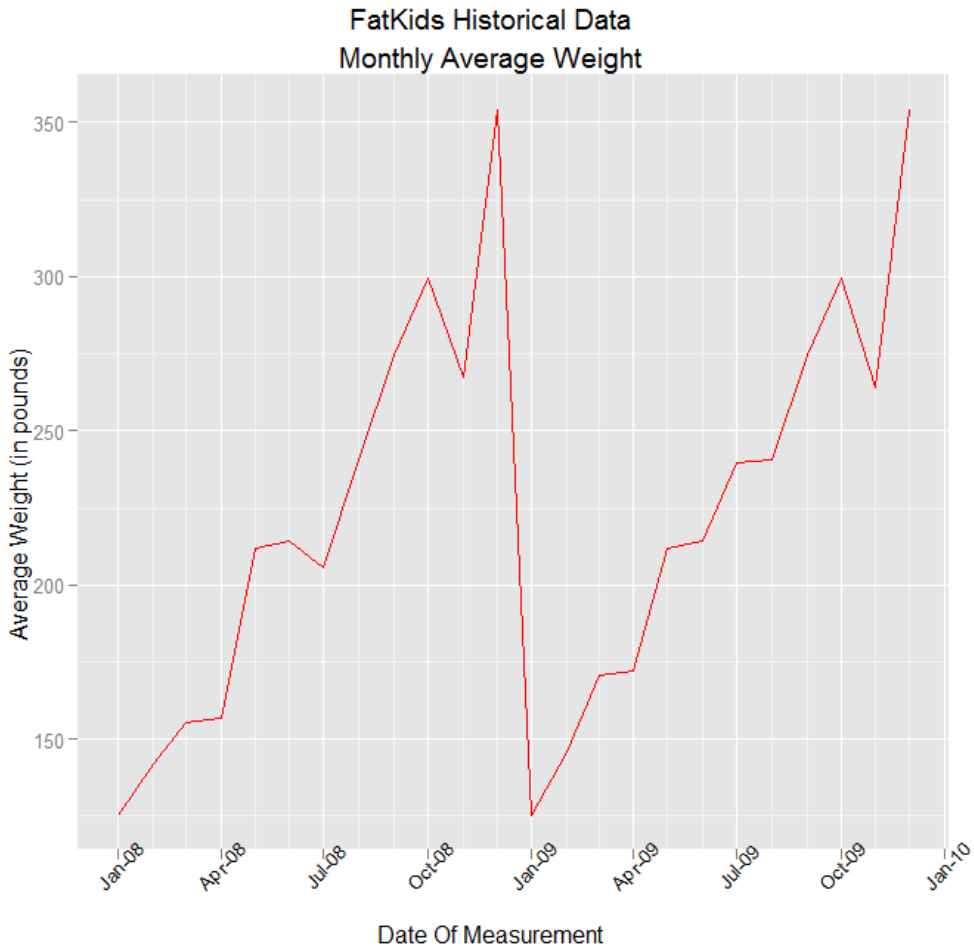
Here is the code:

```
gFatKidsHist2 <- ggplot(dfFatKidsHist_AverageWeight_by_Month,aes(x=DateOfMeas,y=AverageWeight))
gFatKidsHist2 + geom_line(color="red") +
  labs(x="Date Of Measurement",y="Average Weight (in pounds)") +
  opts(title="FatKids Historical Data\nMonthly Average Weight",
       axis.text.x = theme_text(colour='black',angle=45))
```

You'll notice three things about this graph:

1. The center of the date text is under the tick mark for the month.
2. The last month is Jan-10 and we want to stick to a range of Jan-08 to Dec-09 (since that's the data we have).
3. We don't have all 24 months showing up.

Let's handle each one of these problems.



Graphics Using the ggplot2 Package

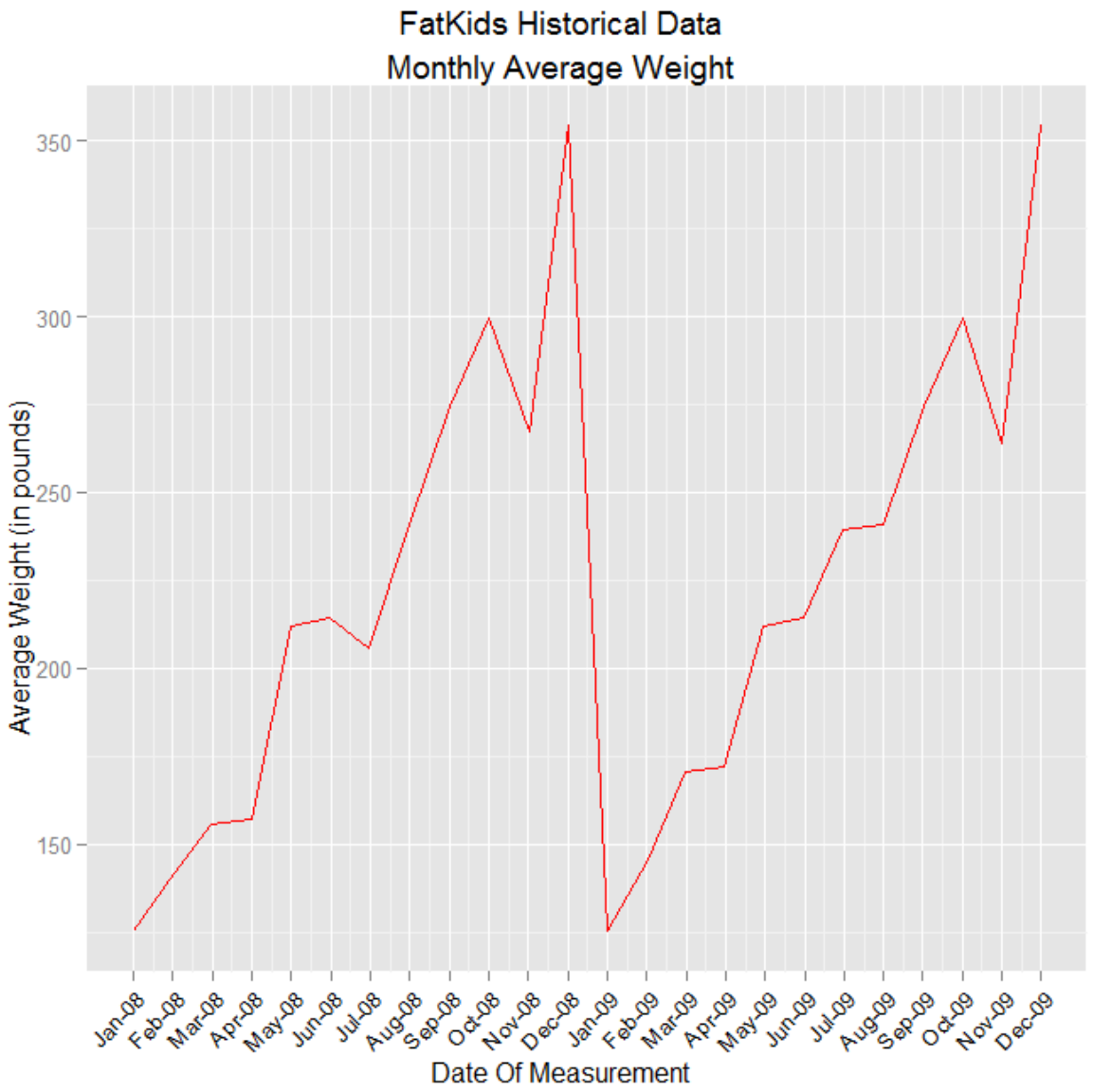
First, in order to nudge the date text down and to the left, we have to add two additional axis.text.x options: hjust (for horizontal adjustment) and vjust (for verticle adjustment). In our case, we want to move the text down and to the left a bit

The Second and Third problems can be answered by setting the x-axis cut points using the `scale_x_continuous()` function.

Here is the code:

```
# Create numeric date ranges for use with breaks= below
d1 <- as.numeric(seq(as.Date("2008-01-01"),as.Date("2009-12-01"),length.out=24))

# Create text dates in mon-yy format for use with labels= below
d2 <- format(seq(as.Date("2008-01-01"),as.Date("2009-12-01"),by="1 month"),"%b-%y")

gFatKidsHist2 <- ggplot(dfFatKidsHist_AverageWeight_by_Month,aes(x=DateOfMeas,y=AverageWeight))

gFatKidsHist2 + geom_line(color="red") +
 labs(x="Date Of Measurement",y="Average Weight (in pounds)") +
 opts(title="FatKids Historical Data\nMonthly Average Weight",
      axis.text.x = theme_text(angle=45,hjust=1,vjust=1))  +
      scale_x_continuous(breaks=d1,
                         labels=d2,
                         limits=c(as.numeric(as.Date("2008-01-01")),as.numeric(as.Date("2009-12-01"))))
```

Graphics Using the `ggplot2` Package



Graphics Using the ggplot2 Package

There's a lot more to the ggplot2 package than shown here. I urge you to purchase the book ggplot2 by Hadley Wickham (see the reference section at the end of the presentation).

ggplot2 allows you to produce maps as well as the graphs shown in this section of the presentation. See the maps package.

ggplot2 also allows you to set up your own "themes" of colors and styles so that you can produce consistent graphs.

ggplot2 does not allow you to produce 3-D graphs, but there may be a new release of ggplot2 that will eventually have that functionality. Keep watching CRAN!

Graphics using the `lattice` Package

Graphics Using the `lattice` Package

The `lattice` package, unlike the `ggplot2` package and similar to the graphics package, contains a series of high-level functions which plot your data. There is no need to use a plus-sign delimited series of functions to produce your graphs, as in the `ggplot2` package; and there is no need to issue several low-level functions to produce your graphs, as in the `graphics` package. To use the `lattice` package, issue `library(lattice)` at the R command line.

The following high-level functions are available in the lattice package (please check the documentation for more information):

```
histogram() - produces a histogram
densityplot() - produces a kernel density plot
qqmath() - produces a theoretical quantile plot
qq() - produces a two-sample quantile plot
stripplot() - produces a stripchart
bwplot() - produces box and whisker plots
dotplot() - produces a Cleveland dot plot
barchart() - produces a bar plot
xyplot() - produces a scatter plot
splot() - produces a scatter plot matrix
contourplot() - produces a contour plot of surfaces
levelplot() - produces a false color level plot of surfaces
wireframe() - produces a three-dimensional perspective plot of surfaces
cloud() - produces a three-dimensional scatter plot
parallel() - produces a parallel coordinates plot
```

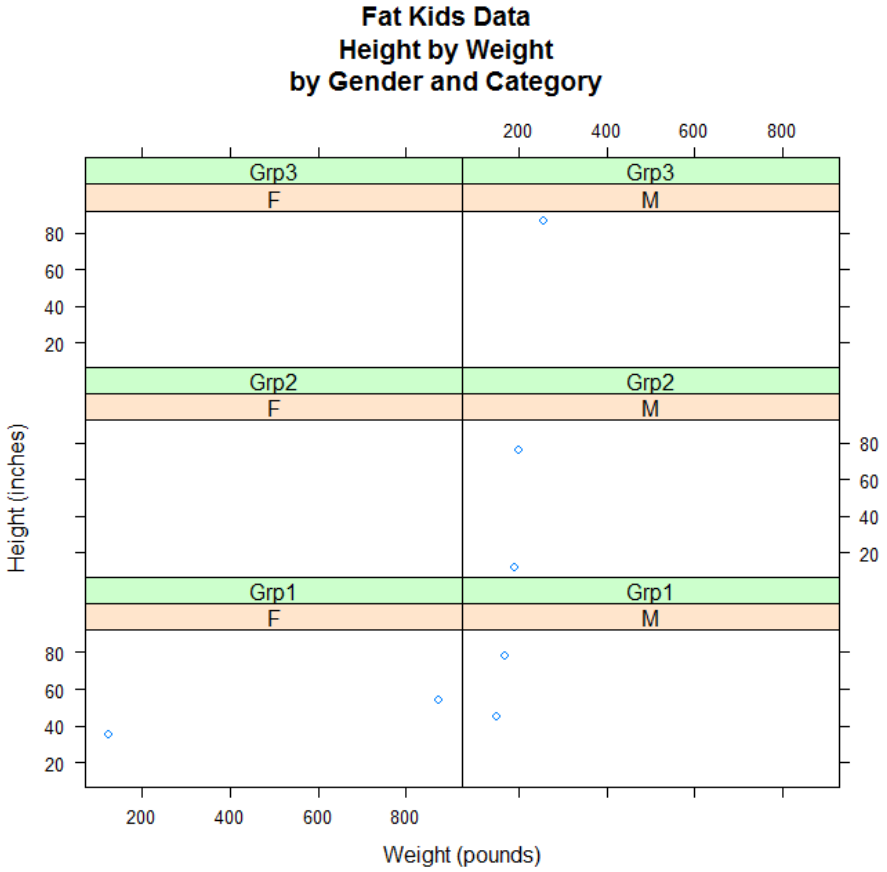
Additional modifications can take place by using graphics parameters as well as using the panel functions alluded to in the overview.

Graphics Using the `lattice` Package

A nice feature of the `lattice` package is the ability to create a trellis plot, similar to faceting in `ggplot2`. The `lattice` package seems to produce nicer looking plots, especially when multiple variables are requested in the trellising.

For example, let's create a graph of the Height and Weight from Fat Kids data by Gender and by Group:

```
> xyplot(Height ~ Weight | Gender + CatGrp, dfFatKids2,
         main="Fat Kids Data\nHeight by Weight\nby Gender and Category",
         xlab="Weight (pounds)",
         ylab="Height (inches)")
```



The `xyplot()` function starts with a formula followed by a vertical bar indicating the trellising variables. In this case, we have two variables, Gender and CatGrp. Note that the first trellis variable Gender appears lowest in the graph, followed by the CatGrp appearing above Gender. The second parameter is the name of the data frame you want to use.

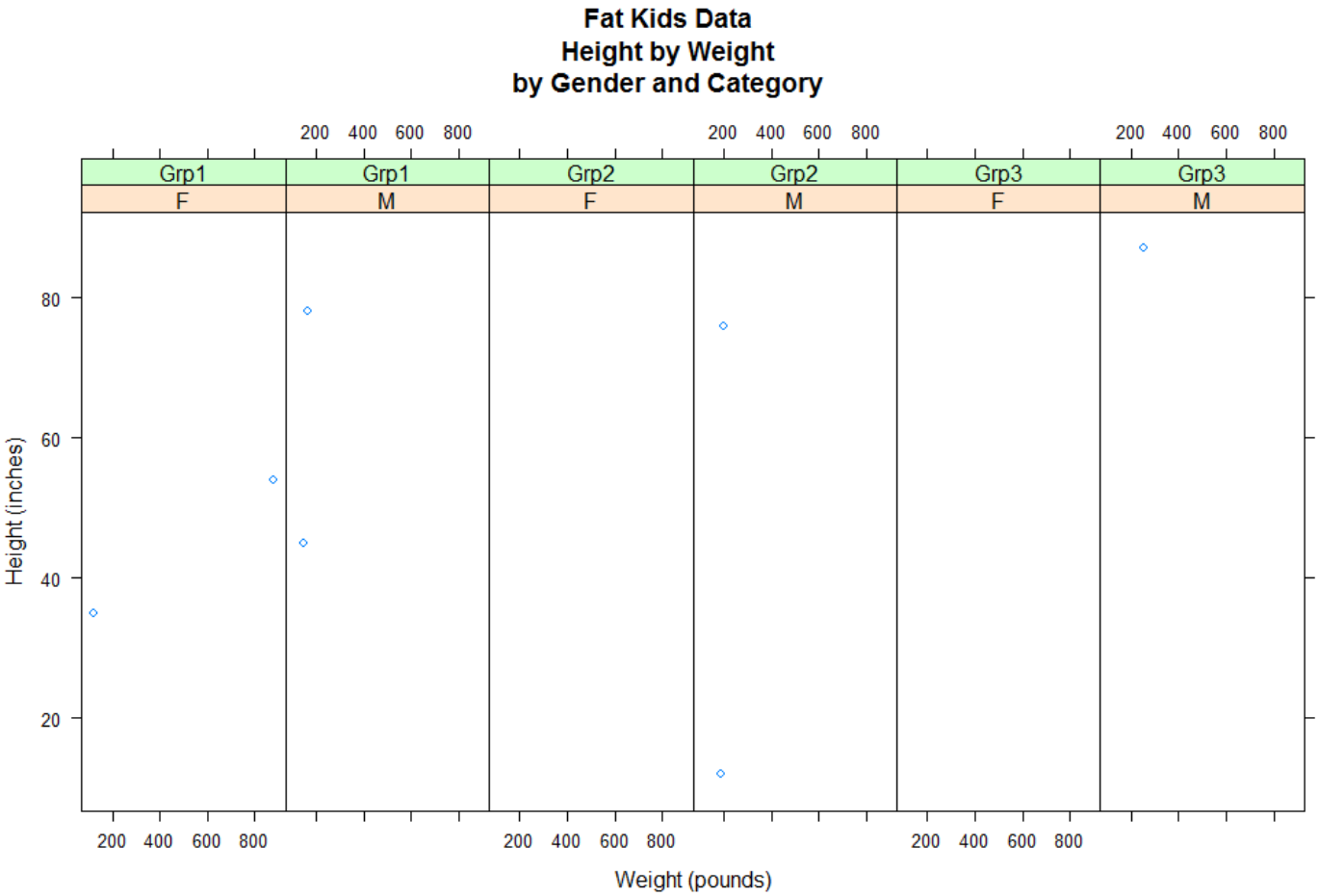
Note that the layout of the panels is done automatically, but you can override it by using the `layout=` vector.

Graphics Using the `lattice` Package

Let's re-create the previous graph with all six panels in one row:

```
> xyplot(Height ~ Weight | Gender + CatGrp, dfFatKids2,
         main="Fat Kids Data\nHeight by Weight\nby Gender and Category",
         xlab="Weight (pounds)",
         ylab="Height (inches)",
         layout=c(6,1))
```

where layout=c(6,1) indicates that you want 6 columns and 1 row to be produced:

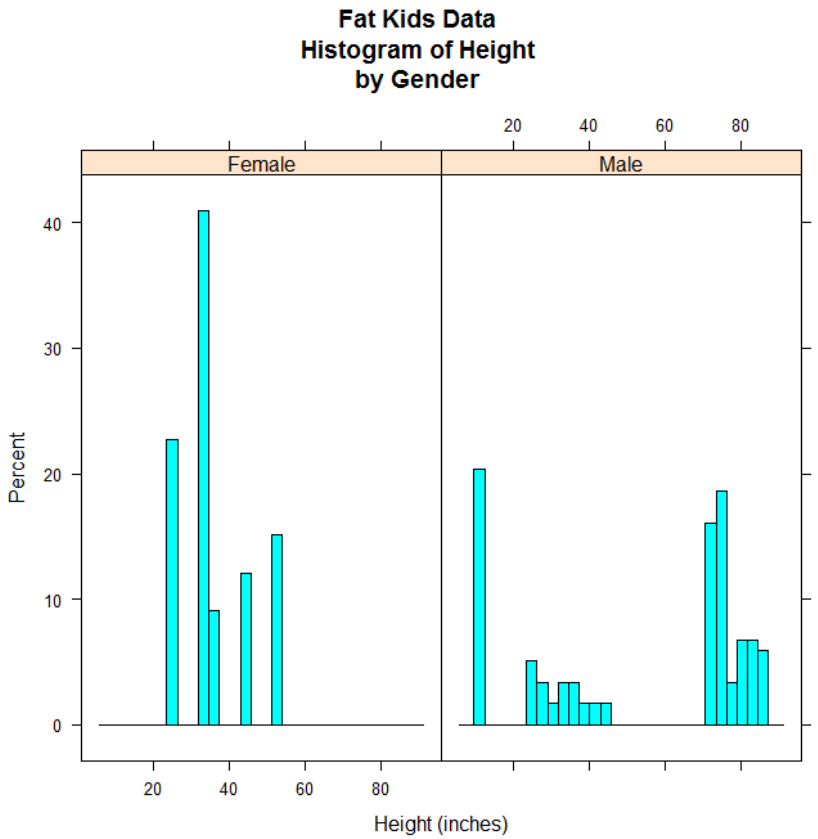


Graphics Using the `lattice` Package

Let's create a histogram of the Height conditioning on Gender:

```
> histogram( ~ Height | Gender, dfFatKidsHist,
             type="percent",
             main="Fat Kids Data\nHistogram of Height\nby Gender",
             xlab="Height (inches)",
             ylab="Percent",
             nint=30)
```

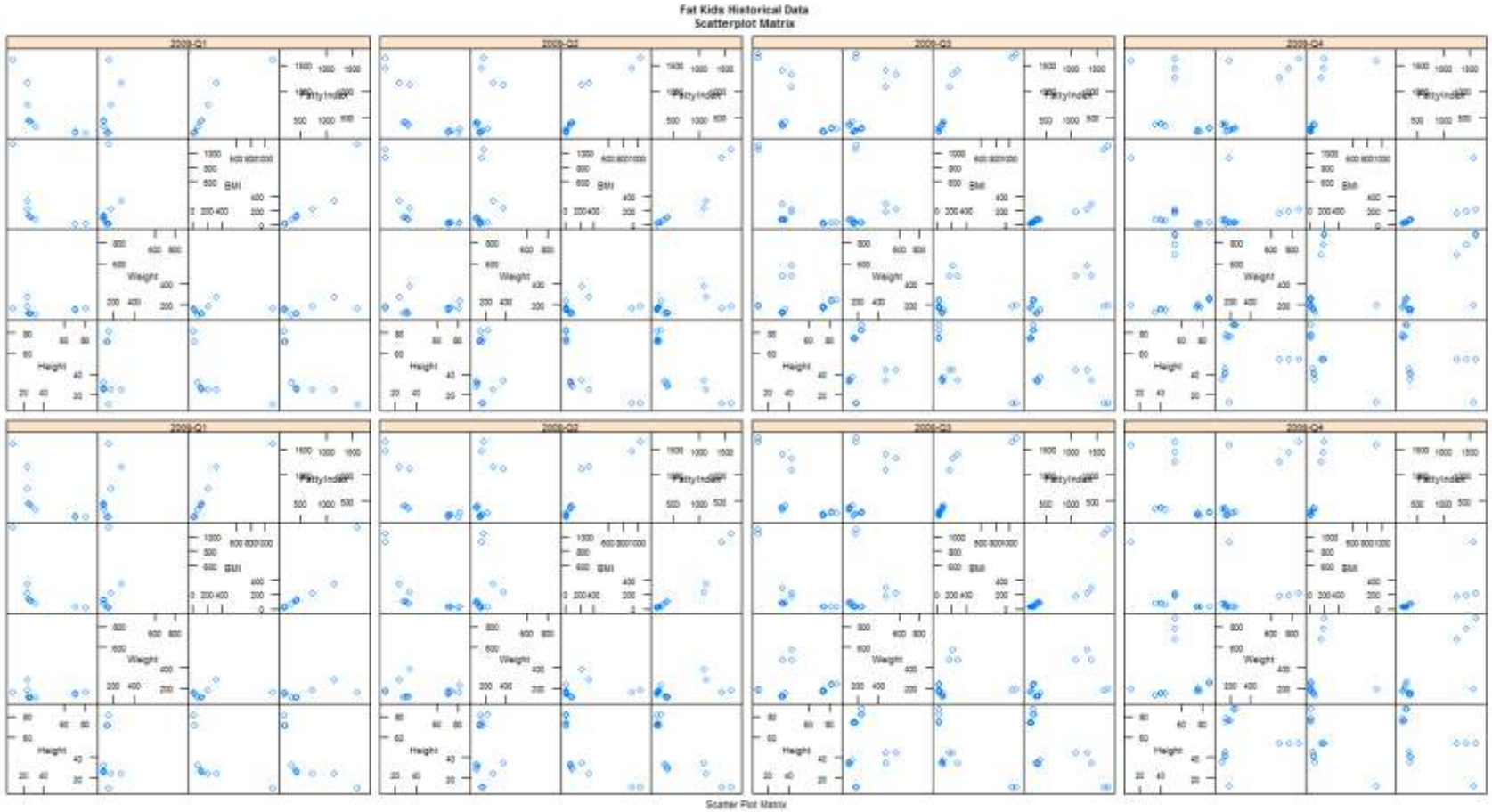
Here nint=30 indicates the number of bins and type indicates that the histogram is showing `percent` on the Y-Axis as opposed to `count`.



Graphics Using the `lattice` Package

We can produce a scatterplot matrix using the `splom()` function and still use the concept of trellising to produce the graph (notice how we are reducing the overall fontsize down by using the `par.settings=` parameter (see `trellis.par.get()` to get a list of modifiable settings):

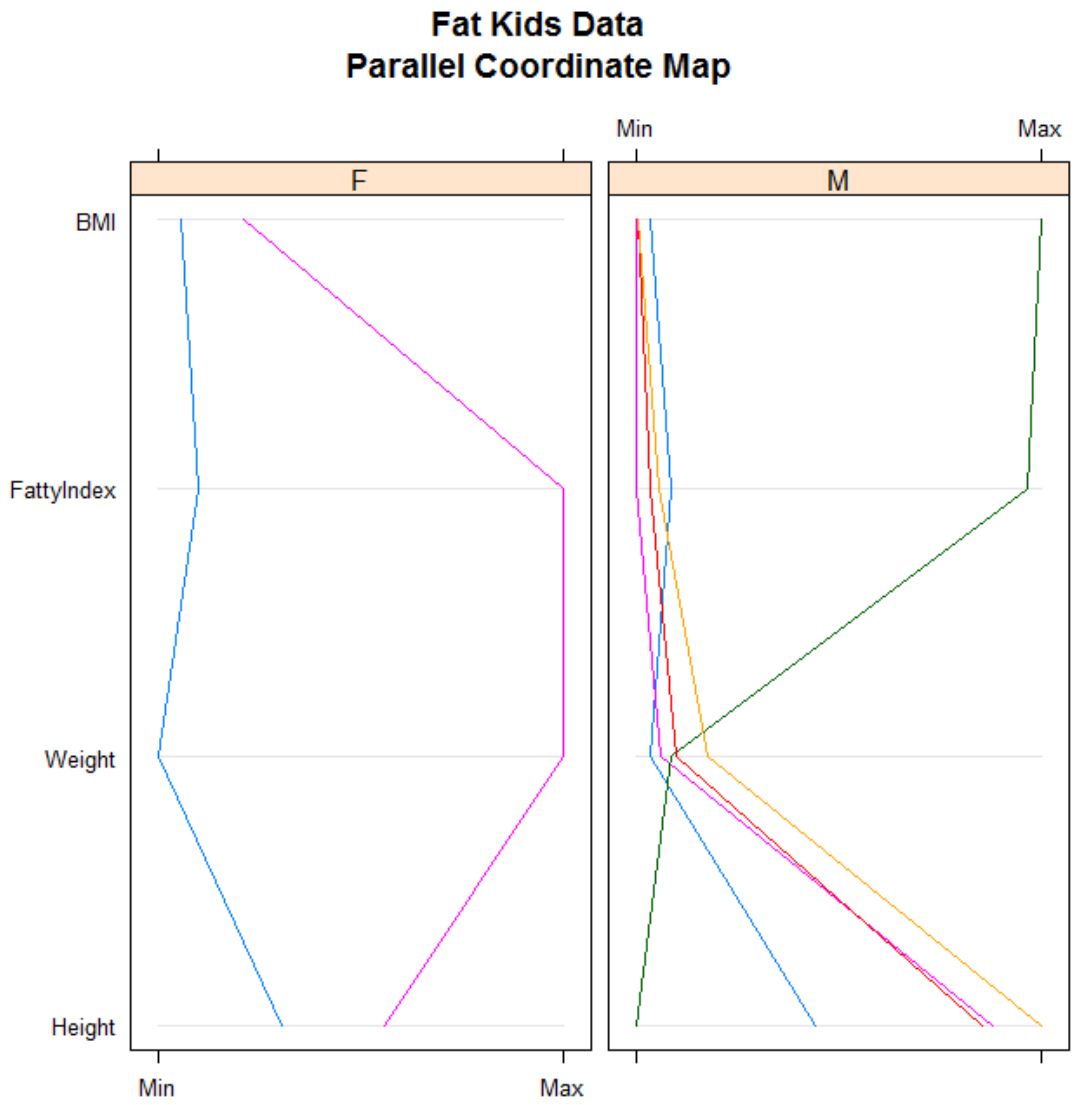
```
splom( ~ dfFatKidsHist[c(3,4,8,9)]| MeasQtr,dfFatKidsHist,main="Fat Kids Historical Data\nScatterplot
Matrix",layout=c(4,2),par.settings=list(fontsize=list(text=7)))
```



Graphics Using the `lattice` Package

Another nice graph is the parallel coordinates graph. Let's plot the Height, Weight, BMI and FattyIndex for all of the fat kids by Gender:

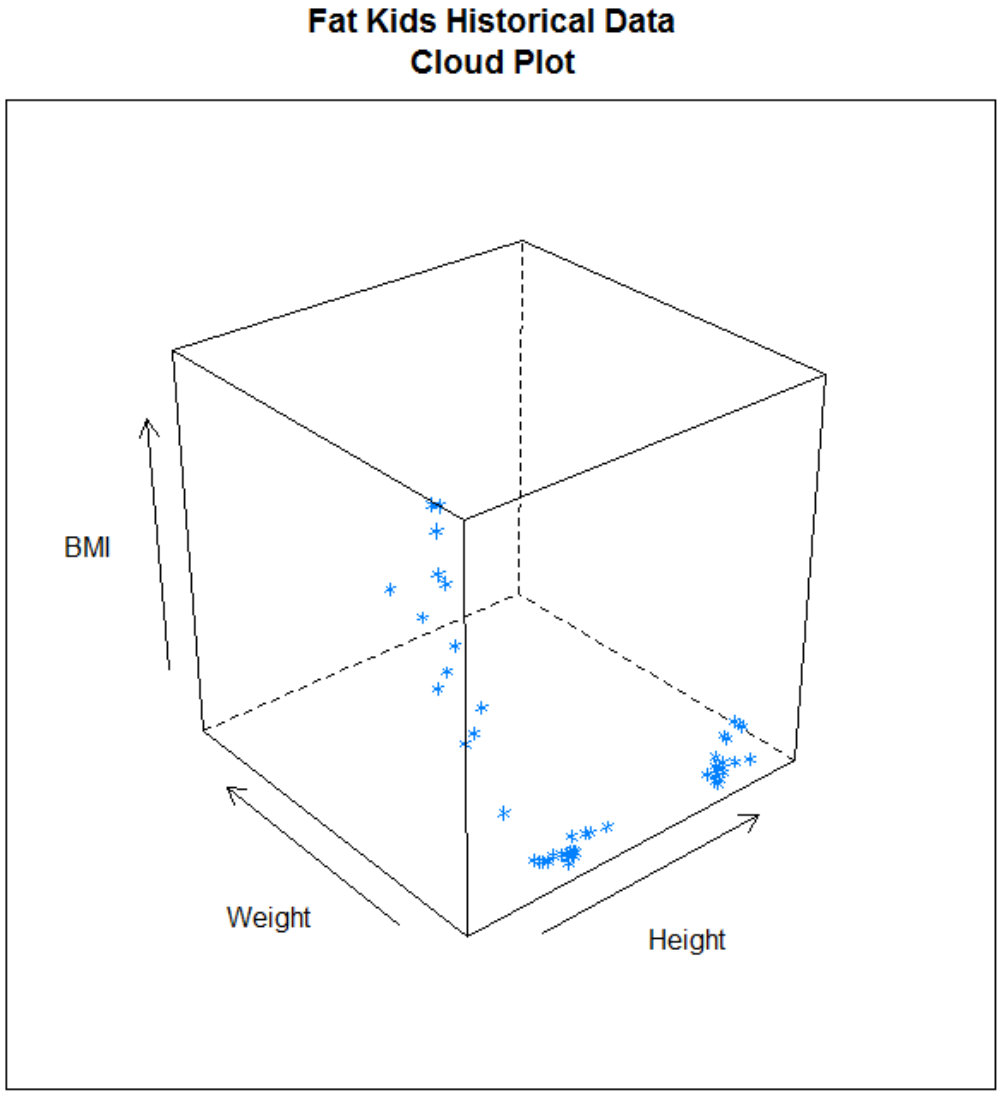
```
parallel( ~ dfFatKids2[c(2,3,4,5)]| Gender,dfFatKids2,main="Fat Kids Data\nParallel Coordinate Map")
```



Graphics Using the `lattice` Package

The lattice package can produce three-dimensional graphs using the `cloud()` function (as well as the `wireframe()` function):

```
cloud(BMI ~ Height * Weight,dfFatKidsHist,main="Fat Kids Historical Data\nCloud Plot")
```



Graphics Using the `lattice` Package

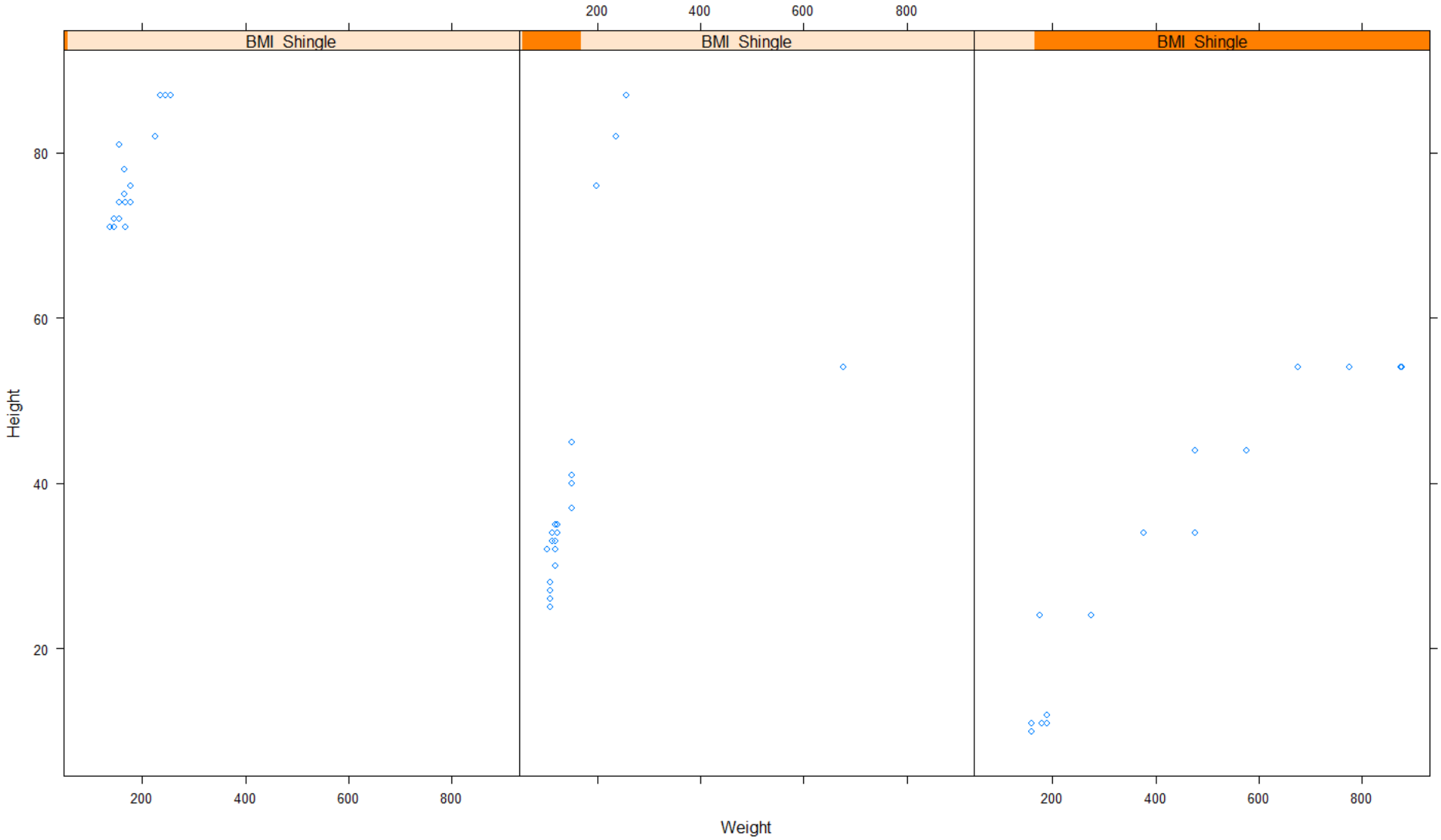
Another nice feature of the lattice package is the *shingle*. A shingle is an overlapping dark rectangle that shows up on the strip. This can represent an additional continuous variable such as time or height. Let's create a shingle based on Body Mass Index and graph the Height and Weight of the Fat Kids historical data. First, we need to create the cuts used in the shingles. Instead of using the `cut()` function, we can use the `equal.count()` function to divide up the data even by row counts and is provided by the `lattice` package. By default, there is a 0.5 fractional overlap between the shingles (hence, the shingles will lay slightly on top of each other, similar to roof shingles), but we can override that by using the `overlap=0` parameter:

```
dfFatKidsHist$BMI_Shingle <- equal.count(dfFatKidsHist$BMI,3,overlap=0)
xyplot(Height ~ Weight | BMI_Shingle,dfFatKidsHist,layout=c(3,1))
```

See next slide for graph.

Note on `equal.count()`: Despite setting `overlap=0`, there still seems to be some overlap. Setting `overlap=-0.01` seems to remove the overlap without dropping any of the data. Note that this function produces data of class "shingle". Typing in the name of the column `dfFatKidsHist$BMI_Shingle` produces information about the data used and the group minimums and maximums.

Graphics Using the `lattice` Package



As you see above, as the body mass index increases (see the orange rectangle), the height and weight seem to increase. This seems to make sense since the BMI formula is 703*Weight/(Height^2).

Graphics Using the `lattice` Package

There's a lot more to the `lattice` package, so pick up the book (Lattice by Deepayan Sarkar) and read it.

Using Graphics Devices to Save Your Graphs

Using Graphics Devices to Save Your Graphs

Up to now, we've been copying our graphs by using the File…Copy to the Clipboard menu item. In this section, we show you how to use graphics devices to save your graphs to an Adobe Portable Document Format (PDF) file, JPEG file, etc. This is very similar to using SAS's Output Delivery System (ODS) to save your output to one of several file types. The following graphics devices are currently available:

- windows: The graphics device for Windows (on screen, to printer and Windows metafile).
- postscript: Writes PostScript graphics commands to a file
- pdf: Write PDF graphics commands to a file
- pictex: Writes LaTeX/PicTeX graphics commands to a file
- png: PNG bitmap device
- jpeg: JPEG bitmap device
- bmp: BMP bitmap device
- tiff: TIFF bitmap device
- xfig: Device for XFIG graphics file format
- bitmap: bitmap pseudo-device via GhostScript (if available)

In order to send your graphs to one of these devices, you must start the device by calling the appropriate function. For example, to send graphs to a PDF file, use the `pdf(file="file-name")` function. Create your graphs, and then call `dev.off()` to stop sending graphs to the pdf file. Below is an example.

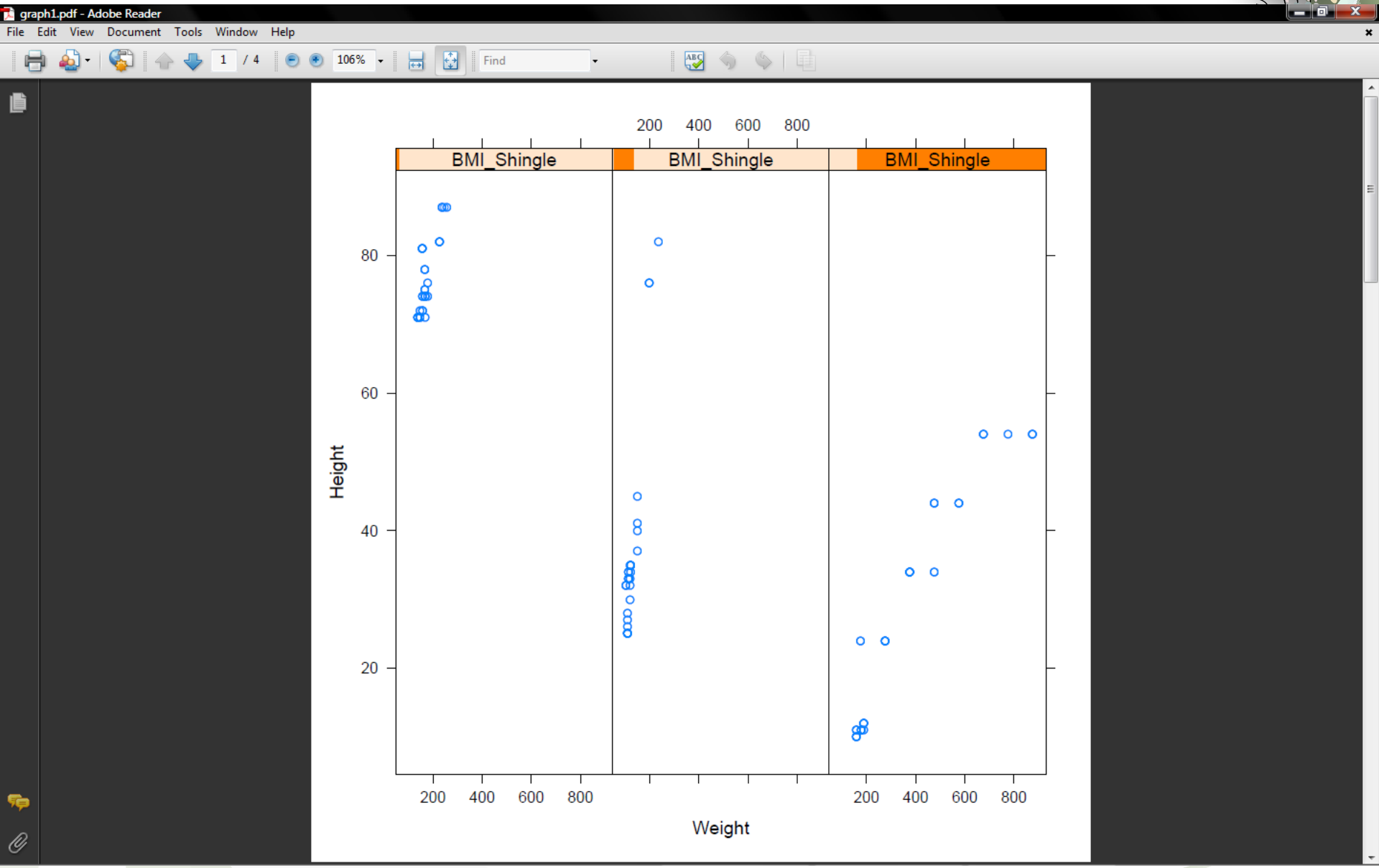
Using Graphics Devices to Save Your Graphs

Sending Multiple Graphs to a PDF File

```
pdf(file="C:\\TEMP\\graph1.pdf")
xyplot(Height ~ Weight | BMI_Shingle,dfFatKidsHist,layout=c(3,1))
cloud(BMI ~ Height * Weight,dfFatKidsHist,main="Fat Kids Historical Data\nCloud Plot")
parallel( ~ dfFatKids2[c(2,3,4,5)]| Gender,dfFatKids2,main="Fat Kids Data\nParallel Coordinate Map")
splom( ~ dfFatKidsHist[c(3,4,8,9)]| MeasQtr,dfFatKidsHist,main="Fat Kids Historical Data\nScatterplot
Matrix",layout=c(4,2),par.settings=list(fontsize=list(text=7)))
dev.off()
```

The PDF file is created and it will contain 4 graphs. The next slide shows the first graph. Please see the documentation (`?pdf`) for more on the pdf device and its options.

Using Graphics Devices to Save Your Graphs



Appendix

Appendix A: References

Click the titles below to be taken to Amazon.com's website.

- SAS and R, 1st Edition, Ken Kleinman and Nicholas J. Horton (ISBN: 9781420070576)
- R for SAS and SPSS Users, 1st Edition, Robert A. Muenchen (ISBN: 9780387094175)
- Data Manipulation with R, 1st Edition, Phil Spector (ISBN: 9780387747309)
- R In A Nutshell, Joseph Adler (ISBN: 9780596801700)
- Software for Data Analysis: Programming with R, John M. Chambers (ISBN: 9780387759357)
- R Through Excel, Richard Heiberger and Erich Neuwirth, (ISBN: 9781441900517)
- The R Book, Michael J. Crawley (ISBN: 9780470510247)
- Interactive and Dynamic Graphics for Data Analysis with R and GGobi, Dianne Cook and Deborah F. Swayne (ISBN: 9780387717616)
- Lattice: Multivariate Data Visualization with R, Deepayan Darkar (ISBN: 9780387759685)
- ggplot2: Elegant Graphics for Data Analysis, Hadley Wickham (ISBN: 9780387981406)
- Introductory Statistics with R, 2nd Edition, Peter Dalgaard (ISBN: 9780387790541)
- Modern Applied Statistics with S, W.N. Venables and B.D. Ripley (ISBN: 9781441930088)
- Manuals Provided with R Software: An Introduction to R, R Data Import/Export, R Language Definition, R Installation and Administration, R Internals, Writing R Extensions
- GGobi Manual (http://www.ggobi.org/rggobi/introduction.pdf), Deborah F. Swayne, Hadley Wickham, et. al.
- cran.r-project.org/web/packages/RcmdrPlugin.HH/RcmdrPlugin.HH.pdf: Documentation for the RcmdrPlugin.HH plug-in to R Commander.

Appendix B: R-Related Websites

- www.r-project.org: This is the main R Software Site and contains the software itself for various platforms as well as documentation.
- cran.r-project.org: This is the website of the Comprehensive R Archive Network (CRAN) and it houses all of the R packages you could ever possibly want.
- journal.r-project.org: This is the website of the R Journal which is a refereed journal of the R Project and contains articles on a variety of topics related to R.
- rwiki.sciviews.org/doku.php: This is the R Wiki website and houses a variety of searchable content.
- sourceforge.net: This is the main SourceForge website and contains free software not necessarily related to R (but does contain a lot of R-related software).
- r-forge.r-project.org: This is the main Rforge website and contains R-related development software such as packages, graphical user interfaces, etc.
- rcom.univie.ac.at: R and Friends website with RCOM package and also R bundled with RCOM.
- www.ggobi.org/downloads: GGobi website with free download of GGobi.
- www.ggobi.org/demos: GGobi demo movies and screen shots.
- cran.r-project.org/web/packages/RcmdrPlugin.HH: Website for the RcmdrPlugin.HH plug-in to R Commander.

Appendix C: graphics Package Function List

Information on package 'graphics'

Axis	Generic function to add an Axis to a Plot
abline	Add Straight Lines to a Plot
arrows	Add Arrows to a Plot
assocplot	Association Plots
axTicks	Compute Axis Tickmark Locations
axis	Add an Axis to a Plot
axis.POSIXct	Date and Date-time Plotting Functions
barplot	Bar Plots
box	Draw a Box around a Plot
boxplot	Box Plots
boxplot.matrix	Draw a Boxplot for each Column (Row) of a Matrix
bxp	Draw Box Plots from Summaries
cdplot	Conditional Density Plots
clip	Set Clipping Region
contour	Display Contours
coplot	Conditioning Plots
curve	Draw Function Plots
dotchart	Cleveland Dot Plots
filled.contour	Level (Contour) Plots
fourfoldplot	Fourfold Plots
frame	Create / Start a New Plot Frame
graphics-package	The R Graphics Package
grconvertX	Convert between Graphics Coordinate Systems
grid	Add Grid to a Plot
hist	Histograms
hist.POSIXt	Histogram of a Date or Date-Time Object
identify	Identify Points in a Scatter Plot
image	Display a Color Image
layout	Specifying Complex Plot Arrangements
legend	Add Legends to Plots
lines	Add Connected Line Segments to a Plot

Appendix C: graphics Package Function List

Information on package 'graphics'

locator	Graphical Input
matplot	Plot Columns of Matrices
mosaicplot	Mosaic Plots
mtext	Write Text into the Margins of a Plot
pairs	Scatterplot Matrices
panel.smooth	Simple Panel Plot
par	Set or Query Graphical Parameters
persp	Perspective Plots
pie	Pie Charts
plot	Generic X-Y Plotting
plot.data.frame	Plot Method for Data Frames
plot.default	The Default Scatterplot Function
plot.design	Plot Univariate Effects of a 'Design' or Model
plot.factor	Plotting Factor Variables
plot.formula	Formula Notation for Scatterplots
plot.histogram	Plot Histograms
plot.table	Plot Methods for 'table' Objects
plot.window	Set up World Coordinates for Graphics Window
plot.xy	Basic Internal Plot Function
points	Add Points to a Plot
polygon	Polygon Drawing
rect	Draw One or More Rectangles
rug	Add a Rug to a Plot
screen	Creating and Controlling Multiple Screens on a Single Device
segments	Add Line Segments to a Plot
smoothScatter	Scatterplots with Smoothed Densities Color Representation
spineplot	Spine Plots and Spinograms
stars	Star (Spider/Radar) Plots and Segment Diagrams
stem	Stem-and-Leaf Plots

Appendix C: graphics Package Function List

Information on package 'graphics'

stripchart	1-D Scatter Plots
strwidth	Plotting Dimensions of Character Strings and Math Expressions
sunflowerplot	Produce a Sunflower Scatter Plot
symbols	Draw Symbols (Circles, Squares, Stars, Thermometers, Boxplots) on a Plot
text	Add Text to a Plot
title	Plot Annotation
xinch	Graphical Units
xspline	Draw an X-spline

Appendix D: ggplot2 Package Function List

Information on package ggplot2'

Coord	See website for documentation
aes	Generate aesthetic mappings
borders	Map borders.
comma	Comma formatter
coord_cartesian	coord_cartesian
coord_equal	coord_equal
coord_flip	coord_flip
coord_map	coord_map
coord_polar	coord_polar
coord_trans	coord_trans
cut_interval	Discretise continuous variable, equal interval length.
cut_number	Discretise continuous variable, equal number of points.
diamonds	Prices of 50,000 round cut diamonds
dollar	Currency formatter
economics	US economic time series
expand_limits	Expand the plot limits with data.
expand_range	Expand range
facet_grid	facet_grid
facet_wrap	facet_wrap
fortify	Fortify a model with data
fortify.SpatialPolygonsDataFrame	Fortify spatial polygons and lines
fortify.lm	Fortify a linear model with its data
fortify.map	Fortify a map
geom_abline	geom_abline
geom_area	geom_area
geom_bar	geom_bar
geom_bin2d	geom_bin2d
geom_blank	geom_blank

Appendix D: ggplot2 Package Function List

Information on package ggplot2'

geom_boxplot	geom_boxplot
geom_contour	geom_contour
geom_crossbar	geom_crossbar
geom_density	geom_density
geom_density2d	geom_density2d
geom_errorbar	geom_errorbar
geom_errorbarh	geom_errorbarh
geom_freqpoly	geom_freqpoly
geom_hex	geom_hex
geom_histogram	geom_histogram
geom_hline	geom_hline
geom_jitter	geom_jitter
geom_line	geom_line
geom_linerange	geom_linerange
geom_path	geom_path
geom_point	geom_point
geom_pointrange	geom_pointrange
geom_polygon	geom_polygon
geom_quantile	geom_quantile
geom_rect	geom_rect
geom_ribbon	geom_ribbon
geom_rug	geom_rug
geom_segment	geom_segment
geom_smooth	geom_smooth
geom_step	geom_step
geom_text	geom_text
geom_tile	geom_tile
geom_vline	geom_vline
ggfluctuation	Fluctuation plot
ggmissing	Missing values plot
ggorder	Order plot

Appendix D: ggplot2 Package Function List

Information on package ggplot2'

ggpcp	Parallel coordinates plot.
ggplot	Create a new plot
ggplot.data.frame	Create a new plot
ggsave	ggsave
ggstructure	Structure plot
label_both	Label facets with value and variable
label_bquote	Label facet with 'bquoted' expressions
label_parsed	Label facets with parsed label.
label_value	Label facets with their value
labs	Change axis labels and legend titles
last_plot	Retrieve last plot modified/created.
map_data	Map data
midwest	Demographic information of midwest counties
movies	Movie information and user ratings from IMDB.com
mpg	Fuel economy data from 1999 and 2008 for 38 popular models of car
msleep	An updated and expanded version of the mammals sleep dataset
opts	Plot options
percent	Percent formatter
plotmatrix	Code to create a scatterplot matrix (experimental)
position_dodge	position_dodge
position_fill	position_fill
position_identity	position_identity
position_jitter	position_jitter
position_stack	position_stack
presidential	Terms of 10 presidents from Eisenhower to Bush W.
qplot	Quick plot.

Appendix D: ggplot2 Package Function List

Information on package ggplot2'

rescale	Rescale numeric vector
scale_alpha_continuous	scale_alpha_continuous
scale_brewer	scale_brewer
scale_continuous	scale_continuous
scale_date	scale_date
scale_datetime	scale_datetime
scale_discrete	scale_discrete
scale_gradient	scale_gradient
scale_gradient2	scale_gradient2
scale_gradientn	scale_gradientn
scale_grey	scale_grey
scale_hue	scale_hue
scale_identity	scale_identity
scale_linetype_discrete	scale_linetype_discrete
scale_manual	scale_manual
scale_shape_discrete	scale_shape_discrete
scale_size_continuous	scale_size_continuous
scientific	Scientific formatter
seals	Vector field of seal movements
stat_abline	stat_abline
stat_bin	stat_bin
stat_bin2d	stat_bin2d
stat_binhex	stat_binhex
stat_boxplot	stat_boxplot
stat_contour	stat_contour
stat_density	stat_density
stat_density2d	stat_density2d
stat_function	stat_function
stat_hline	stat_hline

Appendix D: ggplot2 Package Function List

Information on package ggplot2'

stat_identity	stat_identity
stat_qq	stat_qq
stat_quantile	stat_quantile
stat_smooth	stat_smooth
stat_spoke	stat_spoke
stat_sum	stat_sum
stat_summary	stat_summary
stat_unique	stat_unique
stat_vline	stat_vline
theme_blank	Theme element: blank
theme_bw	Black and white theme
theme_gray	Gray theme
theme_line	Theme element: line
theme_rect	Theme element: rectangle
theme_segment	Theme element: segments
theme_text	Theme element: text
theme_update	Get, set and update themes.
update_geom_defaults	Update geom defaults
update_stat_defaults	Update geom defaults
xlim	Set x limits
ylim	Set y limits

Appendix E: lattice Package Function List

Information on package lattice'

axis.default	Default axis annotation utilities
banking	Banking
barchart.table	table methods for barchart and dotplot
barley	Yield data from a Minnesota barley trial
cloud	3d Scatter Plot and Wireframe Surface Plot
current.row	Accessing Auxiliary Information During Plotting
draw.colorkey	Produce a Colorkey for levelplot
draw.key	Produce a Legend or Key
environmental	Atmospheric environmental conditions in New York City
equal.count	shingles
ethanol	Engine exhaust fumes from burning ethanol
histogram	Histograms and Kernel Density Plots
Lattice	Lattice Graphics
lattice.options	Low-level Options Controlling Behaviour of Lattice
latticeParseFormula	Parse Trellis formula
level.colors	A function to compute false colors representing a numeric or categorical variable
levelplot	Level plots and contour plots
llines	Replacements of traditional graphics functions
lset	Interface to modify Trellis Settings - Deprecated
make.groups	Grouped data from multiple vectors
melanoma	Melanoma skin cancer incidence
oneway	Fit One-way Model
packet.panel.default	Associating Packets with Panels
panel.axis	Panel Function for Drawing Axis Ticks and Labels
panel.barchart	Default Panel Function for barchart
panel.bwplot	Default Panel Function for bwplot

Appendix E: lattice Package Function List

Information on package lattice'

panel.cloud	Default Panel Function for cloud
panel.densityplot	Default Panel Function for densityplot
panel.dotplot	Default Panel Function for dotplot
panel.functions	Useful Panel Functions
panel.histogram	Default Panel Function for histogram
panel.identify	Functions to Interact with Lattice Plots
panel.levelplot	Default Panel Function for levelplot
panel.pairs	Default Superpanel Function for splom
panel.parallel	Default Panel Function for parallel
panel.qqmath	Default Panel and Prepanel Function for qqmath
panel.qqmathline	Useful panel function with qqmath
panel.smoothScatter	Lattice panel function analogous to smoothScatter
panel.stripplot	Default Panel Function for stripplot
panel.superpose	Panel Function for Display Marked by groups
panel.violin	Panel Function to create Violin Plots
panel.xyplot	Default Panel Function for xyplot
prepanel.default.bwplot	Default Prepanel Functions
prepanel.lmline	Useful Prepanel Function for Lattice
print.trellis	Plot and Summarize Trellis Objects
qq	Quantile-Quantile Plots of Two Samples
qqmath	Q-Q Plot with Theoretical Distribution
rfs	Residual and Fit Spread Plots
Rows	Extract rows from a list
simpleKey	Function to generate a simple key
simpleTheme	Function to generate a simple theme
singer	Heights of New York Choral Society singers
splom	Scatter Plot Matrices
strip.default	Default Trellis Strip Function
tmd	Tukey Mean-Difference Plot

Appendix E: lattice Package Function List

Information on package lattice'

trellis.device	Initializing Trellis Displays
trellis.object	A Trellis Plot Object
trellis.par.get	Graphical Parameters for Trellis Displays
update.trellis	Retrieve and Update Trellis Object
utilities.3d	Utility functions for 3-D plots
xyplot	Common Bivariate Trellis Plots
xyplot.ts	Time series plotting methods

Support sheepsqueezers.com

If you found this information helpful, please consider supporting sheepsqueezers.com. There are several ways to support our site:

- ☐ Buy me a cup of coffee by clicking on the following link and donate to my PayPal account: Buy Me A Cup Of Coffee?.
- ☐ Visit my Amazon.com Wish list at the following link and purchase an item: http://amzn.com/w/3OBK1K4EIWIR6

Please let me know if this document was useful by e-mailing me at `comments@sheepsqueezers.com`.