Handout 4: Basic Graphics in R

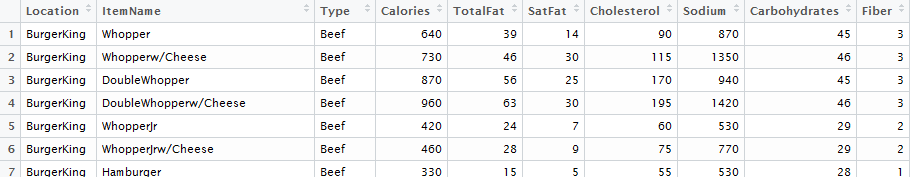
This handout will provide an introduction to creating graphics in R. Here, we will cover only the more basic, traditional graphics. You should be aware, however, that more advanced users can create more complex and interesting graphical summaries of data using R.

First, to see some examples of graphs that can be created in R, enter the following command at the prompt.

> demo(graphics)

**Histograms and Density Smoothers**

Read the NutritionData file into R.



Once this data set has been attached, the names in this data frame are as follows.

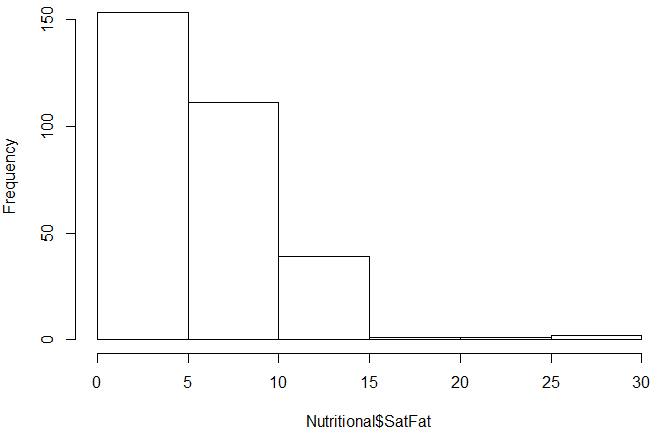
> names(NutritionData)

[1] "Location" "ItemName" "Type" "Calories" "TotalFat"

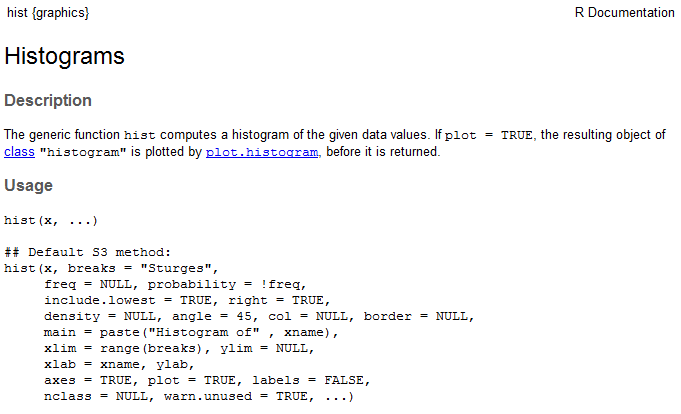
[6] "SatFat" "Cholesterol" "Sodium" "Carbohydrates" "Fiber"

The most basic form of the hist() function is employed below.

> hist(NutritionData$SatFat)

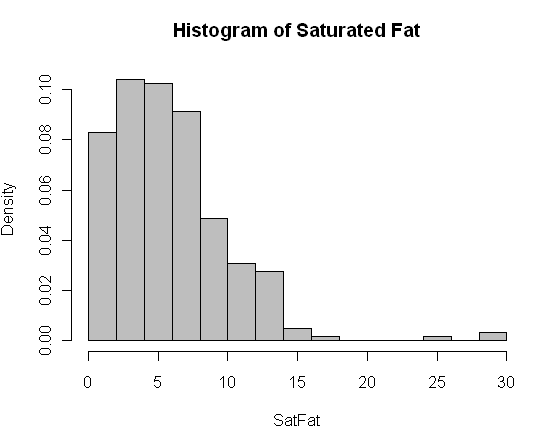


As shown in the following documentation, several optional arguments exist that can be used to modify the resulting plot.



For example, enter the following options to the hist() function.

> hist(NutritionData$SatFat,breaks=20,freq=F,xlab="Saturated Fat", main="Histogram of Saturated Fat",col='gray')



Questions:

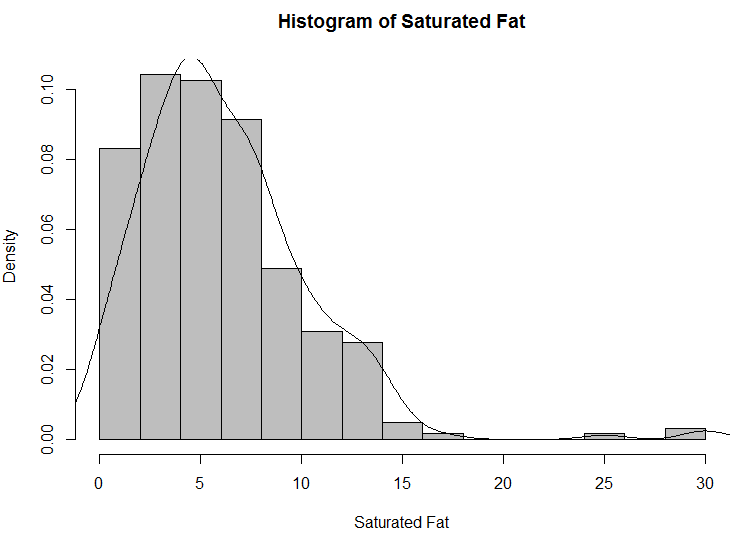
1. What happens when the freq= option is set to TRUE. What changes?
2. Can you figure out how to change the breaks option so that there are breakpoints at 0, 5, 10, 15, 20, 25, and 30? Hint: These values will need to be entered as a vector.

**Adding a Density Smoother to a Histogram in R**

The following command will add a “trend” to the histogram. This trend line is called a density smoother.

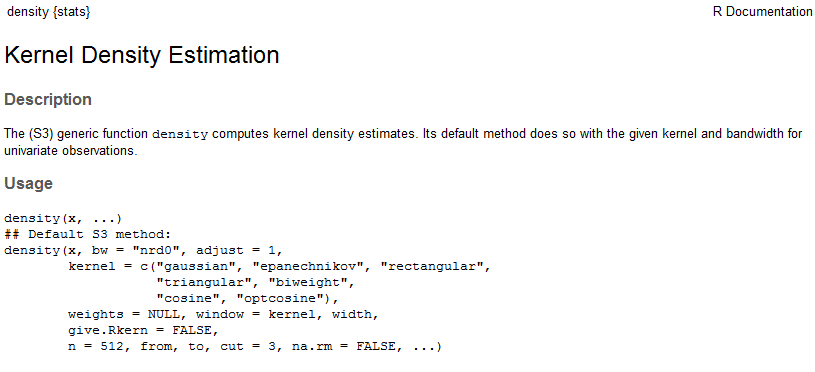
> hist(NutritionData$SatFat,breaks=20,freq=F,xlab="Saturated Fat", main="Histogram of Saturated Fat",col='gray')

> lines(density(NutritionData$SatFat))



Once again, several optional arguments exist that can be used to modify the resulting density smoother.

> help(density)



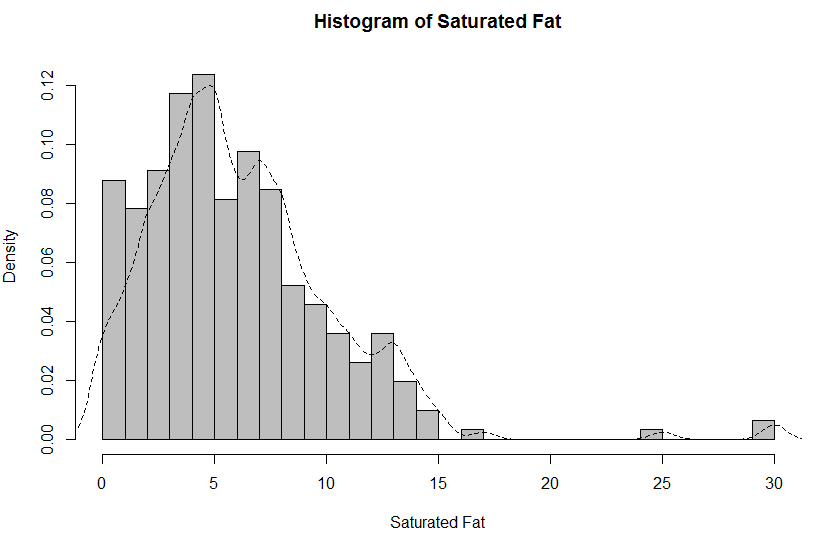
> help(lines)



For example, we can modify the appearance of the histogram/density smoother as follows:

> hist(NutritionData$SatFat,breaks=40,freq=F,xlab="Saturated Fat", main="Histogram of Saturated Fat", col='gray')

> lines(density(NutritionData$SatFat,adjust=0.50),lty=2)



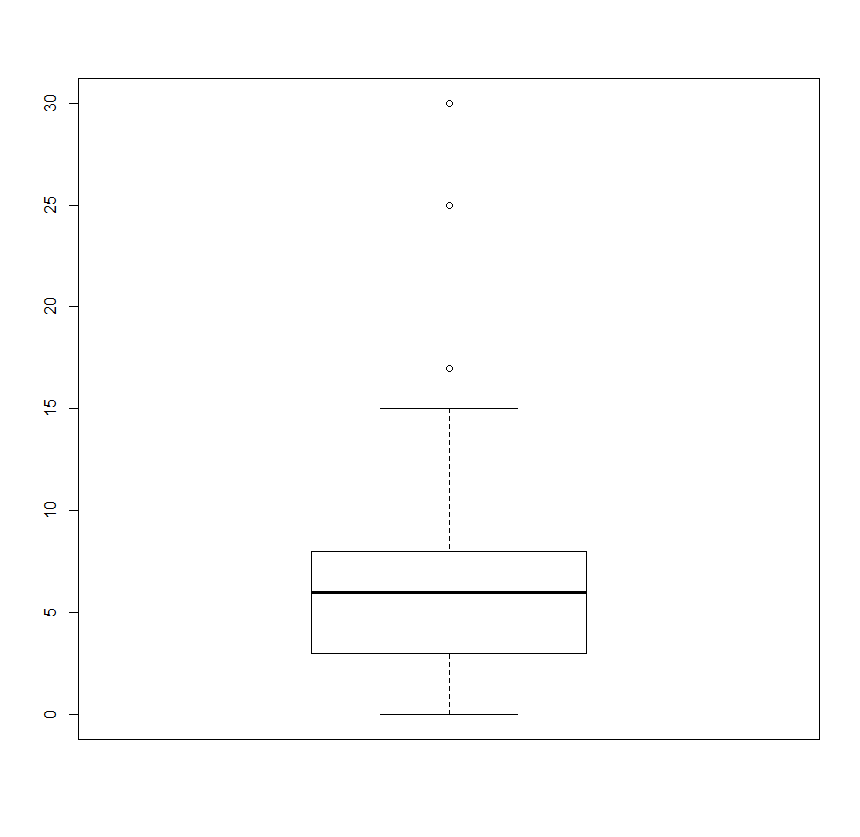
Questions:

1. Change the adjust= option to a few different values. What changes?
2. Change the lty= option to 5 and then to “dotted”. What changes?

**Boxplots in R**

The most basic form of the boxplot() function is employed below.

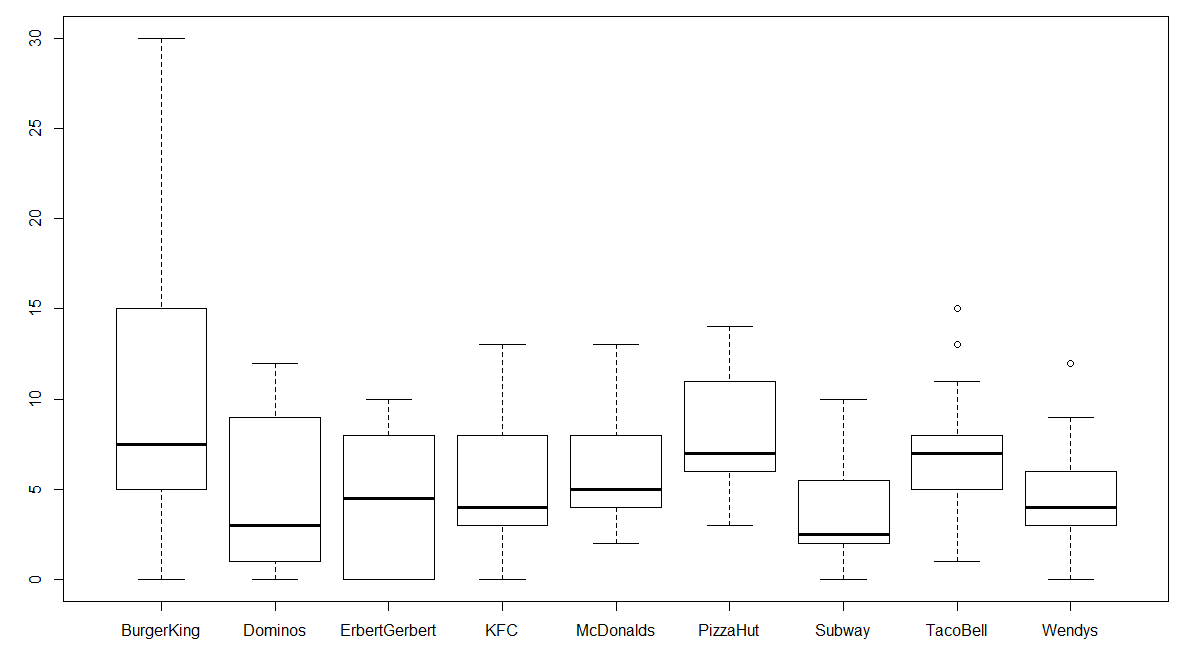
> boxplot(NutritionData$SatFat)



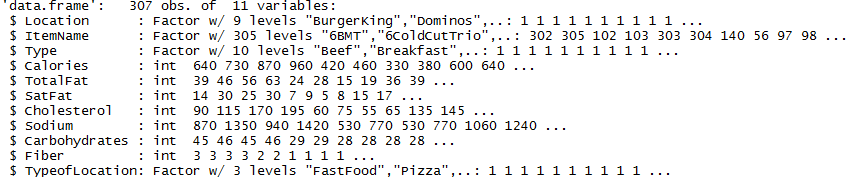
You can learn more about the optional arguments from the help documentation.

> help(boxplot)

The following can be used to obtain boxplots for Saturated Fat for each restaurant location.

> boxplot(NutritionData$SatFat ~ NutritionData$Location)  


The order in which the restaurants are plotted is determine by the factor levels. Most often these are alphabetical in order.



To existing order can be identified using the levels() function.

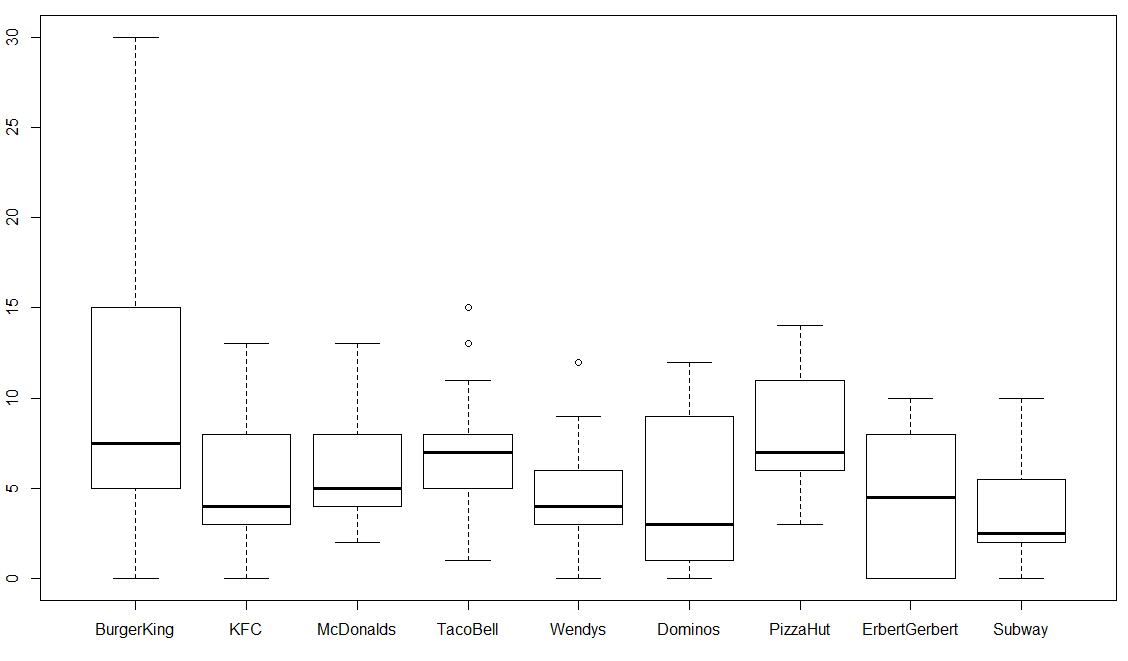
> levels(NutritionData$Location) 

A new vector for the factor level settings can be used to reassign levels. The following will reorder the levels so that all the Fast Food locations are listed first, the Pizza locations second, followed by the Subs locations.



> #Change order of factor levels

> NutritionData$Location<-factor(NutritionData$Location,levels=c("BurgerKing","KFC","McDonalds","TacoBell","Wendys","Dominos","PizzaHut","ErbertGerbert","Subway"))



**Bar Charts / Pie Charts in R**

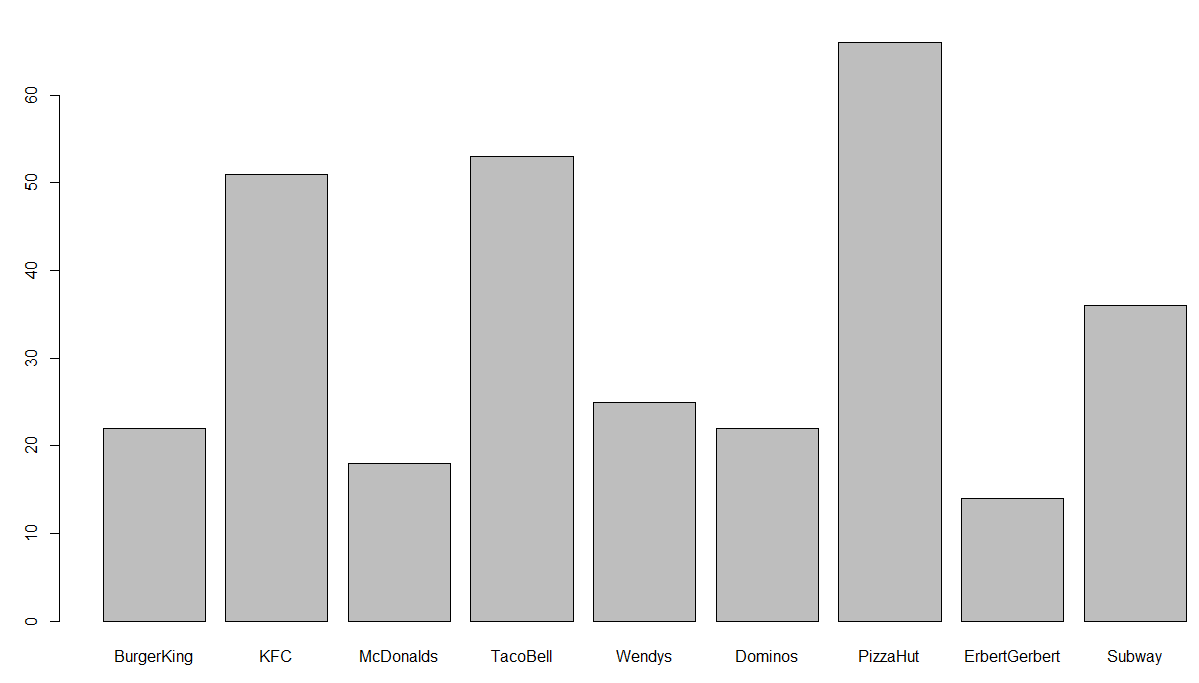
Barchart are commonly used to display counts from categorical data. The table() function is used to obtain counts in R.

> table(NutritionData$Location)



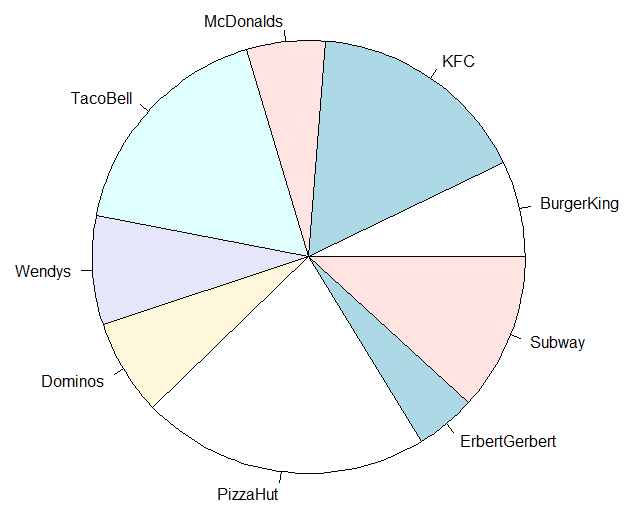
We can pass the output from the table() function directly in the barplot() function.

> barplot(table(NutritionData$Location))



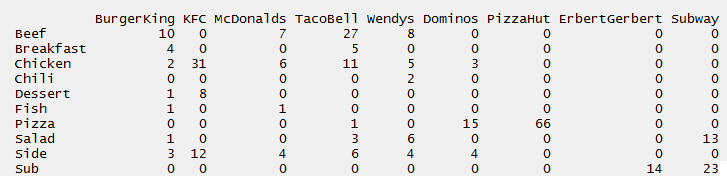
A pie chart can be obtained in a similar way.

> pie(table(NutritionData$Location))



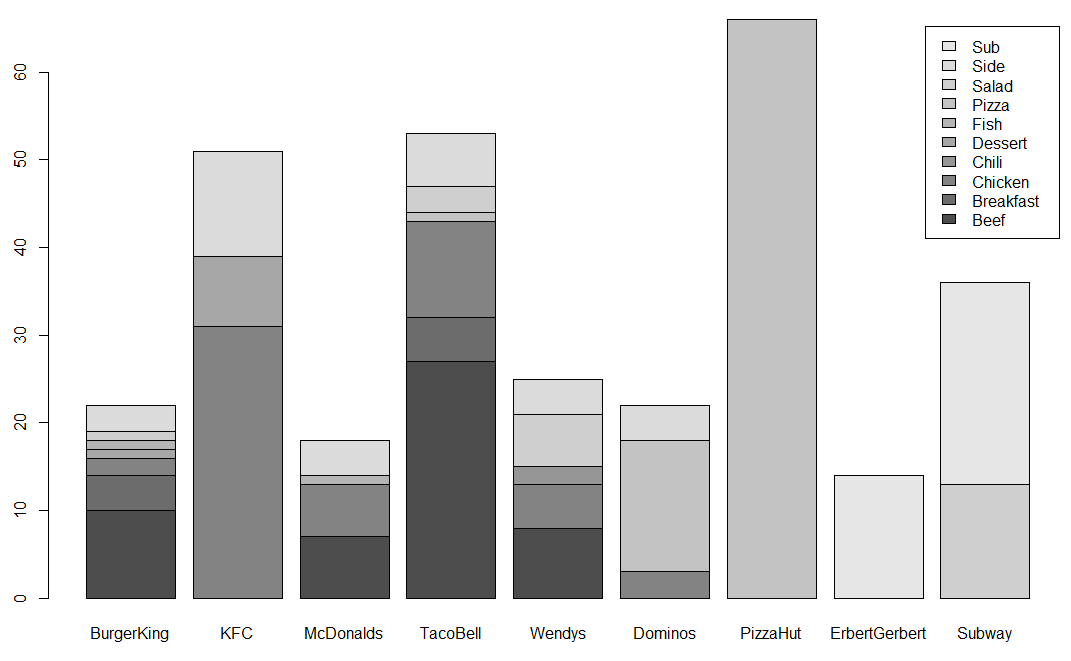
Plots for two-dimensional tables are

> table(NutritionData$Type, NutritionData$Location)



Getting a stacked barchart for this table. A more appropriate 100% stacked barchart is provided below.

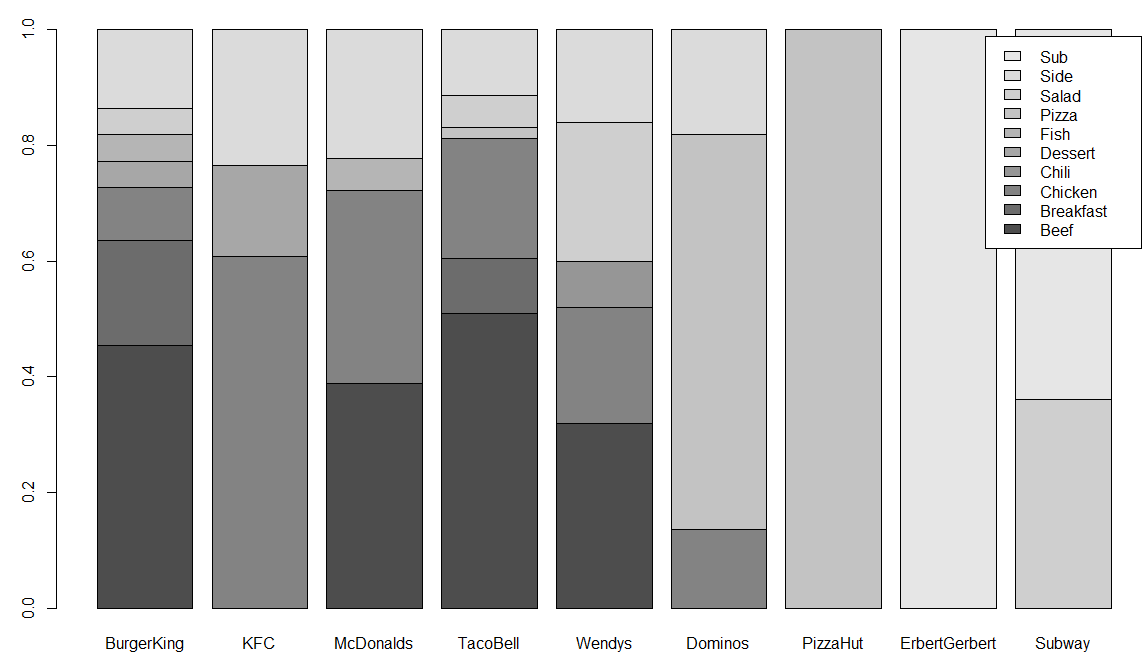
> barplot(table(NutritionData$Type, NutritionData$Location), legend=TRUE)



A 100% stacked barchart can be obtained as follows.

> table.of.counts<-table(NutritionData$Type,NutritionData$Location)

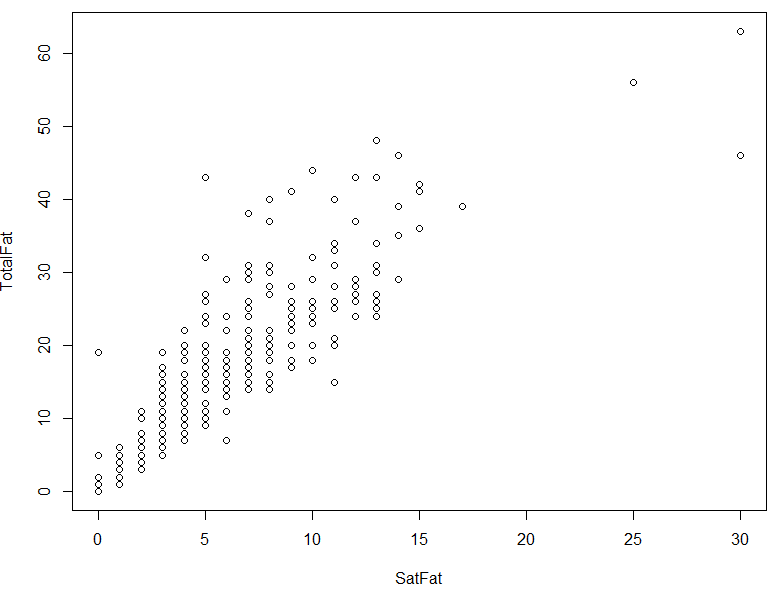
> barplot(prop.table(table.of.counts,2),legend=TRUE)



**Scatterplots and Smoothers**

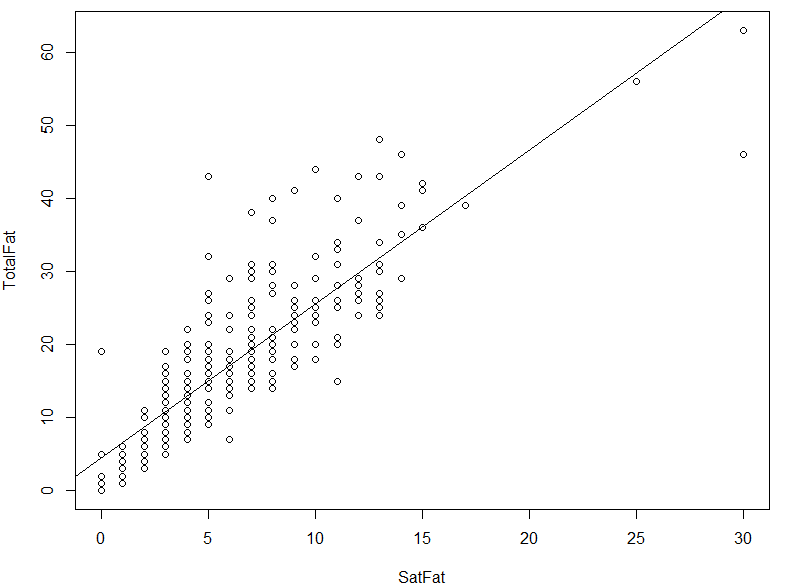
Scatterplots are simple to create in R using the plot() function. For example, we could examine the relationship between *Saturated Fat* and *Total Fat* by creating the following plot.

> plot(TotalFat~SatFat,data=NutritionData)



To add a trend line (i.e., the regression line) to this plot, you can use the abline() and lm() function.

> abline(lm(TotalFat~SatFat,data=NutritionData))



Loess smoothers are a common alternative to regression models in data science. Loess models are more flexible than regression-type smoothers. There are advantages and disadvantages to all models.

Loess Regression Wiki page: <https://en.wikipedia.org/wiki/Local_regression>



# The loess() function in R can be used to obtain a loess smoother in R.

> #Creating the intial plot

> plot(TotalFat~SatFat,data=NutritionData)

>

> #Fitting a loess model

> fit<-loess(TotalFat~SatFat,data=NutritionData)

>

> #Using a generic sequence for x to obtain a prediction line

> #lines(seq(0,30,1), predict(fit,newdata=data.frame(SatFat=seq(0,30,1))))

>

> #Using actual SatFat data valus to obtain a prediction

> lines(sort(NutritionData$SatFat), predict(fit,newdata=NutritionData[order(NutritionData$SatFat),]))

The span= option within the loess() function controls the amount of smoothing being done. The lower the span value, the less smoothing that is done.

> fit<-loess(TotalFat~SatFat,data=NutritionData, span=.4)

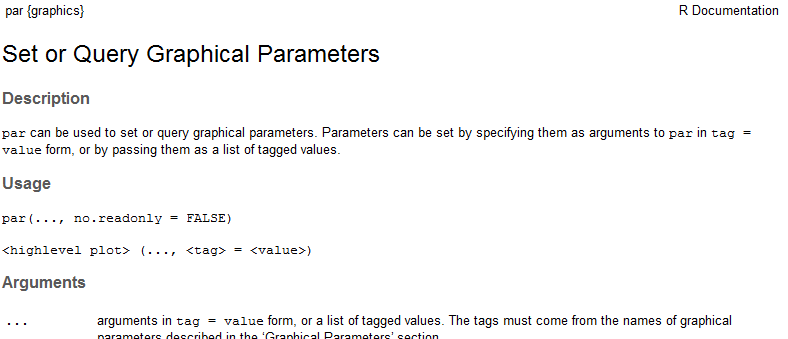
|  |  |
| --- | --- |
| Span = 0.75 (default setting) | Span = 0.3 |

**(Optional) Graphical Parameters**

> par(mfrow=c(1,2)) #Create a 1 x 2 grid for plotting

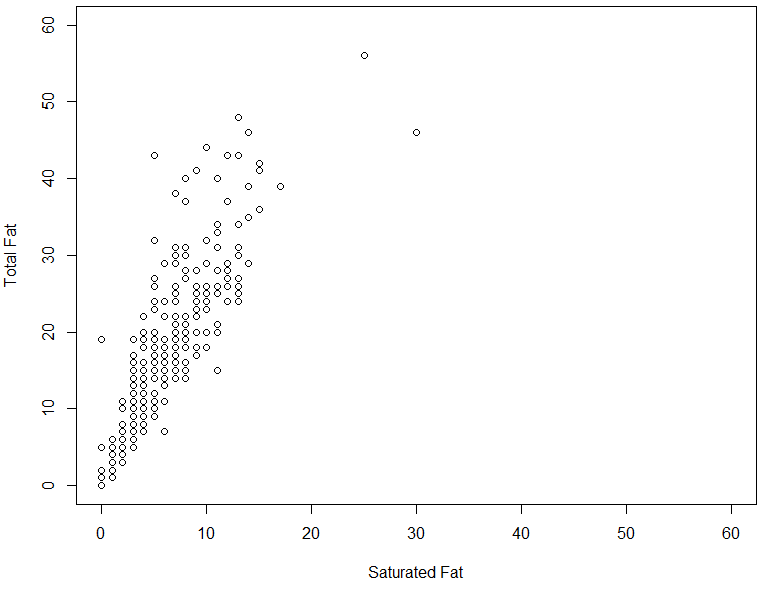
> par(pty="s") #Force square plotting region, par(pty="m") will change back to default

>help(par) # for others – several to choose from



The following command will change the range on the x- and y-axis from the defaults.

> plot(TotalFat~SatFat,data=NutritionData, xlab="Saturated Fat", ylab="Total Fat", xlim=c(0,60), ylim=c(0,60))

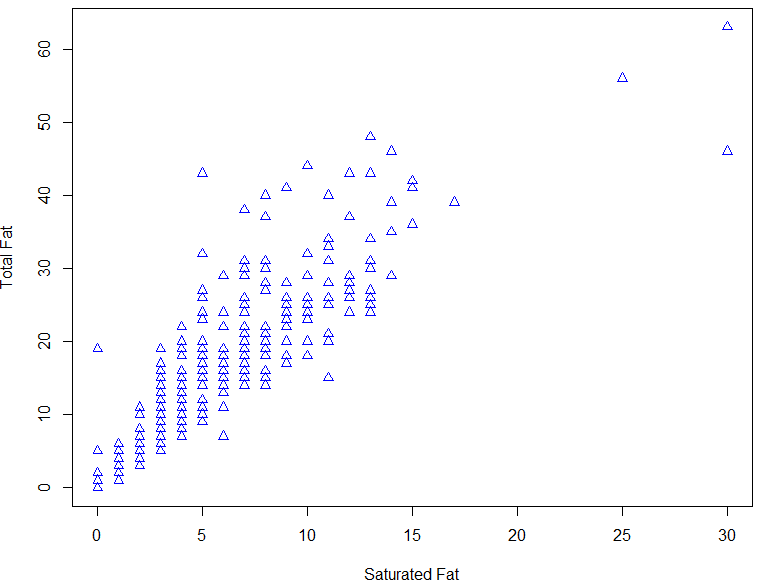


Options can be specified for the points as well. The col= option can be used to change the color and the pch= option specifies the plotting character. The list of plotting characters is provided here.



To following change the color to blue and uses plotting character 2, i.e. a triangle.

> plot(TotalFat~SatFat,data=NutritionData, xlab="Saturated Fat", ylab="Total Fat", col="blue",pch=2)



For the next set of plots, we will create a subset of the NutritionData

> BeefChickenData <- NutritionData[NutritionData$Type %in% c("Beef","Chicken"), ]

|  |  |
| --- | --- |
|  | Consider the following subset for next plot… |

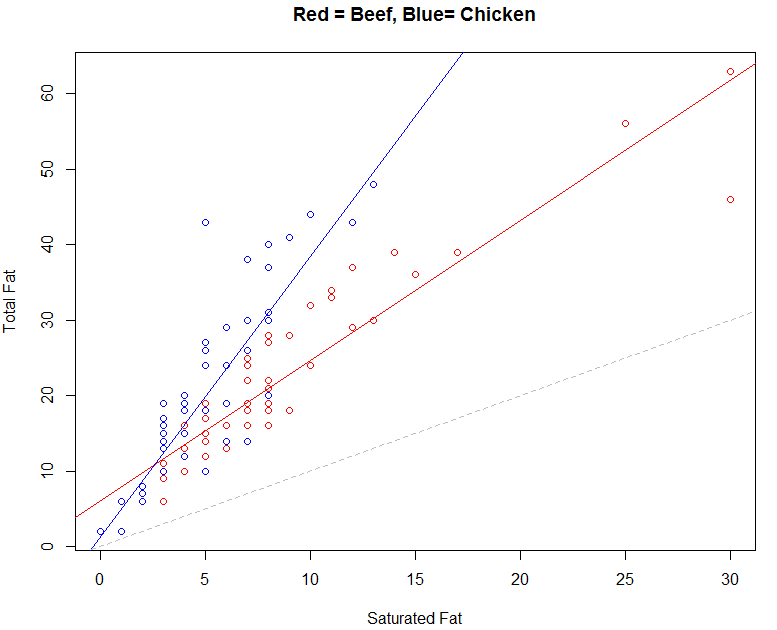
The following sequence of commands will create the plot below.

> plot(TotalFat~SatFat,data=BeefChickenData,xlab="Saturated Fat", ylab="Total Fat", col=c("red","blue")[match(Type,c("Beef","Chicken"))])

> abline(lm(TotalFat~SatFat,data=BeefChickenData[BeefChickenData$Type=="Beef",]),col="red")

> abline(lm(TotalFat~SatFat,data=BeefChickenData[BeefChickenData$Type=="Chicken",]),col="blue")

> abline(0,1,col="gray",lty=2)



A legend can be added to this plot using the legend() function.

> legend("bottomright",legend=c("Beef","Chicken"),fill=c("red","blue"))

The following code is used to add loess smoothers to the scatterplot.

> #Plots with Loess smoothers

> plot(TotalFat~SatFat,data=BeefChickenData,xlab="Saturated Fat", ylab="Total Fat",

+ col=c("red","blue")[match(Type,c("Beef","Chicken"))])

>

> fit.beef<-loess(TotalFat~SatFat,data=BeefChickenData[BeefChickenData$Type=="Beef",])

> lines(sort(BeefChickenData[BeefChickenData$Type == "Beef",6]), sort(predict(fit.beef,newdata=BeefChickenData[BeefChickenData$Type == "Beef",])), col="red")

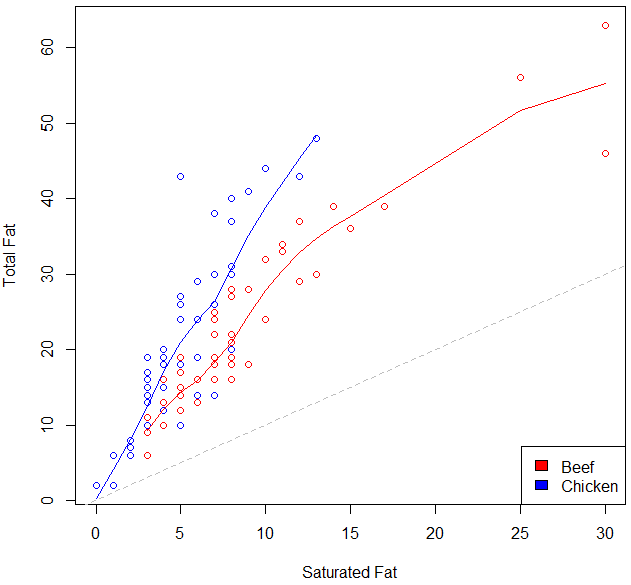
>

> fit.chicken<-loess(TotalFat~SatFat,data=BeefChickenData[BeefChickenData$Type=="Chicken",])

> lines(sort(BeefChickenData[BeefChickenData$Type == "Chicken",6]), sort(predict(fit.chicken,newdata=BeefChickenData[BeefChickenData$Type == "Chicken",])), col="blue")

> abline(0,1,col="gray",lty=2)

> legend("bottomright",legend=c("Beef","Chicken"),fill=c("red","blue"))



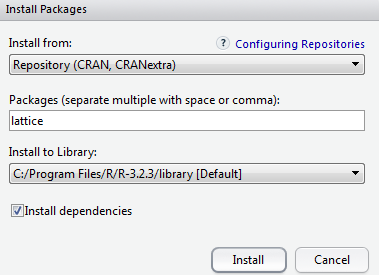
The type=”n” option in the plot() function prevents the points from being plotted.

> plot(TotalFat~SatFat,data=BeefChickenData,xlab="Saturated Fat", ylab="Total Fat",

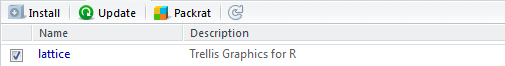
+ col=c("red","blue")[match(Type,c("Beef","Chicken"))], type="n")

**Advanced Plotting**

The lattice package is a powerful graphical package available in R. The most cutting-edge package for plotting is ggplot2. The ggplot2 package will not be covered in this handout.



To load this (or any other) package in R, go to the lower right-hand window of the R Studio window. You can search for the package of interest.



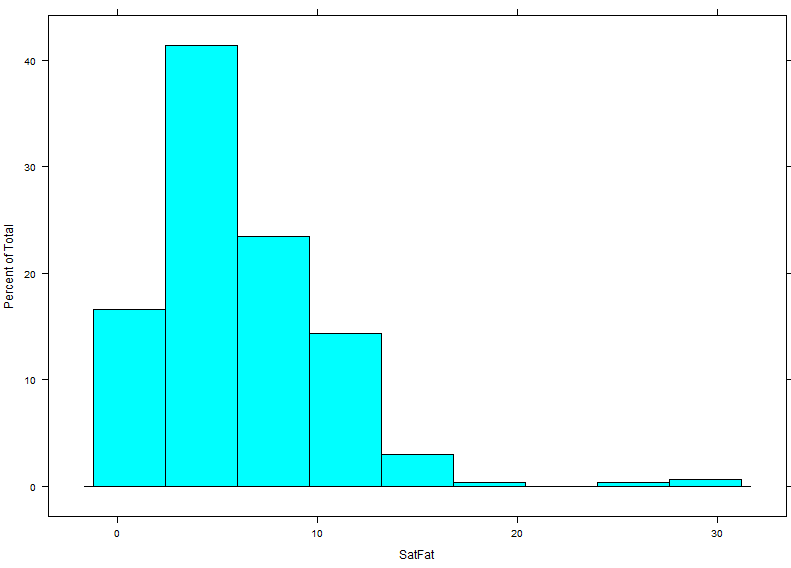
Here, you can check the box next to “lattice” and note that R automatically runs the following command.

> library(lattice)

**Obtaining a Histogram Using the Lattice Package**

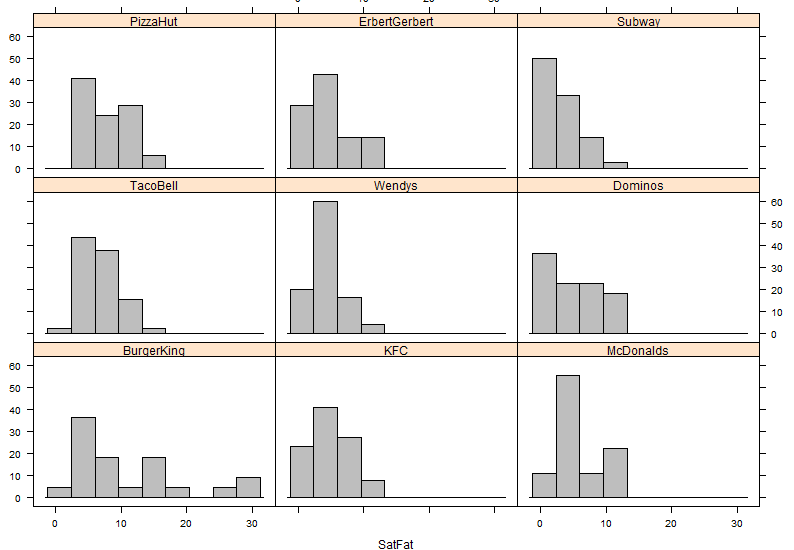
You can use the histogram() function once the lattice package has been installed:

> histogram( ~ SatFat, data=NutritionData)



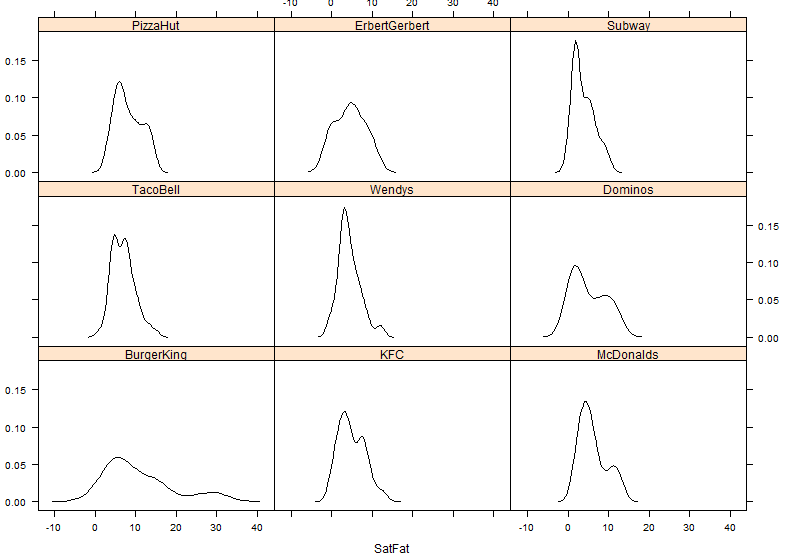
A more interesting display would compare the distribution of Saturated Fat across Location. This is easily implemented with lattice graphics.

> histogram(~SatFat|Location,data=NutritionData, col="gray")



**Obtaining a Density Plot Using the Lattice Package**

> densityplot(~SatFat|Location,data=NutritionData, col="black", plot.points=FALSE)



Question: Re-submit the above command with the plot.points argument omitted. What happens?

Note that instead of displaying the density plots in a separate panels for each location, R can alternatively overlay the density plots as following by way of the groups= option.

> densityplot(~SatFat, data=NutritionData, groups=Location, plot.points=FALSE, auto.key=TRUE)

|  |  |
| --- | --- |
|  |  |

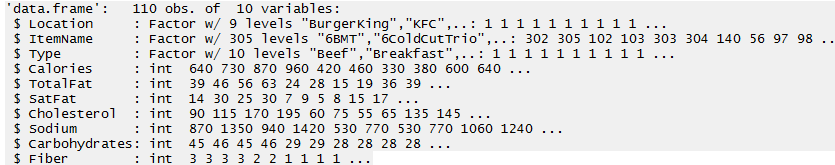
An example with more clarity is given for the Beef and Chicken subset datasets.

> densityplot(~SatFat, data=BeefChickenData, groups=Type, plot.points=FALSE, auto.key=TRUE)

|  |  |
| --- | --- |
|  |  |

Notice that the legend is incorrect and appears to be using the factor levels from the NutritionData and not the Beef and Chicken subset.

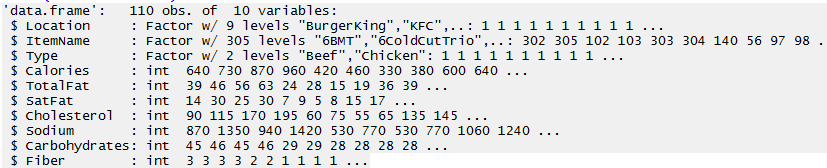
> str(BeefChickenData)



Update the factor level settings for Type in the Beef and Chicken subset.

> BeefChickenData$Type<-factor(BeefChickenData$Type,levels=c("Beef","Chicken"))

> str(BeefChickenData)



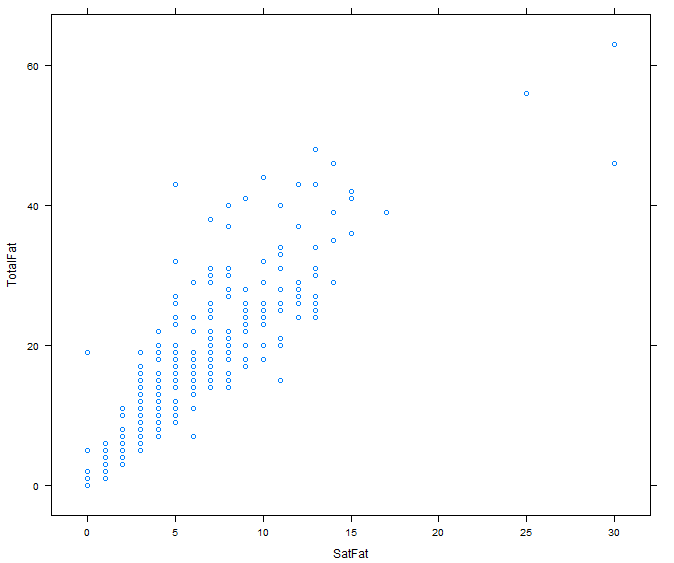
The legend now correctly corresponds with the plot.

> densityplot(~SatFat, data=BeefChickenData, groups=Type, plot.points=FALSE, auto.key=TRUE)

|  |  |
| --- | --- |
|  |  |

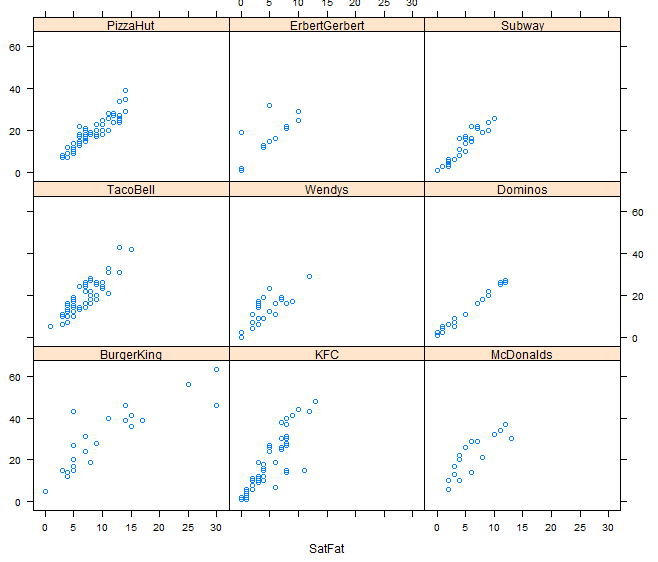
**Obtaining a Scatterplot Using Lattice**

> xyplot(TotalFat~SatFat, data=NutritionData)

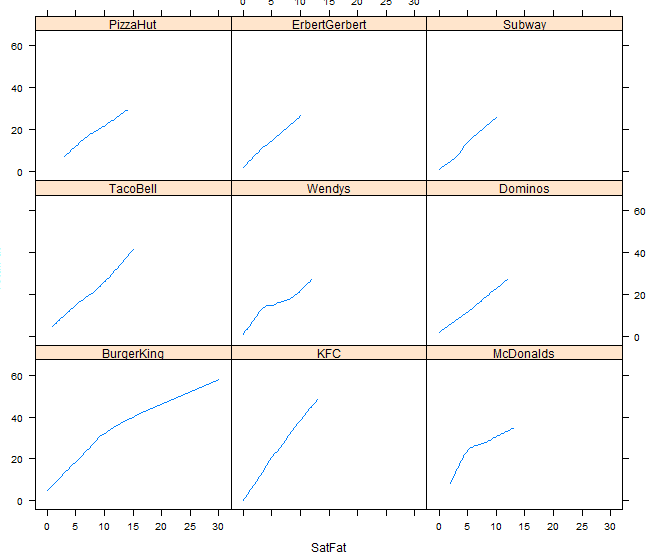


Next, note that you can also obtain the scatterplot above for each location fairly easily using the built-in conditioning functionality provided by the lattice package.

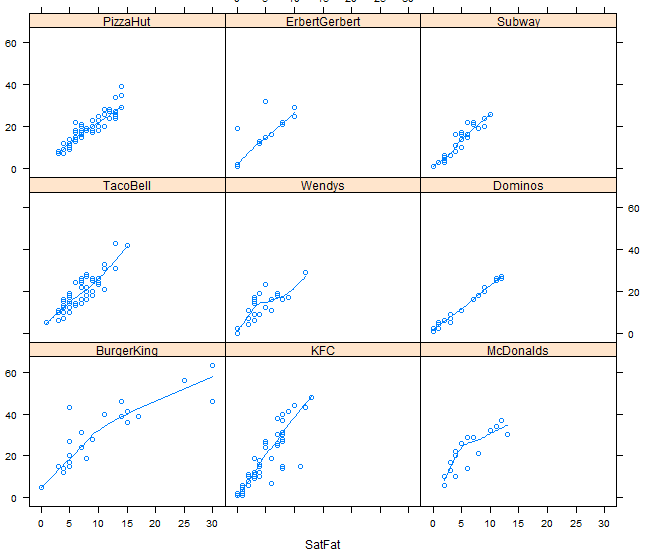
> xyplot(TotalFat~SatFat | Location, data=NutritionData)



> xyplot(TotalFat~SatFat|Location,data=NutritionData, type="smooth")



> xyplot(TotalFat~SatFat|Location,data=NutritionData, type=c("p","smooth"))



Finally, a completely custom panel function can be specified. The following custom panel function will use plotting character 2, the size of the points will be reduced using cex, the loess fit will be blue and thicker than usual through the use of the lwd option.

> panel.smoother <- function(x, y) {

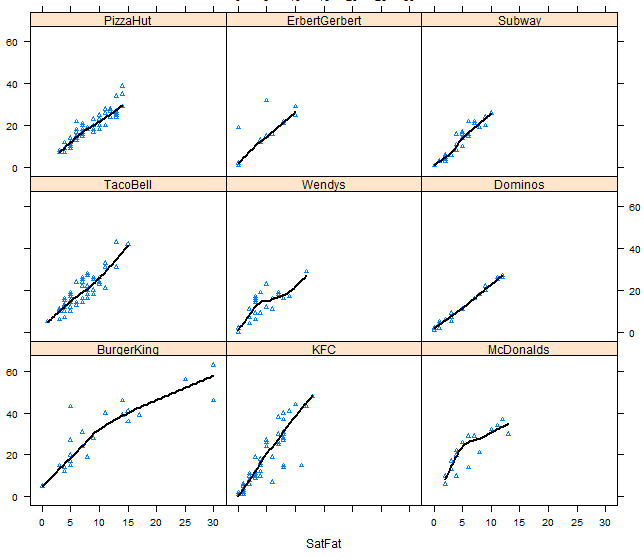
+ panel.xyplot(x, y, pch=2, cex=0.5) # show points

+ panel.loess(x, y, col="black", lwd=2) # show smoothed line

+ }

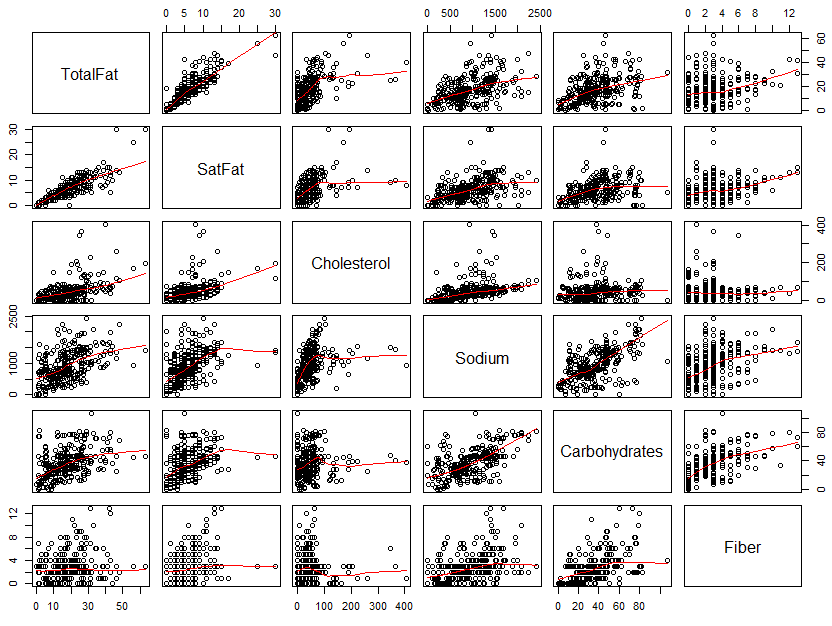
This custom function is then called with the panel= option in the xyplot.

> xyplot(TotalFat~SatFat|Location,data=NutritionData, panel = panel.smoother)

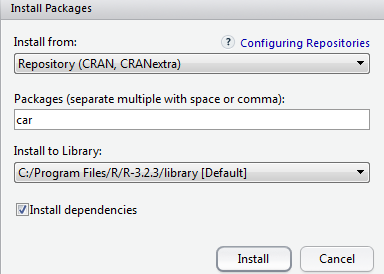


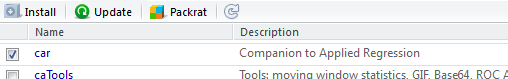
**Obtaining Scatterplot Matrices without Lattice Package**

> pairs(NutritionData[,5:10], panel=panel.smooth)



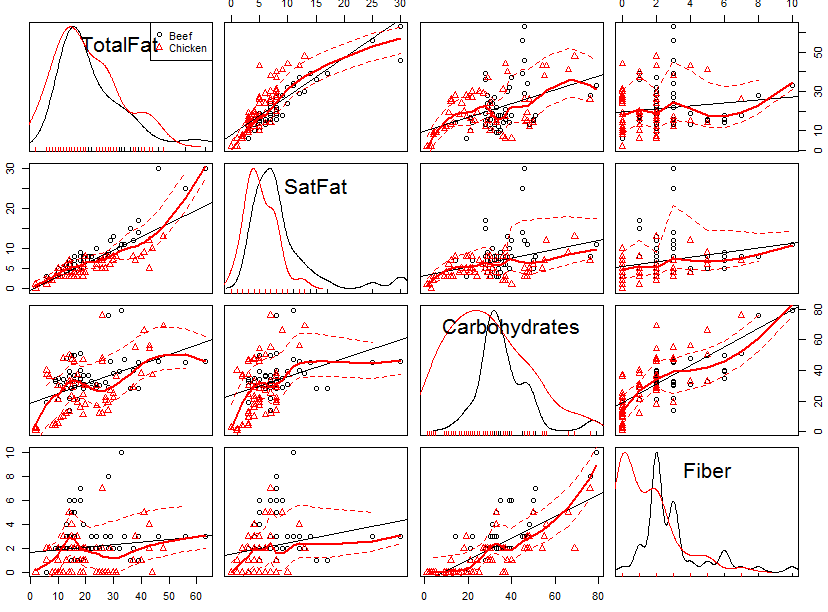
There is a function that accompanies the “An R Companion to Applied Regression.” This package is called car. Install this package and load it contents into R.





After the car package is downloaded and installed, the scatterplotMatrix() function can be used to create a scatterplot matrix with loess smoothers in each panel. The syntax of this function is akin to the lattice package.

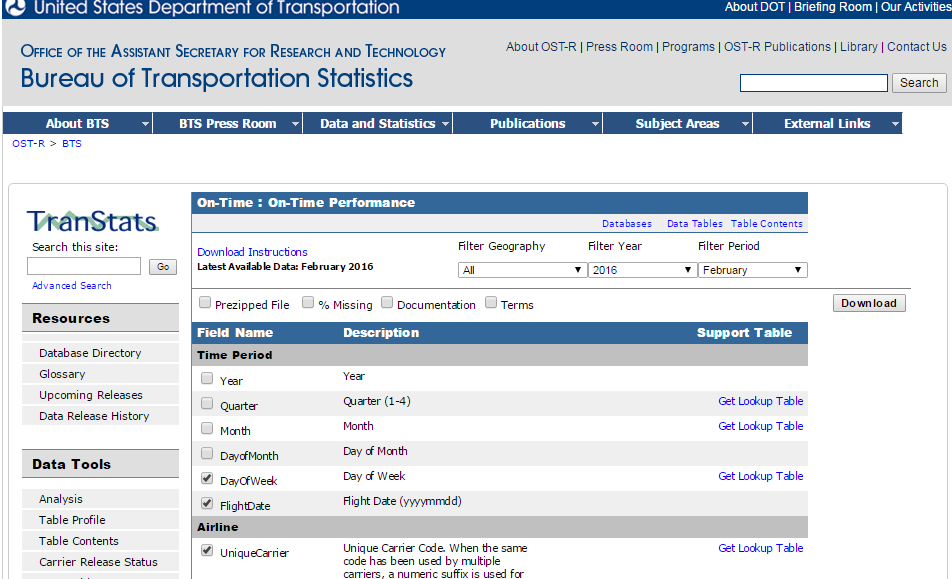
> scatterplotMatrix( ~ (TotalFat+SatFat+Carbohydrates+Fiber)|Type, data=BeefChickenData)



**Task**

For this task, we will consider the airline data. Of interest is the rate in which flights are cancelled or delayed. This data can be downloaded from the following site:

<http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time>



You must specify which Field Names to include. The following fields should be downloaded.

|  |  |  |
| --- | --- | --- |
| **Fields/Variables to download** | | |
| * Day of Week * FlightDate * UniqueCarrier * Origin * Dest | * CRSDepTime * DepTime * DepDelay * CRSArrTime * ArrTime * ArrDelay | * Cancelled * CarrierDelay * WeatherDelay * NASDelay * SecurityDelay * LateAircraftDelay |

Questions

1. Create a barplot to show the cancellation rate of flights by day of week. Which day of the week has the largest percentage of flights?
2. Create a barplot to show the cancellation rate of flights by carrier, i.e. airline. Which carrier appears to cancel the largest percentage of flights?

For the following you may want to create a subset of the full dataset. You can create a subset however you’d like, e.g. narrow your focus to just Chicago O’Hare – Code: ORD. The following can be used to create a random subset of 10000 observations from a data.frame named Delay.

> mysubset<-Delay[sample(1:dim(Delay)[1],size=10000),]

1. Use a command similar to this to obtain the average delay across all flights for Carrier delay, weather delay, NAS delay, security delay, and late aircraft delay. These columns are contained within columns 13:17 for my data.frame.

> apply(Delay[,13:17],2,function(x){mean(x,na.rm=TRUE)})

Which type of delay is the longest on average? Why is the na.rm required for the apply function to work?

1. Create some type of plot for the averages from above. What type of plot did you decide to create?
2. Consider the above problem, which type of delay has the most amount of variation, i.e. largest standard deviation?
3. Your friend insists on flying early in the day to avoid delays. Is there any truth that flights earlier in the day are less likely be delayed? Create a plot to communicate your findings.
4. How often does a flight leave early and arrive early? How often does a flight leave late, but arrive early? Hint: You will likely need to create new variables for this and then use the table() function on your new variables.