R Notebook: Descriptive Analysis

Winter 2024

Contents

This tutorial tackles the first marketing analytics problem: summarizing and exploring a data set with descriptive statistics (mean, standard deviation, and so forth) and visualization methods.

We will use below packages:

- psych
- car
- gpairs
- grid
- lattice
- corrplot
- gplots

1 Data

```
store.df <- read.csv("Data_Descriptive.csv", stringsAsFactors=TRUE)
str(store.df)
```

```
## 'data.frame':    2080 obs. of  10 variables:
##  $ storeNum: int  101 101 101 101 101 101 101 101 101 101 ...
##  $ Year    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Week    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ p1sales : int  127 137 156 117 138 115 116 106 116 145 ...
##  $ p2sales : int  106 105 97 106 100 127 90 126 94 91 ...
##  $ p1price : num  2.29 2.49 2.99 2.99 2.49 2.79 2.99 2.99 2.29 2.49 ...
##  $ p2price : num  2.29 2.49 2.99 3.19 2.59 2.49 3.19 2.29 2.29 2.99 ...
##  $ p1prom  : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ p2prom  : int  0 0 0 0 1 0 0 0 0 0 ...
##  $ country : Factor w/ 7 levels "AU","BR","CN",..: 7 7 7 7 7 7 7 7 7 7 ...
```

The *str()* command shows that the *store.df* is a "data.frame".

2 Summarize a single variable

2.1 Discrete variables

Frequency counts

```
table(store.df$p1price)
```

```
## 
## 2.19 2.29 2.49 2.79 2.99 
##  395  444  423  443  375
```

One of the most useful features of R is that most functions produce an object that you can save and use for further commands.

```
p1.table <- table(store.df$p1price)
p1.table
```

```
## 
## 2.19 2.29 2.49 2.79 2.99 
##  395  444  423  443  375
```

```
str(p1.table)
```

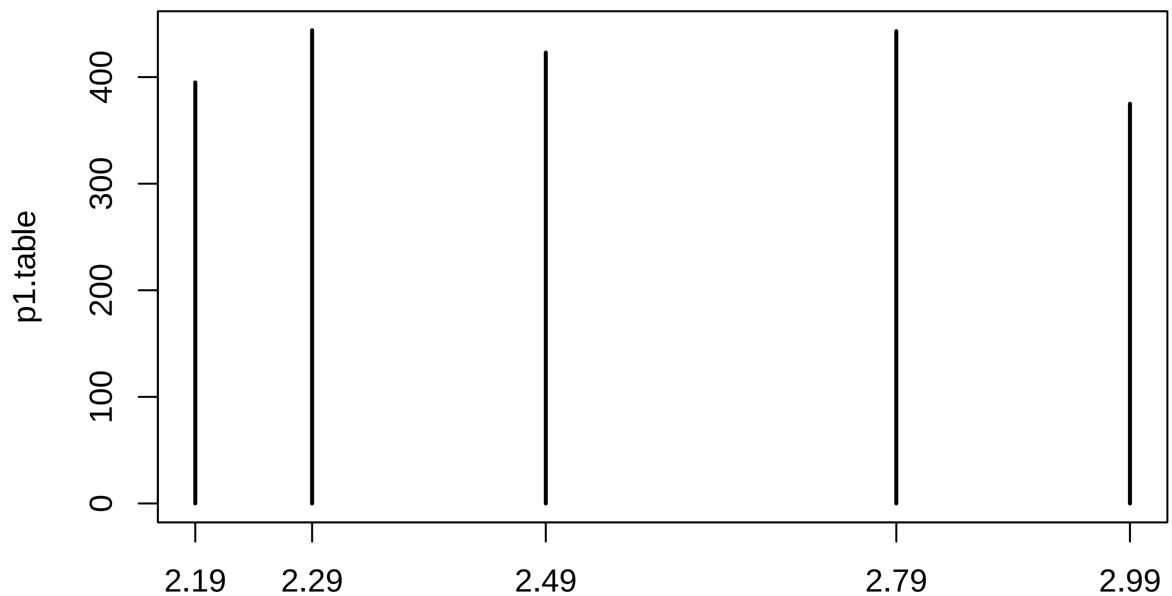
```
##  'table' int [1:5(1d)] 395 444 423 443 375
```

```
##  - attr(*, "dimnames")=List of 1
##   ..$ : chr [1:5] "2.19" "2.29" "2.49" "2.79" ...
```

The *str()* command shows us that the object produced by *table()* is a special type called *table*. You will find many functions in R that produce objects of special types.

You can easily pass *p1.table* to the *plot()* function to produce a quick plot:

```
plot(p1.table)
```



R chose a type of plot suitable for our *table* object, but it is fairly unattractive and the labels could be clearer. We will show how to modify a plot to get better results soon.

An analyst might want to know how often each product was promoted at each price point. The *table()* command produces two-way *cross tabs* when a second variable is included.

```
table(store.df$p1price, store.df$p1prom)
```

```
##
##         0   1
##   2.19 354  41
##   2.29 398  46
##   2.49 381  42
##   2.79 396  47
##   2.99 343  32
```

We can compute the exact fraction of times product 1 on promotion at each price point. We assign the table to a variable and then divide the second column of the table by the sum of the first and second columns:

```r
p1.table2 <- table(store.df$p1price, store.df$p1prom)
p1.table2[ ,2] / (p1.table2[ ,1] + p1.table2[ ,2])
```

```
##       2.19       2.29       2.49       2.79       2.99
## 0.10379747 0.10360360 0.09929078 0.10609481 0.08533333
```

2.2 Continuous variables

Counts are useful when we have a small number of categories, but with continuous variables, it is more helpful to summarize the data in terms of its distribution. The most common way to do that is with mathematical functions that describe the **range** of the data, its **center**, the degree to which it is concentrated or **dispersed**, and the specific points that may be of interest (such as the 90th percentile):

Describe	Function	Value
Extremes	min(x)	Minimum value
	max(x)	Maximum value
Central tendency	mean(x)	Arithmetic mean
	median(x)	Median
Dispersion	var(x)	Variance around the mean
	sd(x)	Standard deviaion (*sqrt(var(x))*)
	IQR(x)	Interquartile range, 75th-25th percentile
	mad(x)	Median absolute deviation
Points	quantile(x, probs=c(. . .))	Percentiles

```r
min(store.df$p1sales)
```

```
## [1] 73
```

```r
max(store.df$p1sales)
```

```
## [1] 263
```

```r
mean(store.df$p1prom)
```

```
## [1] 0.1
```

```r
median(store.df$p2sales)
```

```
## [1] 96
```

```r
var(store.df$p1sales)
```

```
## [1] 805.0044
```

```r
sd(store.df$p1sales)
```

```
## [1] 28.3726
```

```r
IQR(store.df$p1sales)
```

```
## [1] 37
```

```r
mad(store.df$p1sales)
```

```
## [1] 26.6868
```

```r
quantile(store.df$p1sales, probs = c(0.25, 0.5, 0.75))
```

```
## 25% 50% 75%
## 113 129 150
```

```r
#the 25h, 50th (median), and 75th percentiles
```

For skewed and asymmetric distributions that are common in marketing, such as unit sales or household income, the arithmetic *mean()* and standard deviation *sd()* may be misleading; in those cases, the *median()* and the interquartile range *IQR()* (the range of the middle 50% of data) are often used to summarize a distribution.

```r
quantile(store.df$p1sales, probs = c(0.05, 0.95)) # central 90% data
```

```
##  5% 95%
##  93 184
```

```r
quantile(store.df$p1sales, probs = 0:10/10)
```

```
##    0%   10%   20%   30%   40%   50%   60%   70%   80%   90%  100%
##  73.0 100.0 109.0 117.0 122.6 129.0 136.0 145.0 156.0 171.0 263.0
```

The second example shows that we may use sequences in many places in R. In this case, we find every 10th percentile by creating a simple sequence of *0:10* and dividing by 10 to yield the vector *0, 0.1, 0.2, . . . 1.0*. You could also do it by using the sequence function (*seq(from=0, to=1, by=0.1)*). *0:10/10* is shorter and more commonly used.

Suppose we wanted a summary of the sales for product 1 and product 2 based on their median and interquartile range. We might assemble these summary statistics into a data frame that is easier to read than the one-line-at-a-time output above. We create a data frame to hold our summary statistics. We name the columns and rows and fill in the cells with function values:

```r
mysummary.df <- data.frame(matrix(NA, nrow=2, ncol=2)) # 2 by 2 empty matrix
names(mysummary.df) <- c("Median Sales", "IQR") # name columns
rownames(mysummary.df) <- c("Product 1", "Product 2") # name rows
mysummary.df["Product 1", "Median Sales"] <- median(store.df$p1sales)
mysummary.df["Product 2", "Median Sales"] <- median(store.df$p2sales)
mysummary.df["Product 1", "IQR"] <- IQR(store.df$p1sales)
mysummary.df["Product 2", "IQR"] <- IQR(store.df$p2sales)
mysummary.df
```

Such code might be a good candidate for a custom function you can reuse. We will see a shorter way to create this summary soon.

3 Summarize data frames

R provides a variety of ways to summarize data frames without writing extensive code.

3.1 summary()

```r
summary(store.df)
```

```
##     storeNum          Year          Week          p1sales       p2sales
##  Min.   :101.0   Min.   :1.0   Min.   : 1.00   Min.   : 73   Min.   : 51.0
##  1st Qu.:105.8   1st Qu.:1.0   1st Qu.:13.75   1st Qu.:113   1st Qu.: 84.0
##  Median :110.5   Median :1.5   Median :26.50   Median :129   Median : 96.0
##  Mean   :110.5   Mean   :1.5   Mean   :26.50   Mean   :133   Mean   :100.2
##  3rd Qu.:115.2   3rd Qu.:2.0   3rd Qu.:39.25   3rd Qu.:150   3rd Qu.:113.0
##  Max.   :120.0   Max.   :2.0   Max.   :52.00   Max.   :263   Max.   :225.0
```

```
##
##     p1price         p2price         p1prom         p2prom       country
##  Min.   :2.190   Min.   :2.29   Min.   :0.0   Min.   :0.0000   AU:104
##  1st Qu.:2.290   1st Qu.:2.49   1st Qu.:0.0   1st Qu.:0.0000   BR:208
##  Median :2.490   Median :2.59   Median :0.0   Median :0.0000   CN:208
##  Mean   :2.544   Mean   :2.70   Mean   :0.1   Mean   :0.1385   DE:520
##  3rd Qu.:2.790   3rd Qu.:2.99   3rd Qu.:0.0   3rd Qu.:0.0000   GB:312
##  Max.   :2.990   Max.   :3.19   Max.   :1.0   Max.   :1.0000   JP:416
##                                                                US:312
```

It works similarly for single vectors:

```
summary(store.df$Year)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     1.0     1.0     1.5     1.5     2.0     2.0
```

The *digit=* argument is helpful if you wish to change the precision of the display:

```
summary(store.df, digits = 2)
```

```
##     storeNum         Year          Week        p1sales       p2sales
##  Min.   :101   Min.   :1.0   Min.   : 1   Min.   : 73   Min.   : 51
##  1st Qu.:106   1st Qu.:1.0   1st Qu.:14   1st Qu.:113   1st Qu.: 84
##  Median :110   Median :1.5   Median :26   Median :129   Median : 96
##  Mean   :110   Mean   :1.5   Mean   :26   Mean   :133   Mean   :100
##  3rd Qu.:115   3rd Qu.:2.0   3rd Qu.:39   3rd Qu.:150   3rd Qu.:113
##  Max.   :120   Max.   :2.0   Max.   :52   Max.   :263   Max.   :225
##
##     p1price       p2price       p1prom        p2prom     country
##  Min.   :2.2   Min.   :2.3   Min.   :0.0   Min.   :0.00   AU:104
##  1st Qu.:2.3   1st Qu.:2.5   1st Qu.:0.0   1st Qu.:0.00   BR:208
##  Median :2.5   Median :2.6   Median :0.0   Median :0.00   CN:208
##  Mean   :2.5   Mean   :2.7   Mean   :0.1   Mean   :0.14   DE:520
##  3rd Qu.:2.8   3rd Qu.:3.0   3rd Qu.:0.0   3rd Qu.:0.00   GB:312
##  Max.   :3.0   Max.   :3.2   Max.   :1.0   Max.   :1.00   JP:416
##                                                           US:312
```

```
# digits=2 means two significant positions, not two decimal places
```

The most important use for *summary()* is: *after importing data, use summary() to do a quick quality check.* Check the *min* and *max* for outliers or miskeyed data, and check the *mean* and *median* are reasonable.

3.2 *describe()**

describe() from the *psych* package reports a variety of statistics for each variable in a data set, including *n*, the count of observations; *trimmed mean*, the mean after dropping a small proportion of extreme values; and statistics such as *skew* and *kurtosis* that are useful when interpreting data concerning to normal distributions.

```
library(psych) # install if needed
describe(store.df)
```

By comparing the trimmed mean to the overall mean, one might discover when outliers are skewing the mean with extreme values. *describe()* is especially recommended for summarizing survey data with discrete values such as 1-7 Likert scale items from the survey.

Note that there is a * next to the label of the *country*. This is a warning; *country* is a factor, and the summary may not make sense to them. You can select the variables (columns) that are numeric. For instance, if we wished to describe only columns 2 and 4 through 9:

```r
describe(store.df[,c(2, 4:9)])
```

Recommended approach to inspect data

We can now recommend a general approach to inspect a data set after importing it, replacing *my.data* and *DATA* with the names of your objects:

1. Import your data with *read.csv()*;
2. Convert it to a data frame if needed (*my.data <- data.frame(DATA)*), and set column names if needed (*names(my.data) <- c(...)*);
3. Examine *dim()* to check the data frame has the expected number of rows and columns;
4. Use *head(my.data)* and *tail(my.data)* to check the first few and last few rows; make sure the header rows at the beginning and blank rows at the end were not included accidentally.
5. Use *some()* from the *car* package to examine a few sets of random rows;
6. Check the data frame structure with *str()* to ensure the variable types and values are appropriate. Change the type of variables as necessary - especially to *factor* types;
7. Run *summary()* and look for unexpected values, especially *min* and *max* that are unexpected.
8. Load the *psych* library and examine basic descriptive with *describe()*. Reconfirm the observation counts by checking that *n* is the same for each variable, and check trimmed mean and skew (if relevant).

3.3 apply()

apply(x= DATA, MARGIN = MARGIN, FUN= FUNCTION) runs any functions that you specify on each of the rows and/or columns of an object. The term *margin* is a two-dimensional metaphor that denotes which "direction" you want to do something: either along the rows (*MARGIN=1*) or columns (*MARGIN=2*), or both simultaneously (*MARGIN= c(1,2)*).

Suppose we want to find the mean of every column of *store.df*, except for *store.df&Store*, which is not a number and does not have a mean:

```r
apply(store.df[ 2:9], MARGIN = 2, FUN = mean)
```

```
##        Year        Week     p1sales     p2sales     p1price     p2price 
##   1.5000000  26.5000000 133.0485577 100.1567308   2.5443750   2.6995192 
##      p1prom      p2prom 
##   0.1000000   0.1384615
```

```r
apply(store.df[ , 2:9], 2, sum)
```

```
##     Year     Week  p1sales  p2sales  p1price  p2price   p1prom   p2prom 
##   3120.0  55120.0 276741.0 208326.0   5292.3   5615.0    208.0    288.0
```

```r
apply(store.df[ , 2:9], 2, sd)
```

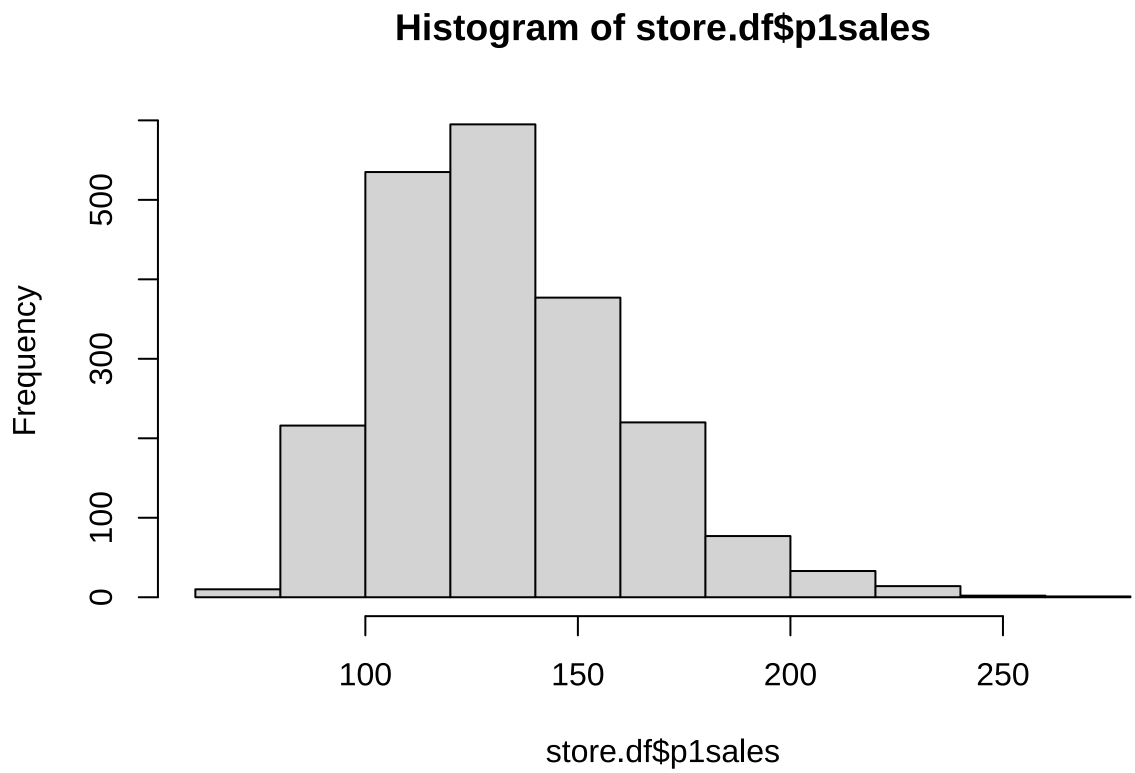
```
##       Year       Week    p1sales    p2sales    p1price    p2price     p1prom 
##  0.5001202 15.0119401 28.3725990 24.4241905  0.2948819  0.3292181  0.3000721 
##     p2prom 
##  0.3454668
```

4 Single variable visualisation

4.1 Histograms

A fundamental plot for a single continuous variable is *histogram*.

```r
hist(store.df$p1sales)
```

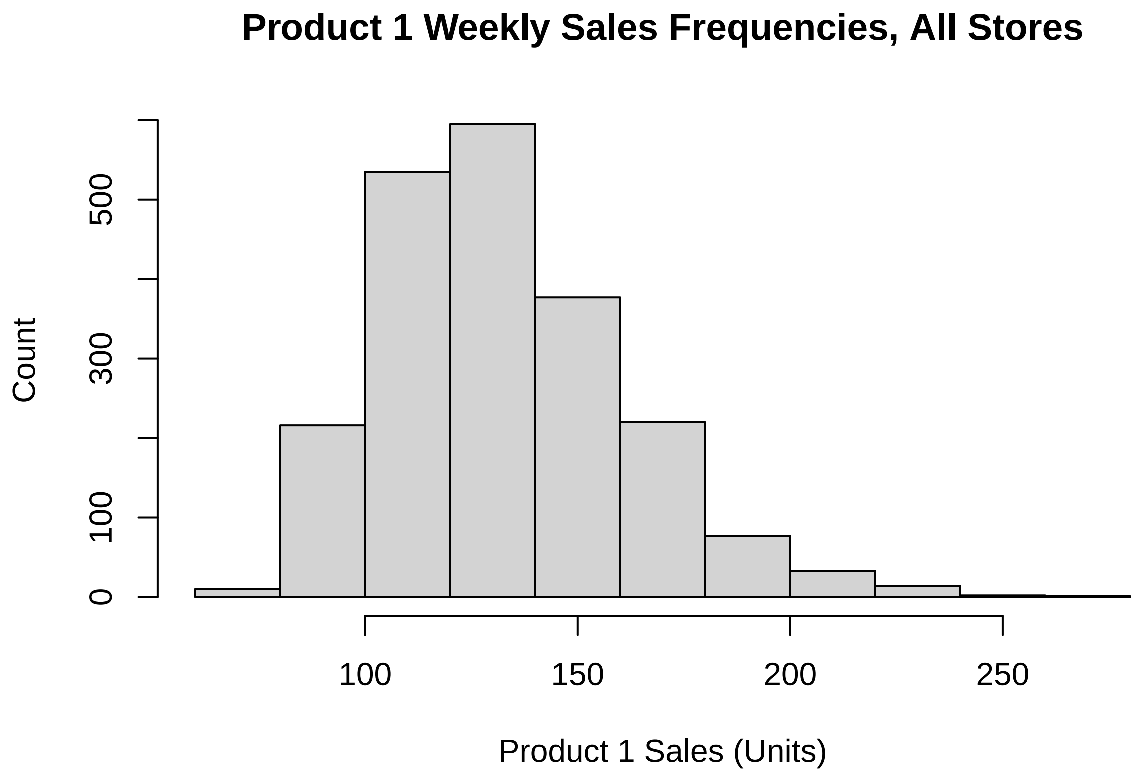


There are four things you should understand about graphics in R:

- R graphics are produced through commands that often seem tedious and require trial and iteration.
- Always use a text editor when working on plot commands; they rapidly become too long to type, and you will often want to try slight variants and copy and paste them for reuse.
- Despite the difficulties, R graphics can be high quality, portable in format, and even beautiful.
- Once you have a code for a useful graphic, you can reuse it with new data. It is often helpful to tinker with previous plotting code when building a new plot rather than recreating it.

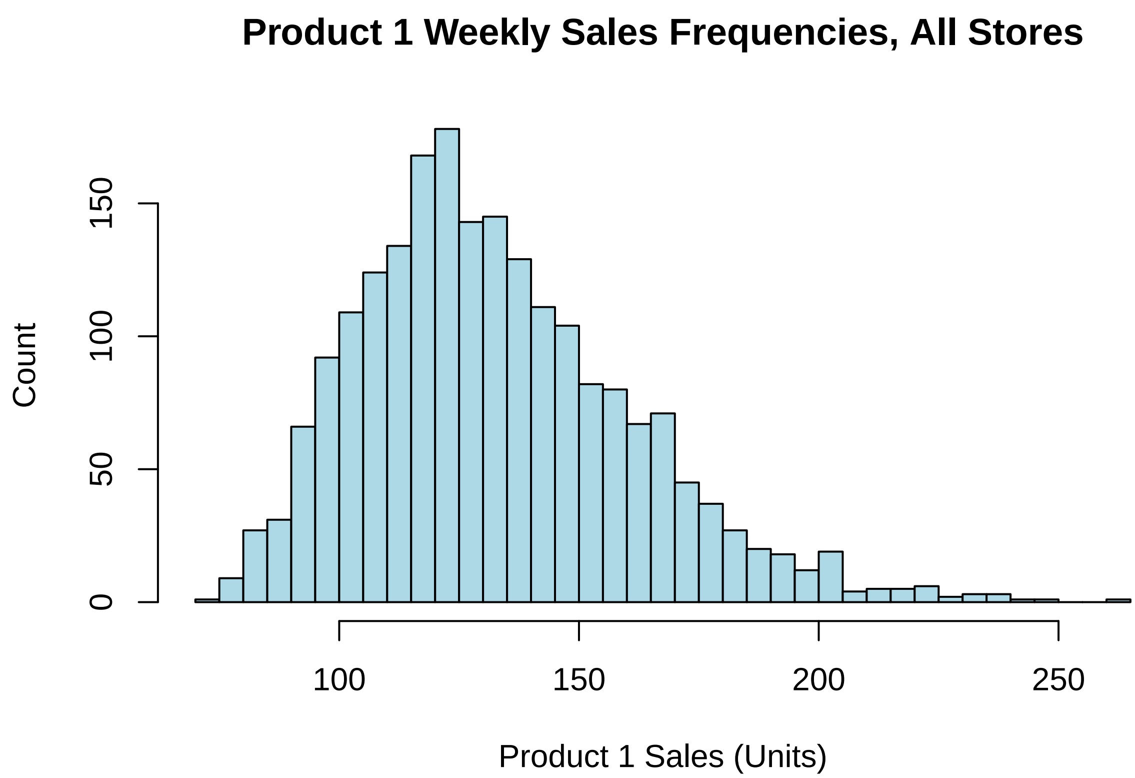
In our case, we can add the title and axis labels to our plot command:

```
hist(store.df$p1sales,
     main = "Product 1 Weekly Sales Frequencies, All Stores",
     xlab = "Product 1 Sales (Units)",
     ylab = "Count")
```



We can have more granularity (more bars) and color the histogram bars. R knows many colours by name, including the most common ones in English (“res”, “blue”, “green”, etc.) and less common ones (e.g., “coral”, and “burlywood”). Many of these can be modified by adding the prefix “light” or “dark”. For a list of built-in colour names, run the “colors()* command.

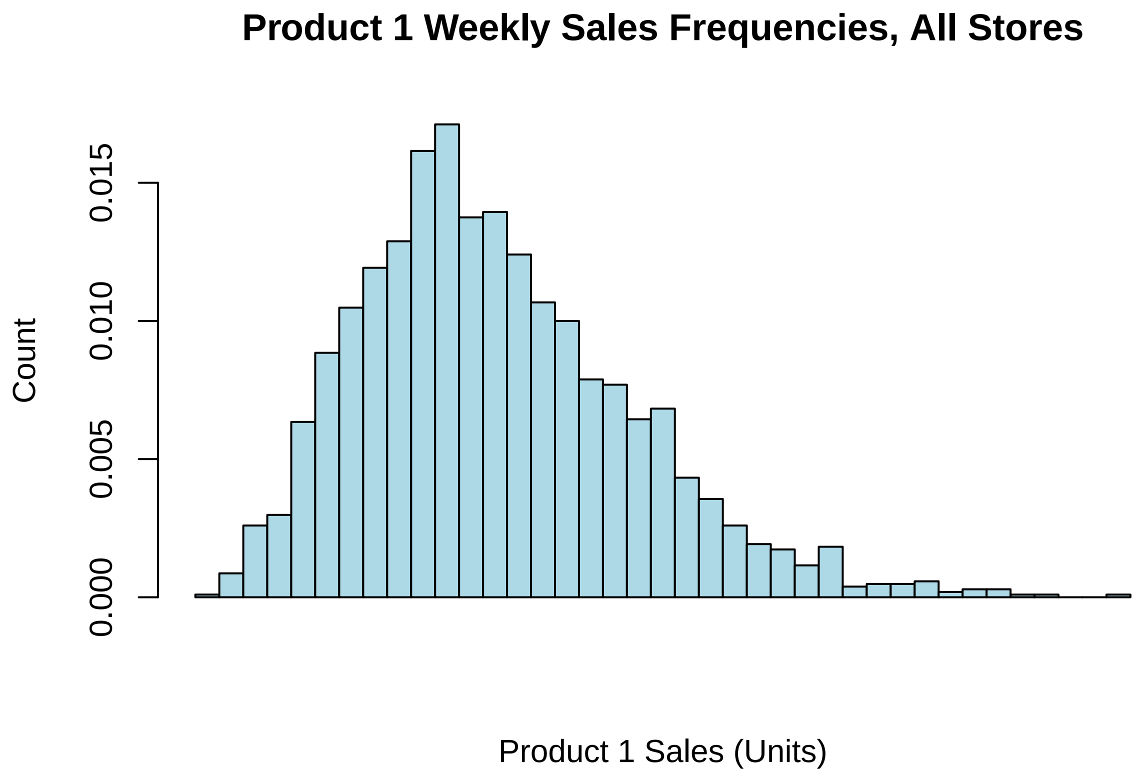
```
hist(store.df$p1sales,
     main = "Product 1 Weekly Sales Frequencies, All Stores",
     xlab = "Product 1 Sales (Units)",
     ylab = "Count",
     breaks = 30, # more columns
     col = "lightblue" # colore the bars
     )
```



The y-axis value for the height of the bars changes according to count. The count depends on the number of bins and the sample size. We can make it absolute by using *relative frequencies* (technically, the *density* estimate) instead of counts for each point. This makes the Y-axis comparable across different-sized samples.

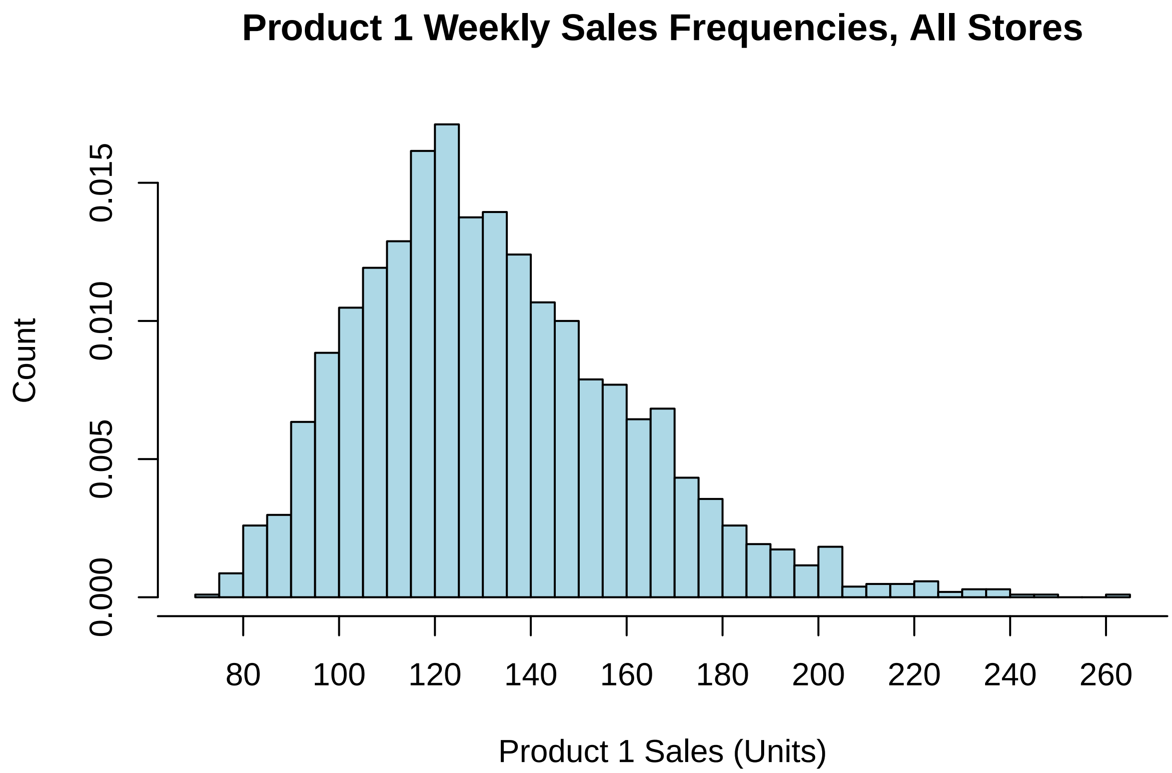
We can also remove the X-axis text to replace it with the one we want.

```
hist(store.df$p1sales,
     main = "Product 1 Weekly Sales Frequencies, All Stores",
     xlab = "Product 1 Sales (Units)",
     ylab = "Count",
     breaks = 30,
     col = "lightblue",
     freq = FALSE, # means plot density, not counts
     xaxt="n" # means x-axis tick mark is set to "none"
     )
```



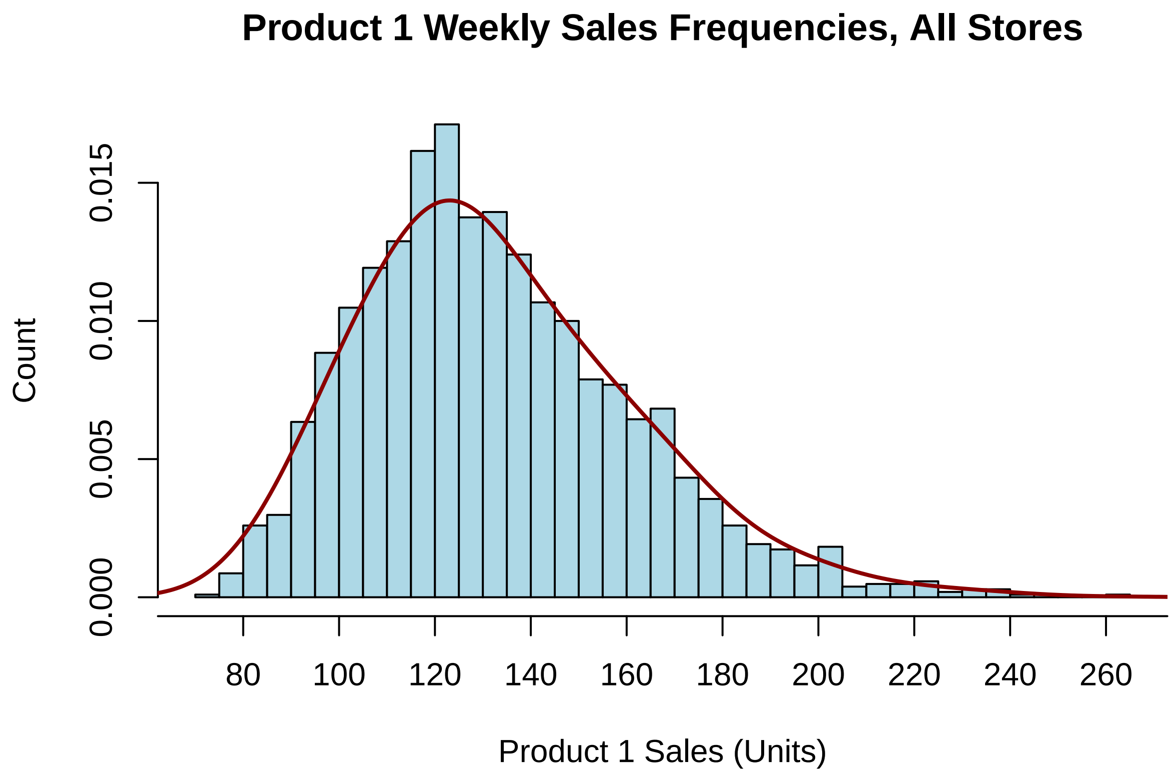
With *axis()*, we specify which axis to change using an argument: *side=1* alters the X axis, while *side=2* alters the Y axis (the top and right axes are *side=3* and *side=4*, respectively). We have to tell it where to put the labels, and the argument "at=VECTOR* specifies the new tick marks for the axis. They are easily made with the *seq()* function to generate a sequence of numbers:

```
hist(store.df$p1sales,
     main = "Product 1 Weekly Sales Frequencies, All Stores",
     xlab = "Product 1 Sales (Units)",
     ylab = "Count",
     breaks = 30,
     col = "lightblue",
     freq = FALSE, # means plot density, not counts
     xaxt="n" # means x-axis tick mark is set to "none"
     )

axis(side = 1, at=seq(60, 300, by=20)) # add "60", "80", ...
```



Finally, we can add a smoothed estimation line. We use the *density()* function to estimate the density value for the *p1sales* vector and add those to the chart with the *lines()* command. The *lines()* command adds elements to the current plot in the same way as *axis()* command.

```
hist(store.df$p1sales,
     main = "Product 1 Weekly Sales Frequencies, All Stores",
     xlab = "Product 1 Sales (Units)",
     ylab = "Count",
     breaks = 30,
     col = "lightblue",
     freq = FALSE, # means plot density, not counts
     xaxt="n" # means x-axis tick marks is set to "none"
     )

axis(side = 1, at=seq(60, 300, by=20)) # add "60", "80", ...
lines(density(store.df$p1sales, bw=10), # "bw=..." adjusts the smoothing
      type="l", col = "darkred", lwd=2) # lwd=line width
```



The final graph is now very informative.

The process we have shown to produce this graphic represents how analysts use R for visualisation. You start with a default plot, change some of the options, and use functions like *axis()* and *density()* to alter features of the plot with complete control. Although, at first, this will seem cumbersome compared to the drag-and-drop methods of other visualisation tools, it really is not much more time-consuming if you use a code editor and become familiar with the functions. It has the great advantage that once you have written the code, you can reuse it with different data.

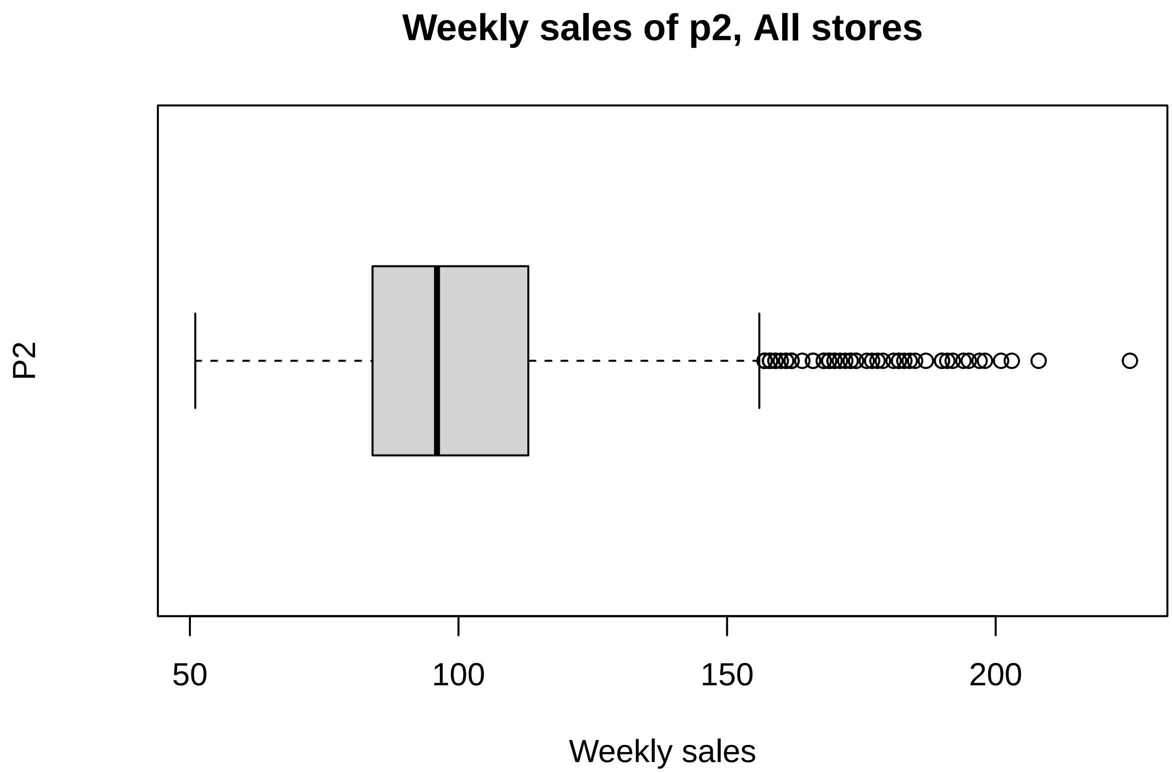
** Exercise**

Modify the code to create the same histogram for product 2. It requires only minor changes to the code, whereas with a drag-and-drop tool, you would start all over.

4.2 Boxplots

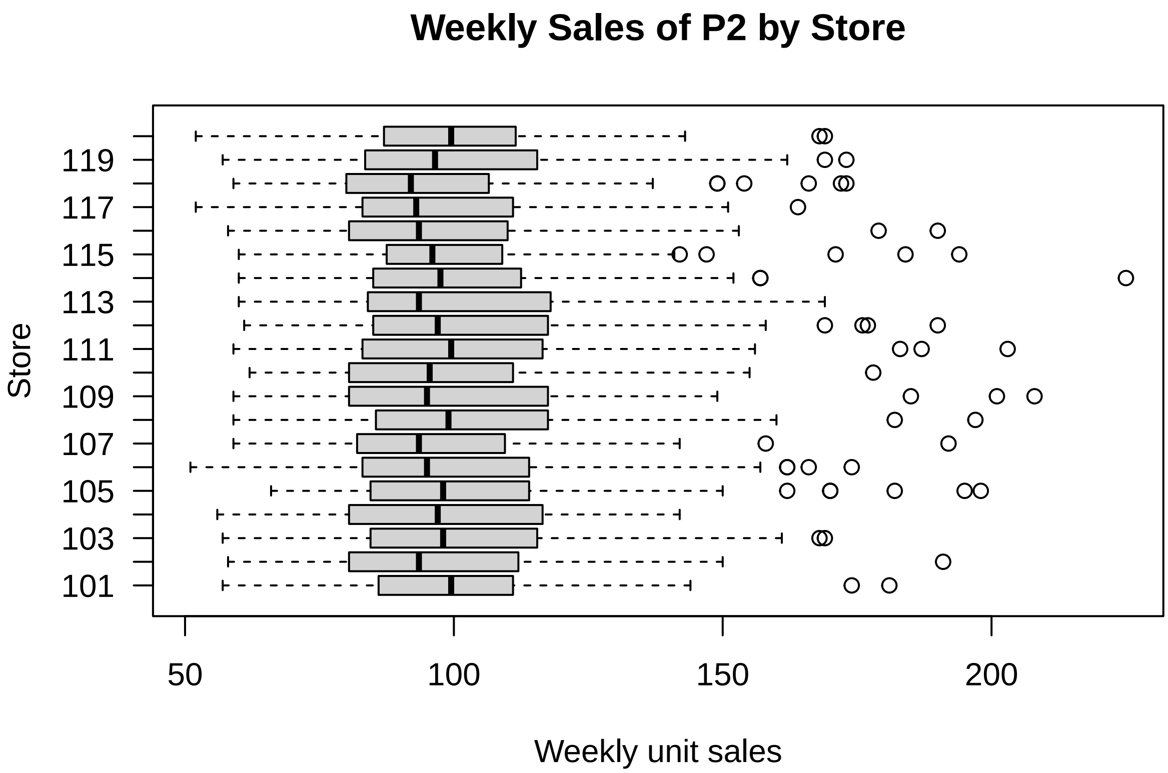
Boxplots are a compact way to represent a distribution. We add labels and use the option *horizontal =TRUE* to rotate the plot 90 degrees to look better.

```
boxplot(store.df$p2sales, xlab = "Weekly sales", ylab = "P2",
        main = "Weekly sales of p2, All stores", horizontal = TRUE)
```



Boxplots are even more useful when you compare distributions by some other factors. How do different stores compare on sales of product 2? The *boxplot()* command makes it easy to compare these by specifying a response *formula* using a *tilde (~)* to separate the *response variable* (sometimes called a *dependent* variable) from the *explanatory variable* (sometimes called an *independent* variable). In this case, our response variable is *p2sales*, and we want to plot it with regard to the explanatory variable *storeNum*.

```
boxplot(store.df$p2sales ~ store.df$storeNum, # boxplot p2sales by Store
        horizontal = TRUE, ylab = "Store", xlab = "Weekly unit sales",
        las=1, main = "Weekly Sales of P2 by Store")
```



We added one other argument to the plot: *las=1*. That forces the axes to have text in the horizontal direction, making the store numbers more readable.

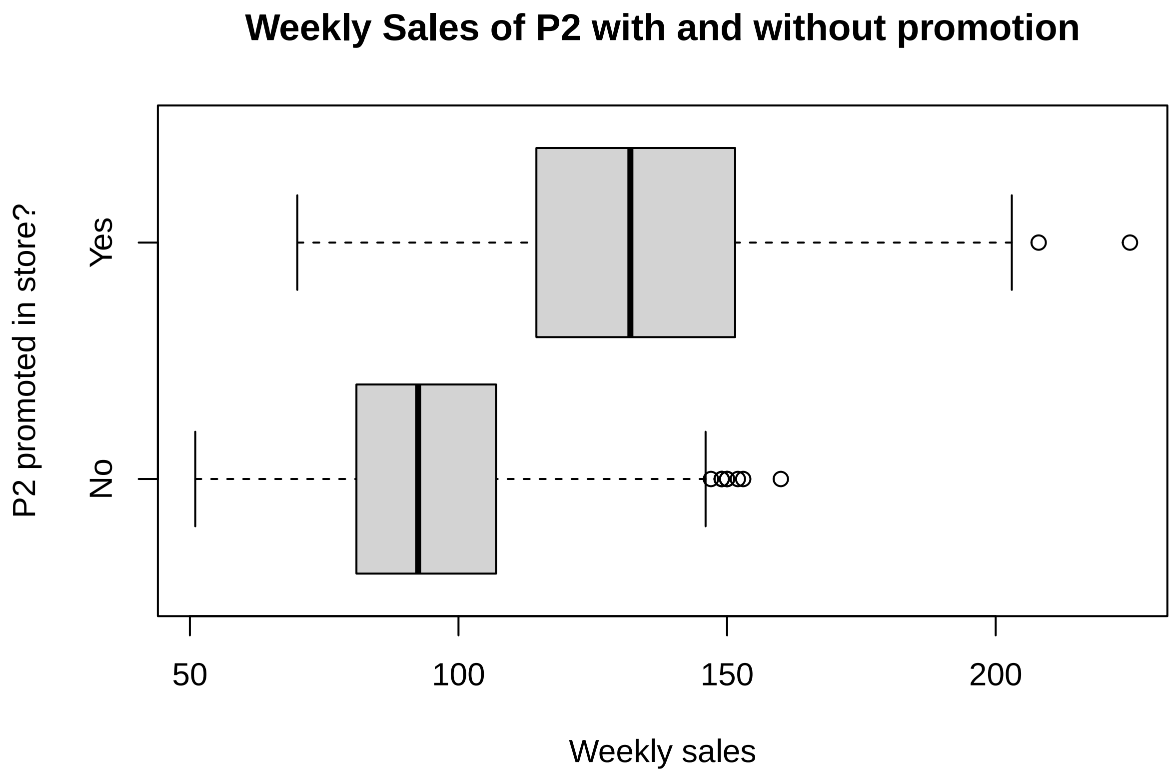
We see the stores are roughly similar in sales of product 2.

Shortcut commands that make life easier:

- Many commands for statistics and plotting to understand the *data=DATAFRAME* argument and will use variables from *data* without specifying the full name of the data frame. This makes it easy to repeat analyses on different datasets that include the same variables. All you need to do is change the argument for *data=*.

Exercise Do P2 sales differ in relation to in-store *promotion*?

```
boxplot(p2sales ~ p2prom, data = store.df, horizontal = TRUE, yaxt="n",
        ylab = "P2 promoted in store?", xlab = "Weekly sales",
        main = "Weekly Sales of P2 with and without promotion")
axis(side=2, at=c(1,2), labels=c("No", "Yes"))
```




```
# we replace the default Y axis with one that is more informative.
```

4.3 Aggregate()

Sometimes, we want to break out data by factors and summarize it. For example, how can we compute the mean sale by the store?

```
p1sales.sum <- aggregate(store.df$p1sales,
                         by=list(country=store.df$country), sum)
p1sales.sum
```

5 Takeaways

- Always check your data for proper structure and data quality using *str()*, *head()*, *summary()*, and other basic inspection commands.
- Describe discrete (categorical) data with *table()* and continuous data with *describe()* from the *psych* package.
- Histograms and boxplots are good for initial data visualization.
- Use *aggregate()* to break out your data by grouping variables.