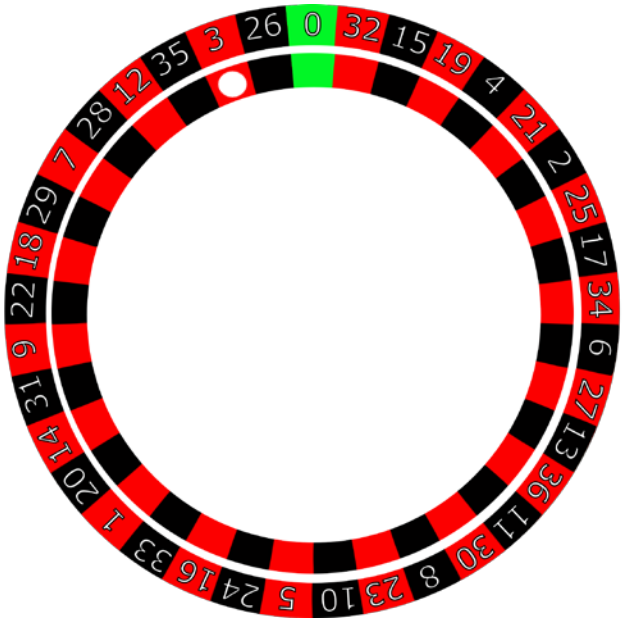
The roulette case

Roulette is a betting game which rewards the player's correct prediction of its outcome. The game consists of a ball spinning around a wheel which rotates in the opposite direction. The wheel features 37 numbered pockets. Each of the number has a color (18 are red, 18 are black and one, the zero, is green). The aim of the game is to bet on one or several outcomes regarding the pocket on which the ball lands. Numbers can range from 0 to 36, and several types of bets are available such as the color of the number, it being even or odd, and several other characteristics related to the number or the position on the wheel (as marked on the betting grid). The image below is a representation of an European roulette wheel. The ball is represented by the tiny white circle. In this example it landed on the pocket corresponding to the number 3.



A representation of a roulette wheel

Numbers are ordered on the wheel in such a way that the position of a number on the wheel is as unrelated as possible to the possible bets, (except of course bets on the position itself, which we will not describe here). The order of the numbers, starting from 0 is visible on the image above. As you can notice, the color of the numbers alternates when moving forward on the wheel (red, black, red, and so on). Also, the order seems to be unrelated to the betting grid. We will see if this is the case at the end of the chapter.

The table below is a schematic representation of the betting grid at European roulette. Red numbers are italicized. Betting on each number returns 35 times the amount if the number is drawn (plus the initial bet). Betting on color (red or black), odd vs even, 1-18 vs 19-36 return each the betted amount (plus the initial bet), if the drawn number corresponds to that attribute. The probability of occurrence of any of these is 18/37 or 0.487. Betting on the 1st dozen, 2nd dozen, 3rd dozen and each of the 2:1 column returns for each 2 times the amount if the drawn number falls in that category (plus the initial bet). The probability of occurrence of any of these is 12/37 or 0.32. Bets are lost if the drawn number fails to be within the betting category. In the example above, the ball stopped on number 3. Examples of winning bets in the depicted example are Red, first dozen, 1-18, the 3rd column, and of course betting on number 3.

The following table depicts a betting grid at roulette.

		0		
1-18	1st 12	*1*	2	*3*
		4	*5*	6
EVEN		*7*	8	*9*
		10	11	*12*
RED	2nd 12	13	*14*	15
		16	17	*18*
BLACK		*19*	20	*21*
		22	*23*	24
ODD	3rd 12	*25*	26	*27*
		28	29	*30*
19-36		31	*32*	33
		34	35	*36*
		2-1	2-1	2-1

A representation of a betting grid at roulette

Histograms and bar plots

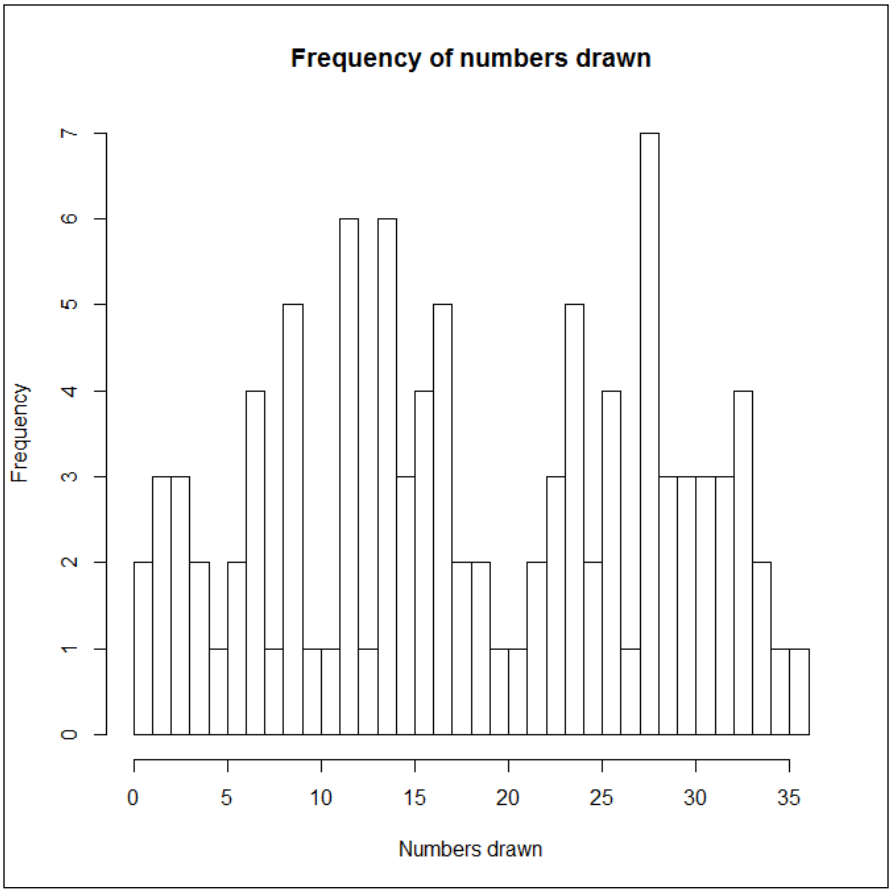
Roulette is a fascinating example of a betting game using random outcomes. In order to explore some properties of roulette spins, let's visualize some randomly drawn numbers in the range of those in an European roulette game (0 to 36). Histograms allow the graphic representation of the distribution of variables. Let's have a look at it! Type in the following code:

```
1  set.seed(1)
2  drawn = sample(0:36, 100, replace = T)
3  hist(drawn, main = "Frequency of numbers drawn",
4     xlab = "Numbers drawn", breaks=37)
```

Here we first set the seed number to `1` (see line 1). For reproducibility reasons, computer generated random numbers are generally not really random (they are in fact called pseudo-random). Exceptions exist, such as numbers generated on the website `http://www.random.org` (which bases the numbers on atmospheric variations). Setting the seed number to `1` (or any number really) makes sure the numbers we generate here will be the same as you will have on your screen, when using the same code and the same seed number. Basically, setting it allows the reproduction of random drawings. On line 2, we use the `sample()` function to generate 100 random numbers in a range of 0 to 36 (`0:36`). The `replace` argument is set to true (`T`), which means that the same number can be drawn several times.

The `hist()` function (lines 3 and 4) will plot the frequency of these numbers. The `hist()` function takes a multitude of arguments, of which we use 4 here; `main`, which sets the title of the graphic, `xlab`, which sets the title of the horizontal axis (similarity, `ylab` would set the title of the vertical axis), and `breaks`, which forces the display of 37 breaks (corresponding to the number of possible outcomes of the random drawings). For more information about the `hist()` function, you can simply type `?hist()` in your R console.

As you can notice on the graph below, the frequencies are quite different between numbers, even though each number has an equal theoretical probability to be drawn on each roll. The output is provided in the figure below:



A histogram of the frequency of numbers drawn

Let's dwell a little upon the representation of mean values using bar plots. This will allow us to have a look at other properties of the roulette drawings. The mean values will represent the proportions of presence of characteristics of the roulette outcomes (for example, proportion of red number drawn). We will therefore build some new functions.

The `buildDf()` function will return a data frame with a number of rows that correspond to how many numbers we want to be drawn, and a number of columns that correspond to the total number of attributes we are interested in (the number drawn, its position on the wheel and several possible bets), totaling 14 columns. The matrix is first filled with zeroes, and will be populated at a later stage:

```
1    buildDf = function(howmany) {
2       Matrix=matrix(rep(0, howmany * 14), nrow=howmany,ncol=14)
3       DF=data.frame(Matrix)
4       names(DF)=c("number","position","isRed","isBlack",
5         "isOdd","isEven","is1to18","is19to36","is1to12",
6         "is13to24","is25to36","isCol1","isCol2","isCol3")
7       return(DF)
8    }
```

Let's examine the code in detail: on line one, we declare the function, which we call `buildDF`. We tell R that it will have an argument called `howmany`. On line 2, we assign a matrix of `howmany` rows and `14` columns to an object called `Matrix`. The matrix is at this stage filled with zeroes. On line 3, we make a data frame called `DF` of the matrix, which will make some operations easier later. On lines 4 to 6, we name the columns of the data frame using `names()` functions. The first column will be the number drawn, the second the position on the wheel (the position for `0` will be `1`, the position for `32` will be `2`, and so on). The other names correspond to possible bets on the betting grid. We will describe these later when declaring the function that will fill in the matrix. On line 7, we specify that we want the function to return the data frame. On line 8, we close the function code block (using a closing bracket), which we opened on line 1 (using an opening bracket).

Our next function, `attributes()`, will fill the data frame with numbers drawn from the roulette, their position on the roulette, their color, and other attributes (more about this below):

```
1    attributes = function(howmany,Seed=9999) {
2       if (Seed != 9999) set.seed(Seed)
3       DF = buildDf(howmany)
4       drawn = sample(0:36, howmany, replace = T)
5       DF$number=drawn
6       numbers = c(0, 32, 15, 19, 4, 21, 2, 25, 17, 34, 6, 27,
7          13, 36, 11, 30, 8, 23, 10, 5, 24, 16, 33, 1, 20, 14,
8          31, 9, 22, 18, 29, 7, 28, 12, 35, 3, 26)
```

The function is not fully declared at this stage. We will break it down in several parts in order to explain what we are doing here. On line 1, we assign the function to object attributes, specifying that we have 2 arguments; `howmany` for the number of rows corresponding to how many numbers we want to be drawn, and `Seed` for the seed number we will use (with default value `9999`). On line 2, we set the seed to the provided seed number if it is not `9999` (as we need the function to be able not to set the seed for analyses we will do later). On line 3, we create the data frame by calling the function `buildDf()` we created before. On line 4, we sample the specified amount of numbers. On line 5, we assign these numbers to the column of the data frame called `drawn`. On line 6, we create a vector called `numbers`, which contains the numbers `0` to `36`, in the order featured on the roulette wheel (starts with 0, then 32, 15 …).

In the remaining of the function (presented below), we populate the rest of the attributes:

```
9       for (i in 1:nrow(DF)){
10         DF$position[i]= match(DF$number[i],numbers)
11         if (DF$number[i] != 0) { if (DF$position[i]%%2) {
12            DF$isBlack[i] = 1} else {DF$isRed[i] = 1}
13         if (DF$number[i]%%2) { DF$isOdd[i]=1}
14         else {DF$isEven[i]=1}
15         if (DF$number[i] <= 18){ DF$is1to18[i]=1}
16         else { DF$is19to36[i]=1}
17         if(DF$number[i] <= 12){ DF$is1to12[i]=1}
18         else if (DF$number[i]<25) { DF$is13to24[i] = 1}
19            else { DF$is25to36[i] = 1}
20         if(!(DF$number[i]%%3)){ DF$isCol3[i] = 1}
21         else if ((DF$number[i] %% 3 ) == 2) {
22           DF$isCol2[i] = 1}
23           else { DF$isCol1[i] = 1}
24           }
25         }
26       return(DF)
27    }
```

On line 9, we create a loop, meaning that the code block will iterate from i = 1, to i = the number of numbers we have drawn (the number of rows of the data frame). We open the code block using an opening bracket. On line 10, we assign to the attribute `position` of the drawn number on the wheel, using function `match()`. On lines 11 to 12, we create a nested condition, stating that if the number is not 0, we assign 1 to attribute `isBlack` is the position of the number is even, or 1 to `isRed` if the position is odd (remember the color of the numbers alternate – red, black, red ...). On line 13 and 14, we assign `1` to attribute `isOdd` if the number if odd, or `1` to attribute `isEven` if the number is even. On lines 15 and 16, we assign `1` to attribute `is1to18` if the number is smaller or equal to `18`, or `1` to attribute is`19to36` if the number is higher than `18`. On lines 17 to 18, we assign `1` to either `is1to12`, `is13to24` or `is25to36` depending on the value of the number (that's self-explanatory). Finally, on lines 20 to 26, we assign the column number on the betting grid, by setting the value of either `isCol1`, `isCol2`, or `isCol3` (on the table representing the betting grid, `isCol1` is the left 2:1 column, `isCol2` the middle 2:1 column and `isCol3` the right one). As we have used nested conditions here, we close the code block on lines 24 and 25. On line 26, we tell R that we want the function to return the resulting data frame. On line 27, we close the code block of the function (that we opened on line 1).

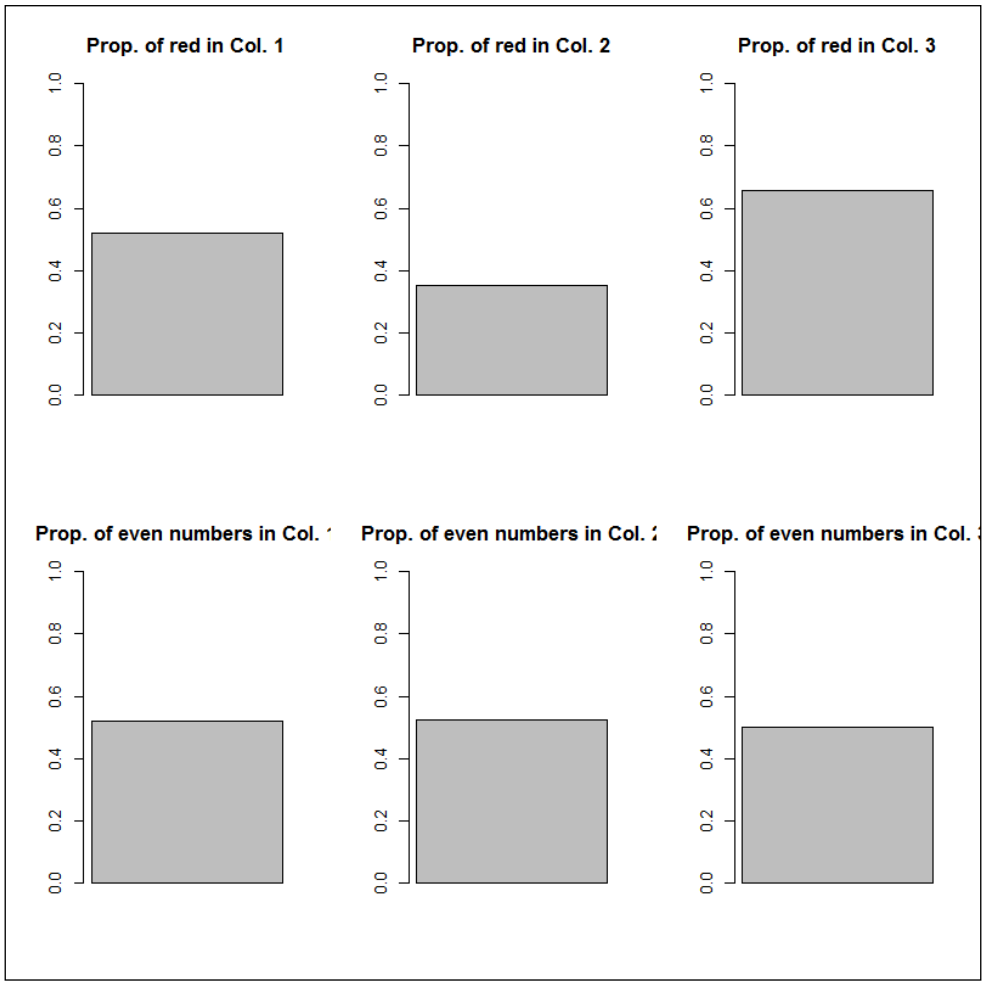
Now that we have our functions ready, we can now focus on visualizing some data. The following code will generate 1,000 roulette spins (let's use a seed number of `2` so that the calculation of the random number is the same on your machine as in mine):

```
Data=attributes(1000,2)
```

It is now time to explore the relationship between our variables in the following graph. We will first ask R to plot several graphs on the plotting area. To do so, we will rely on the `mfrow` argument of the `par()` function (line 1). We then tell R to plot `2` rows of `3` graphs corresponding to the proportion of red numbers (the mean of the values, as these are represented by `1` for presence and `0` for absence) in the 2:1 columns 1, 2 and 3 on the first row, and the proportion of even number in columns 1, 2 and 3 in the second row. Notice that for all 6 plots we use subsetting (using `subset()` function here) to select the portion of the data we are interested in. We use attribute `ylim` to define the range of the plotting area (from `0` to `1`), and attribute `main` to print the title of the plots.

```
1    par(mfrow = c(2,3))
2    barplot(mean(subset(Data, isCol1 == 1)$isRed), ylim=(c(0,1)),
3       main = "Prop. of red in Col. 1")
4    barplot(mean(subset(Data, isCol2 == 1)$isRed), ylim=(c(0,1)),
5       main = "Prop. of red in Col. 2")
6    barplot(mean(subset(Data, isCol3 == 1)$isRed), ylim=(c(0,1)),
7       main = "Prop. of red in Col. 3")
```

```
8    barplot(mean(subset(Data, isCol1 == 1)$isEven), ylim=(c(0,1)),
9       main = "Prop. of even numbers in Col. 1")
10    barplot(mean(subset(Data, isCol2 == 1)$isEven), ylim=(c(0,1)),
11       main = "Prop. of even numbers in Col. 2")
12    barplot(mean(subset(Data, isCol3 == 1)$isEven), ylim=(c(0,1)),
13       main = "Prop. of even numbers in Col. 3")
```



Bar plots of the proportion of red, and even numbers drawn from Columns 1, 2 and 3

As can be seen on the graphs, the proportion of red numbers drawn from columns 1, 2 and 3 is different, whereas the proportion of even numbers is relatively similar between all the columns. This can be expected from the betting grid.

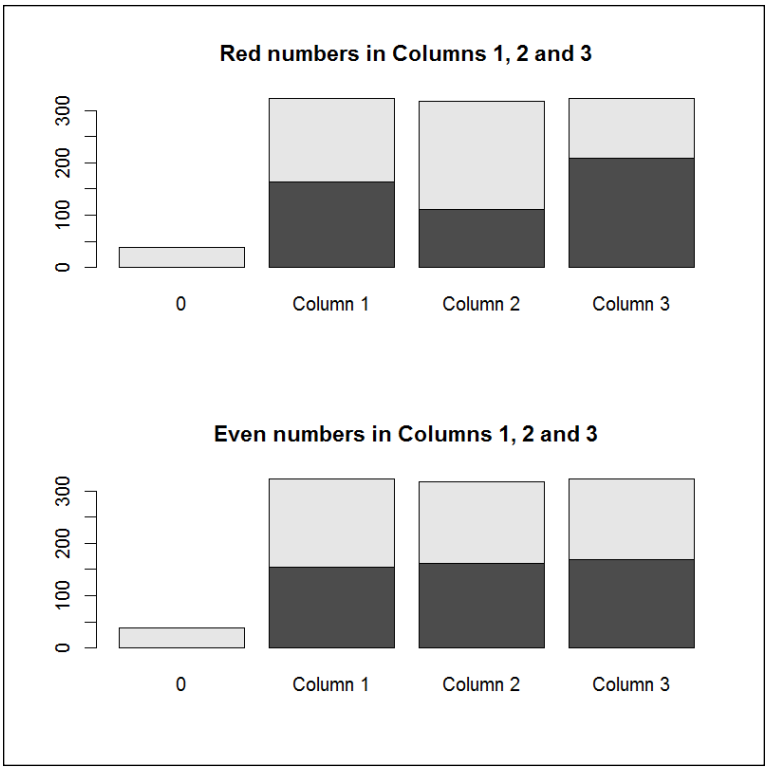
You might have noticed that we have lost important information in the process; the total of numbers drawn from each column, and the number of zeros; and we needed to produce one bar plot per column, which is a bit tricky. Let's solve these problems by first adding a single attribute which indicates the membership of the drawn numbers to `Column 1`, `Column 2` and `Column 3`.

```
1    for (i in 1:nrow(Data)){
2       if(Data$isCol1[i]== 1){ Data$Column[i]=1 }
3          else if (Data$isCol2[i] == 1 ) { Data$Column[i] = 2 }
4          else if (Data$isCol3[i] == 1 ) { Data$Column[i] = 3 }
5          else {Data$Column[i] = 0 }
6    }
```

On line 1, we start a `for` loop that will iterate from i = 1 to i = the number of rows in data frame `Data`. We use nested condition in lines 2 to 5 to determine the column number (1 if attribute `isCol1` equals to `1`, `2` if attribute `isCol2` equals to `1`, `3` if attribute `isCol3` equals to `1`, or `0` if neither of these conditions is satisfied. We close the code block on line 6.

We now can plot the column in relation to the proportion of red, and even numbers. For now, our attributes `isRed` and `isEven` are ordered with `0` coming first and `1` second. We want just the opposite, as we want the number of numbers coded `1` to appear at the bottom of the graph. We therefore reorder the values of our attributes using the levels attribute of the `factor()` function (lines 1 and 2). We will use `par()` again to get both graphs on the same plotting area. We then generate the stacked bar plots using the `barplot()` function again. Notice we do not plot mean values this time, but the content of the table in which the cells correspond to the intersections of the attributes `Column` and `isRed` or `isEven`. We rely on the argument `name.arg` to name the sections of the plots:

```
1    Data$isRed = factor(Data$isRed, levels = c(1,0))
2    Data$isEven = factor(Data$isEven, levels = c(1,0))
3    par(mfrow = c(2,1))
4    barplot(table(Data$isRed,Data$Column),
5       main = "Red numbers in Columns 1, 2 and 3",
6       names.arg = (c("0","Column 1", "Column 2", "Column 3")) )
7    barplot(table(Data$isEven,Data$Column),
8       main = "Even numbers in Columns 1, 2 and 3",
9       names.arg = (c("0","Column 1", "Column 2", "Column 3")) )
```



A bar plot of the number of Red and Even numbers drawn

As can be seen on this stacked bar plot, approximately the same amount of numbers have been drawn from each of the columns. The number 0 has been drawn around 50 times, which is about twices often as expected given its theoretical probability equal to those of the other numbers (*1000 * (1/37) = 27*).

Scatterplots

Until now we have observed frequencies of the relationship between categorical membership (nominal attributes) and frequencies or means. It is also useful to have a look at relationships between numerical attributes. We will rely on scatterplots for this purpose. This will require a little scripting again, as we will examine the relationships between proportions. Let me first introduce the function `proportions()` which will generate the proportions for us, for all of our nominal attributes. This function takes one argument, `DF`, and call our `attributes()` function by default. We could instead give as an argument the data frame with the numbers we have previously drawn and the attributes.

The body of the function computes and returns the transpose of the means of each nominal attributes:

```
1    proportions = function(n = 100) {
2       DF=attributes(n)
3       return(data.frame(t(colMeans(DF[3:ncol(DF)]))))
4    }
```

The body of this function calls our `attributes()` function and passes the number of roulette draws to it (line 2). It then returns a data frame which contains the transpose means of all the columns, except columns 1 and 2 (which are not of interest here).

Our next function `multisample()` will return a data frame containing the proportions of each attribute of each of `k` samples (one sample per row) of `n` numbers drawn. It will by default draw `100` samples of `100` numbers. After starting the function declaration on line 1, we set the seed to the provided value, or the default value on line 2. We then create a vector containing the values returned by a first call to the `proportions()` function. In the following loop, we append iteratively values returned by function `proportions()` (lines 4 to 7). Finally, we return the resulting data frame (line 8), and close the function code block (line 9).

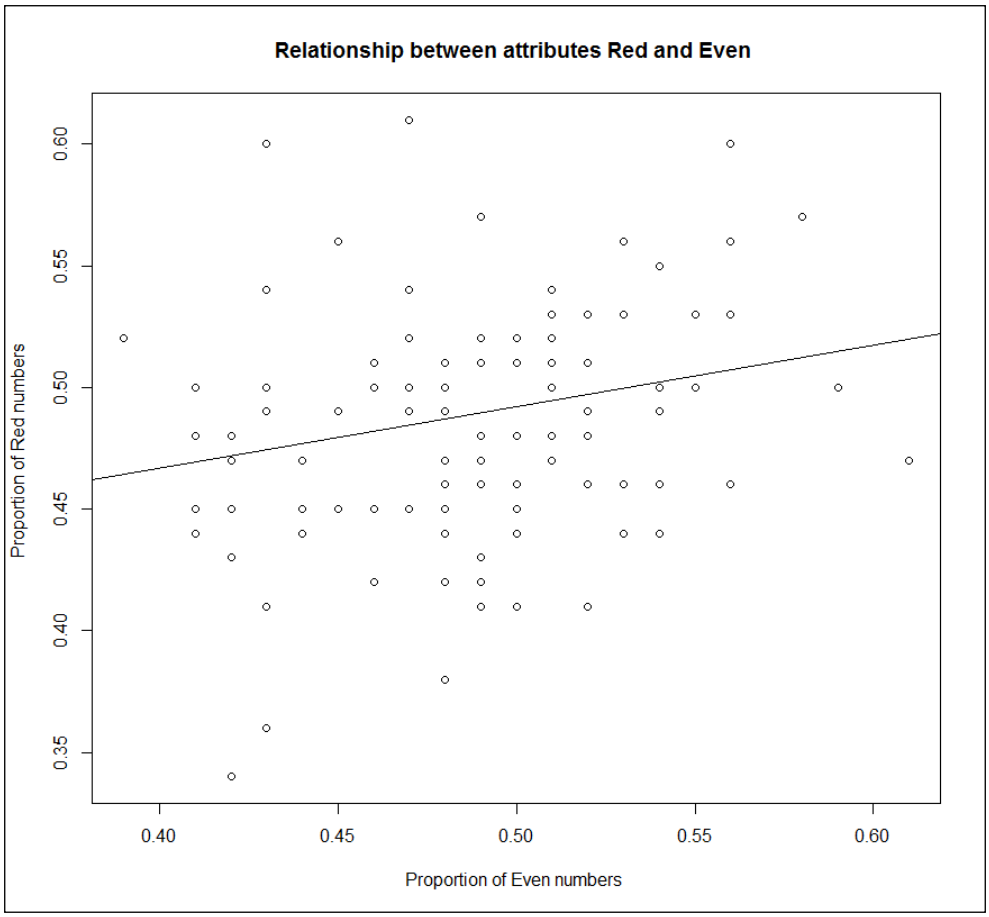
```
1    multisample = function(n=100,k=100, Seed=3){
2       set.seed(Seed)
3       ColMeans.df=proportions(n)
4       for (i in 1:k-1){
5          ColMeans.df=rbind(ColMeans.df,
6             proportions(n))
7       }
8       return(ColMeans.df)
9    }
```

We are now able to examine the relationship between proportions of numbers using scatterplots. Scatterplots display each observation on a plane by plotting the values of two attributes. On line 1, we first create a data frame of proportions using the default arguments for `multisample()` function. This will not take too long to compute. Having a look at the roulette grid, one can see that 10 out of the 18 red numbers are odd. Will we be able to spot this relationship from the random drawings? We will investigate this visually. We plot the proportions of red and the proportions of even numbers using a scatterplot (lines 3 to 6). The `main` argument set the title of the graph (line 4). The `xlab` argument sets the title of the horizontal axis (line 5). The `ylab` argument sets the title of the vertical axis (line 6). We also add and a line (called slope) showing the direction of the relationship using `abline()` function on line 7.

The function here uses the coefficients of a linear model as argument. I will discuss the `lm()` function which provides such coefficients in the chapter about regression:

```
samples = multisample()
par(mfrow=c(1,1))
plot(samples$isOdd,samples$isRed,
   main = "Relationship between attributes Red and Even ",
   xlab = "Proportion of Even numbers",
   ylab = "Proportion of Red numbers")
abline(lm(samples$isOdd~samples$isRed))
```

The output is provided below:



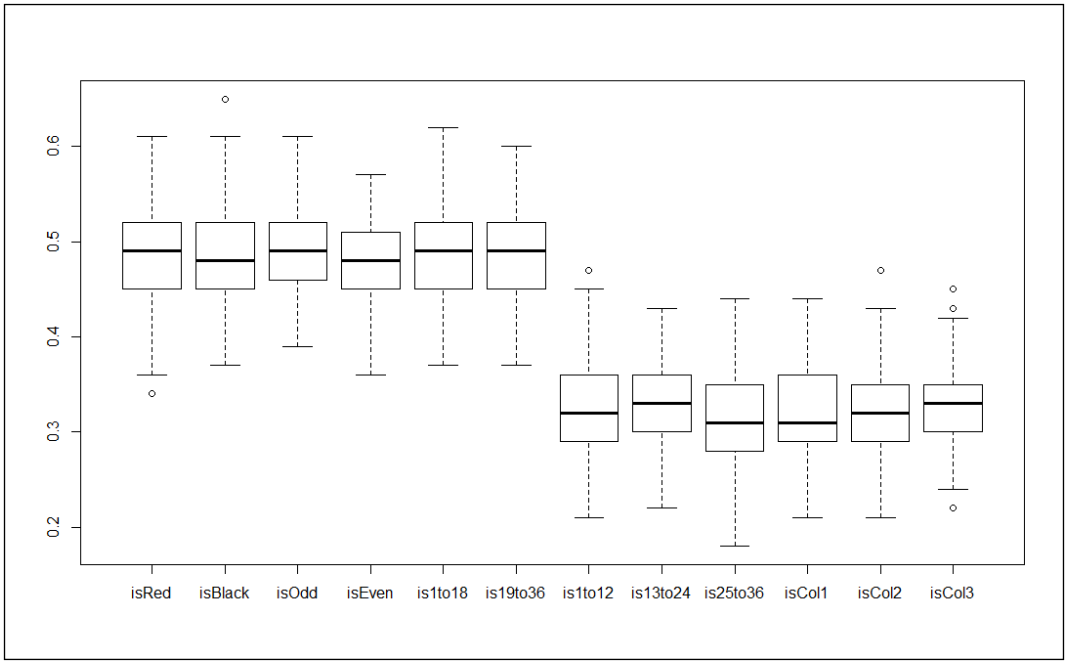
A scatterplot showing the relationship between the proportion of Red and Even numbers

The graph depicts the values on our two attributes (the proportion of even numbers on the *x* axis and the proportion of red numbers on the *y* axis) for each of our samples. The line represents the linear relationship between these two proportions; the higher the proportion of even numbers, the higher the proportion of red numbers. Again, this can be expected from the betting grid.

Boxplots

As we can also notice from the scatterplot, whereas most of the samples have a relatively balanced proportion of red or even numbers, these proportions are very small or large in some cases. We could examine the dispersion of those values using a histogram again, but the boxplot is much more interesting, so we will use it instead. Boxplots are representations of the distribution of an attribute. We could have a look at only one attribute by specifying its name as an argument from the `boxplot()` function. We will instead look at all the arguments at once by giving the data frame as an argument:

```
boxplot(samples)
```



Boxplots of all the attributes

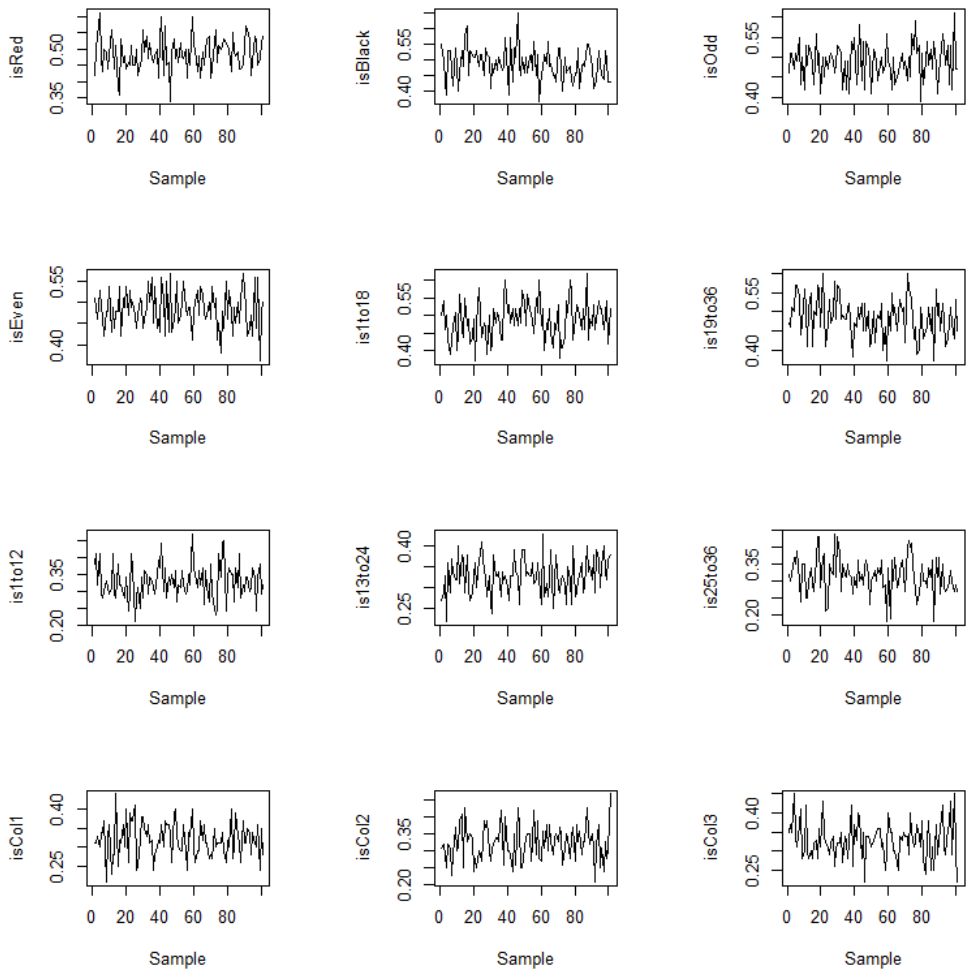
As can be seen from the boxplots, the proportions of red, black, odd, even, numbers below 18, and numbers higher than 18 are a little below 50% on average, which is what is expected as 18 of 37 numbers are in each of these categories. We can also notice that the average proportion of numbers between 1 and 12, 13 and 24, 25 and 36, as well as numbers on columns 1, 2 and 3 are a bit below 33%, which is expected as well. What might surprise us is that there is a huge variation around these average values. On each boxplot, the bottom box represents the data points that are in the second quartile. The top box represents data points that are in the third quartile. Thus, 50% of our data points fit in the two boxes. The space between the whiskers represents 150% of the interquartile range (the distance between the third and first quartile, or Q3-Q1). Finally, outliers are displayed as separate points on the boxplots. We can visually notice that the space between the whiskers it is about as large for the attributes on the right large of the graph as for attributes on the left side, even though the median proportion is much lower.

Line plots

Line plots provide the same information as bar plots. They might allow to understand relationships between attributes better because the values are linked by lines which give a better feeling of the difference between the values. We will investigate the variability of the proportions of each attribute by plotting its proportion from each sample. On line 1, we will first configure the plotting area contain 12 plots (as we have 12 attributes). Notice we use the `oma` attribute to set the outer margin, and the `mar` attribute to set the inner margin. On line 2, we set the names to be used in the titling of the axis (using the `ylab` attribute, see line 4). We then iteratively create, for each attribute, a graph plotting each value (lines 3 to 5). The type attribute is set to `l` (line 5) in order to plot lines instead of dots as in a scatterplot.

```
1    par(mfrow=c(4,3), oma = rep(0.1,4), mar = rep(4,4))
2    names=colnames(samples)
3    for (i in 1:ncol(samples)){
4       plot(samples[,i], xlab="Sample", ylab=names[i],
5          type = "l")
6    }
```

The output is provided below:



Variability of the proportion of each attribute