

# Introduction to R - part 3



Tiago Nardi

University of Pavia

# Plot design

# Histograms

A histogram is a representation of the distribution of **numerical** data

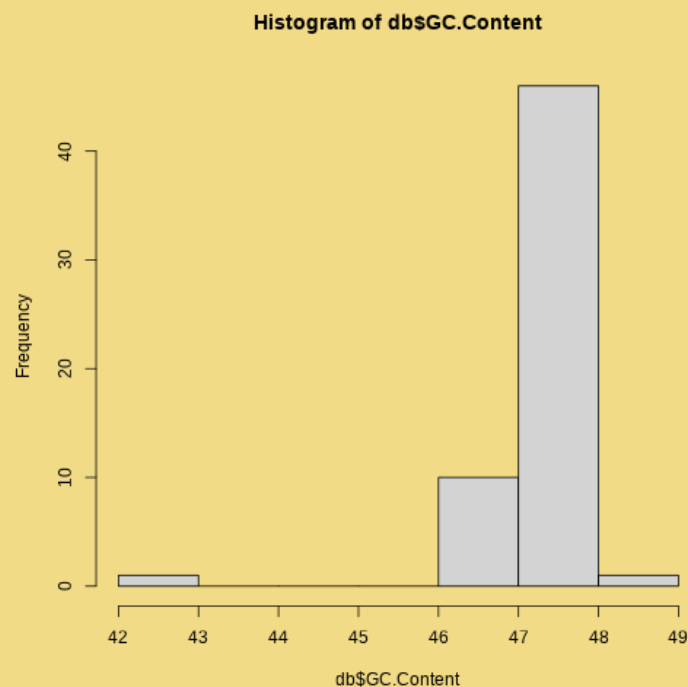
The entire range of values is divided into **intervals** (bins) and then the function counts how many values fall into each interval

The wideness of the column is dependant of the choosen interval

# Histograms

Histogram plot with default values

```
db <- read.csv("patric_redux.csv")
hist(db$GC.Content)
```



# Histograms

```
hist(dataframe_name$column,  
      breaks = "value", col = "colour_name",  
      border = "colour_name", main = "Title",  
      xlab = "x_name", ylab = "y_name")
```

breaks: a single number giving the number of cells for the histogram

col: a colour to be used to fill the bars, you can use the name of ("lightblue", "red", see <http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf> ) the RGB or the hex format

border: the color of the border around the bars. The default is to use the standard foreground color

main, xlab, ylab: these arguments give labels to title and axis

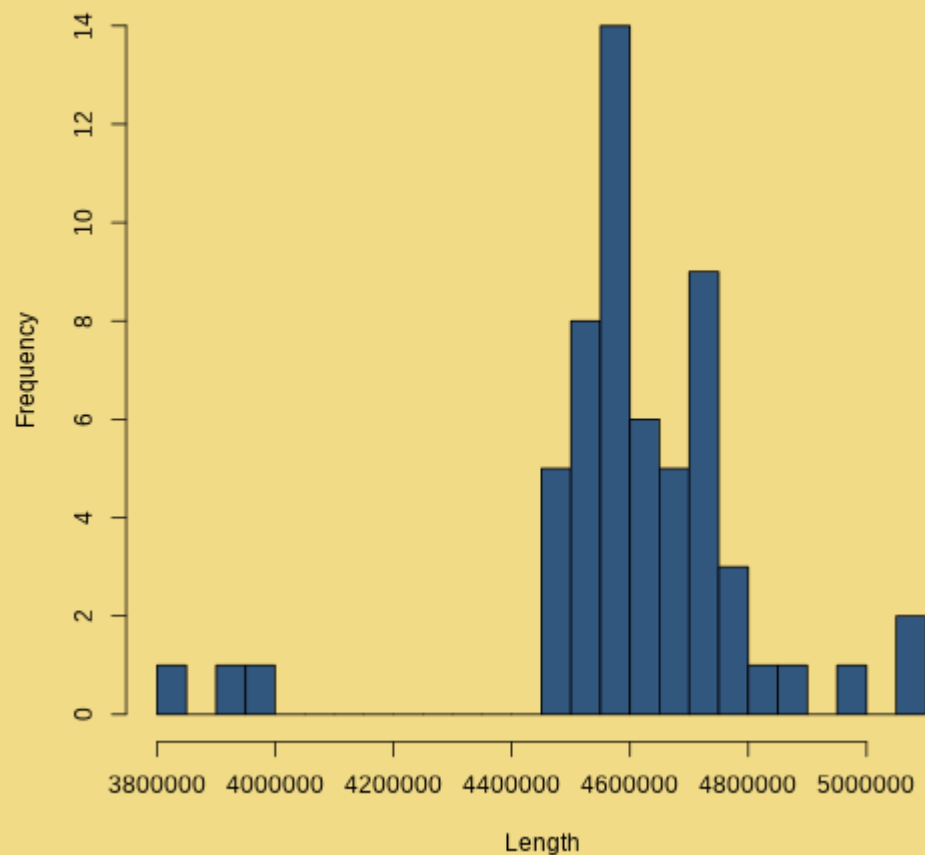
# Exercise

```
hist(dataframe_name$column,
     breaks = "value",
     col = "colour_name",
     border = "colour_name",
     main = "Title",
     xlab = "x_name",
     ylab = "y_name")
```

Try to get a histogram using the `patric_redux.csv`, draw the genome length, use appropriate labels and use a colour



Genome Length Distribution



```
hist(db$Genome.Length, breaks = 20,
     main = "Genome Length Distribution",
     col = "#32577F",border="black",xlab="Length")
```

# Saving graphs

To save a graph

```
pdf("Graph_name.pdf")  
graph_command # any graph, including hist()  
dev.off()
```

This will save whatever you draw (*graph\_command*) in your **present working directory**

You can also use the *export* function in the **Plot tab**



# Barplot

A barplot shows the relationship between categorical variables. Each variable is represented as a bar, and the bar size represents its numeric value

In R, `barplot()` function computes a barplot of the given grouped values

For the bar height you need a quantitative variable

For the bar name you need a qualitative variable

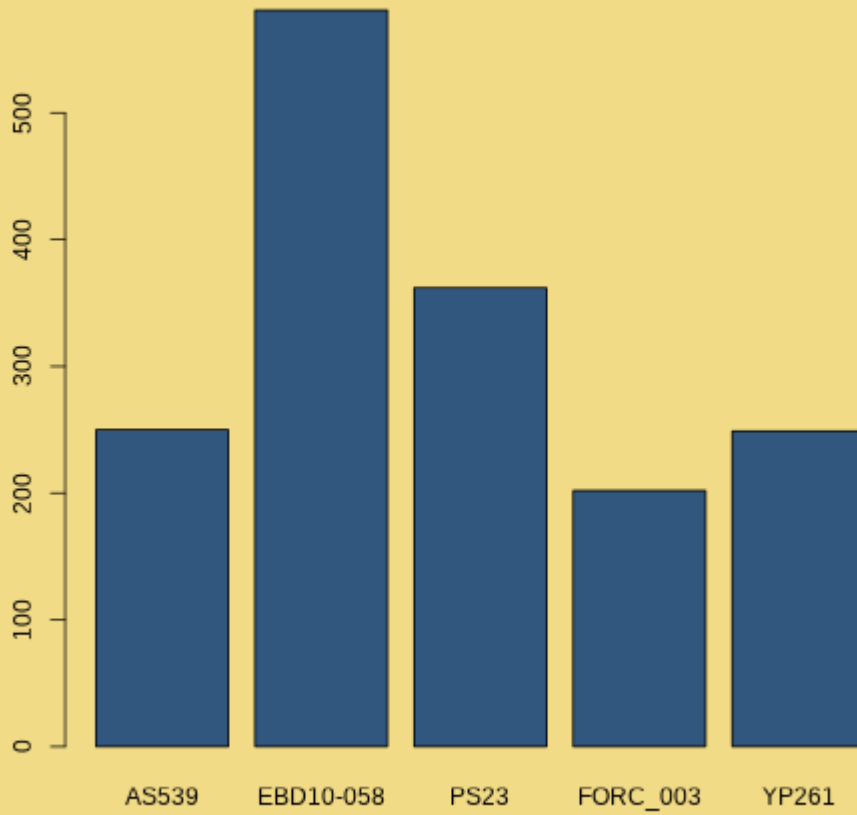
Try to make one barplot,  
using the subsetting  
table, with the *ID* as  
qualitative and *Contigs*  
as quantitative variable



```
# I'm using a smaller table for readability sake
db_sub <- subset(
  db,
  db$Isolation_location == "Barad-dur" |
  db$Isolation_location == "Isengard" )
```

```
barplot(
  height=quantitative_variable,
  names=qualitative_variable)
```

# Barplot



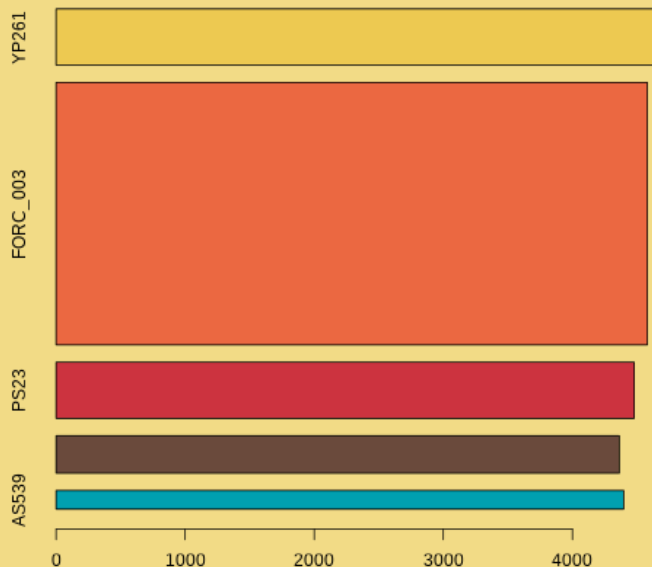
# Barplot

```
barplot(table, col = "colour_name",
        horiz = T, main = "Title",
        xlab = "x_name", ylab = "y_name",
        width=c(0.1,0.2,3,1.5,0.3),
        col=c("color", "#HEX"), space=c(1,3))
```

You can edit labels, as you have done before with the histogram, changing bar colours, display the bars horizontally, modify the space between the bars or even their width

Remember that you can check the **help** using *?barplot*

# A modified barplot



```
barplot(height=db_sub$PATRIC.CDS,
        names=db_sub$ID,
        horiz=T,
        width=c(0.1,0.2,0.3,
                1.4,0.3),
        col=c("#00A0B0",
              "#6A4A3C",
              "#CC333F",
              "#EB6841",
              "#EDC951"))
```

# Scatter plot

A scatter plot shows the relationship between **two numerical** variables

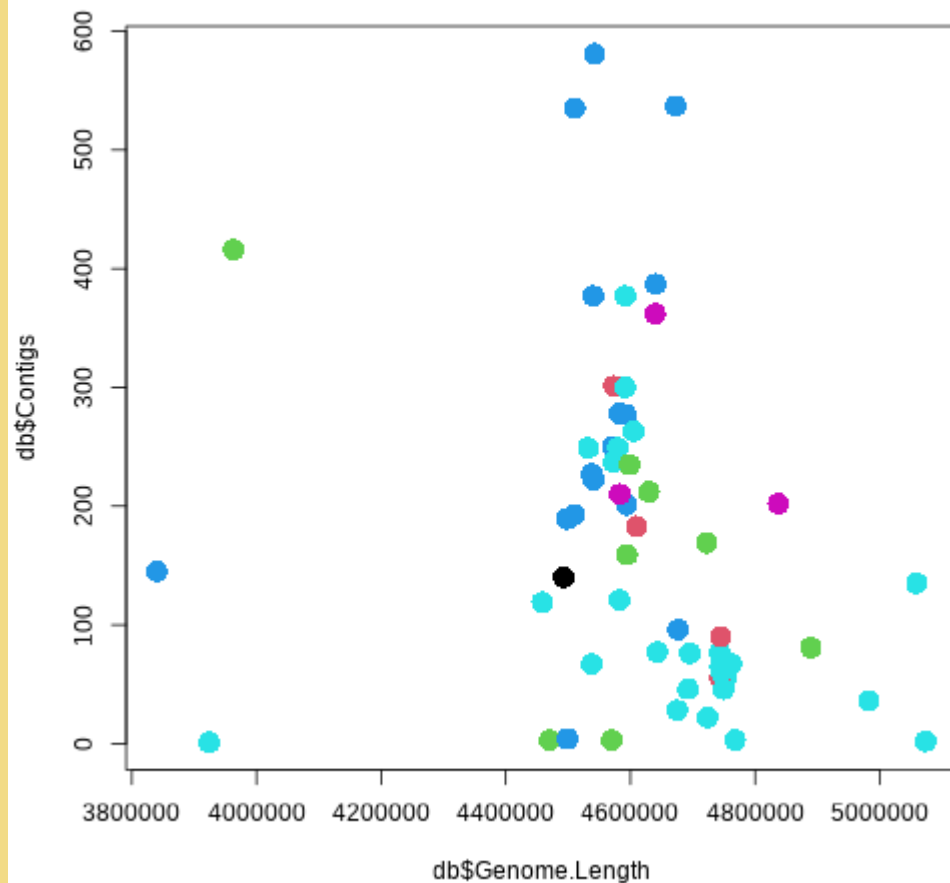
Displayed as a collection of points, each having one variable determining x axis and the other variable determining the position on the y axis

In R, we can use the generic function *plot()*

```
plot(x,y, col = "colour_name",  
main = "Title", xlab = "x_name", ylab = "y_name")
```



Draw a scatterplot with *Genome.Length* and *Contigs* number as the numerical variables



```
plot(db$Genome.Length,db$Contigs,
      col= as.numeric(as.factor(db$Source)),cex = 2,pch=16)
```

# Colouring - not mandatory

To assign a colour we need to use a **factor** (qualitative variable)

The variable can be already assigned (es. *Source*) or we can create one using the values in the table

We need a **vector** of colours of the same length of the list of factors

```
levels(as.factor(db$Source))
```

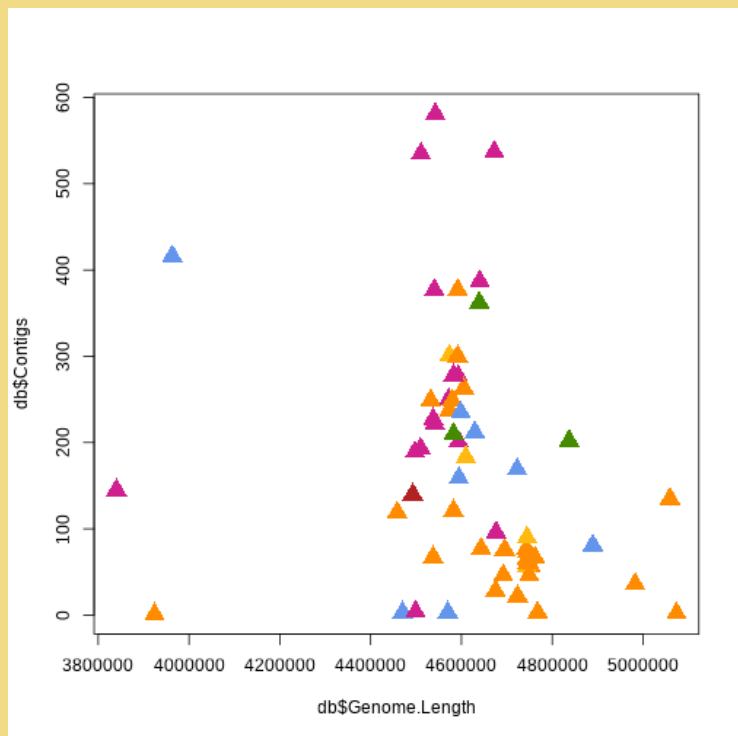
```
## [1] "Dragon" "Dwarf"  "Elf"    "Hobbit" "Human"  "Orc"
```

**Factors** are assigned in alphabetical order, so the colours should follow the same order

```
s_colours <- c("firebrick","darkgoldenrod1","cornflowerblue",  
              "violetred","darkorange","chartreuse4")
```



# Coloured by Source



```
plot(db$Genome.Length,db$Contigs,
     col= s_colours[
       (as.factor(db$Source))],
     cex = 2,pch=17)
```

I use a **vector** of colours, each value is assigned to a row of the data frame depending on the **level** of *db\$Source*

# Using the df values

Instead of groups already present in the dataframe you can also create grouping basing on values

```
db$group <- factor(ifelse(db$Genome.Length < 4400000, "low",  
                          ifelse(db$Genome.Length > 4800000 ,  
                                "high", "medium")))
```

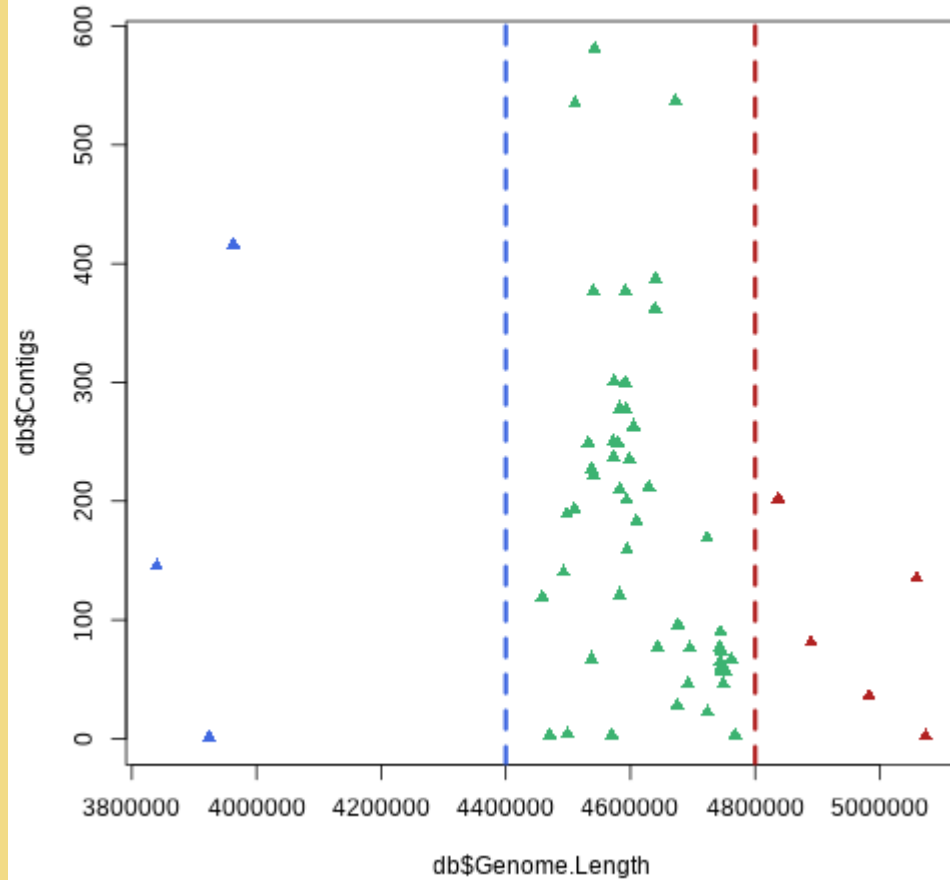
We can divide the samples according to *Genome.Length*, creating a new column

I used a logical function that assign different values on the logical condition

```
ifelse(condition, if TRUE then, if FALSE then)
```

Then we assign to a **vector** three colours already present in R

```
plot_colours <- c("firebrick", "royalblue", "mediumseagreen")
```



```
plot(db$Genome.Length,db$Contigs,col=plot_colours[db$group],pch=17)
abline(v=4400000, col="royalblue", lwd=3, lty=2)
abline(v=4800000, col="firebrick", lwd=3, lty=2)
```

# Legend

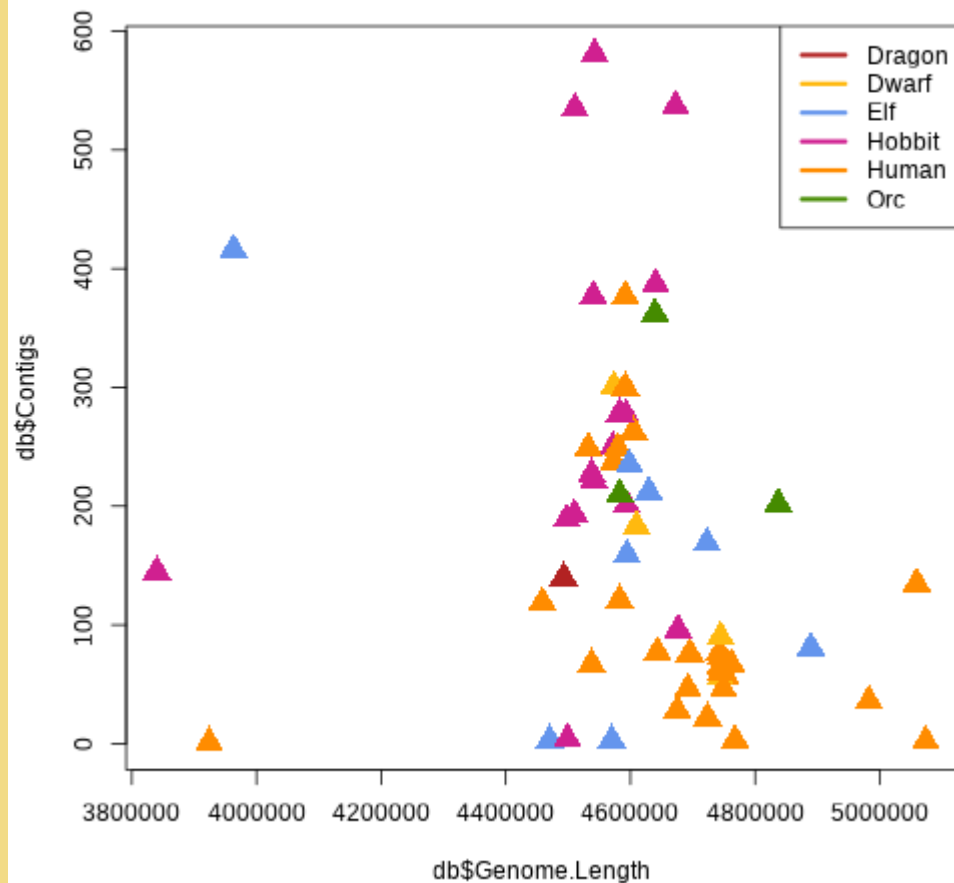
We can add a legend to a plot

First we draw the plot, then we use the function *legend*

```
legend(position, legend=vector_of_labels, col=vector_of_colours)
```

The position can be specified with any of these **strings**

"bottomright", "bottom", "bottomleft", "left",  
"topleft", "top", "topright", "right" "center"



```
legend("topright", legend=levels(as.factor(db$Source)),
      lwd = 3, col = s_colours)
```

# Exercises

Make a histogram of the *PATRIC.CDS*

















Make a barplot to draw the number of samples for each *Species*

Draw a scatterplot using the *GC.Content* and *PATRIC.CDS*

# Extra - Plotting

You can chose how to draw points using different **pch** values

Combine shapes and colours to make the plot more readable

<b>0</b> 	<b>1</b> 	<b>2</b> 	<b>3</b> 	<b>4</b> 	
<b>5</b> 	<b>6</b> 	<b>7</b> 	<b>8</b> 	<b>9</b> 	
<b>10</b> 	<b>11</b> 	<b>12</b> 	<b>13</b> 	<b>14</b> 	
<b>15</b> 	<b>16</b> 	<b>17</b> 	<b>18</b> 	<b>19</b> 	
<b>20</b> 	<b>21</b> 	<b>22</b> 	<b>23</b> 	<b>24</b> 	<b>25</b> 

# Extra - Plotting

The same is valid for **lty**, used to specify the type of lines in a plot

6. 'twodash'	- - - - -
5. 'longdash'	- - - - - .
4. 'dotdash'	- . - . - . - . - .
3. 'dotted'	. . . . .
2. 'dashed'	- - - - -
1. 'solid'	—————
0. 'blank'	



# Extra - Data visualisation

You can manipulate single elements in the plot

You can add lines, curves, labels and other elements

For example you can add a line at a specific value

```
abline(h = 1, lty = 2)
```

## Extra - resources

You can control many settings and aspect in the graphical output using the library **ggplot2**

This library is extremely useful for data visualisation

<https://r-graphics.org/> is a good reference for what you make with **ggplot2**

## Extra - Other resources

<https://socviz.co/>

Introduction to Data visualisation, principles to use to make effective plots