Intro to R - Part I

Install RStudio

First, install the R framework. Go to the CRAN website and in the section *Download and Install R* choose a distribution for your operating system.

In order to install the RStudio, go to the download page of the RStudio website. Download *Installers for Supported Platforms*, choose an installer for your operating system. Run the downloaded installer and complete the installation process.

Vectors

A datum occurring by itself in an expression is taken as a vector of length one. Following are the most used classes of vectors.

Numeric

```
x <- 41.5
x

## [1] 41.5

print(class(x))

## [1] "numeric"
```

Integer

```
x <- 5
x

## [1] 5

print(class(x))

## [1] "numeric"

x <- 5L
print(class(x))

## [1] "integer"
```

Character

```
x <- "Hello"
x

## [1] "Hello"
```

```
print(class(x))

## [1] "character"

x == 'Hello' # use either single or double quotes

## [1] TRUE
```

Logical (true/false)

```
x <- TRUE
x

## [1] TRUE

print(class(x))

## [1] "logical"
```

Creating vector with more elements

When you want to create a vector with more than one element, use the *c()* (short for concatenate) function:

```
v <- c(1.5, 5, 3)
v

## [1] 1.5 5.0 3.0
```

Vector arithmetics

An arithmetic operation on a vector results in a vector containing values of that operation applied to each element of the vector.

```
# (+, -, *, /, ^)
v + 1

## [1] 2.5 6.0 4.0

-v

## [1] -1.5 -5.0 -3.0

2*v + 2

## [1]  5 12  8

v/2

## [1] 0.75 2.50 1.50

v^2

## [1]  2.25 25.00  9.00
```

Common arithmetic functions are available: *log*, *exp*, *sin*, *cos*, *tan*, *sqrt*, …

max and *min* select the largest and smallest elements of a vector, respectively. *range* is a function that returns a vector containing the minimum and maximum of all the given arguments, namely *c(min(v), max(v))*. The function *length(v)* returns the number of elements in a vector, while the *sum(v)* gives the total of the elements in a vector, and *prod(v)* their product [2].

```
v1 <- c(1, 2, 3, 4, 5)

# get the max value
max(v1)

## [1] 5

# get the min value
min(v1)

## [1] 1

# get the range of values
range(v1)

## [1] 1 5

# calculate the sum of all elements
sum(v1)

## [1] 15

# calculate the product of all elements
prod(v1)

## [1] 120
```

Generating regular sequences

For regularly spaced sequences involving integers, it is simplest to use the colon (:) operator. For generating sequences with numeric values, the *seq* should be used.

```
# generate a sequence from 1 to 10
a <- 1:10
a

##  [1]  1  2  3  4  5  6  7  8  9 10

# generate a sequence from 10 to 1
b <- 10:1
b

##  [1] 10  9  8  7  6  5  4  3  2  1
```

```
rev(a)

##  [1] 10  9  8  7  6  5  4  3  2  1

# generate a sequence from 3.2 to 4.7, with a step 0.2
seq(3.2, 4.7, by = 0.2)

## [1] 3.2 3.4 3.6 3.8 4.0 4.2 4.4 4.6
```

Factor values

Factors are used to represent categorical variables (variables with a limited number of different values). Factors are stored as a vector of integer values with a corresponding set of character values to use when the factor is displayed.

```
# create a vector and convert to factor
v2 <- c("cold", "mild", "mild", "hot", "cold")
temp <- as.factor(v2)

# print unique factor values (levels)
levels(temp)

## [1] "cold" "hot"  "mild"

# specify the order of levels
temp <- factor(v2, levels = c('cold', 'mild', 'hot'))
levels(temp)

## [1] "cold" "mild" "hot"

# print the summary
summary(temp)

## cold mild  hot
##    2    2    1
```

Missing values

When an element or a value is “not available” or is a “missing value” in the statistical sense, a place within a vector may be reserved for it by assigning it with the special value *NA*. The function *is.na(v)* gives a logical vector of the same size as the vector with value TRUE if and only if the corresponding element in the vector is *NA*. [2].

```
v3 <- c(1, NA, 3, 4, NA)

# check which values are NAs
is.na(v3)

## [1] FALSE  TRUE FALSE FALSE  TRUE

sum(is.na(v3))
```

```
## [1] 2
```

The second kind of *missing* values which are produced by numerical computation, the so-called *Not a Number* (NaN) values [2].

```
# NaN missing value
x <- 0/0
is.na(x)

## [1] TRUE

is.nan(x)

## [1] TRUE
```

is.na(xx) is TRUE both for NA and NaN values. *is.nan(xx)* is only TRUE for NaNs.

Data frame

Data frames are matrix-like structures, where the columns can be of different types. It consists of a list of vectors of equal length. It is used for storing data tables.

```
# create a data frame
data1 <- data.frame(
  days = c("Mon", "Tue", "Wen", "Thu", "Fri"),
  temp = c(25, 28, 30, 29, 34))
data1

##   days temp
## 1  Mon   25
## 2  Tue   28
## 3  Wen   30
## 4  Thu   29
## 5  Fri   34

# create a data frame from existing vectors
data2 <- data.frame(v1, v2, v3)
data2

##   v1   v2 v3
## 1  1 cold  1
## 2  2 mild NA
## 3  3 mild  3
## 4  4  hot  4
## 5  5 cold NA
```

Task 1

Create a dataframe called *co2.emissions* containing two columns: *years* (2000, 2002, 2004, 2006, 2008, 2010) and *emission* (2.7, 2.9, 4, 4.9, 5.3, 6.2).

Answer:

```
co2.emissions <- data.frame(
  year = c(2000, 2002, 2004, 2006, 2008, 2010),
  emission = c(2.7, 2.9, 4, 4.9, 5.3, 6.2)
)
co2.emissions

##   year emission
## 1 2000      2.7
## 2 2002      2.9
## 3 2004      4.0
## 4 2006      4.9
## 5 2008      5.3
## 6 2010      6.2
```

Loading data frame from a file

Data can be loaded from a file. The most used file formats are CSV (Comma Separated Values) and TSV (Tab Separated Values). In order to read a CSV file *data/beatles_v1.csv*, we can use the *read.csv* function. By setting the argument *stringsAsFactors* to FALSE, strings will not be converted to factors (which is the default setting).

read.csv

```
# reading a data frame from the CSV file
beatles <- read.csv("data/beatles_v1.csv", stringsAsFactors = FALSE)
beatles

##                         Title Year Duration
## 1             12-Bar Original 1965      174
## 2           A Day in the Life 1967      335
## 3          A Hard Day's Night 1964      152
## 4 A Shot of Rhythm and Blues 1963      104
## 5           A Taste of Honey 1963      163
## 6         Across the Universe 1968      230
## 7              Act Naturally 1965      139
## 8             Ain't She Sweet 1961      150
## 9          All I've Got to Do 1963      124
```

Inspecting data frame

The *head* and *tail* functions print first and last several items, respectively, and are often used to get a first glance at a loaded dataset.

```
# get first several rows
head(beatles)

##                         Title Year Duration
## 1             12-Bar Original 1965      174
## 2           A Day in the Life 1967      335
## 3          A Hard Day's Night 1964      152
## 4 A Shot of Rhythm and Blues 1963      104
## 5           A Taste of Honey 1963      163
## 6        Across the Universe 1968      230
```

```
# get last several rows
tail(beatles)
```

```
##                         Title Year Duration
## 4 A Shot of Rhythm and Blues 1963      104
## 5           A Taste of Honey 1963      163
## 6        Across the Universe 1968      230
## 7              Act Naturally 1965      139
## 8            Ain't She Sweet 1961      150
## 9         All I've Got to Do 1963      124
```

The *str* function is used to inspect the structure of a data frame.

```
# str function
str(beatles)
```

```
## 'data.frame':    9 obs. of  3 variables:
##  $ Title   : chr  "12-Bar Original" "A Day in the Life" "A Hard Day's 
Night" "A Shot of Rhythm and Blues" ...
##  $ Year    : int  1965 1967 1964 1963 1963 1968 1965 1961 1963
##  $ Duration: int  174 335 152 104 163 230 139 150 124
```

```
# print the number of rows
nrow(beatles)
```

```
## [1] 9
```

```
# print the number of columns
ncol(beatles)
```

```
## [1] 3
```

Produce a summary of data.

```
# summary function
summary(beatles)
```

```
##     Title                Year         Duration    
##  Length:9           Min.   :1961   Min.   :104.0  
##  Class :character   1st Qu.:1963   1st Qu.:139.0  
##  Mode  :character   Median :1964   Median :152.0  
##                     Mean   :1964   Mean   :174.6  
##                     3rd Qu.:1965   3rd Qu.:174.0  
##                     Max.   :1968   Max.   :335.0
```

colnames() function prints column names.

```
# get column names
names(beatles)
```

```
## [1] "Title"    "Year"     "Duration"
```

We can similarly change column names.

```
# make a copy of the original data frame
beatles1 <- beatles

# update column names
names(beatles1) <- c("song_name", "release_year", "duration")
beatles1
```

```
##                     song_name release_year duration
## 1             12-Bar Original         1965      174
## 2           A Day in the Life         1967      335
## 3          A Hard Day's Night         1964      152
## 4 A Shot of Rhythm and Blues         1963      104
## 5            A Taste of Honey         1963      163
## 6         Across the Universe         1968      230
## 7               Act Naturally         1965      139
## 8             Ain't She Sweet         1961      150
## 9          All I've Got to Do         1963      124
```

Removing columns

```
# remove column duration
beatles1$duration <- NULL
beatles1
```

```
##                     song_name release_year
## 1             12-Bar Original         1965
## 2           A Day in the Life         1967
## 3          A Hard Day's Night         1964
## 4 A Shot of Rhythm and Blues         1963
## 5            A Taste of Honey         1963
## 6         Across the Universe         1968
## 7               Act Naturally         1965
## 8             Ain't She Sweet         1961
## 9          All I've Got to Do         1963
```

Subsetting

Print a specific element from a data frame. Indexes start from 1.

```
# get the title (column 1) of the 3rd song (3rd row)
song3_title <- beatles[3, 1]
song3_title
```

```
## [1] "A Hard Day's Night"
```

Retrieve a specific row or several rows by index.

```
# get the third row
beatles.subset <- beatles[3,]
beatles.subset

##                Title Year Duration
## 3 A Hard Day's Night 1964      152

# get rows at positions from 3 to 6
beatles.subset1 <- beatles[3:6,]
beatles.subset1

##                        Title Year Duration
## 3         A Hard Day's Night 1964      152
## 4 A Shot of Rhythm and Blues 1963      104
## 5           A Taste of Honey 1963      163
## 6        Across the Universe 1968      230

# get rows at positions 3 and 6
beatles.subset2 <- beatles[c(3,6),]
beatles.subset2

##                 Title Year Duration
## 3  A Hard Day's Night 1964      152
## 6 Across the Universe 1968      230
```

Retrieve a specific column by index or by name

```
# get the second column
years <- beatles[,2]
years

## [1] 1965 1967 1964 1963 1963 1968 1965 1961 1963

# get the Year column
years <- beatles$Year
years

## [1] 1965 1967 1964 1963 1963 1968 1965 1961 1963
```

Lastly, we can retrieve rows with a logical index vector.

```
# retrieve all songs released before year 1965
songsBefore1965 <- beatles[beatles$Year < 1965,]
songsBefore1965

##                        Title Year Duration
## 3         A Hard Day's Night 1964      152
## 4 A Shot of Rhythm and Blues 1963      104
## 5           A Taste of Honey 1963      163
## 8            Ain't She Sweet 1961      150
## 9         All I've Got to Do 1963      124
```

```
# alternative
songsBefore1965 <- subset(beatles, Year < 1965)
songsBefore1965

##                         Title Year Duration
## 3         A Hard Day's Night 1964      152
## 4 A Shot of Rhythm and Blues 1963      104
## 5           A Taste of Honey 1963      163
## 8            Ain't She Sweet 1961      150
## 9         All I've Got to Do 1963      124

# retrieve all songs released before year 1965 with duration lower than 150
seconds
shortSongsBefore1965 <- beatles[beatles$Year < 1965 & beatles$Duration <
150,]
shortSongsBefore1965

##                         Title Year Duration
## 4 A Shot of Rhythm and Blues 1963      104
## 9         All I've Got to Do 1963      124

# alternative
shortSongsBefore1965 <- subset(beatles, Year < 1965 & Duration < 150)
shortSongsBefore1965

##                         Title Year Duration
## 4 A Shot of Rhythm and Blues 1963      104
## 9         All I've Got to Do 1963      124
```

NOTE: Logical operators & and && both perform logical AND operation. But the shorter form performs elementwise comparisons in much the same way as arithmetic operators (it performs logical AND on all elements of both vectors). The longer form evaluates left to right examining only the first element of each vector. The similar difference is between the | and || operators. Therefore, *use the shorter form* both for AND (&) and OR (|)

Task 2

Print a number of songs from 1963 that last more than 2 minutes (120 seconds).

Answer:

```
nrow(beatles[beatles$Year == 1965 & beatles$Duration > 120,])

## [1] 2
```

Plotting

The most basic function to create a plot is the *plot(x, y)*, where x and y are numeric vectors denoting the x- and y-axes.

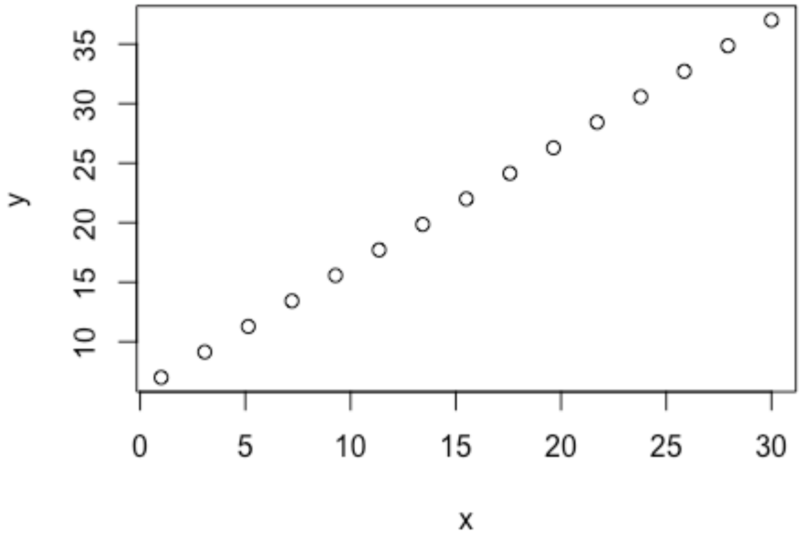
```
# render a basic plot for the given vectors
x <- seq(1, 30, length.out=15)
x

##  [1]  1.000000  3.071429  5.142857  7.214286  9.285714 11.357143 13.428571
##  [8] 15.500000 17.571429 19.642857 21.714286 23.785714 25.857143 27.928571
## [15] 30.000000

y <- seq(5, 35, length.out=15) + 2
y

##  [1]  7.000000  9.142857 11.285714 13.428571 15.571429 17.714286 19.857143
##  [8] 22.000000 24.142857 26.285714 28.428571 30.571429 32.714286 34.857143
## [15] 37.000000

plot(x, y)
```



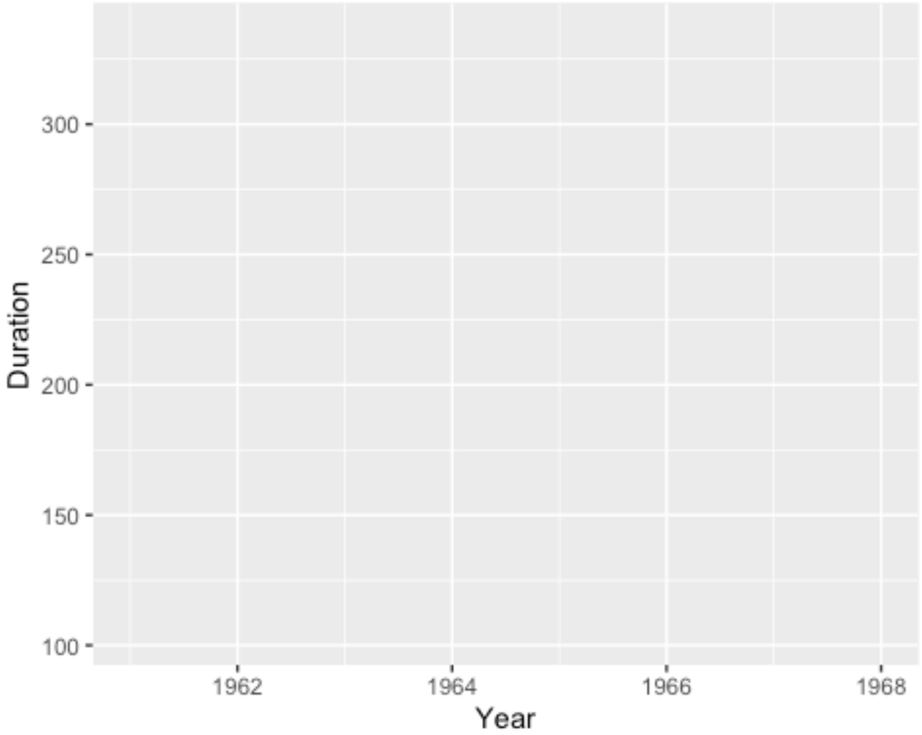
For more advanced plots, ggplot2 library is recommended. *ggplot* function works with data frames and not individual vectors.

When using the ggplot2 for the first time on a specific machine, it needs to be installed, i.e. appropriate packages need to be downloaded from the online repository. After that, the library should be imported (included) with the *library* function.

```
# include ggplot2 library
#install.packages("ggplot2")
library(ggplot2)
```

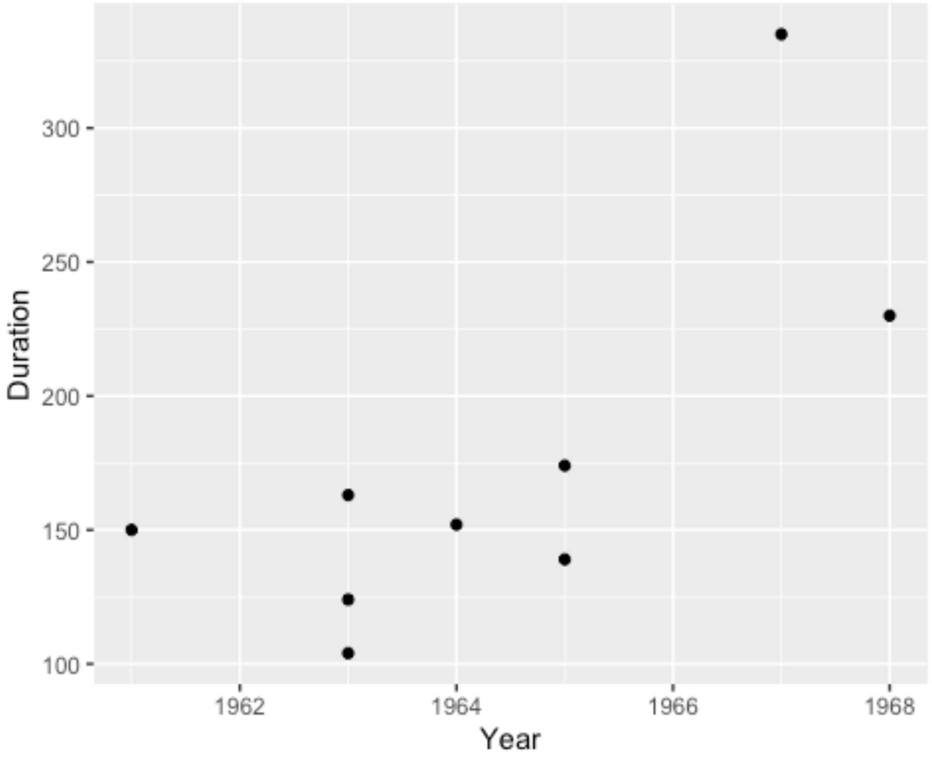
Information that is a part of the data frame has to be specified inside the *aes()* (short for aesthetics) function that specifies x- and y-axes.

```
# render a plot for the given data frame
ggplot(beatles, aes(x=Year, y=Duration))
```



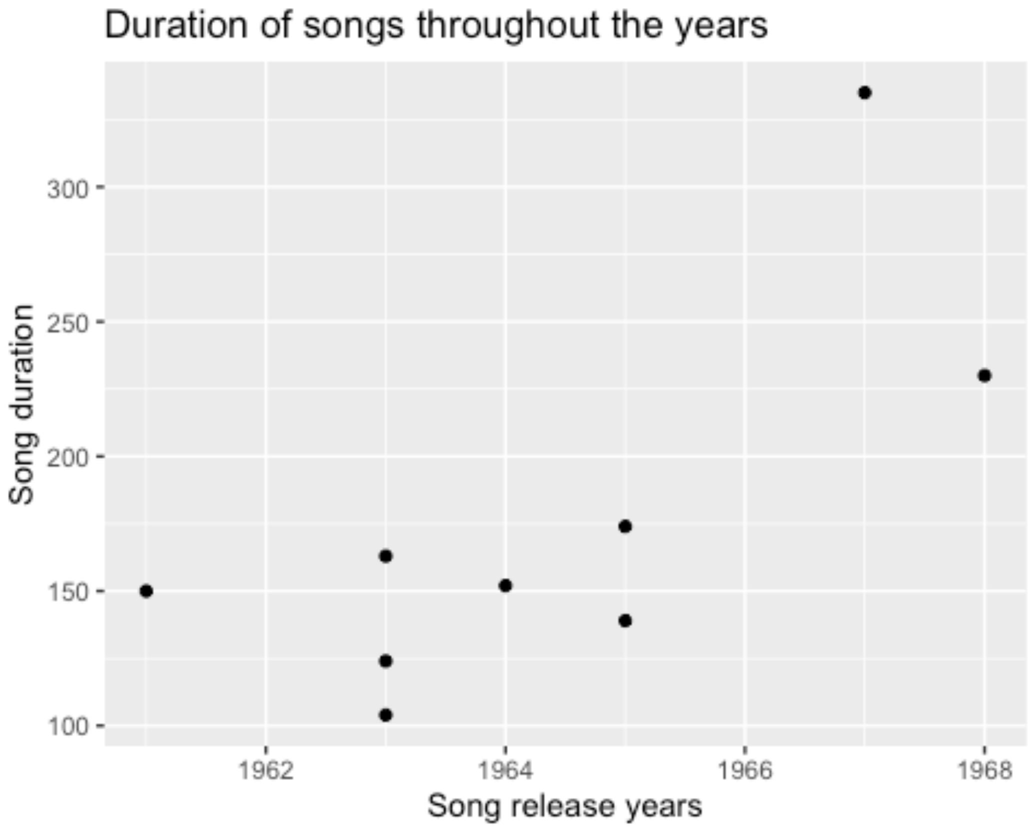
A blank plot is drawn. Even though the x and y are specified, there are no points or lines in it. This is because, *ggplot* doesn't assume which plot we want - a scatterplot, a line plot or some other type of plot. We have only told the *ggplot* what dataset to use and what columns should be used for x- and y-axis. We haven't explicitly asked it to draw any points. Plots are drawn by adding layers to the basic plot generated by the *ggplot* function. More specifically, in order to generate a scatter plot, we use the *geom_point()* layer.

```
# render a scatter plot for Year and Duration
ggplot(beatles, aes(x=Year, y=Duration)) + geom_point()
```



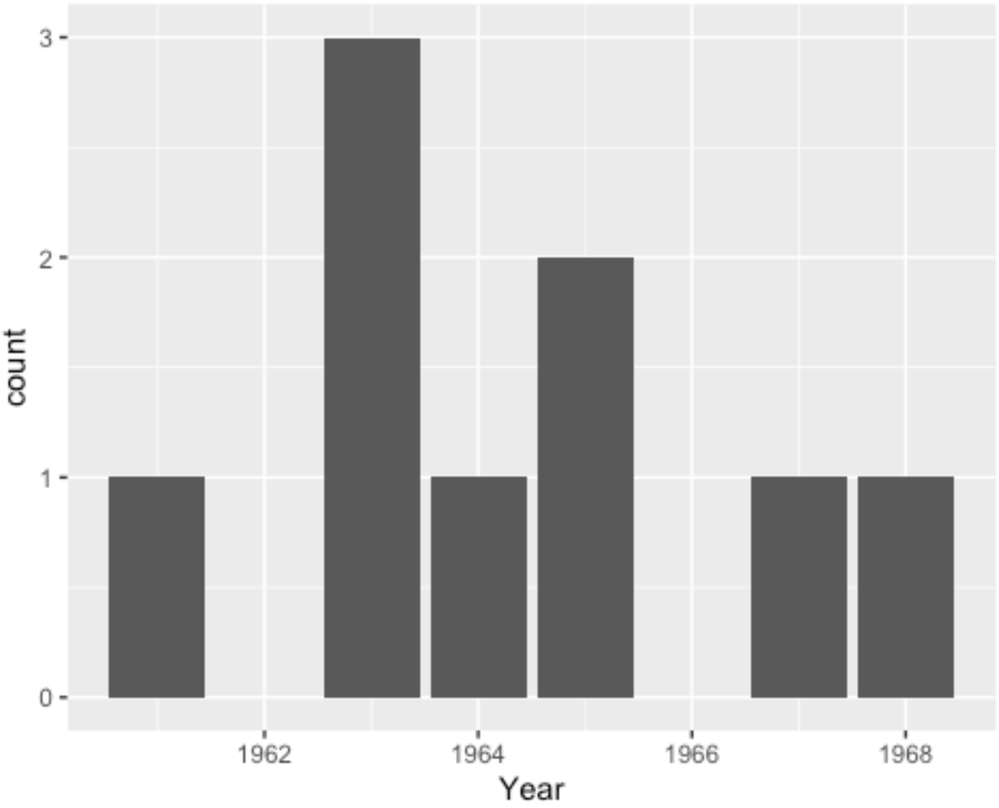
The plot can be further enhanced by adding a custom x- and y-axis labels, and a title.

```r
ggplot(beatles, aes(x=Year, y=Duration)) +
  geom_point() +
  xlab("Song release years") + ylab("Song duration") +
  ggtitle("Duration of songs throughout the years")
```



We can plot a bar chart by adding a *geom_bar()* layer instead of *geom_point()*. *geom_bar()* makes the height of bars proportional to the number of cases in each group, i.e. it makes a bar chart of counts. In this case, we provide only one variable for the x-axis.

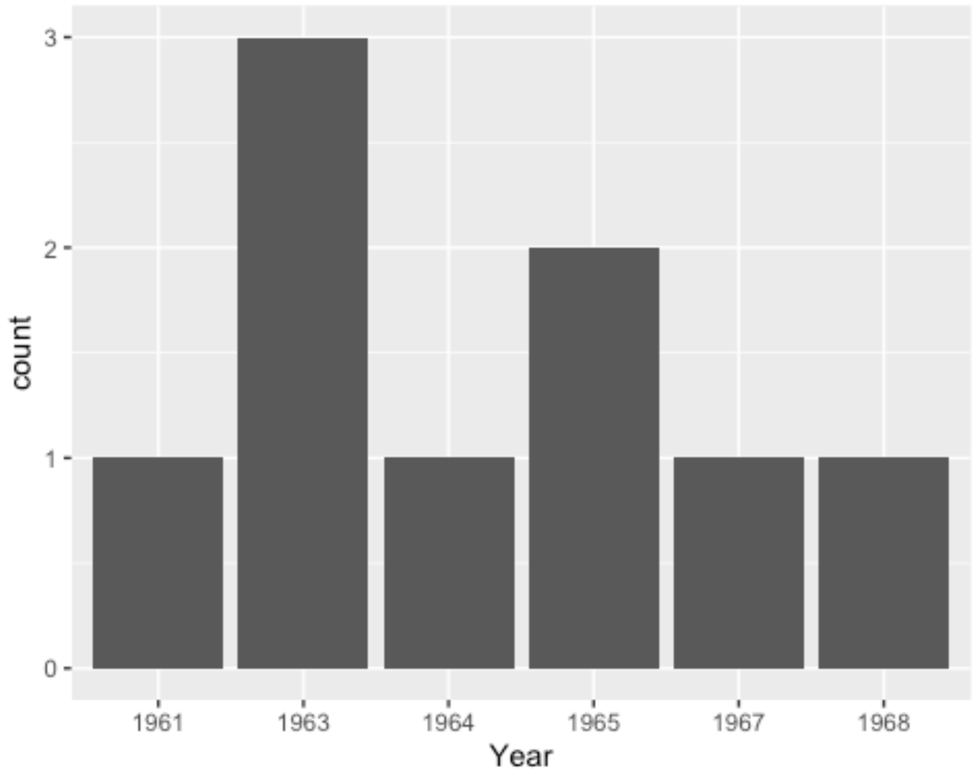
```
# render a bar chart displaying the number of songs in each year
ggplot(beatles, aes(x=Year)) +
  geom_bar()
```



In the previous bar chart, we can observe that the x-axis has all values from 1961 to 1968, which is the value interval of the *Year* attribute. These include the years 1962 and 1966 with no songs. If we want to omit these two values, we can convert the *Year* variable into a factor, and now the *Year* variable will have only the following values: 1961, 1963, 1964, 1965, 1967, and 1968. Only these values will be plotted.

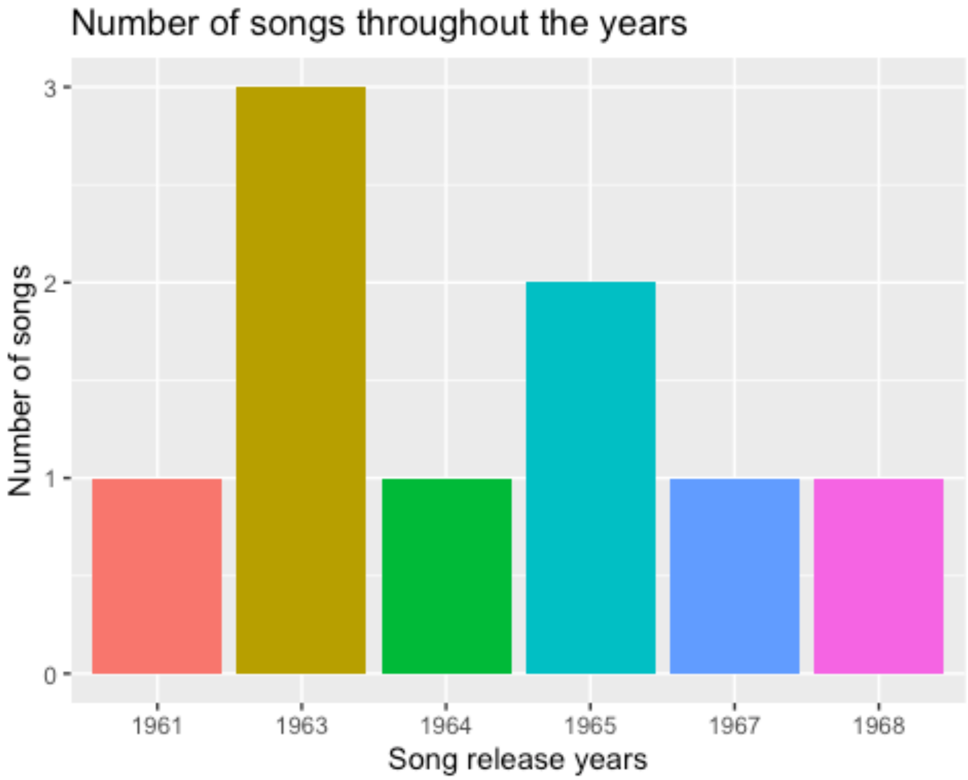
```
# convert the Year variable to factor
beatles$Year <- factor(beatles$Year)

# render a bar chart
ggplot(beatles, aes(x=Year)) +
  geom_bar()
```



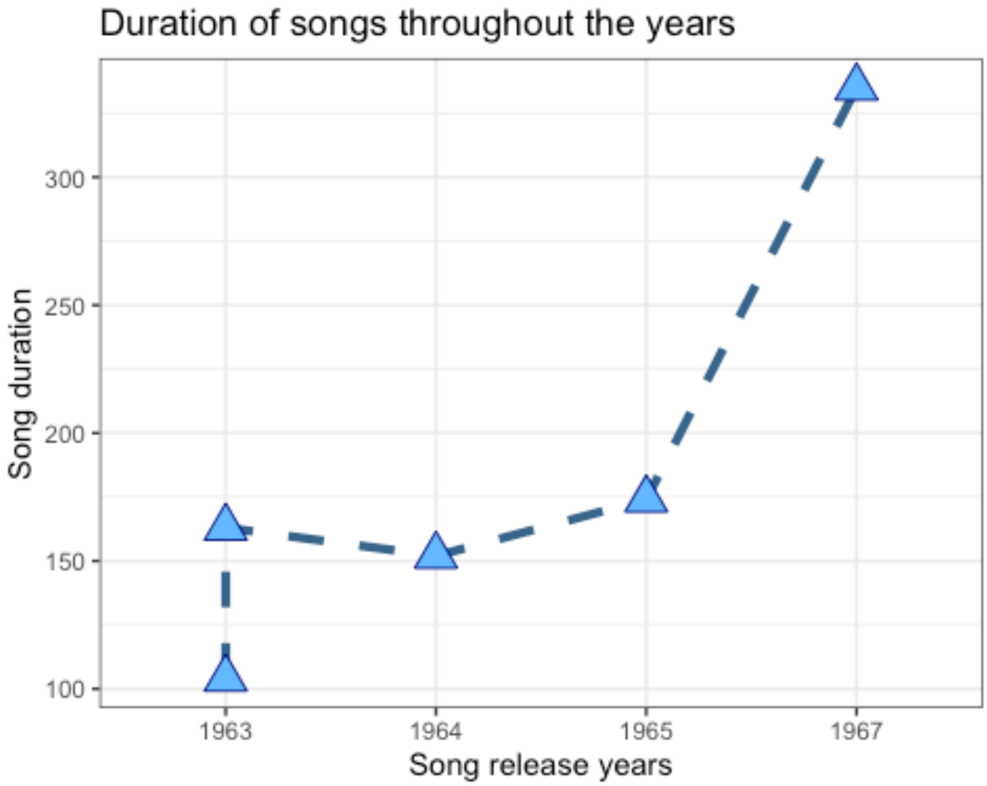
Further enhance the chart by adding a custom x- and y-axis labels, and a chart title. The aesthetic *fill* parameter will take different colouring scales by setting the *fill* to be equal to a factor variable.

```
ggplot(beatles, aes(x=Year, fill = Year)) +
  geom_bar(show.legend = FALSE) +
  xlab("Song release years") + ylab("Number of songs") +
  ggtitle("Number of songs throughout the years")
```



A line chart can be created by using the *geom_line* layer. *geom_line()* tries to connect data points that belong to the same group. Different levels of a factor variable belong to different groups. By specifying *group=1* we indicate we want a single line connecting all the points.

```
# render a line chart for the first five songs with specific line and ponts
properties
ggplot(beatles[1:5,], aes(x=Year, y=Duration, group = 1)) +
  geom_line(colour = "steelblue4", linetype = "dashed", size = 1.5) +
  geom_point(colour="navyblue", size = 5, shape = 24, fill = "steelblue1") +
  xlab("Song release years") + ylab("Song duration") +
  ggtitle("Duration of songs throughout the years") +
  theme_bw()
```



For different shapes and line types, see this reference page: http://www.cookbook-r.com/Graphs/Shapes_and_line_types/

For color palettes, see this R Color Cheatsheet.

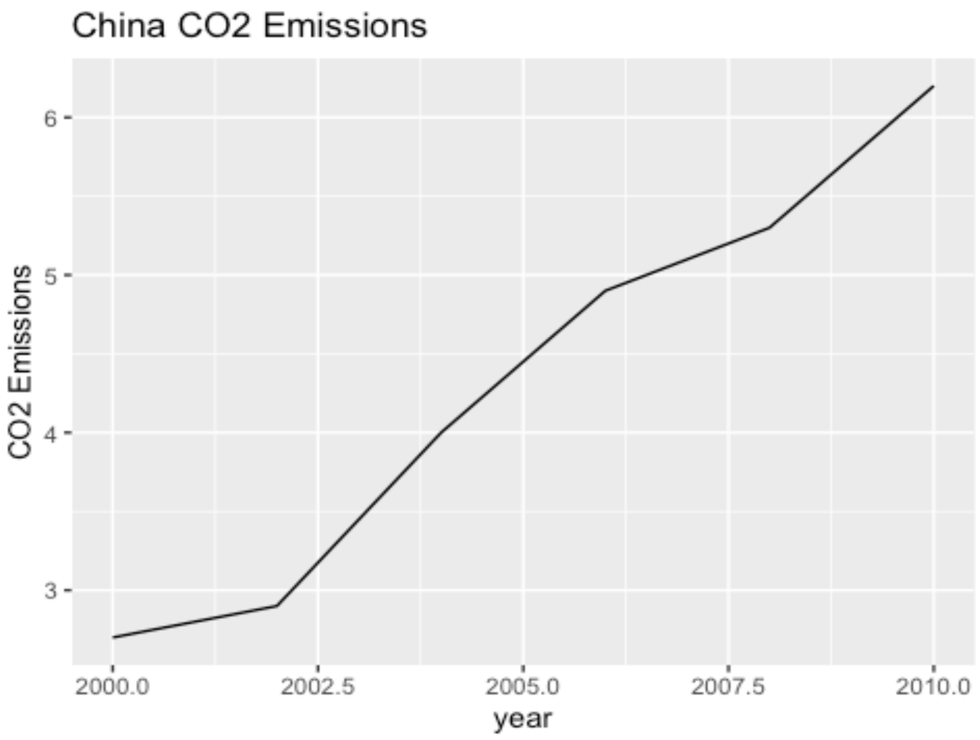
Task 3

Create a line chart from a dataset *co2.emissions* (created in Task 1) with the x-axis representing the years, and the y-axis representing the values of CO2 emissions.

Chart title should be "China CO2 Emissions" and y-axis should have a label "CO2 Emissions".

Answer:

```
ggplot(co2.emissions, aes(x = year, y = emission, group = 1)) +
  geom_line()  +
  ggtitle("China CO2 Emissions") +
  ylab("CO2 Emissions")
```



Homework - Complete interactive R tutorials with Swirl

Swirl is an interactive R tutorial that teaches you R from the R console. All you need to do is install Swirl package for R and issue a *swirl* command which will start the tutorial.

```
#install.packages("swirl")
library("swirl")

##
## | Hi! I see that you have some variables saved in your workspace. To keep
## | things running smoothly, I recommend you clean up before starting swirl.
##
## | Type ls() to see a list of the variables in your workspace. Then, type
## | rm(list=ls()) to clear your workspace.
##
## | Type swirl() when you are ready to begin.

#swirl()
```

Once a Swirl session is started, you will be prompted with an option to install a Swirl course. For our course, you need to install the *R Programming: The basics of programming in R* Swirl course and go through the following tutorials:

1. Basic Building Blocks
2. Workspace and Files
3. Sequences of Numbers

4. Vectors
5. Missing Values
6. Subsetting Vectors
7. Matrices and Data Frames
8. Logic
9. Functions
10. Looking at Data

References

[1] https://www.tutorialspoint.com/r/r_data_types.htm [2] An Introduction to R, https://cran.r-project.org/doc/manuals/R-intro.pdf