R_commands

PACKAGES

Install and load a package

```
install.packages("package_name")
library(package_name)
```

VARIABLES AND VECTORS

Create a variable

```
variable_name <- "object_name"
```

Create a vector

```
vector_name <- c("object_1", "object_2", "object_3", ...)
```

DATAFRAMES

Read a file

```
read.csv("File_name", header = T, sep="symbol", quote = "symbol",
         dec = "symbol", na.strings = value/values)
read.table("File_name", header = T, sep = "symbol",
           quote = "symbol", dec = "symbol", na.strings = value/values)
# requires to load the package "readxl"
read_excel("File_name", sheet = number, na = value/values)
```

Write a file

```
write.csv(dataframe_name, file="Output_name.csv", row.names=F)
```

To compactly display the structure of a dataframe

```
head(dataframe_name)
str(dataframe_name)
summary(dataframe_name)
```

Get row names and column names

```
rownames(dataframe_name)
colnames(dataframe_name)
```

Check the object type

```
class(object_name)
```

Change the object type

```
as.character(dataframe_name$column_name)
as.factor(dataframe_name$column_name)
as.numeric(dataframe_name$column_name)
as.integer(dataframe_name$column_name)
as.data.frame(matrix_name)
```

Check if a value is present in a column

```
"value" %in% dataframe_name$column_name
```

Get an object from a column:

```
dataframe_name$column_name[row_position]
```

Extract the first location that has the value of interest

```
match("value", dataframe_name$column_name)
```

Extract all locations that have the value of interest

```
which(dataframe_name$column_name %in% "value")
```

Modify an object from a column

```
dataframe_name$column_name[position in the column] <- "new_value"
```

Extract objects from a dataframe

```
Specific object:dataframe_name["row_name", "column_name"]
dataframe_name[row_index, column_index]
```

Single column

```
dataframe_name$column_name
dataframe_name[ , "column_name"]
dataframe_name[ , column_index]
```

Single row

```
dataframe_name["row_name", ]
dataframe_name[row_index, ]
```

Get more columns

```
dataframe_name[ , c("column_1_name", "column_2_name")]
dataframe_name[ , c(first_column_index, second_column_index)]
```

Get more rows

```
dataframe_name[c("row_1_name", "row_2_name"), ]
dataframe_name[c(first_row_index, second_row_index), ]
```

Get a range of columns

```
dataframe_name[ , c(first_column_index : last_column_index)]
```

Get a range of rows

```
dataframe_name[c(first_row_index : last_row_index), ]
```

Add a new column

```
dataframe_name$column_name <- new_column
cbind(dataframe_name, new_column)
cbind(new_column, dataframe_name)
cbind(dataframe_name[range_index_previous_column], new_column_name,
dataframe_name[range_indices_following_columns])
```

Remove a column

```
dataframe_name$column_name <- NULL
```

Combine two columns into a new one

```
paste(dataframe_name$column_name_1, dataframe_name$column_name_2, sep="value")
```

Separate one column into two columns

```
library(tidyr)
separate(dataframe_name, column_name, into=c("column_name_1", "column_name_2"), sep= "value")
```

Get the largest element of a column

```
max(dataframe_name$column_name)
```

Get the smallest element of a column

```
min(dataframe_name$column_name)
```

Get minimum and maximum of all the given arguments:

```
range(dataframe_name$column_name)
```

Get the length of a column

```
length(vector_name)
```

Get a vector of unique values

```
unique(dataframe_name$column_name)
```

Sort the values in a column

```
sort(dataframe_name$column_name)
```

Combine two data frames

```
rbind(dataframe_1_name, dataframe_2_name)
```

Rename a column

```
colnames(dataframe_name)[column_index] <- "new_column_name"
```

Create a new dataframe from another one

```
cbind(dataframe_name$column_1, dataframe_name$column_2, dataframe_name$column_3)
```

Merge two dataframes on the basis of a column

```
new_dataframe_name <- merge(dataframe_1_name, dataframe_2_name, by = column_name)
```

Subset

```
subset(dataframe_name, dataframe_name$column logical_operator logical expression)
subset(dataframe_name, dataframe_name$column logical_operator logical expression &
dataframe_name$column2 logical_operator logical expression2)
subset(dataframe_name, dataframe_name$column logical_operator logical expression |
dataframe_name$column2 logical_operator logical expression2)
```

Drop levels:

```
subset_dataframe_name <- droplevels(subset_dataframe_name)
```

Build a contingency table

```
table(dataframe_name$column_1, dataframe_name$column_2)
```

GRAPHS

Visualize a distribution:

```
# Only the column is mandatory
hist(dataframe_name$column, breaks = "value", col = "colour_name",
     border = "color_name", main = "Title",xlab = "x_name", ylab = "y_name")
```

Visualize a barplot

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

Visualize a distribution of grouped values

```
# Only the table value is mandatory
barplot(table, col = "colour_name", beside = TRUE, cex.names = number,
        las = number, main = "Title", xlab ="x_name", ylab = "y_name")
```

Visualize a scatterplot

```
# Only the x and y axis are mandatory
plot(x,y, col = "colour_name", main = "Title", xlab = "x_name", ylab = "y_name")
```

Add the legend to a plot

```
legend(coordinates, legend_text, fill = "colour_name", col = "colour_name",
       bg = "colour_name", pch=number)
legend("topright", legend_text, col = "colour_name", pch = 15)
```

Save a graph

```
pdf("Graph_name.pdf")
graph_command
legend_command
dev.off()
```

STATISTICAL ANALYSIS

```
shapiro.test(numeric_vector_data_values)
wilcox.test(first_numeric_vector, second_numeric_vector))
t.test(first_numeric_vector, second_numeric_vector)
table_chi_square <- table(dataframe_name$column_1, dataframe_name$column_2)
chisq.test(table_chi_square)
```

HELP

```
?function_name
help(function_name)
```