Exercise 2

Exercise introduction

This is Exercise 2 in Part 1 of the course.

The purpose of the exercise is to give students more familiarity with `R`.

Basics of R

Use the **sudoku** package to view a data matrix.

```
activatePkgs('sudoku')
```

```
## Loading required package: sudoku
```

```
library(sudoku)
puz <- fetchSudokuUK()
puz
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
##  [1,]    0    0    3    0    0    0    2    1    0
##  [2,]    0    0    0    2    3    0    0    0    4
##  [3,]    8    0    2    0    0    0    0    6    9
##  [4,]    0    0    0    0    0    2    0    0    8
##  [5,]    0    0    0    6    0    1    0    0    0
##  [6,]    9    0    0    5    0    0    0    0    0
##  [7,]    7    5    0    0    0    0    4    0    3
##  [8,]    4    0    0    0    1    7    0    0    5
##  [9,]    0    3    8    0    0    0    1    0    0
```

```
class(puz)
```

```
## [1] "matrix" "array"
```

Basic data aggregations

Going through some basic data creation and aggregations. This chunk includes vectors as well as a dataframe.

```
x <- c(2, 3, 4, 5, 4, 7, 9, 6, 7, 2)
x
```

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

```
mean(x)
```

```
## [1] 4.9
```

```
median(x)
```

```
## [1] 4.5
```

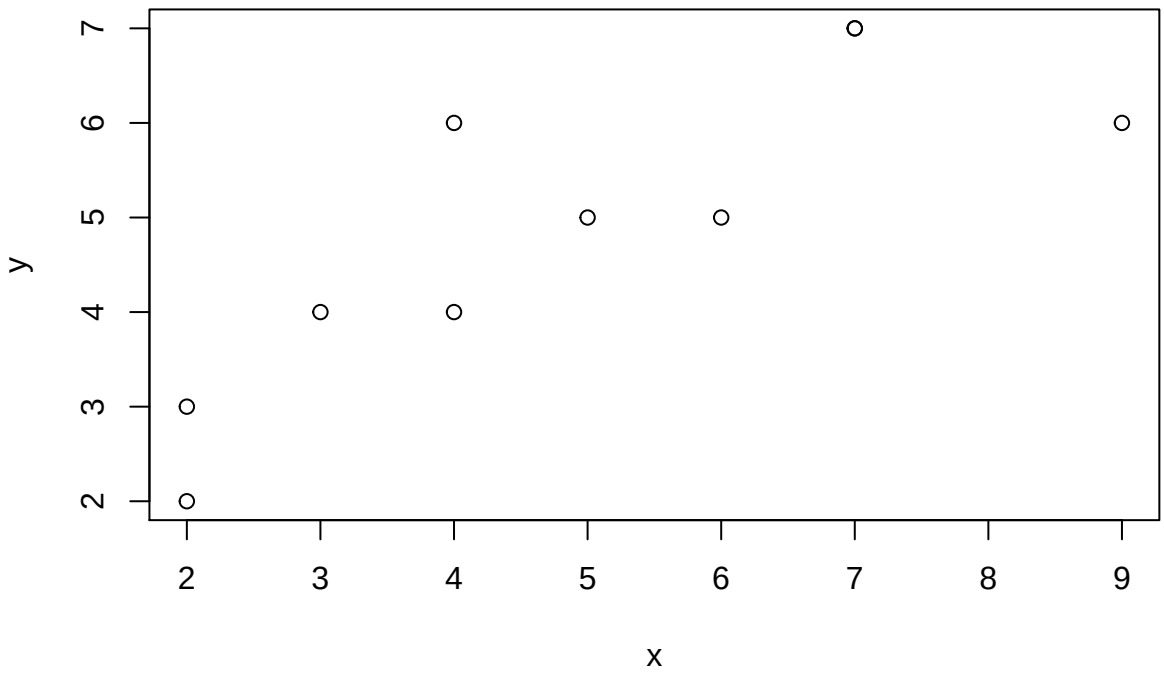
```
y <- c(3, 4, 6, 5, 4, 7, 6, 5, 7, 2)
z <- c(1, 1, 1, 1, 1, 2, 2, 2, 2, 2)

df <- data.frame(x, y, z)

df
```

```
##    x y z
## 1  2 3 1
## 2  3 4 1
## 3  4 6 1
## 4  5 5 1
## 5  4 4 1
## 6  7 7 2
## 7  9 6 2
## 8  6 5 2
## 9  7 7 2
## 10 2 2 2
```

Plotting

R's default `plot` command with continuous data is a scatterplot.

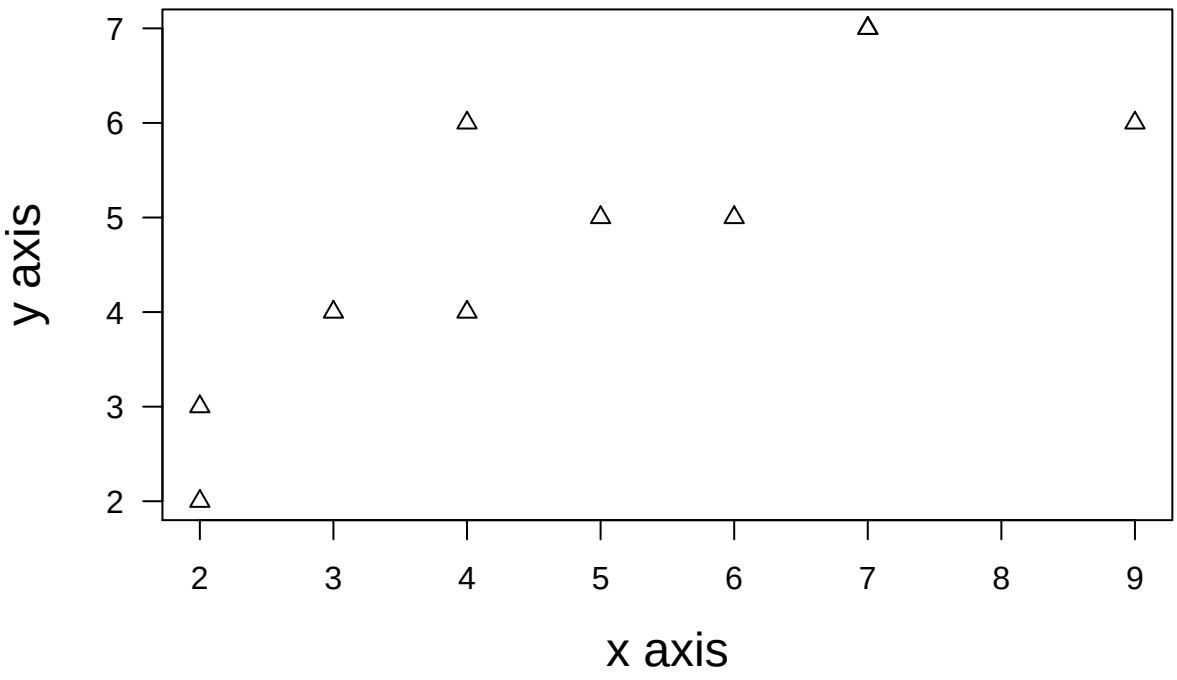
```
plot(y ~ x, data = df)
```



Modifying plot parameters

Modifying parameters will modify the output from the `plot` function.

```
plot(y ~ x, data = df, las = 1, xlab = 'x axis', ylab = 'y axis', cex.lab = 1.5, pch = 2)
```



Creating a simple linear model

Use the function `lm` to create a simple linear model. The `summary` function called upon the linear model will provide information about the residuals, the coefficients for the model, and even a basic t-test.

```
reg <- lm(y ~ x, data = df)

summary(reg)
```

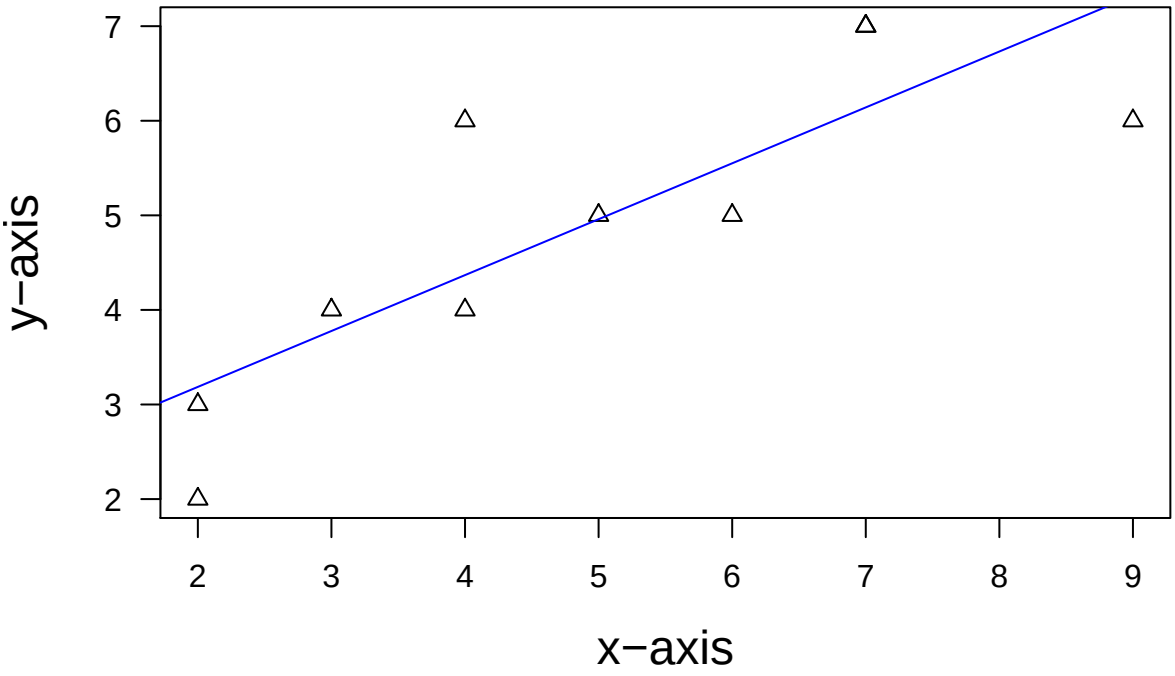
```
##
## Call:
## lm(formula = y ~ x, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.3231 -0.5046 -0.0726  0.6999  1.6319
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.0041     0.7601   2.637  0.02987 *
## x             0.5910     0.1414   4.180  0.00308 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.9887 on 8 degrees of freedom
## Multiple R-squared:  0.6859, Adjusted R-squared:  0.6467
## F-statistic: 17.47 on 1 and 8 DF,  p-value: 0.003079
```

Plotting the linear model

Use the `abline` function to after creating a plot in order to add the linear model to the scatterplot.

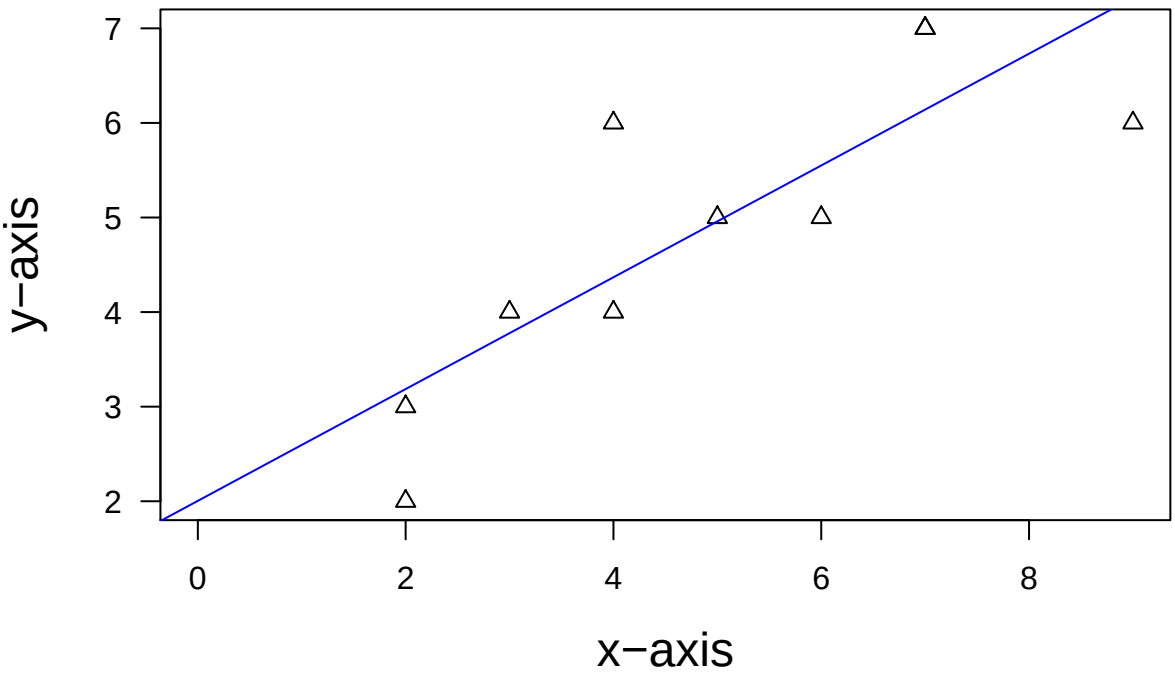
```
plot(y ~ x, data = df, las = 1, xlab = 'x-axis', ylab = 'y-axis', cex.lab = 1.5, pch = 2)
abline(reg, col = 'blue')
```



Adjusting plot limits

If the plot is too short or narrow, one can modify the parameters `ylim` and `xlim` to modify the plot limits.

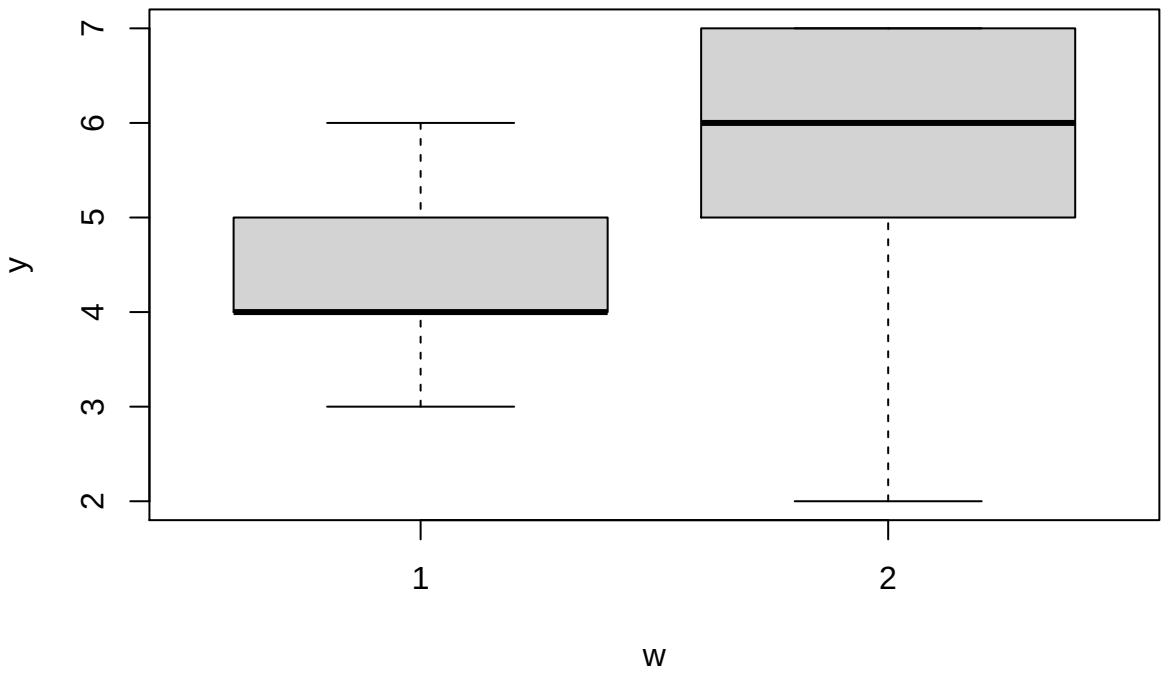
```
plot(y ~ x, data = df, las = 1, xlab = 'x-axis', ylab = 'y-axis', cex.lab = 1.5, pch = 2, xlim = c(0, 9
abline(reg, col = 'blue')
```



Plotting continuous ~ categorical data

If you have continuous data in two or more groups/categories, then a boxplot is a standard way to visualize the results. This is the default output of `R`'s `plot` function if the 'independent' variable in formula is of type `factor`.

```
df$w <- as.factor(df$z)

plot(y ~ w, data = df)
```



gplots package

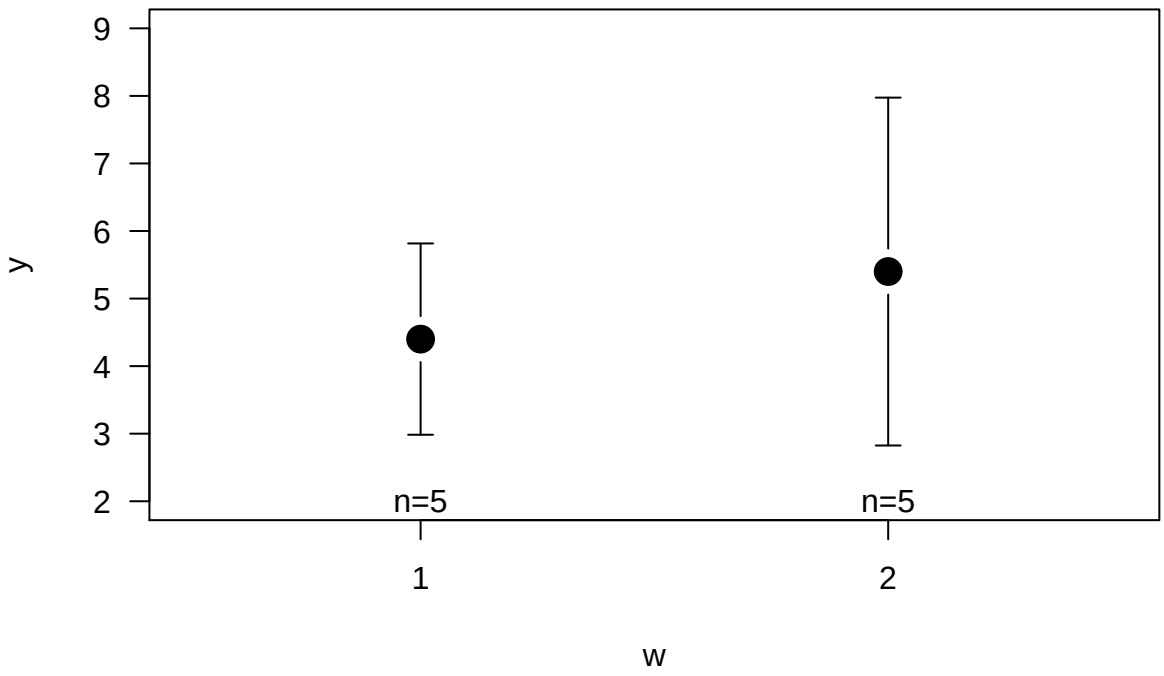
There are many packages for plotting. One is `gplots` which has the function `plotmeans` for plotting means as well as 95% confidence intervals.

```
activatePkgs('gplots')
```

```
## Loading required package: gplots
```

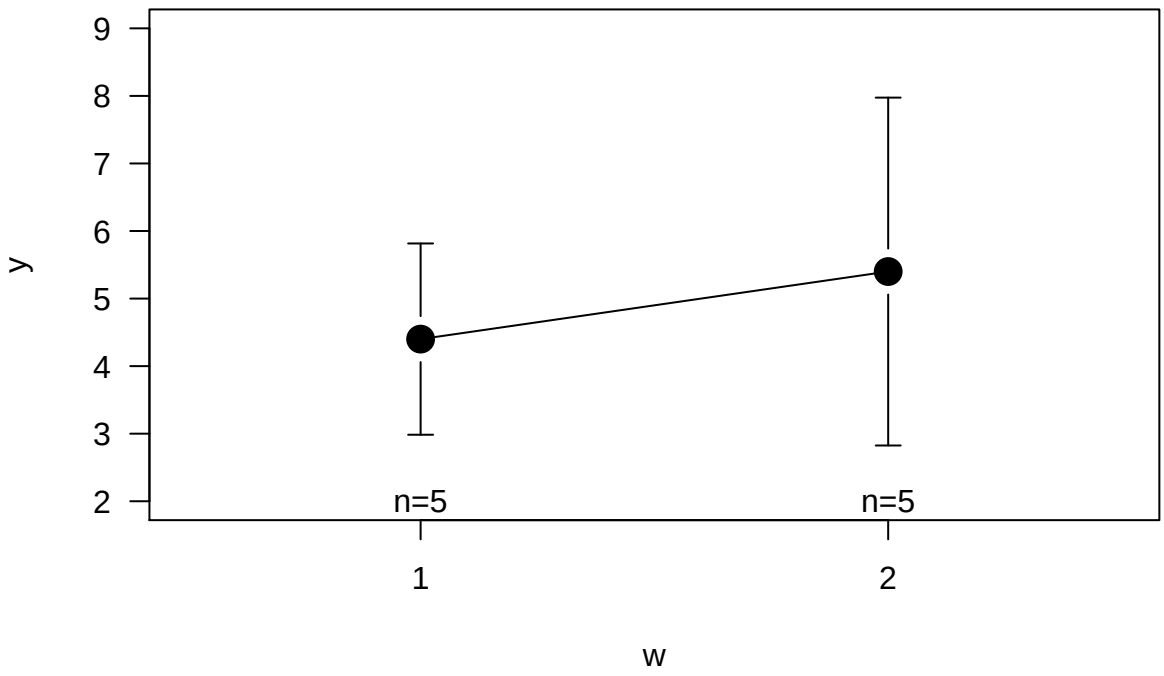
```
## Warning: package 'gplots' was built under R version 4.0.5
```

```
plotmeans(y ~ w, data = df, las = 1, ylim = c(2, 9)
          , connect = F
          , barcol = 'black', cex = 2, pch = 16)
```



The `connect` argument must be set to `FALSE` or else the means will be connected with a line.

```
activatePkgs('gplots')
plotmeans(y ~ w, data = df, las = 1, ylim = c(2, 9)
          # , connect = F
          , barcol = 'black', cex = 2, pch = 16)
```



Aggregations

One can call the `mean` function on multiple vectors in a single call by using the `aggregate` function with the `by` parameter.

```
aggregate(x = df$y, by = list(df$w), FUN = mean)
```

```
##   Group.1   x
## 1       1 4.4
## 2       2 5.4
```

Welch's two-sample t-test

Using `t.test` will also provide the means in addition to the t-test results.

```
t.test(y ~ w, data = df)
```

```
##
##  Welch Two Sample t-test
##
## data:  y by w
## t = -0.94491, df = 6.2161, p-value = 0.38
## alternative hypothesis: true difference in means is not equal to 0
```

```
## 95 percent confidence interval:
##  -3.567906  1.567906
## sample estimates:
## mean in group 1 mean in group 2
##             4.4             5.4
```

Key learnings

- Library `gplots` has a relatively convenient function `plotmeans`
- One can use `aggregate` with `by` and `FUN` parameters to accomplish aggregations on group levels

Unresolved questions

- Which type of t-test is used when calling `summary` on a linear model object?
- How to specify different 'flavors' of t-tests with the `t.test` function? For example, one-sample vs. two-sample, paired vs. unpaired, homoscedasticity assumption fulfilled or not (Levene's adjustment)?