notes5

2022-06-12

Subsetting data frames

```
df1 <- iris[1:3, ] #gives a data frame with only 3 rows of "iris" but all the columns
dim(df1)
```

```
## [1] 3 5
```

```
df2 <- iris[c(1,4), c("Sepal.Length", "Species")]
#here rows 1 and 4 + columns "Sepal.Length" and "Species" have been specifically selected
dim(df2)
```

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

Another way of subsetting is the following:

```
#let's start by selecting the first 5 rows and all columns. This is the data frame
#we're going to subset:
df3 <- iris[1:5, ]

#initialize a logical vector with 5 elements, as the number of rows selected in df3:
x <- c(TRUE, TRUE, FALSE, FALSE, TRUE)

#we subset df3 using x: this will keep only the former two rows and the latter
#(which are "TRUE"), so overall we'll have 3 rows and 5 col
df3[x, ]
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
```

```
#likewise for columns:
df3[ , x]
```

```
##   Sepal.Length Sepal.Width Species
## 1          5.1         3.5  setosa
## 2          4.9         3.0  setosa
## 3          4.7         3.2  setosa
## 4          4.6         3.1  setosa
## 5          5.0         3.6  setosa
```

The previous strategy can be applied when we want to select rows satisfying a certain condition:

```
x <- df3$Sepal.Length >= 5  # x is again a logical vector, as before
x
```

```
## [1]  TRUE FALSE FALSE FALSE  TRUE
```

```
df3[x, ]  # now we've selected only those rows for which Sepal.Length is >= 5
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
```

Which as exactly the same as doing everything at the same time:

```
df3[df3$Sepal.Length>=5, ]
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
```

More advanced:

```
df4 <- iris[iris$Sepal.Length>=5 & iris$Petal.Length>=5, ]
#to select rows that have both Sepal.Length and Petal.Length >= 5

head(df4)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 78           6.7         3.0          5.0         1.7 versicolor
## 84           6.0         2.7          5.1         1.6 versicolor
## 101          6.3         3.3          6.0         2.5  virginica
## 102          5.8         2.7          5.1         1.9  virginica
## 103          7.1         3.0          5.9         2.1  virginica
## 104          6.3         2.9          5.6         1.8  virginica
```

```
dim(df4)
```

```
## [1] 46  5
```

46 flowers (rows) respect both conditions.

```
df5 <- iris[iris$Sepal.Length>=5 | iris$Petal.Length>=5, ]
#different from before as here rows can have either Sepal.Length or Petal.Length >= 5
#(just one of the two conditions, if respected, allows rows selection)

head(df5)
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1         3.5          1.4         0.2  setosa
## 5           5.0         3.6          1.4         0.2  setosa
## 6           5.4         3.9          1.7         0.4  setosa
## 8           5.0         3.4          1.5         0.2  setosa
## 11          5.4         3.7          1.5         0.2  setosa
## 15          5.8         4.0          1.2         0.2  setosa
```

Lapply function

Same as `apply()` but in this case the function is applied to all elements within a list.

```
x <- list(char="String", num=c(2, 3), log=c(FALSE, TRUE, TRUE))  # we generate a list
x
```

```
## $char
## [1] "String"
##
## $num
## [1] 2 3
##
## $log
## [1] FALSE  TRUE  TRUE
```

```
lply <- lapply(x, length)  # we apply the function length() on the list we just generated
```

```
class(lply)
```

```
## [1] "list"
```

```
lply
```

```
## $char
## [1] 1
##
## $num
## [1] 2
##
## $log
## [1] 3
```

`lply` is still a list but with info related to its associated function (length).

Factors

Data frames are lists. Thus, lapply can be used on them too:

```
lapply(iris, class)  # this applies to all the columns of the dataset iris
```

```
## $Sepal.Length
## [1] "numeric"
##
## $Sepal.Width
## [1] "numeric"
##
## $Petal.Length
## [1] "numeric"
##
```

```
## $Petal.Width
## [1] "numeric"
##
## $Species
## [1] "factor"
```

Species column is a "factor" type of object. This class is used in R to represent categorical data indeed. **The possible values a factor can assume are called *levels*.**

```
levels(iris$Species)  # how many levels are there for Species?
```

```
## [1] "setosa"     "versicolor" "virginica"
```

```
table(iris$Species)  # to know how many flowers for each level of Species
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

```
iris_red <- iris[iris$Species == "setosa" | iris$Species == "virginica", ]
#to select rows of flowers of either "setosa" or "virginica" Species

dim(iris_red)
```

```
## [1] 100   5
```

```
#should "versicolor" have disappeared from this newly created data frame?
levels(iris_red$Species)
```

```
## [1] "setosa"     "versicolor" "virginica"
```

It did not, but:

```
table(iris_red$Species)
```

```
##
##     setosa versicolor  virginica
##         50          0         50
```

No flowers for "versicolor". To really forget about this level just type:

```
iris_red$Species <- factor(iris_red$Species)
levels(iris_red$Species)
```

```
## [1] "setosa"    "virginica"
```

Regression with categorical variables

When we introduced regression we dealt with *numeric* variables (both as dependent and independent variables). However, we could also use *categorical* variables. There are two cases:

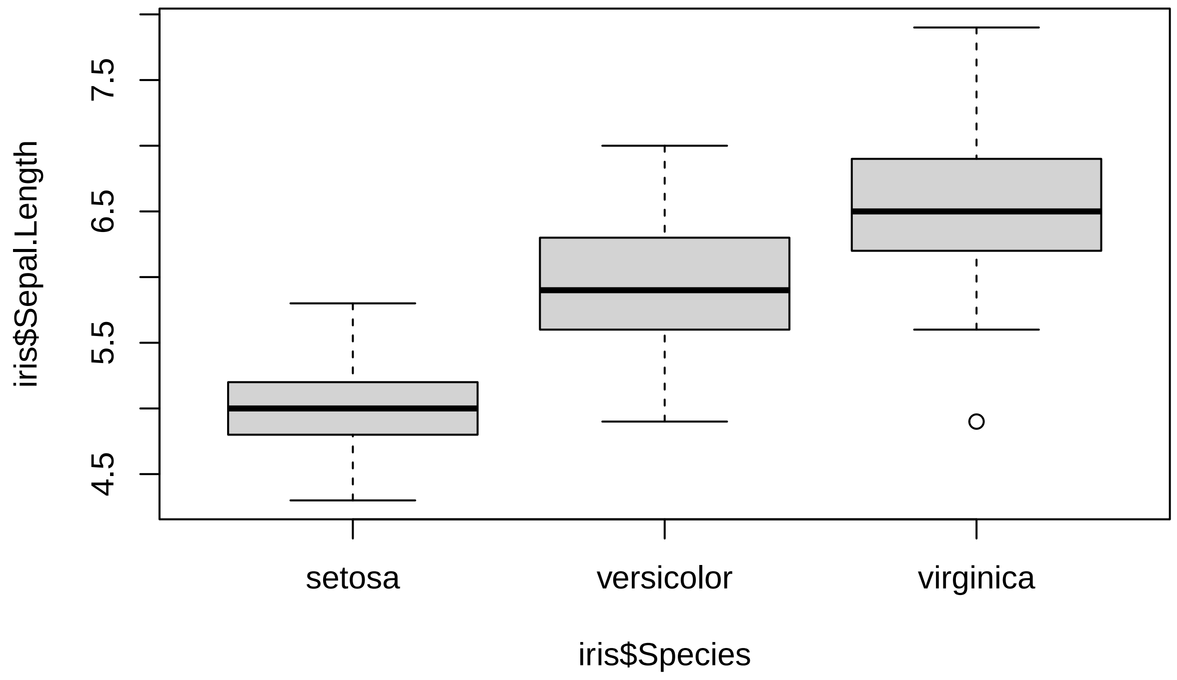
1. prediction of a numerical variable from a categorical variable using linear regression
2. prediction of a categorical variable from a numerical or categorical variable using logistic regression

The choice of the kind of regression is related to the type of dependent variable.

1. Linear regression with independent categorical variables

Question: does the Species (categorical variable) predict Sepal.Length (numerical variable)?

```
plot(iris$Sepal.Length ~ iris$Species)
```



```
#note that this plot() is different from that of before,
#that was plot(x, y) and printed a scatter plot
```

The Species is a *predictor* / *regressor* of Sepal.Length. But how good is this prediction? We'll use *binary* categorical variable for simplicity: thus we need to reduce the iris dataset to include only 2 species ("setosa" and "virginica"), the previously defined "iris_red".

Why Species as a binary variable? Because now we treat it as it was a numerical variable: "setosa" = 0 and "virginica" = 1. This way, an ordinary linear regression can be applied between Sepal.Length and Species:

```r
linreg <- lm(iris_red$Sepal.Length ~ iris_red$Species)
summary(linreg)
```

```
##
## Call:
## lm(formula = iris_red$Sepal.Length ~ iris_red$Species)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6880 -0.2880 -0.0060  0.2985  1.3120
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 5.0060     0.0727   68.85   <2e-16 ***
## iris_red$Speciesvirginica   1.5820     0.1028   15.39   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5141 on 98 degrees of freedom
## Multiple R-squared:  0.7072, Adjusted R-squared:  0.7042
## F-statistic: 236.7 on 1 and 98 DF,  p-value: < 2.2e-16
```

Let's verify that this transformation actually occurred:

```r
#add a new variable (column) to iris_red.
#The 0 will be recycled for all the rows of the dataset (see output)
iris_red$num_species <- 0

head(iris_red)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species num_species
## 1          5.1         3.5          1.4         0.2  setosa           0
## 2          4.9         3.0          1.4         0.2  setosa           0
## 3          4.7         3.2          1.3         0.2  setosa           0
## 4          4.6         3.1          1.5         0.2  setosa           0
## 5          5.0         3.6          1.4         0.2  setosa           0
## 6          5.4         3.9          1.7         0.4  setosa           0
```

```r
#we assign 1, in the num_species column, only to virginica rows (flowers):
iris_red[iris_red$Species == "virginica", "num_species"] <- 1

tail(iris_red)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species num_species
## 145          6.7         3.3          5.7         2.5 virginica           1
## 146          6.7         3.0          5.2         2.3 virginica           1
## 147          6.3         2.5          5.0         1.9 virginica           1
## 148          6.5         3.0          5.2         2.0 virginica           1
## 149          6.2         3.4          5.4         2.3 virginica           1
## 150          5.9         3.0          5.1         1.8 virginica           1
```

```
table(iris_red$num_species)
```

```
##
##  0  1
## 50 50
```

```
#to see how many flowers (rows) have 1 or 0 in the num_species column
#(i.e. how many flowers are "virginica" and how many "setosa")
```

Now we do the linear regression, it should give the same output as before:

```
linreg2 <- lm(iris_red$Sepal.Length ~ iris_red$Species)
s <- summary(linreg2)
s$coefficients
```

```
##                            Estimate Std. Error  t value     Pr(>|t|)
## (Intercept)                   5.006 0.07270432 68.85423 8.453626e-85
## iris_red$Speciesvirginica    1.582 0.10281944 15.38620 6.892546e-28
```

We do have the same results.

Interpretation: being “virginica” adds about 1.58 cm to Sepal.Length with respect to being “setosa”. Uncertainty (Std.error) on this value (1.58) is about 0.1 cm. P-value is small so we’re confident the effect is real and not stochastic.

Differential expression as a regression problem

```
setwd("C:/Users/seren/Desktop/Provero")
load("expr.RData")
head(expr)
```

```
##             mes1     mes2     mes3     mes4     mes5     mes6     neu1     neu2
## RNF14  8.102109 6.431024 7.233297 5.536356 7.980071 8.120170 6.807137 7.481830
## UBE2Q1 9.535052 9.531678 6.666299 6.934365 9.379369 9.046204 9.466937 9.369915
## RNF17  4.149430 4.374377 5.037466 5.135278 4.172273 4.529581 4.609084 4.456974
## RNF10  7.142845 6.882857 5.524656 6.236647 7.379514 6.893025 6.624718 7.057433
## RNF11  9.563294 8.782727 8.522499 6.927014 8.938691 8.707519 8.289256 9.274853
## RNF13  9.552312 9.237758 9.556759 7.951833 9.873957 9.457725 9.294424 9.016314
##              neu3      neu4     neu5      neu6
## RNF14   9.232656  8.474975 9.032406  8.901127
## UBE2Q1  9.171364  9.165563 9.742695  9.236809
## RNF17   4.161883  4.132093 4.293262  4.159367
## RNF10   6.897234  7.229054 6.788821  6.940517
## RNF11   9.317841  9.505812 9.941937  9.478019
## RNF13  10.465560 10.228730 9.599728 10.633883
```

We consider only one gene (the most significantly, differentially expressed one from last time):

```
gene <- expr["SERPINC1", ]
#we did not set 'drop=FALSE', meaning that now 'gene' is no longer a matrix but a vector.

class(gene)
```

```
## [1] "numeric"
```

```
gene
```

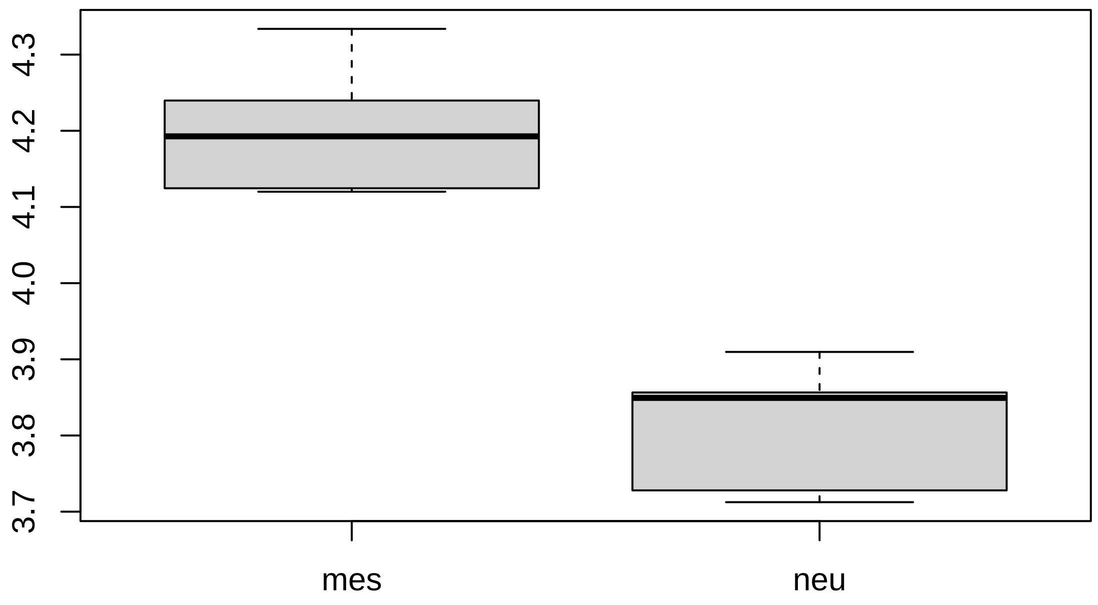
```
##     mes1     mes2     mes3     mes4     mes5     mes6     neu1     neu2
## 4.333796 4.124529 4.239739 4.120013 4.146627 4.238776 3.850684 3.856474
##     neu3     neu4     neu5     neu6
## 3.848018 3.712498 3.909699 3.727955
```

A t-test found this gene differentially expressed (between "mes" and "neu"):

```
t.test(gene[1:6], gene[7:12])
```

```
##
##  Welch Two Sample t-test
##
## data:  gene[1:6] and gene[7:12]
## t = 8.1037, df = 9.9484, p-value = 1.086e-05
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.2776378 0.4884131
## sample estimates:
## mean of x mean of y
##  4.200580  3.817555
```

```
boxplot(gene[1:6], gene[7:12], names = c("mes", "neu"))
```



This is a **regression problem**: is tumour type (categorical variable: either "mes" or "neu") a *predictor* / *regressor* of the expression level of SERPINC1 (numerical variable)?

```
type <- factor(c(rep("mes", 6), rep("neu", 6)))
#to first define the categorical variable tumour type,
#that should be a factor as Species for the iris dataset

class(type)
```

```
## [1] "factor"
```

```
levels(type)
```

```
## [1] "mes" "neu"
```

```
type
```

```
##  [1] mes mes mes mes mes mes neu neu neu neu neu neu
## Levels: mes neu
```

Now that we defined the categorical (binary) variable, let's do the linear regression with our gene:

```r
gene_linreg <- lm(gene ~ type)
summary(gene_linreg)
```

```
##
## Call:
## lm(formula = gene ~ type)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.10506 -0.07718  0.03180  0.03898  0.13322
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.20058    0.03342 125.685  < 2e-16 ***
## typeneu     -0.38303    0.04727  -8.104 1.05e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.08187 on 10 degrees of freedom
## Multiple R-squared:  0.8678, Adjusted R-squared:  0.8546
## F-statistic: 65.67 on 1 and 10 DF,  p-value: 1.052e-05
```

Interpretation: being "neu" decreases expression of SERPINC1 by 0.38 with respect to being "mes". Uncertainty on this value is about 0.05. P-value suggests this effect is real.

Let's compare this latter P-value with that retrieved with the t-test at the beginning:

```r
t.test(gene[1:6], gene[7:12])$p.value
```

```
## [1] 1.085867e-05
```

```r
summary(gene_linreg)$coefficients[2,4]
```

```
## [1] 1.051641e-05
```

The slight difference we can observe is due to the fact that the Welch t-test we used doesn't assume the variance is independent of the *regressor* (`type` in this case) value, while the linear regression assumes same variance in "mes" and "neu". To make the t-test assume this too, type:

```r
t.test(gene[1:6], gene[7:12], var.equal = TRUE)
```

```
##
##  Two Sample t-test
##
## data:  gene[1:6] and gene[7:12]
## t = 8.1037, df = 10, p-value = 1.052e-05
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.277712 0.488339
## sample estimates:
## mean of x mean of y
##  4.200580  3.817555
```

```
#now it assumes the variance is the same in "mes" and "neu"
```

```
t.test(gene[1:6], gene[7:12], var.equal = TRUE)$p.value
```

```
## [1] 1.051641e-05
```

```
summary(gene_linreg)$coefficients[2,4]
```

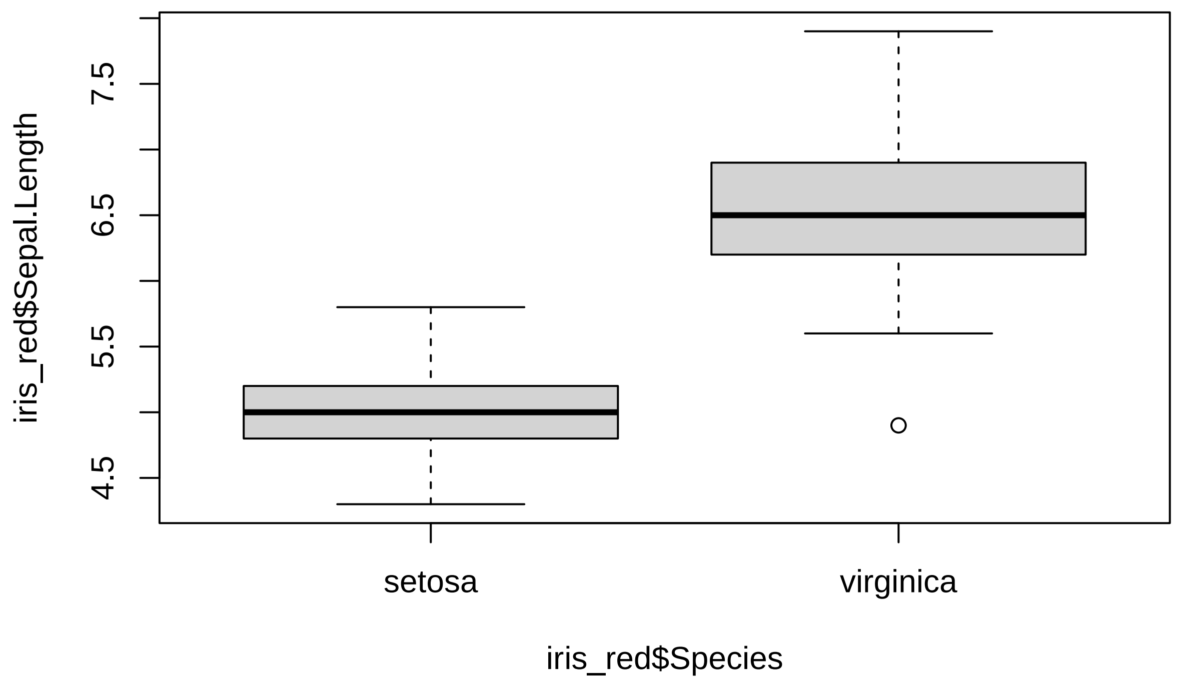
```
## [1] 1.051641e-05
```

Now the P-values are identical. **Linear regression of a numerical variable on a binary categorical variable is exactly the same as a Welsch t-test (with equal variance) comparing the two levels of the categorical variable.**

2. Logistic regression

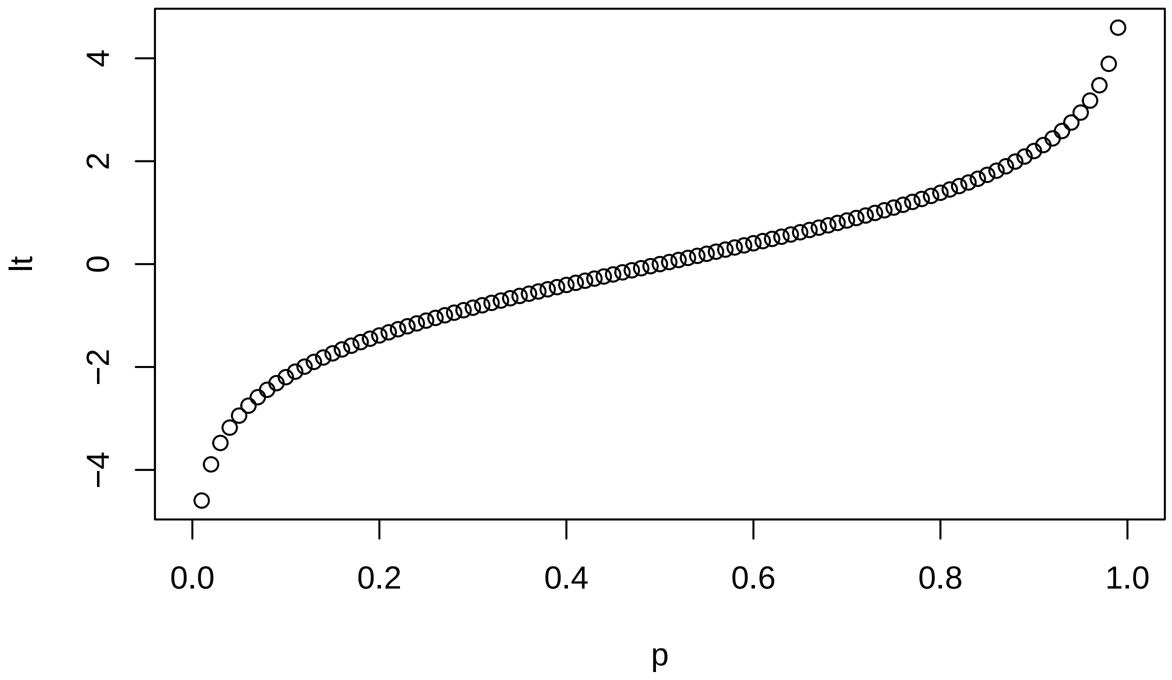
We've said you can guess Sepal.Length better if you know Species, as Species is a significant *predictor* of Sepal.Length. Also the reverse is true (i.e. by knowing Sepal.Length you make a safer bet on the Species), so now we'll regress/predict the categorical variable Species (dependent) on the numerical variable Sepal.Length (independent): this is done by **logistic regression** instead of linear regression.

```
plot(iris_red$Sepal.Length ~ iris_red$Species)
```



Logistic regression tells you the probability of being either “virginica” (=1) or “setosa” (=0) based on Sepal.Length. Since “virginica” = 1 and “setosa” = 0, we’re regressing a number (probability: 1 or 0, represented by the variable *num_species*) on a number (length of sepals). Probability is not that easy to regress (since it stays between 0 and 1), better to use its *logistic transform* (lt): $log(p/1 - p)$

```
p <- seq(from=0,to=1,by=0.01) #the probability is defined: it ranges from 0 to 1 (0.01 steps)
lt <- log(p/(1-p))  #we'll that the logistic transform varies between - infinite and + infinite
plot(x=p, y=lt)
```



This logistic transform of the probability of being either “virginica” (1) or “setosa” (0) has replaced the probability itself, and as the probability it is dependent on Sepal.Length:

$log(p/1 - p) = \beta_0 + \beta_1 x$

Seems like a linear regression except for the fact that homoscedasticity is violated. We do logistic regression using the `glm()` function:

```
logreg <- glm(iris_red$num_species ~ iris_red$Sepal.Length, family = "binomial")
summary(logreg)
```

```
##
## Call:
## glm(formula = iris_red$num_species ~ iris_red$Sepal.Length, family = "binomial")
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.5850  -0.1470  -0.0065   0.0824   3.2276
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)           -38.547      9.557  -4.033 5.50e-05 ***
## iris_red$Sepal.Length    6.805      1.694   4.016 5.91e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 138.629  on 99  degrees of freedom
## Residual deviance:  26.247  on 98  degrees of freedom
## AIC: 30.247
##
## Number of Fisher Scoring iterations: 8
```

Important: `glm()` wants either a *logical* variable or a *numerical* variable taking values 0 or 1 (in our case, num_species is a numerical that can be only 1 or 0). The argument `family` should be specified because this function doesn't do only logistic regression.

Interpretation: every 1 cm increase in Sepal.Length contributes to extra 6.805 units in the logistic transform of num_species (i.e. logistic transform of the probability of being either "virginica" or "setosa").Uncertainty on the 6.805 value is about 1.7. P-value is low.

Logistic *vs* Linear regression

If variable A gives info about variable B (i.e. A is *regressor* of B), also the reverse is true. It means that:

- linear regression to predict Sepal.Length from species (num_species)
- logistic regression to predict species (num_species) from Sepal.Length

both give the same information.