2 Implementing an Edit-Distance Algorithm

Use the program you wrote to score the following strings:
a) “data Science” to “Data Sciency”
b) “organizing” to “orGanising”
c) “AGPRklafsdyweIllIIgEnXuTggzF” to “AgpRkliFZdiweIllIIgENXUTygSF”)
Then:
d) Describe a scenario (3-4 sentences) where implementing the standard Hamming distance algorithm would be applicable.

Firstly write a function

```
Hamming_Distance=function(word1,word2){
  # install and load packages
  options(repos='http://cran.rstudio.org')
  have.packages <- installed.packages()
  cran.packages <- c('base')
  to.install <- setdiff(cran.packages, have.packages[,1])
  if(length(to.install)>0)
    install.packages(to.install)
  library(base)

  # split both words
  A=unlist(strsplit(word1, ""))
  B=unlist(strsplit(word2, ""))
  AA=c(A)
  BB=c(B)

  # start model based on different conditions
  SZ=c("Z","S")
  sz=c("z","s")
  count=sum(AA!=BB)
  for (i in 1:length(AA)) {
    if(i==1){
      if(AA[i]!=BB[i]){
        if (tolower(AA[i])==tolower(BB[i])){
          count=count-1
        } else if(((AA[i] %in% SZ) && (BB[i] %in% SZ))||
                  ((AA[i] %in% sz) && (BB[i] %in% sz))){
          count=count-1
        } else {count=count-0}
      } else {count=count-0}
    } else if(i>1){
      if(AA[i]!=BB[i]){
        if(tolower(AA[i])==tolower(BB[i])){
          count=count-0.5
        } else if(((AA[i] %in% SZ) && (BB[i] %in% SZ))||
                  ((AA[i] %in% sz) && (BB[i] %in% sz))){
          count=count-1
        } else if(((AA[i] == toupper(AA[i]))&&(BB[i] == toupper(BB[i])))||
                  ((AA[i] == tolower(AA[i]))&&(BB[i] == tolower(BB[i])))){
```

```r
          count=count-0
        } else {count=count+0.5}
      } else {count=count-0}
    } else {count=count-0}
  }

  # manage output word
  Text="Our new hamming distance for two strings is:"
  output=list(Text,count)
  print(output)
}
```

a) "data Science" to "Data Sciency"

```r
first_word="data Science"
second_word="Data Sciency"
Hamming_Distance(first_word,second_word)
```

```
## [[1]]
## [1] "Our new hamming distance for two strings is:"
##
## [[2]]
## [1] 1
```

```r
# the distance score here is 1.
```

b) "organizing" to "orGanising"

```r
first_word="organizing"
second_word="orGanising"
Hamming_Distance(first_word,second_word)
```

```
## [[1]]
## [1] "Our new hamming distance for two strings is:"
##
## [[2]]
## [1] 0.5
```

```r
# the distance score here is 0.5.
```

c) "AGPRklafsdyweIllIIgEnXuTggzF" to "AgpRkliFZdiweIllIIgENXUTygSF"

```r
first_word="AGPRklafsdyweIllIIgEnXuTggzF"
second_word="AgpRkliFZdiweIllIIgENXUTygSF"
Hamming_Distance(first_word,second_word)
```

```
## [[1]]
## [1] "Our new hamming distance for two strings is:"
##
## [[2]]
## [1] 8.5
```

```
# the distance score here is 8.5
```

Show them in a table

First_word	Second_word	Hamming Distance
data Science	Data Sciency	1
organizing	orGanising	0.5
AGPRklafsdyweIllIIgEnXuTggzF	AgpRkliFZdiweIllIIgENXUTygSF	8.5

d) Describe a scenario (3-4 sentences) where implementing the standard Hamming distance algorithm would be applicable.

The Hamming distance can be used for two strings of equal length to check the minimum number of substitutions required to change one string into the other.
In this scenario, it requires case sensitive, for example, “a” and “A” are different no matter if it is in the first position.
It doesn’t consider the different spelling method for the same word, but it just cares about pure identity.
It needn’t check if the word sounds same after changing one or more strings, but it just cares about pure identity.