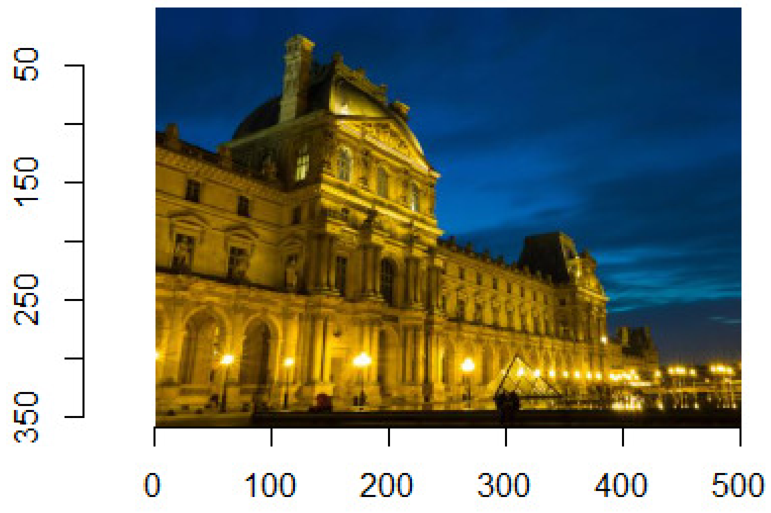
seriationLouvre.r

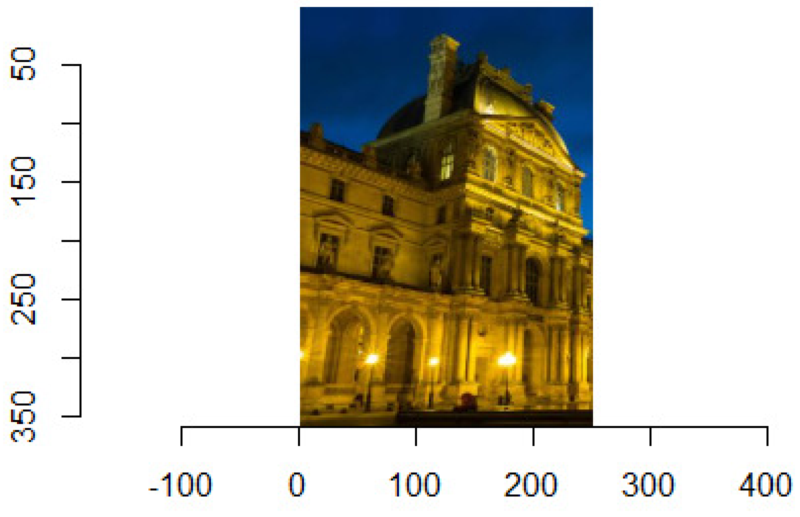
Sun Apr 23 23:34:41 2017

```
#####################################################
## Breitzman
## 4/23/17
## Piping and seriation images
#####################################################


s1 <- "this is a string. "
s2 <- "this is another string. "

toupper(s1)

## [1] "THIS IS A STRING. "

paste(s1,s2)

## [1] "this is a string.  this is another string. "

#install.packages("magrittr")
library(magrittr)

## Warning: package 'magrittr' was built under R version 3.3.3

## fun with pipes.  If you've used a unix shell you may have used pipes
before
## this is equivalent to paste(s1,s2)
s1 %>% paste(s2)

## [1] "this is a string.  this is another string. "

## this is equivalent to toupper(paste(s1,s2))
s1 %>% paste(s2) %>% toupper()

## [1] "THIS IS A STRING.  THIS IS ANOTHER STRING. "

toupper(paste(s1,s2))

## [1] "THIS IS A STRING.  THIS IS ANOTHER STRING. "

## this is silly but it works
## it's obviously equal to tolower(toupper(paste(s1,s2)))
toupper(paste(s1,s2)) %>% tolower()

## [1] "this is a string.  this is another string. "
## use it if you want.
## some people like it
## sometimes it's less readable
(1 + 8) %>% sqrt

## [1] 3

sqrt(1+8)

## [1] 3

## doesn't always work
(sqrt(1+8))^3

## [1] 27

##(1+8) %>% sqrt %>% ^3


##install.packages("imager")
library(imager)

## Warning: package 'imager' was built under R version 3.3.3

## Loading required package: plyr

## Warning: package 'plyr' was built under R version 3.3.3

##
## Attaching package: 'imager'

## The following object is masked from 'package:plyr':
##
##     liply

## The following object is masked from 'package:magrittr':
##
##     add

## The following objects are masked from 'package:stats':
##
##     convolve, spectrum

## The following object is masked from 'package:graphics':
##
##     frame

## The following object is masked from 'package:base':
##
##     save.image

im <- load.image("C:\\Users\\Tony\\Dropbox\\Rowan\\ProgWorkshop-
R\\Lectures\\lecture24louvre.jpg")
```

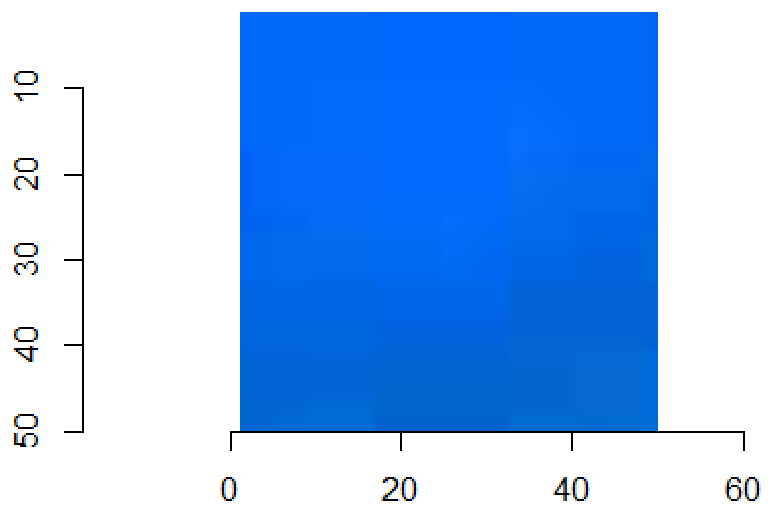
```r
plot(im)
```



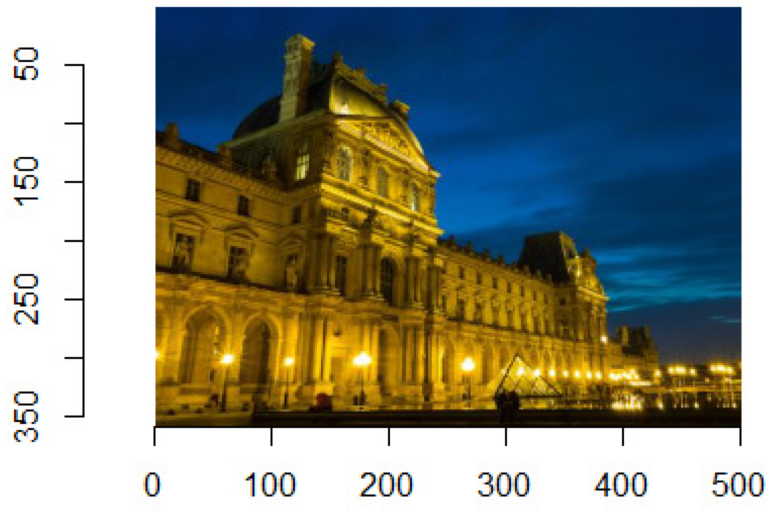
```r
dim(im)

## [1] 501 358   1   3

## 501 pixels wide by 358 pixels tall

t <- imsplit(im,"x")
length(t)

## [1] 501

## we essentially get 501 vectors containing a 1 pixel by 358 tall slice

## if we do
plot(imappend(t[1:250],"x"))
```



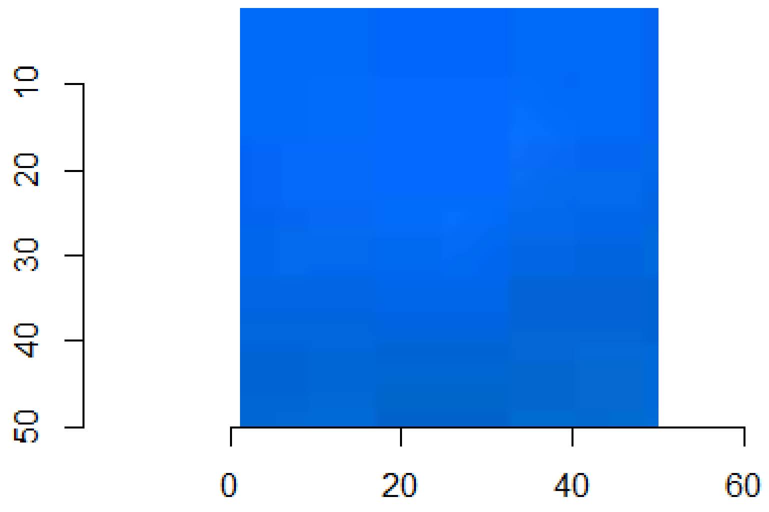
```
## we obviously get half the picture

## we can grab a chunk of blue sky for example by doing the following
chunk <- imsplit(im,"x")
chunk2 <- imsplit(imappend(chunk[1:50],"x"),"y")
plot(imappend(chunk2[1:50],"y"),"y")
```



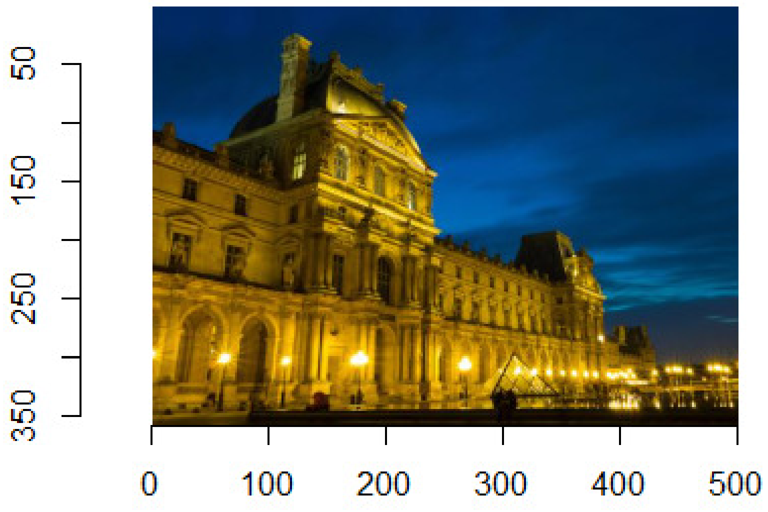
```
## how did that work?  We grabbed the first 50 vertical vectors
## and then grabbed the 50 horizontal vectors from that subset

## we can write a function to plot any subset of the picture
## as follows
plotSubset <- function(image,xRange,yRange){
  chunk <- imsplit(im,"x")
  chunk2 <- imsplit(imappend(chunk[xRange],"x"),"y")
  plot(imappend(chunk2[yRange],"y"),"y")
}


plot(im)
```



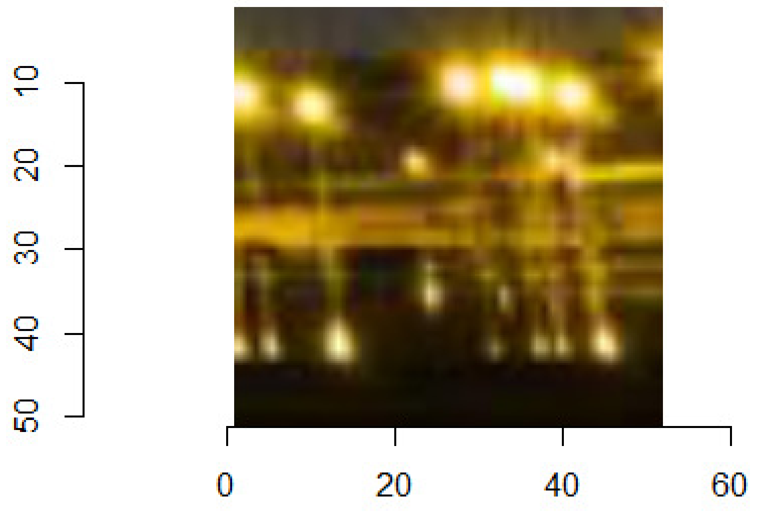
```
## get a chunk of the sky
plotSubset(im,1:50,1:50)
```



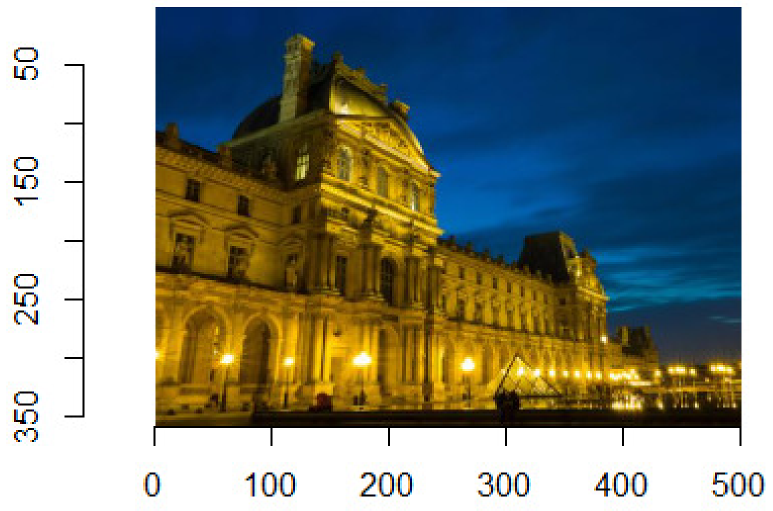
```
plot(im)
```



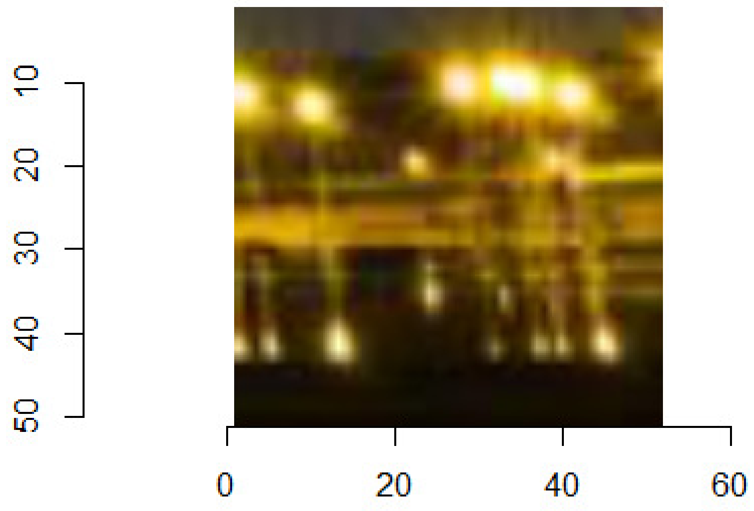
```
## zoom in on the street lights at the bottom right
plotSubset(im,450:501,300:350)
```



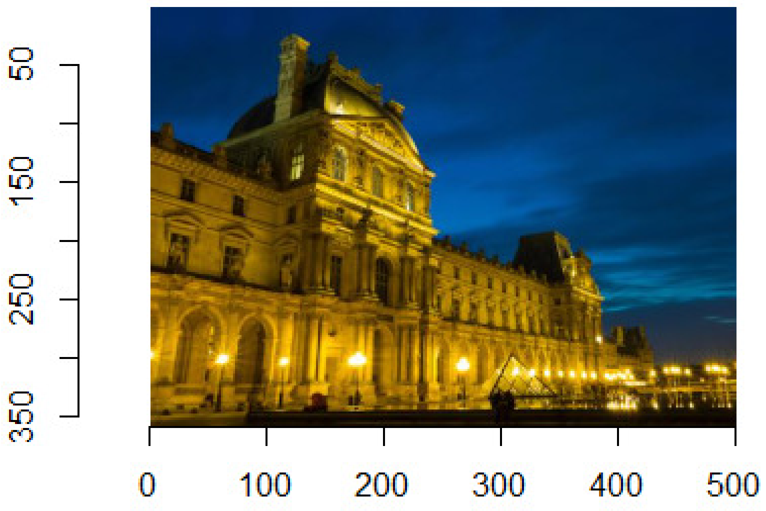
```r
## Note we can rewrite that function with pipes
## i'm not claiming it's more readable
## but it does have the advantage of not using
## the temp variables chunk and chunk2
plotSubset2 <- function(image,xRange,yRange){
  imsplit((imsplit(image,"x")[xRange] %>% imappend("x")),"y")[yRange] %>%
  imappend("y") %>% plot
}


plot(im)
```



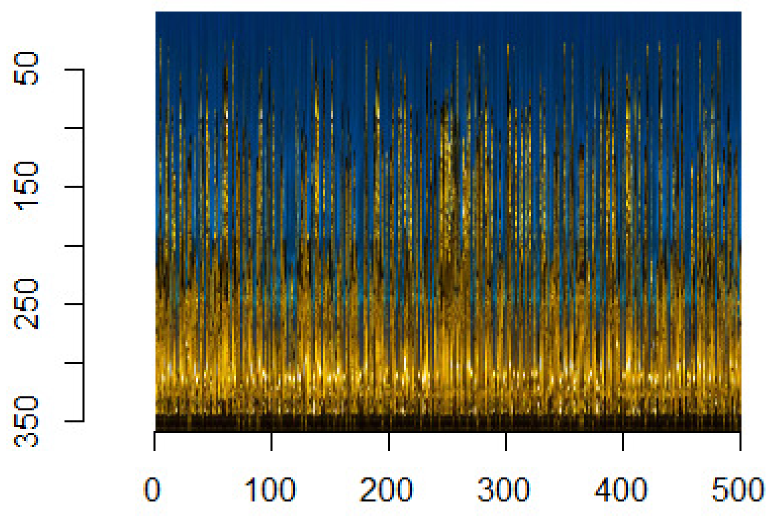
```
plotSubset2(im,450:501,300:350)
```



```r
## look at this
## recall that sample randomly rearranges
## things so this essentially chops up the image
## and puts it back together randomly
scramble <- function(im,axis="x")
{
  imsplit(im,axis) %>% { .[sample(length(.))] } %>% imappend(axis)
}


plot(im)
```

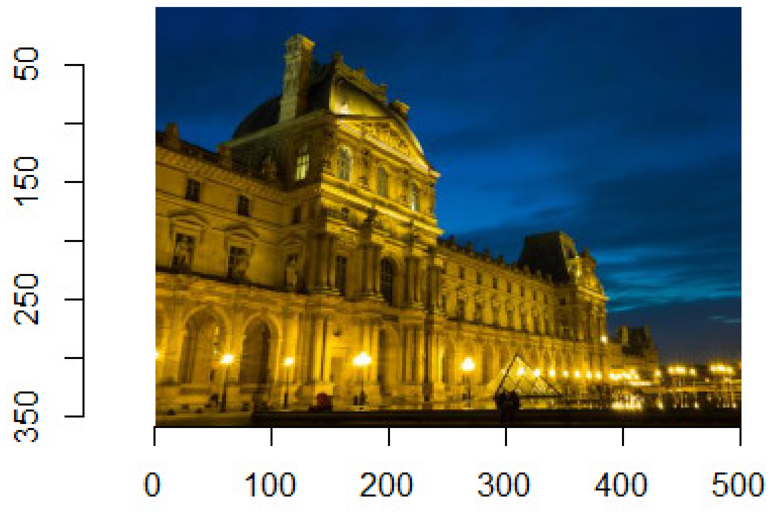


```r
scrambledIm <- scramble(im)

plot(scrambledIm)
```

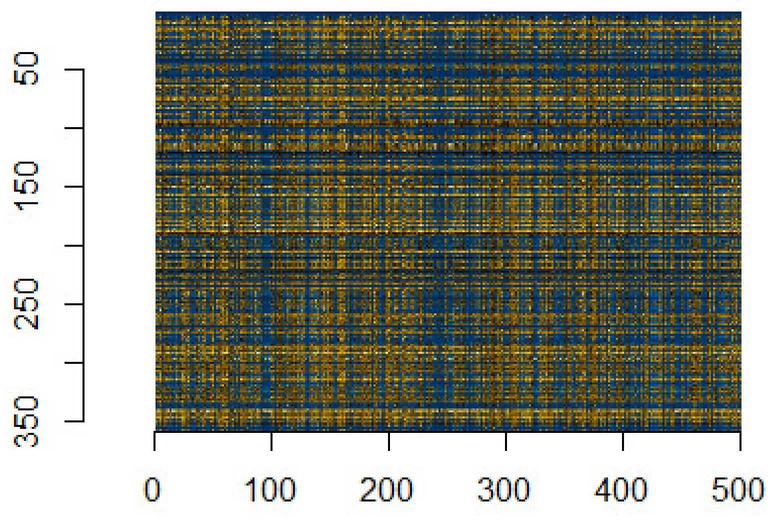


```
## does anybody have any idea how we can unscramble this picture?
## believe it or not it can be done.
## It turns out that any 2
## adjacent vectors are very similar to each other
## since each pixel is just a number we can compare 2 vectors
## and sort the vectors by similarity using seriate.





##install.packages("seriation")
library(seriation)

## Warning: package 'seriation' was built under R version 3.3.3
```

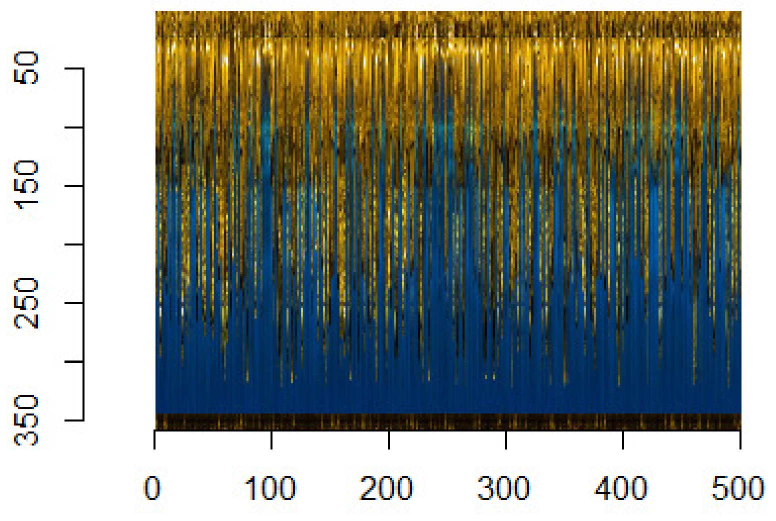
```
##install.packages("purrr")
library(purrr)

## Warning: package 'purrr' was built under R version 3.3.3

##
## Attaching package: 'purrr'

## The following object is masked from 'package:plyr':
##
##     compact

## The following object is masked from 'package:magrittr':
##
##     set_names

unscramble <- function(im.s,axis="x",method="TSP",...)
{
  cols <- imsplit(im.s,axis)
  #Compute a distance matrix (using L1 - Manhattan - distance)
  #Each entry D_ij compares column i to column j
  D <- map(cols,as.vector) %>% do.call(rbind,.) %>% dist(method="manhattan")
  out <- seriate(D,method=method,...)
  cols[get_order(out)] %>% imappend(axis)
}

## note sometimes it will reverse the picture
## because it orders the vectors by closeness
## doesn't know which is first
plot(unscramble(scrambledIm))
```



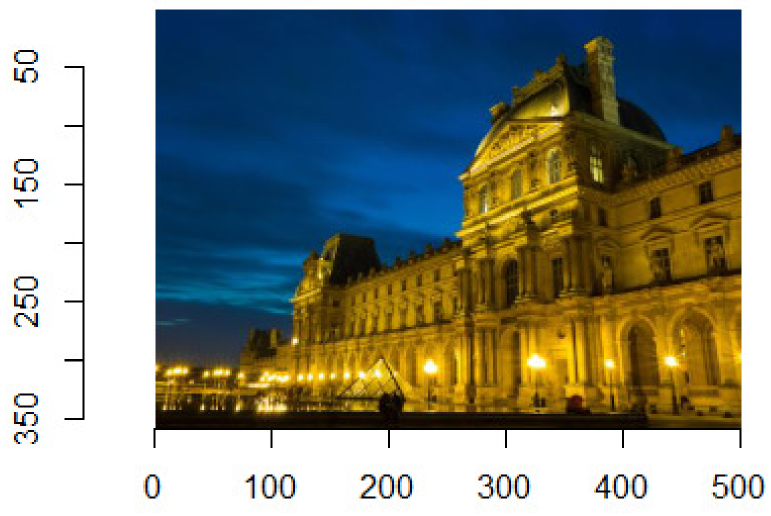
```r
## the surprising thing is it will work with a picture
## that is vertically and horizontally scrambled

scrambledIm2 <- scramble(scramble(im,"x"),"y")
plot(scrambledIm2)
```



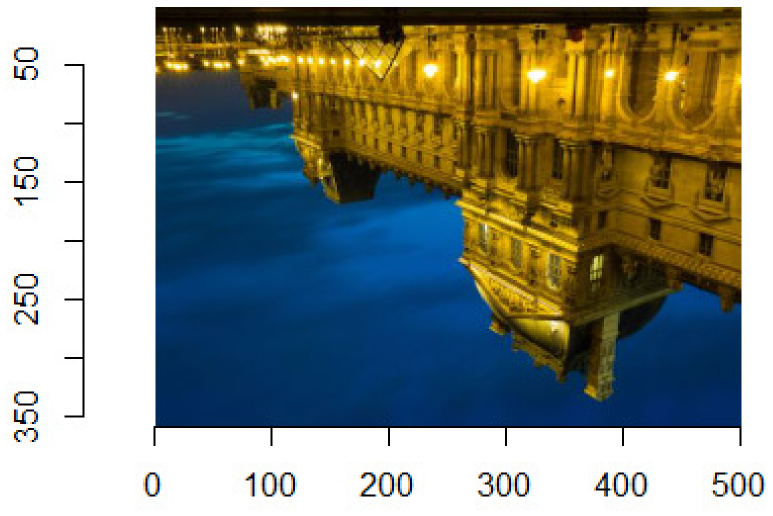
```
## It would be a miracle if we could fix this right?
plot(unscramble(scrambledIm2,"y"))
```



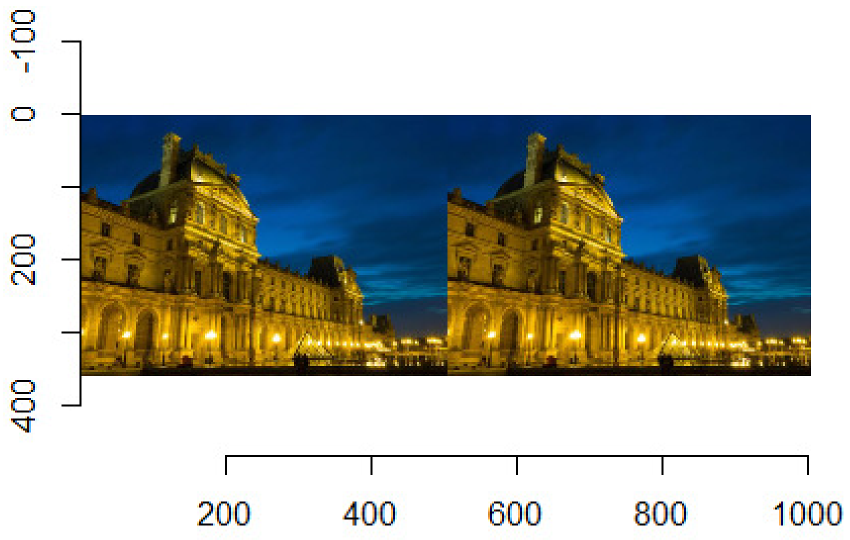
```r
## It looks like we just have to run unscramble twice

plot(unscramble(unscramble(scrambledIm2,"y"),"x"))
```



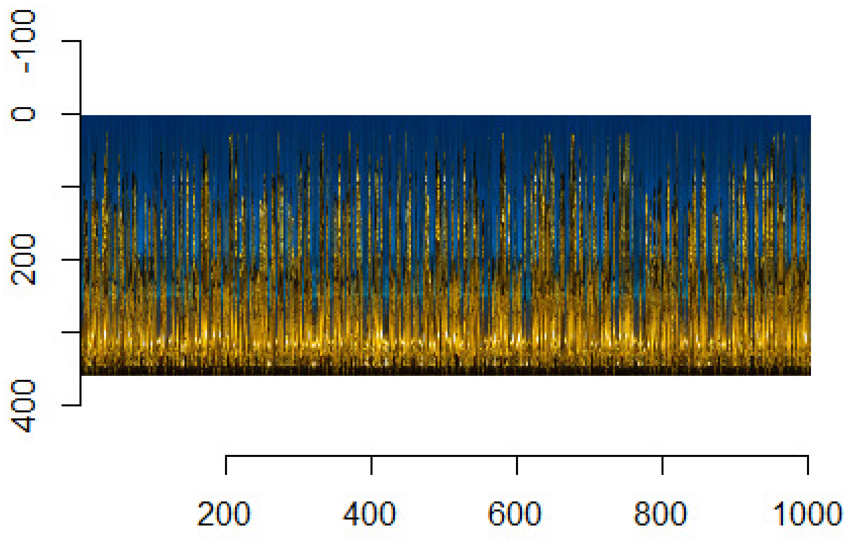
```r
## or if you prefer

unscramble(scrambledIm2,"y") %>% unscramble("x") %>% plot
```



```
## Sometimes we get a picture that’s upside down or backwards

## Note if we have a picture with a lot of symmetry (mirror images)
## this method won't really work

## consider
doubleIm <- imappend(list(im,im),"x")
plot(doubleIm)
```

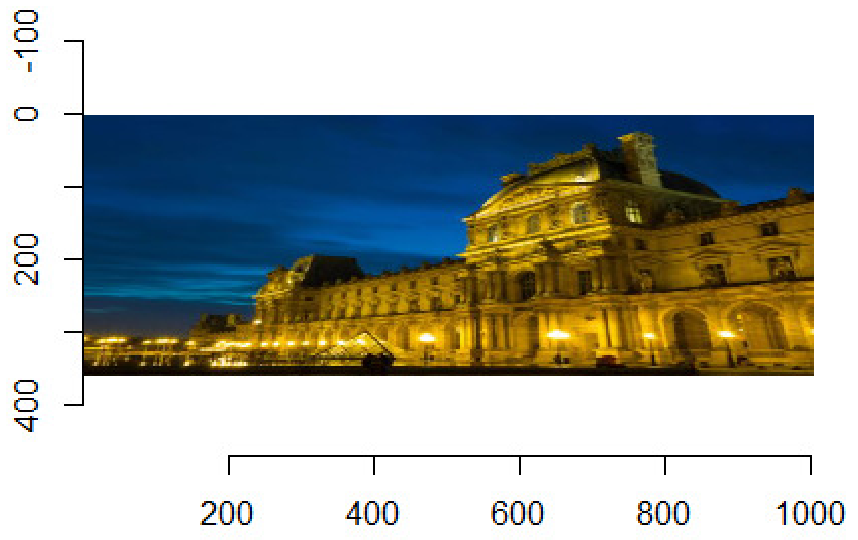


```r
plot(scramble(doubleIm))
```



```
## any guesses on what unscramble will do?
```

```
plot(unscramble(scramble(doubleIm)))
```


-100
0
200
400
200
400
600
800
1000