First, let's construct a 10×5 matrix and replace 10 randomly picked positions within the matrix with NAs (*i.e.* missing values).

```
# Make test data matrix
dat<- matrix(rnorm(50), nrow=10, ncol=5)

# JUST R VALUES NO P !
# Replace 10 NAs within matrix randomly
set.seed (877)
naInd<- sample(1:length(dat), 10)
dat[naInd]<- NA
colnames(dat)<- paste("col", 1:ncol(dat), sep="")
rownames(dat)<- paste("row", 1:nrow(dat), sep="")
dat
#             col1       col2       col3       col4        col5
# row1  -0.8882978 -1.3064945 -0.8559183 -1.2621139  0.28889517
# row2  -1.1934817         NA  1.0021094  0.2707312  2.65574584
# row3   0.5436480 -0.9709940         NA -2.0137933 -0.03901379
# row4  -0.1557453 -1.6251252 -0.2549788 -0.5652703          NA
# row5  -0.7226121  2.7137291 -0.5804944  0.4200483 -0.18883746
# row6          NA -0.9527775  2.1885032 -0.3665413  1.14035680
# row7          NA  1.4430923         NA  0.3362986          NA
# row8   0.4691104 -1.5502340         NA         NA  0.22606033
# row9  -0.3557879  0.1540679 -0.4542577  0.4951978 -1.11224029
# row10  0.6162009         NA -0.9514461 -1.0438710 -1.48530042
```

Normally, we can use the `cor()` function from the `stats` package to get pairwise correlations over the columns of a data frame such as `dat`. However, as demonstrated in the following, due to the NAs in the data frame the results of the correlations will be mainly NA. Note, that parameter settings such as `na.rm=TURE (that` ignores the NAs in functions such as median and mean) are not supported by the `cor()` function. Furthermore, the `cor()` function does not run statistucal tests, hence it does not return p values for the comparisons (execute `?cor` for more info).

```
cor(dat)
#      col1 col2 col3 col4 col5
# col1    1   NA   NA   NA   NA
# col2   NA    1   NA   NA   NA
# col3   NA   NA    1   NA   NA
# col4   NA   NA   NA    1   NA
# col5   NA   NA   NA   NA    1
```

One may be tempted to remove those rows from the data frame which have one or more missing values, however you may be left with too small number of rows in your data to run any meaningful correlation analysis.

```
#Removing rows with NAs
as.data.frame(na.omit(dat))
#            col1       col2       col3       col4       col5
# row1 -0.5903153  1.1200880 -1.4642429  0.2085692  1.1770598
# row5  0.4496017  0.8385497 -0.4793778 -0.1731461  0.8716287
# row9 -0.5647845 -1.6658176 -0.5613469  0.7549264 -1.1794651

#Running correlation on NA filtered data
cor(na.omit(dat))
#            col1       col2       col3       col4       col5
# col1  1.0000000  0.1517447  0.9106435  0.7645283 -0.9994524
# col2  0.1517447  1.0000000  0.5465933  0.7531387 -0.1843677
# col3  0.9106435  0.5465933  1.0000000  0.9625528 -0.9238171
# col4  0.7645283  0.7531387  0.9625528  1.0000000 -0.7854387
# col5 -0.9994524 -0.1843677 -0.9238171 -0.7854387  1.0000000
```

We would, of course, prefer to get the most from our data. Therefore, we would like to ignore NAs in our paired correlation tests. One correlation function supported by R's `stats` package that can remove the NAs is `cor.test()` . However, this function only runs correlation on a pair of vectors and does NOT accept a data.frame/matrix as its input (to run correlation on the columns of the data frame and build a a pairwise correlation matrix accordingly). Note that the `cor.test()` function also returns measured p values for the comparisons. Here, we define a function that adapts `cor.test` to run pairwise correlations over all columns of an input data frame and returns two matrices for the r values and p values of the pairwise comparisons.

```
# Returns r and p values but does not accept data frame as input !
cor.test(dat[,1], dat[,2], na.action=na.omit)
# Pearson's product-moment correlation
#
# data:  dat[, 1] and dat[, 2]
# t = -1.063, df = 4, p-value = 0.3477
# alternative hypothesis: true correlation is not equal to 0
# 95 percent confidence interval:
#  -0.9275857  0.5527702
# sample estimates:
#       cor
# -0.4693404

# Defining a correlation function that suits a data frame input
# and returns r and p values
mycor<- function(x,...){
  r<- apply(x, 2, function(j){
    apply(x, 2, function(i){
      as.numeric(cor.test(i,j, …)$estimate)
    })
  })
  P<- apply(x, 2, function(j){
    apply(x, 2, function(i){
      as.numeric(cor.test(i,j, …)$p.value)
    })
```

```
  })
  out<-c()
  out$P<- P
  out$r<- r
  return(out)
}

# Running the defined correlation function and measuring
# its running time
time1<- Sys.time()
myCorDat<- mycor(dat, method="pearson", na.action=na.omit)
time2<- Sys.time()


(runTimeMyCor<- difftime(time2,time1))
#Time difference of 0.02293086 secs
```

One problem with using `cor.test` in a loop or apply function is that it is inefficient (or in other words slow), especially for large data. This problem can also be seen in the `mycor()` function mentioned above, as it takes ~0.02 seconds to run on a data of size 50 (i.e. 10×5 matrix). If you are willing to install the Hmisc R package (available in CRAN) it supports a rcorr() function which: is robust, accepts a numeric matrix (or 2 numeric matrics !) as input, removes NAs according to the pairwise comparisons, and returns both r and p value measures for the pairwise comparisons.

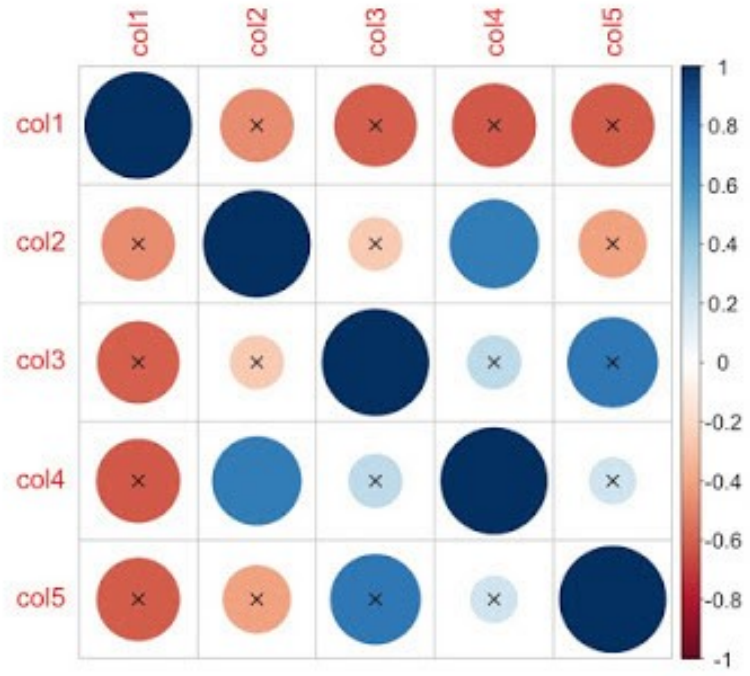
```
#Running rcorr function and measuring run time
ime1<- Sys.time()
rcorrDat<- Hmisc::rcorr(dat, type="pearson")
time2<- Sys.time()
(runTimeRcorr<- difftime(time2,time1))
#Time difference of 0.0009348392 secs

#mycor vs rcorr run time
c(runTimeMyCor, runTimeRcorr)
# Time differences in secs
# [1] 0.0229308605 0.0009348392

rcorrDat
#       col1  col2  col3  col4  col5
# col1  1.00 -0.47 -0.59 -0.61 -0.60
# col2 -0.47  1.00 -0.25  0.69 -0.40
# col3 -0.59 -0.25  1.00  0.25  0.71
# col4 -0.61  0.69  0.25  1.00  0.19
# col5 -0.60 -0.40  0.71  0.19  1.00
#
# n
#      col1 col2 col3 col4 col5
# col1    8    6    6    7    7
# col2    6    8    5    7    6
```

```
# col3    6    5    7    7    6
# col4    7    7    7    9    7
# col5    7    6    6    7    8
#
# P
#        col1   col2   col3   col4   col5
# col1        0.3477 0.2166 0.1428 0.1506
# col2 0.3477        0.6842 0.0854 0.4283
# col3 0.2166 0.6842        0.5851 0.1115
# col4 0.1428 0.0854 0.5851        0.6782
# col5 0.1506 0.4283 0.1115 0.6782
```

Correlation plotting

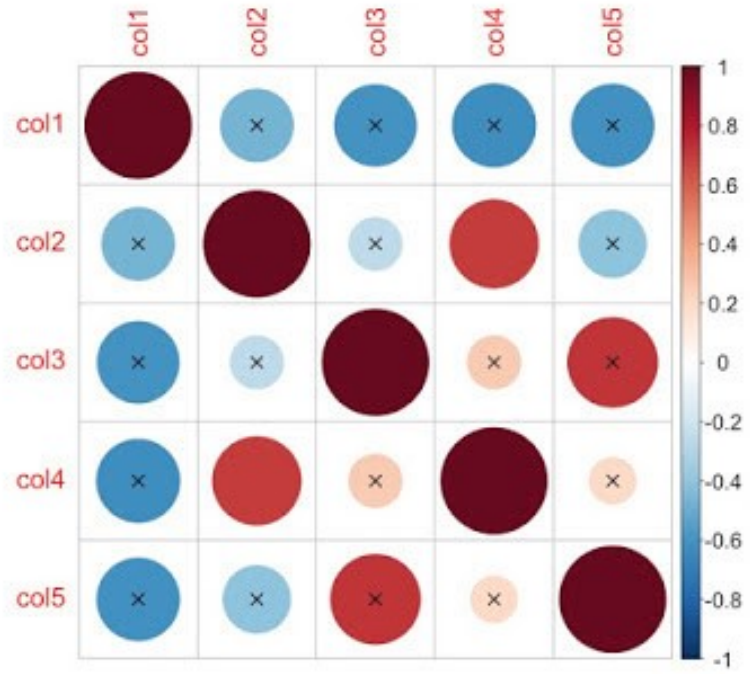
One interesting plotting method is supported by the `corrplot()` function supported by the `corrplot` R package. This function mainly visualizes the r measurements for the paired correlations. The size and colour of circles in the figure represent the r. If the p value is higher than the defined sig.level threshold parameter, an X sign shows that the correlation is NOT significant.

```
library(corrplot)
jpeg("corplotTest.jpg", width=800, height=800, quality=100,
pointsize=24)
corrplot(corr=myCorDat$r,
         p.mat = myCorDat$P,
         type="full", insig="pch", sig.level =.1, pch.cex = .9)
dev.off()
```



The default colour palette is not too popular in our lab however! As a heat colour, it is preferred that a higher value is associated with red and lowere value with blue. Using the following code I usually build a custom colour palette function that is in the reverse order as the default colours used by `corrplot`.

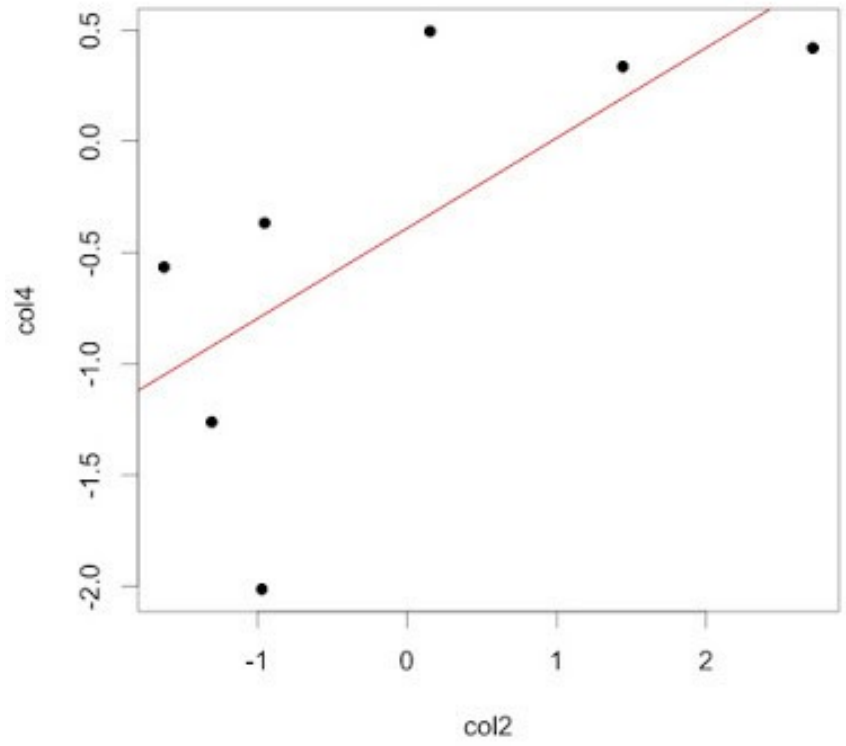
```
# make function that makes set colors
col2 <- colorRampPalette(c("#053061", "#2166AC", "#4393C3",
                           "#92C5DE", "#D1E5F0", "#FFFFFF",
                           "#FDDBC7", "#F4A582", "#D6604D",
                           "#B2182B", "#67001F"))
jpeg("corplotTest2.jpg", width=800, height=800, quality=100,
pointsize=24)
corrplot(corr=myCorDat$r,
         p.mat = myCorDat$P,
         type="full", insig="pch", sig.level =.1, pch.cex = .9,
col=col3(200))
dev.off()
```



Since we generated the values in matrix randomly, the moderately high and significant correlation of col4 with col2 may come as a surprise. First of all, note that its pvalue is actually ~0.085 which is not less than 0.05. We however, chose a more relaxed 0.1 pvalue significance threshold in the plotting. Furthermore, this shows that as the number of comparisons increases in relation to the size of data, the chances of aceiving an accidental signifcant correlation increases. Pvalue adjustments can be helpful to correct for multiple testing. In the following we first recheck the correlation of col4 vs col2. Next, we adjust the pvalues acheived by the correlation tests using Benjamini and Hochberg method. Finally, we plot col4 vs col2 abd fit a line to the data points in the plot. Note that since the paiwise correlation matrices are symmetric with predictable diagonals, I will run the p adjustment over the lower triangular of the P value matrix. As you can see non eo fthe adjusted p values are significant any more.

```
# Recheck correlation of col4 vs col2
cor.test(x=dat[,"col2"], y=dat[,"col4"], method="pearson")
#
# Pearson's product-moment correlation
#
# data:  dat[, "col2"] and dat[, "col4"]
# t = 2.1394, df = 5, p-value = 0.08538
# alternative hypothesis: true correlation is not equal to 0
# 95 percent confidence interval:
#  -0.1287915  0.9498704
# sample estimates:
#       cor
# 0.6913155

p.adjust(myCorDat$P[lower.tri(myCorDat$P, diag = FALSE)], method="BH")
```

```
# [1] 0.9401500 0.8955621 0.9401500 0.9401500 0.9401500 0.6116986
# [7] 0.6116986 0.7738987 0.6116986 0.9401500

# Plot col4 (y-axis) vs col2 (x-axis)
jpeg("highCorPlot.jpg", width=800, height=800, quality=100,
pointsize=24)
plot(dat[,"col2"], dat[,"col4"], pch=16, xlab="col2", ylab="col4")
abline(lm(dat[,"col4"]~dat[,"col2"]), col="red", lwd=2)
dev.off()
```

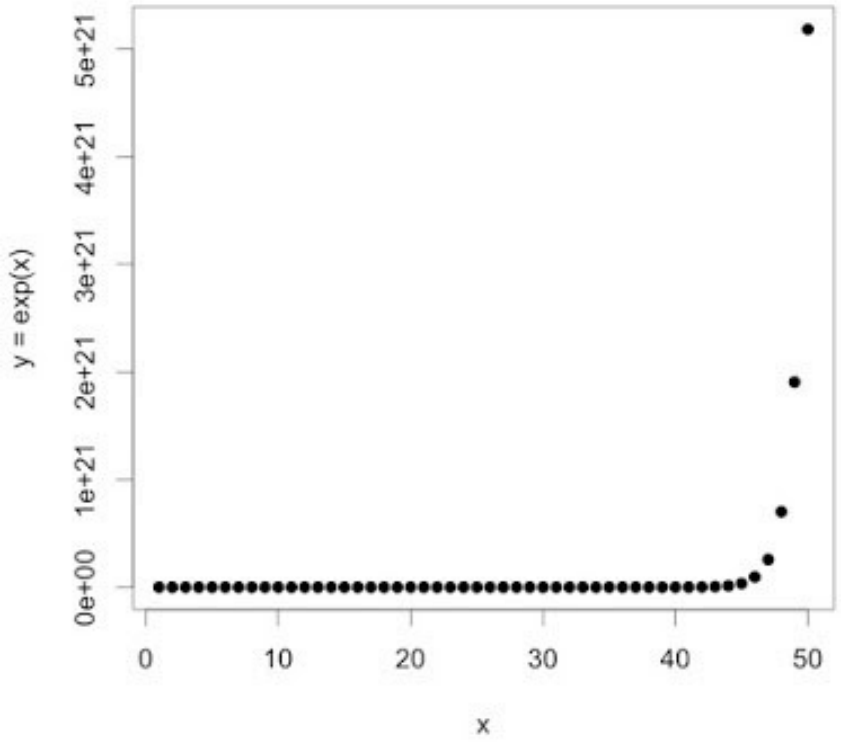


Person vs Spearman correlation

All the codes above use Pearson which is the default method used in most correlation related functions. Many of these functions do however support Spearman as well which is a ranked correlation measurement method. As demostrated by using a simple exponential data, Spearman correlation is more forgiving towards skewness and extreme outliers within the data and mamnages to detect the strong correlation between the defined x and y var.

```
# Defining an unskewed data
x<- 1:50
# Defining a highly skewed data
y<-exp(x)

e1071::skewness(x)
#[1] 0
e1071::skewness(y)
```

```
#[1] 5.574573

cor.test(x=x, y=y, method="pearson")
#
# Pearson's product-moment correlation
#
# data:  x and y
# t = 2.6098, df = 48, p-value = 0.01205
# alternative hypothesis: true correlation is not equal to 0
# 95 percent confidence interval:
#  0.08223784 0.57449344
# sample estimates:
#       cor
# 0.3525162

cor.test(x=x, y=y, method="spearman")
#
# Spearman's rank correlation rho
#
# data:  x and y
# S = 0, p-value < 2.2e-16
# alternative hypothesis: true rho is not equal to 0
# sample estimates:
# rho
#   1

jpeg("whySpearman.jpg", width=800, height=800, quality=100,
pointsize=24)
plot(x, y, pch=16, xlab="x", ylab="y = exp(x)")
dev.off()
```


y = exp(x)
0e+00
1e+21
2e+21
3e+21
4e+21
5e+21
0
10
20
30
40
50
x
