```r
# The apply family

# Apply functions allow you to apply a function across different subsets of data.
# For example, you may want to obtain the mean (max, min, or meidan) for each column of a data.frame.
# It Offers an alternative to explicit iteration using for() loop. Can be faster.

# The apply family includes
# apply(): apply a function to rows or columns of a matrix or data.frame
# lapply(): apply a function to elements of a list or data.frame
# sapply(): same as lapply(), but simplifies the output
# tapply(): apply a funciton to levels of a factor vector.

# 1. apply()
# apply(x, MARGIN=1, FUN=my.fun): apply my.fun() across rows of a matrix or data.frame x.
# apply(x, MARGIN=2, FUN=my.fun): apply my.fun() across columns of a matrix or data.frame x.

# Examples

x <- matrix(rnorm(9), 3, 3)
x

apply(x, MARGIN = 1, FUN=min)

for (i in 1:nrow(x)) {
  print(min(x[i,]))
}

apply(x, MARGIN = 2, FUN=sum)
colSums(x)

# Applying a function that takes extra arguments
# Sometimes we want to use a function over rows or columns of a matrix, that takes extra
# arguments (besides the row or column itself). We can put these as inputs to apply().

setwd("C:/Users/elsect_main")
rev_exp0 <- read.csv("district_rev_exp.csv", na.strings = "-")
head(rev_exp0)

maximum <- apply(rev_exp0[,-c(1,3,4)], MARGIN=2, FUN=max, na.rm=TRUE) # na.rm=TRUE is an argument of max.
maximum

first <- apply(rev_exp0[,-c(1,3,4)], MARGIN=2, FUN=which.max)
first
rev_exp0[first[1],]

second <- apply(rev_exp0[-first[1],-c(1,3,4)], MARGIN=2, FUN=which.max)
second
rev_exp0[second[1],]

# We can use this apply() to our own functions.
my.fun <- function(x) {
  m1 <- median(x, na.rm=TRUE)
  m2 <- mean(x, na.rm=TRUE)
  return(c(m1,m2))
}

apply(rev_exp0[,-c(1,3,4)], MARGIN=2, FUN=my.fun)

# lapply(), elements of a list (data.frame)

my.list = list(nums=seq(0.1,0.6,by=0.1), chars=letters[1:12],
               TF=sample(c(TRUE,FALSE), 6, replace=TRUE))
my.list

lapply(my.list, FUN=mean)   # The type of outcome is always list.

```

```
# lapply() with extra arguments

mean.omitting.one = function(i,vec) {
  return(mean(vec[-i]))
}

my.vec = rev_exp0[,"TOTALEXP"]
n = length(my.vec)
my.vec.jack = lapply(1:n, FUN=mean.omitting.one, vec=my.vec)

head(my.vec.jack) # It's a list, and here are the first 6 elements

# sapply(), elements of a list or data.frame
# The sapply() function works just like lapply(), but tries to simplify the return value whenever possible.
# E.g., most common is the conversion from a list to a vector

my.vec.jack1 = sapply(1:n, FUN=mean.omitting.one, vec=my.vec)
length(my.vec.jack1)

sqrt((n-1)/n * sum((my.vec.jack1 - mean(my.vec.jack1))^2))  # Jackknife standard error
sqrt((n-1)^2/n) * sd(my.vec.jack1)

sd(my.vec)/sqrt(n)                 # conventional standard error

# tapply(), levels of a factor vector
# tapply(x, INDEX=my.index, FUN=my.fun): apply my.fun() to subsets of elements in x that share a common level in my.index

str(rev_exp0)

# tapply is designed to work on a single vector (column), while aggregate can work with multiple columns.
tapply(rev_exp0[,"TOTALREV"], INDEX=list(ST=rev_exp0$STATE, EnR = rev_exp0$ENROLL > 1000), FUN=mean, na.rm=TRUE)
aggregate(rev_exp0[ ,c("ENROLL", "TOTALREV", "TOTALEXP")], by=list(ST = rev_exp0$STATE, EnR = rev_exp0$ENROLL > 1000), FUN=mean, na.rm=TRUE)

## Exercise 1
################################################################################################

president <- c("Obama:2009-2007", "Bush:2001-2009", "Clinton:1993-2001", "Bush:1989-1993", "Reagan:1981-1989")

# Using strsplit(), split names from terms.


# Using lapply(), convert to uppercase strings.



select_first <- function(x) {
  x[1]
}

# Using lapply() and select_first, select each president name.



## Exercise 2
################################################################################################

mtcars
n <- nrow(mtcars)

# Using the Jackknife procedure introduced above, calculate the standard error of the slope coefficient.
```

```
# Create a function that runs a regression with omitting one observation.
# Define x=mtcars$wt, y=mtcars$mpg
# Using sapply(), apply this function to all possible subsamples leaving one
observation out.
# Calculate the Jackknife standard error, using the formula introduced above.

# Compare this with the conventional standard error.
```