

//A R program to TFA Confidence Intervals

```
>newstr<-    r programming!
```

```
Print(newstr)
```

```
Mystring<mystring>
```

```
$Rscript test.R
```

```
//The data set involve in the program with its declaration
```

```
the.data = ToothGrowth  
the.data$dose = as.factor(the.data$dose)  
nt = nrow(the.data)  
a = length(unique(the.data[,2]))  
b = length(unique(the.data[,3]))  
names(the.data) = c("Y","A","B")
```

```
//finding the values of all multilplirs invove in the data set
```

```
find.mult = function(alpha,a,b,dfSSE,g,group){  
  if(group == "A"){  
    Tuk = round(qtukey(1-alpha,a,dfSSE)/sqrt(2),3)  
    Bon = round(qt(1-alpha/(2*g), dfSSE ),3)  
    Sch = round(sqrt((a-1)*qf(1-alpha, a-1, dfSSE)),3)  
  }else if(group == "B"){  
    Tuk = round(qtukey(1-alpha,b,dfSSE)/sqrt(2),3)
```

```

    Bon = round(qt(1-alpha/(2*g), dfSSE ),3)
    Sch = round(sqrt((b-1)*qf(1-alpha, b-1, dfSSE)),3)
  }else if(group == "AB"){
    Tuk = round(qtukey(1-alpha,a*b,dfSSE)/sqrt(2),3)
    Bon = round(qt(1-alpha/(2*g), dfSSE ),3)
    Sch = round(sqrt((a*b-1)*qf(1-alpha, a*b-1, dfSSE)),3)
  }
  results = c(Bon, Tuk,Sch)
  names(results) = c("Bonferroni","Tukey","Scheffe")
  return(results)
}

```

//finding Cis

```

find.means = function(the.data,fun.name = mean){
  a = length(unique(the.data[,2]))
  b = length(unique(the.data[,3]))
  means.A = by(the.data[,1], the.data[,2], fun.name)
  means.B = by(the.data[,1],the.data[,3],fun.name)
  means.AB = by(the.data[,1],list(the.data[,2],the.data[,3]),fun.name)
  MAB = matrix(means.AB,nrow = b, ncol = a, byrow = TRUE)
  colnames(MAB) = names(means.A)
  rownames(MAB) = names(means.B)
  MA = as.numeric(means.A)
  names(MA) = names(means.A)
  MB = as.numeric(means.B)
  names(MB) = names(means.B)
  MAB = t(MAB)
  results = list(A = MA, B = MB, AB = MAB)
  return(results)
}

scary.CI = function(the.data,MSE,equal.weights =
TRUE,multiplier,group,cs){
  if(sum(cs) != 0 & sum(cs !=0 ) != 1){
    return("Error - you did not input a valid contrast")
  }else{

```

```

the.means = find.means(the.data)
the.ns = find.means(the.data,length)
nt = nrow(the.data)
a = length(unique(the.data[,2]))
b = length(unique(the.data[,3]))
if(group == "A"){
  if(equal.weights == TRUE){
    a.means = rowMeans(the.means$AB)
    est = sum(a.means*cs)
    mul = rowSums(1/the.ns$AB)
    SE = sqrt(MSE/b^2 * (sum(cs^2*mul)))
    N = names(a.means)[cs!=0]
    CS = paste("(",cs[cs!=0],")",sep = "")
    fancy = paste(paste(CS,N,sep = ""),collapse = "+")
    names(est) = fancy
  } else{
    a.means = the.means$A
    est = sum(a.means*cs)
    SE = sqrt(MSE*sum(cs^2*(1/the.ns$A)))
    N = names(a.means)[cs!=0]
    CS = paste("(",cs[cs!=0],")",sep = "")
    fancy = paste(paste(CS,N,sep = ""),collapse = "+")
    names(est) = fancy
  }
} else if(group == "B"){
  if(equal.weights == TRUE){
    b.means = colMeans(the.means$AB)
    est = sum(b.means*cs)
    mul = colSums(1/the.ns$AB)
    SE = sqrt(MSE/a^2 * (sum(cs^2*mul)))
    N = names(b.means)[cs!=0]
    CS = paste("(",cs[cs!=0],")",sep = "")
    fancy = paste(paste(CS,N,sep = ""),collapse = "+")
    names(est) = fancy
  } else{

```

```

b.means = the.means$B
est = sum(b.means*cs)
SE = sqrt(MSE*sum(cs^2*(1/the.ns$B)))
N = names(b.means)[cs!=0]
CS = paste("(",cs[cs!=0],")",sep = "")
fancy = paste(paste(CS,N,sep = ""),collapse = "+")
names(est) = fancy
}
} else if(group == "AB"){
  est = sum(cs*the.means$AB)
  SE = sqrt(MSE*sum(cs^2/the.ns$AB))
  names(est) = "someAB"
}
the.CI = est + c(-1,1)*multiplier*SE
results = c(est,the.CI)
names(results) = c(names(est),"lower bound","upper bound")
return(results)
}
}
//conngsintervals for //
Bon = find.mult(alpha = 0.01, a = 2, b = 3, dfSSE = 60 - 2*3, g = 3,
group = "B")[1]
Tuk = find.mult(alpha = 0.01, a = 2, b = 3, dfSSE = 60 - 2*3, g = 3,
group = "B")[2]
B.cs.1 = c(1,0,0)
B.cs.2 = c(0,1,0)
B.cs.3 = c(0,0,1)
#Confidence interval for first factor level of factor B:
scary.CI(the.data,MSE,equal.weights = FALSE,Bon,"B",B.cs.1)

```

//Using Backward Elimination to Find the most significant Factors

```
backwardElimination <- function(x, sl) {  
  numVars = length(x)  
  for (i in c(1:numVars)){  
    regressor = lm(formula = Y ~ ., data = x)  
    maxVar = max(coef(summary(regressor))[c(2:numVars), "Pr(>|t|)"])  
    if (maxVar > sl){  
      j = which(coef(summary(regressor))[c(2:numVars), "Pr(>|t|)"] ==  
        maxVar)  
      x = x[, -j] }  
    numVars = numVars - 1  
  }  
  return(summary(regressor))  
}
```

SL = 0.05

dataset = dataset[, c(indexes of independent factors separated by a
coma)] backwardElimination(dataset, SL)

*/codes on how to implement and find SSE values

```
all.models = list(AB,A.B,A,B,N)  
SSE = t(as.matrix(sapply(all.models,function(M) sum(M$residuals^2))))  
colnames(SSE) = c("AB","(A+B)","A","B","Empty/Null")
```

```
rownames(SSE) = "SSE"
```

//hypothesis test code implementation

Analysis of Variance Table

##

Model 1: Y ~ A + B

Model 2: Y ~ A * B

Res.Df RSS Df Sum of Sq F Pr(>F)

1 56 820.43

```
## 2    54 712.11 2    108.32 4.107 0.02186 *  
## ---  
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
//finding partial R^2
```

```
Partial.R2 = function(small.model,big.model){  
  SSE1 = sum(small.model$residuals^2)  
  SSE2 = sum(big.model$residuals^2)  
  PR2 = (SSE1 - SSE2)/SSE1  
  return(PR2)  
}
```