Coding and data analysis exercises

1.

```r
require(stats)

mylm <- function(Y,X){

    Y = as.matrix(Y); Xnew = as.matrix(X)

    # Check if numeric
    if(!is.numeric(Y) | !is.numeric(Xnew))
        stop("Y or X is not numeric!\n")

    # Check for dimensions
    dy = dim(Y) ; dx = dim(Xnew)
    if(dy[2] != 1 | dy[1] != dx[1])
        stop("Y or X has wrong dimensions\n")

    # Check for ill conditioned elements
    # we can use is.finite to response only to finite real numbers
    if(FALSE %in% is.finite(Y) | FALSE %in% is.finite(Xnew))
        warning("Y or X is ill conditioned\n")

    # Check if of full rank
    D = cbind(1,Xnew)
    DtD = t(D) %*% D
    if(det(DtD) == 0)
        stop("Design matrix is not full rank\n")

    # Regressing
    DtD.inv = solve(DtD)
    hat.matrix = D %*% DtD.inv %*% t(D)
    beta = DtD.inv %*% t(D) %*% Y
    fitted = D %*% beta
    residuals = Y - fitted

    SS.tot = sum((Y - mean(Y))^2)
    if(SS.tot == 0)
        warning("Y is constant!\n")
    SS.res = sum((Y - fitted)^2)
    SS.reg = SS.tot - SS.res
    R2 = SS.reg / SS.tot

    df = dim(D)[1] - dim(D)[2]
    df2 = dim(D)[2] - 1

    s2 = SS.res / df
    std_error = sqrt(s2 * diag(DtD.inv))
    t_value = beta / std_error
    P_value = 1 - pt(abs(t_value), df) + pt(-abs(t_value), df)
```

```
    K = cbind(rep(0, df2), diag(df2))
    Kbeta = K %*% beta
    Fstat = t(Kbeta) %*% solve(K %*% DtD.inv %*% t(K)) %*% Kbeta
    Fstat = Fstat / (df2 * s2)
    P_value_F = 1 - pf(Fstat, df2, df)

    beta_names = c("(Interception)")
    for(i in 1:(dim(D)[2] - 1)){
        beta_names = c(beta_names, sprintf("beta%d",i))
    }
    t_summary = data.frame("Estimate"=beta, "Std.Error"=std_error,
                           "t.value"=t_value, "Pvalue"=P_value)
    row.names(t_summary) = beta_names

    summary <- function(){
        cat("T table:\n")
        print(t_summary)
        cat("\nOverall F test:\n")
        cat(sprintf("F-statistics: %f on %d and %d DF,  p-value: %f",
                    Fstat, df2, df, P_value_F))
    }

    internal.t.res = c()
    external.t.res = c()
    PRESS.res = c()
    Cooks.dist = c()
    for(i in 1:dx[1]){
        ir = residuals[i] / sqrt(s2 * (1 - hat.matrix[i,i]))
        er = ir * sqrt((df - 1) / (df - ir^2))
        pr = residuals[i] / (1 - hat.matrix[i,i])
        cd = ir^2 / dim(D)[2] * (hat.matrix[i,i] / (1 - hat.matrix[i,i]))
        internal.t.res = c(internal.t.res, ir)
        external.t.res = c(external.t.res, er)
        PRESS.res = c(PRESS.res, pr)
        Cooks.dist = c(Cooks.dist, cd)
    }

    # Return result
    result = list(beta = beta,
                  fitted = fitted,
                  residuals = residuals,
                  R2 = R2,
                  hatdiag = diag(hat.matrix),
                  summary = summary,
                  internal.t.res = internal.t.res,
                  external.t.res = external.t.res,
                  PRESS.res = PRESS.res,
                  Cooks.distance = Cooks.dist)
    return(result)
}

test.X = cbind(sample(1:100),sample(1:100),sample(1:100))
beta = c(5,-1,0.01,2)
```

```r
test.y = cbind(1,test.X) %*% beta + rnorm(100,0,5)
test.y[50] = test.y[50] + 10
model = lm(test.y ~ test.X)
mymodel = mylm(test.y, test.X)

ir = rstandard(model)
er = rstudent(model)
cook = cooks.distance(model)
pr = model$residuals / (1 - hatvalues(model))

#Test internal standardized residuals:
print(sum(abs(ir - mymodel$internal.t.res)))
```

```
## [1] 9.470064e-13
```

```r
#Test external standardized residuals:
print(sum(abs(er - mymodel$iexternal.t.res)))
```

```
## [1] 0
```

```r
#Test PRESS residuals:
print(sum(abs(pr - mymodel$PRESS.res)))
```

```
## [1] 5.78998e-12
```

```r
#Test Cook's distance:
print(sum(abs(cook - mymodel$Cooks.distance)))
```

```
## [1] 1.669646e-14
```