

# HW4\_\_Salem

*Mohamed Salem*

*September 24, 2019*

I concluded that there is value in adopting a specific style. Personally, given my current coding style, I've resolved to evolve my style to follow the conventions of using BigCamelCase and not using the attach function.

The main recommendation I received was to add spaces between my operators. I was also pointed towards not having my code exceed 80 characters per line.

```
# Read and import the object
data <- readRDS(file = "D:/Vtech/Statistical Programming/HW4_data.rds")

AnalyzeData <- function(x) {
  SumData <- x
  names(SumData) <- c("Observer", "Var_1", "Var_2")
  SumData <- SumData %>% group_by(Observer) %>% summarise(dev1_mean = mean(Var_1,
    na.rm = T), dev2_mean = mean(Var_2, na.rm = T), dev1_sd = sd(Var_1,
    na.rm = T), dev2_sd = sd(Var_2, na.rm = T), dev_corr = cor(Var_1,
    Var_2, method = "pearson"))

  SumDataMelted <- data.frame(matrix(rep(c(1, 2), each = length(SumData$dev1_mean),
    nrow = length(SumData$dev1_mean), ncol = 1)))
  names(SumDataMelted) <- c("Variable")
  RowMeltmean <- rbind(data.frame(means = SumData$dev1_mean), data.frame(means = SumData$dev2_mean))
  RowMeltsd <- rbind(data.frame(means = SumData$dev1_sd), data.frame(means = SumData$dev2_sd))
  SumDataMelted <- data.frame(cbind(as.factor(SumDataMelted$Variable),
    RowMeltmean$means, RowMeltsd))
  names(SumDataMelted) <- c("Variable", "MeanValues", "SdValues")
  SumDataMelted$Variable <- as.factor(SumDataMelted$Variable)

  bxplt <- ggplot(data = SumDataMelted, aes(x = Variable, y = MeanValues)) +
    geom_boxplot()

  plotB <- bxplt + facet_wrap(~Variable, scales = "free")

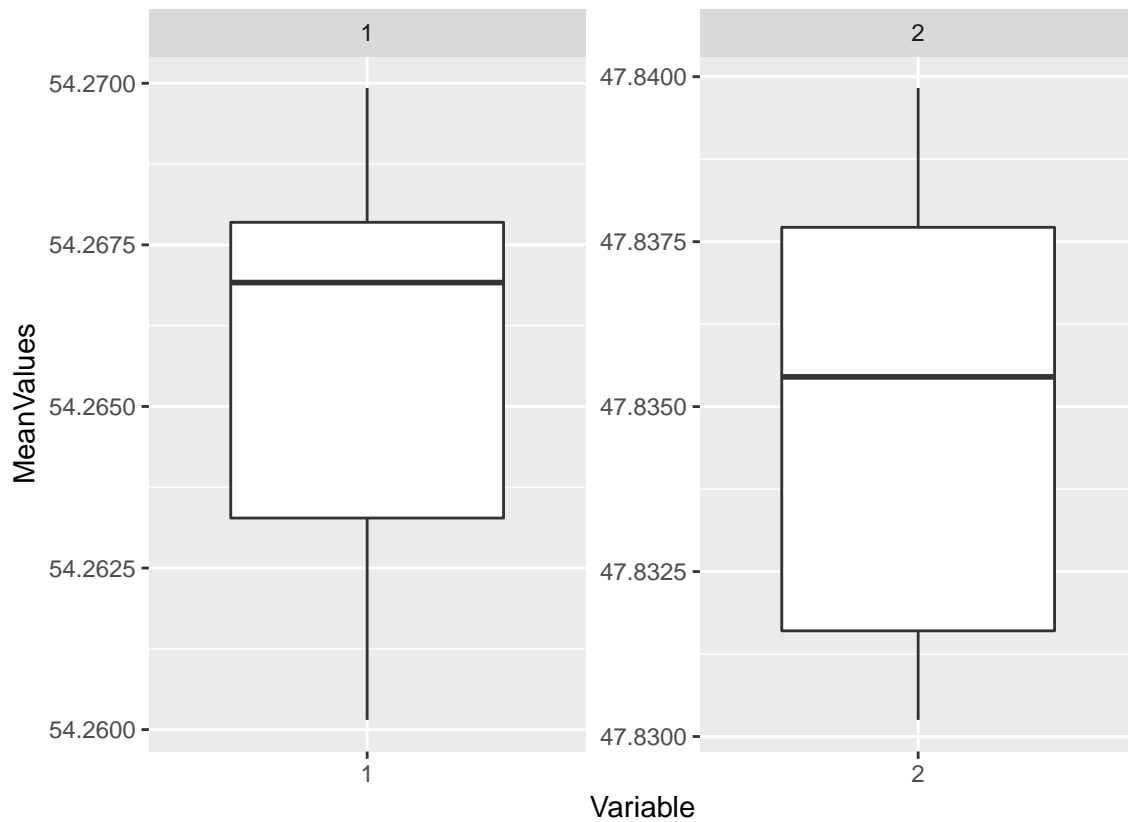
  vlnplot <- ggplot(data = SumDataMelted, aes(x = Variable, y = SdValues)) +
    geom_violin(trim = FALSE)

  plotA <- vlnplot + facet_wrap(~Variable, scales = "free")

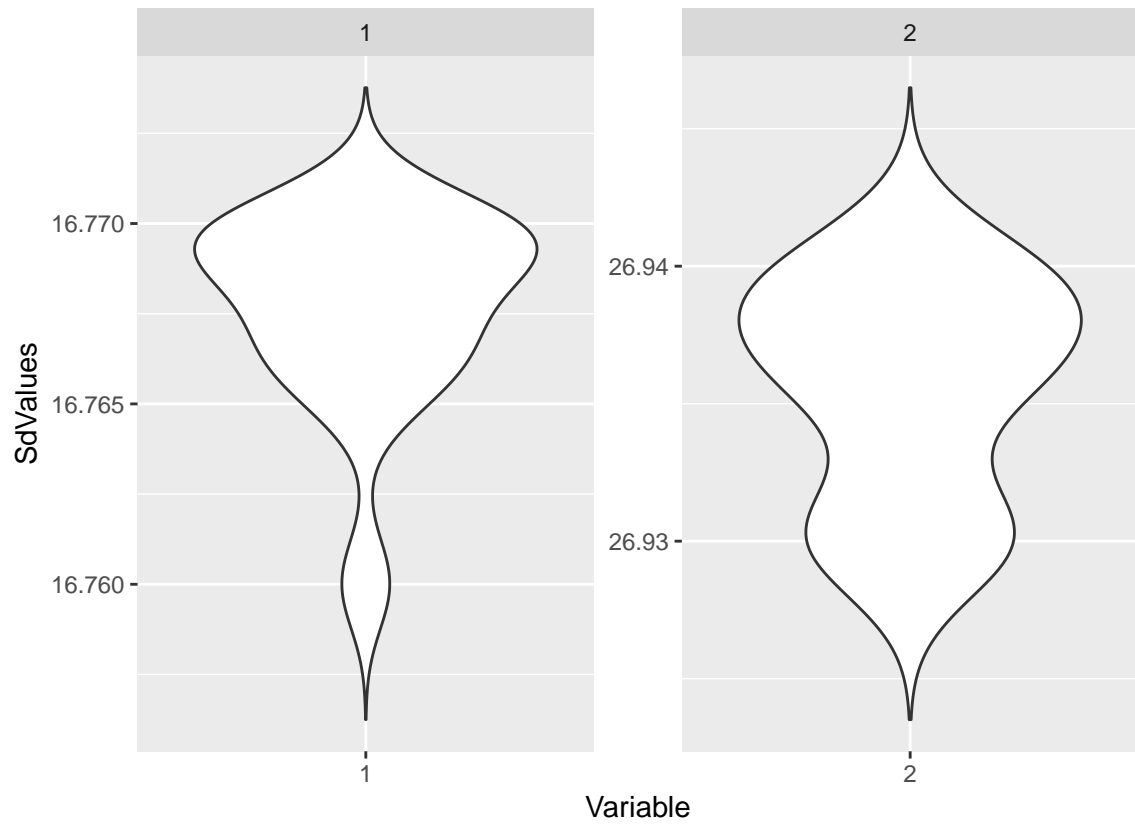
  return(list(SumData, plotB, plotA))
}
```

```
AnalyzeData(data)
```

```
## [[1]]
## # A tibble: 13 x 6
##   Observer dev1_mean dev2_mean dev1_sd dev2_sd dev_corr
##   <dbl>     <dbl>     <dbl>   <dbl>   <dbl>   <dbl>
## 1         1      54.3      47.8    16.8    26.9  -0.0641
## 2         2      54.3      47.8    16.8    26.9  -0.0686
## 3         3      54.3      47.8    16.8    26.9  -0.0683
## 4         4      54.3      47.8    16.8    26.9  -0.0645
## 5         5      54.3      47.8    16.8    26.9  -0.0603
## 6         6      54.3      47.8    16.8    26.9  -0.0617
## 7         7      54.3      47.8    16.8    26.9  -0.0685
## 8         8      54.3      47.8    16.8    26.9  -0.0690
## 9         9      54.3      47.8    16.8    26.9  -0.0686
## 10        10      54.3      47.8    16.8    26.9  -0.0630
## 11        11      54.3      47.8    16.8    26.9  -0.0694
## 12        12      54.3      47.8    16.8    26.9  -0.0666
## 13        13      54.3      47.8    16.8    26.9  -0.0656
##
## [[2]]
```



```
##
## [[3]]
```



```

RamenSum <- function(a, b, f, eps = 1e-06, N = 10000) {
  i = 1
  h = 1
  SliceWidths = numeric(N)
  SliceWidths[1] = h
  IntrmSoln = numeric(N)
  RSum = 0
  cdf = integrate(f, 0, 1)
  while (abs(RSum - cdf$value) > eps) {
    Pts = seq(a, b, by = h)
    NumPts = length(Pts)
    midpts = 0.5 * (Pts[2:NumPts] + Pts[1:(NumPts - 1)])
    RSum = sum(h * f(midpts))
    h = 0.5 * h
    i = i + 1
    SliceWidths[i] = h
  }
  return(list(paste("Solution = ", RSum), paste("Soln Slice Width = ",
    h), paste("True Integral = ", cdf$value), SliceWidths[1:(i -
    1)]))
}

```

```

f <- function(x) {
  exp(-(x^2)/2)
}
RamenSum(0, 1, f)

```

```

## [[1]]
## [1] "Solution = 0.8556247775143"
##
## [[2]]
## [1] "Soln Slice Width = 0.001953125"
##
## [[3]]
## [1] "True Integral = 0.855624391892149"
##
## [[4]]
## [1] 1.00000000 0.50000000 0.25000000 0.12500000 0.06250000 0.03125000
## [7] 0.01562500 0.00781250 0.00390625

```

```

NewtonRhapson <- function(f, eps = 0.001, x0 = 1, N = 10000) {
  h = 0.001
  i = 1
  x1 = x0
  xstatic = x0
  p = numeric(N)
  while (i <= N) {
    derv = (f(x0 + h) - f(x0))/h
    x1 = (x0 - (f(x0)/derv))
    p[i] = x1
    i = i + 1
    if (abs(x1 - x0) < eps)
      break
    x0 = x1
  }

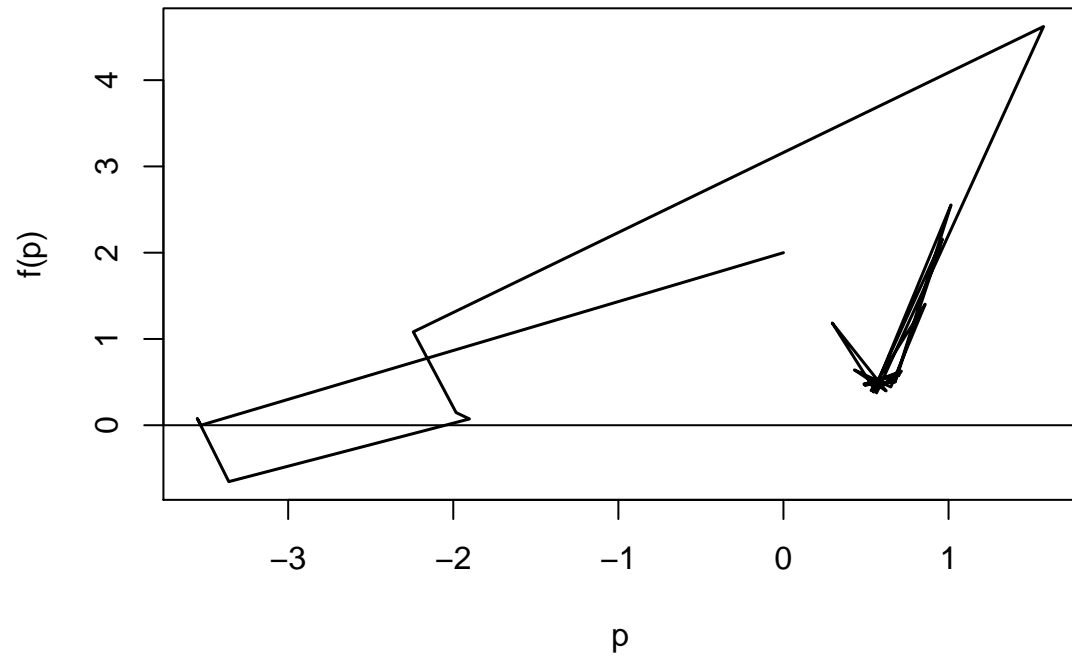
  m <- seq(-50, 50, length = 10000)
  RootGrph <- plot(p, f(p), type = "l", lwd = 1.5, main = expression(f(Newton -
    Rhapson), xlim = c(-9, 9), ylim = c(-5, 5), ylab = "f(x)"))
  abline(h = 0)
  FnGrph <- plot(m, f(m), type = "l", lwd = 1.5, main = expression(f(Domain)),
    xlim = c(-9, 9), ylim = c(-5, 5), ylab = "f(x)")
  abline(h = 0)

  return(list(paste("Solution = ", x1), paste("tolerance = ", eps),
    paste("Initial Guess = ", xstatic), paste("Initial Interval = ",
      xstatic, " to ", xstatic + h), Solution_Iterations = p[1:(i -
        1)], RootGrph, FnGrph))
}

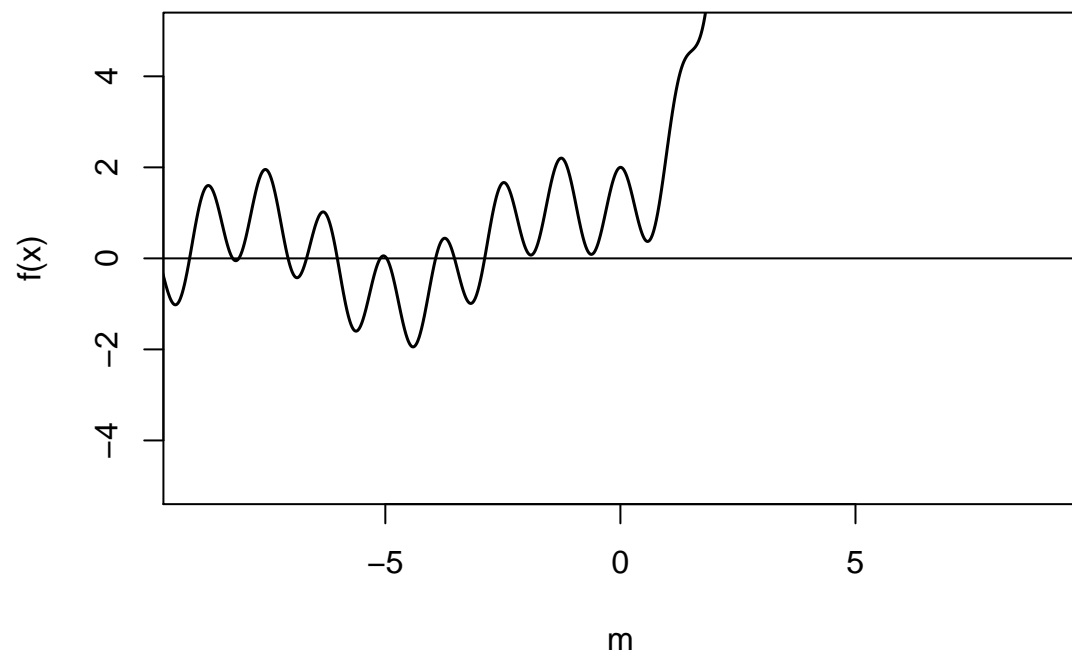
f <- function(x) {
  3^x - sin(x) + cos(5 * x)
}
NewtonRhapson(f)

```

f(Newton – Rhapson)



f(Domain)



```

## [[1]]
## [1] "Solution = -3.52872159591386"
##
## [[2]]
## [1] "tolerance = 0.001"
##
## [[3]]
## [1] "Initial Guess = 1"
##
## [[4]]
## [1] "Initial Interval = 1 to 1.001"
##
## $Solution_Iterations
## [1] 0.6764805 0.4917700 0.7107228 0.5426332 0.9624917 0.6772893
## [7] 0.4932623 0.7141221 0.5466665 1.0147837 0.6741682 0.4873978
## [13] 0.7013121 0.5306938 0.8584602 0.6502529 0.4295528 0.6220424
## [19] 0.2955926 0.5640025 1.5752758 -2.2422711 -1.9828445 -1.9018960
## [25] -3.3596057 -3.5505636 -3.5280662 -3.5287216
##
## [[6]]
## NULL
##
## [[7]]
## NULL

```