STAT 341: Tutorial 8 – Practice with Bootstrap & Prediction

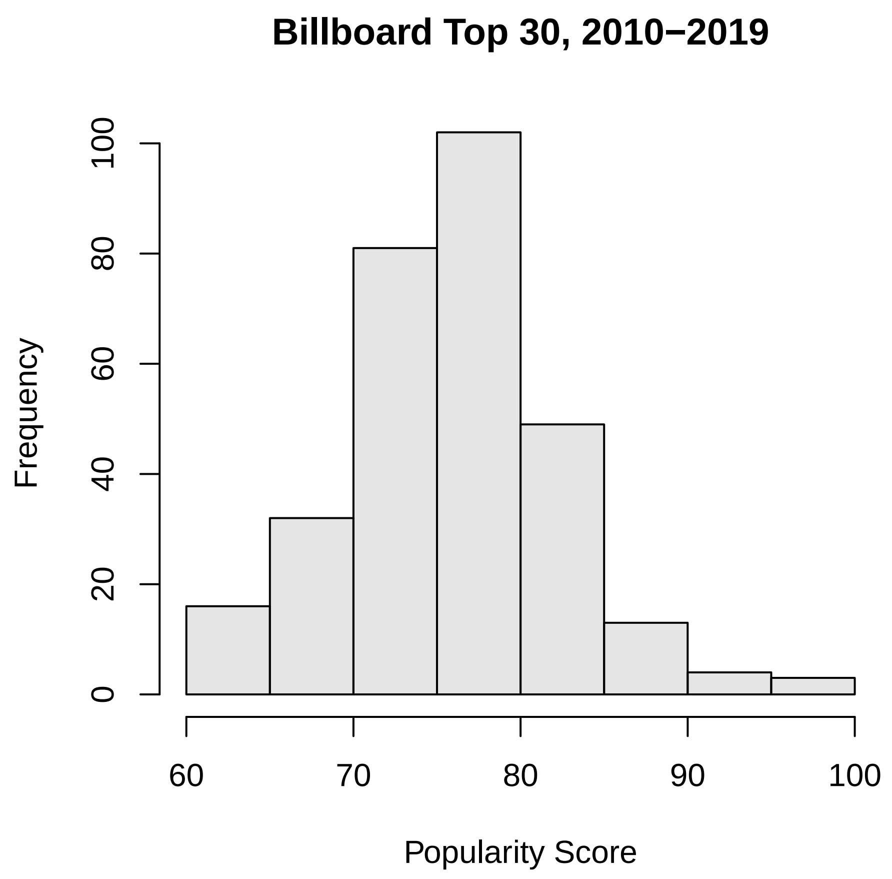
Friday March 27, 2020

Part I: Boostrap Confidence Intervals

Here we will deal with the Billboard Top 30 data again, and in particular the `popularity` score of the $\boxed{N = 300}$ songs.

(a) Load the data and plot a histogram of the `popularity` score.

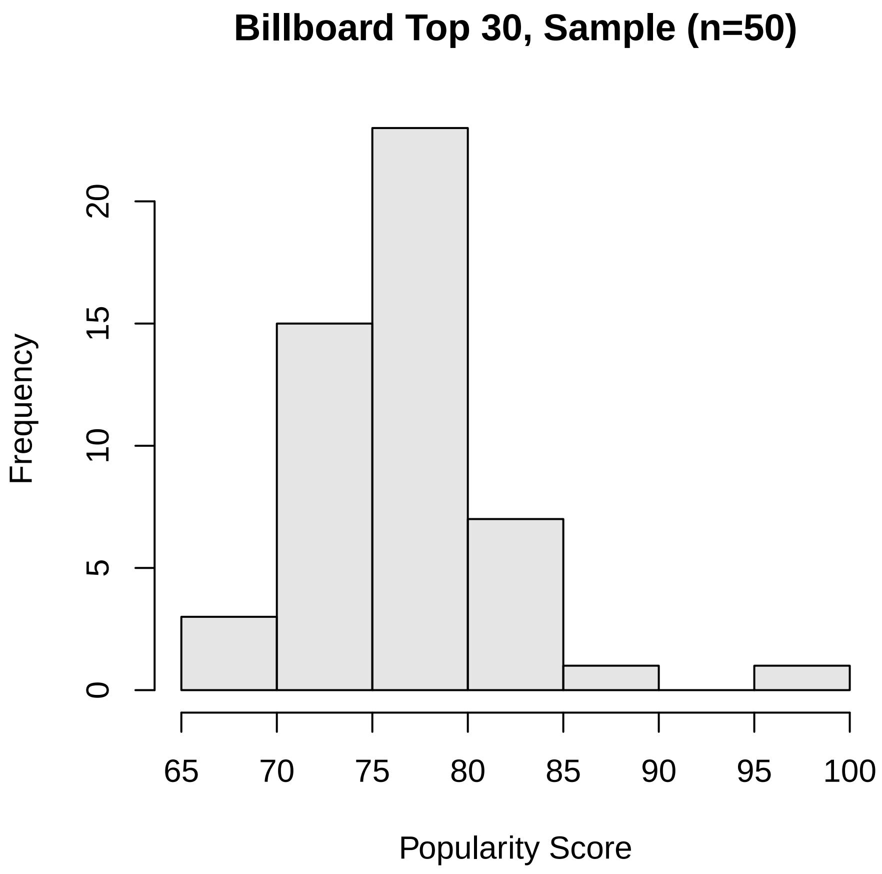
```
spot <- read.csv("/Users/nstevens/Dropbox/Teaching/STAT_341/Assignments/Assignment1/spotify.csv")
hist(spot$popularity, xlab = "Popularity Score", main = "Billboard Top 30, 2010-2019",
    col = adjustcolor("grey", 0.4))
```



(b) Take a sample of size $\boxed{n = 50}$ from this population and plot the `popularity` scores from this sample.

```
# Take the sample
set.seed(341)
N <- dim(spot)[1]
n <- 50
samp.indx <- sample(N, n)
S <- spot[samp.indx, "popularity"]

# Make the plot
```

```r
hist(S, xlab = "Popularity Score", main = "Billboard Top 30, Sample (n=50)",
    col = adjustcolor("grey", 0.4))
```

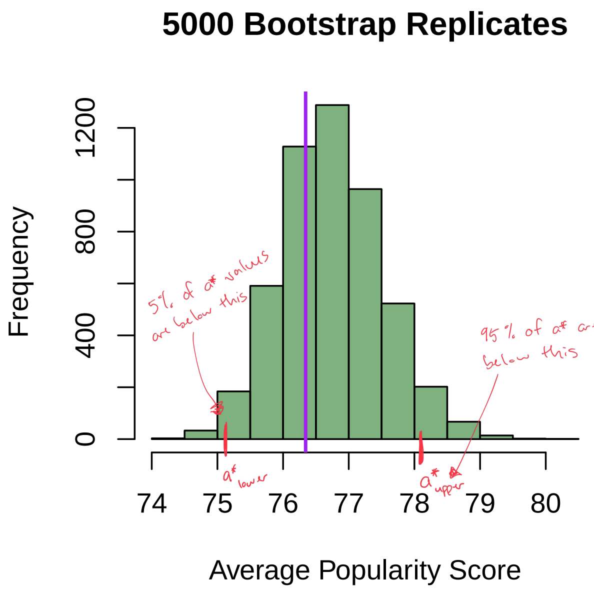


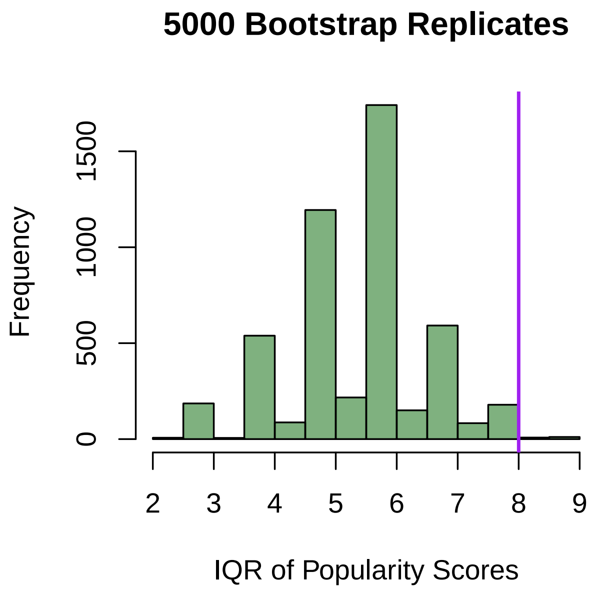
(c) By resampling $\mathcal{S}$ with replacement, construct $\boxed{B = 5000}$ bootstrap samples $\boxed{S_1^\star, S_2^\star, \ldots, S_{5000}^\star}$ and construct histograms of the average and interquartile range calculated on each of those samples. Include vertical lines that indicate the average and IQR in the population.

```r
# Get the bootstrap samples
B <- 5000
Sstar <- sapply(1:B, FUN = function(b) {
    sample(S, n, replace = TRUE)
})
```

* ← returns a 50 × 5000 matrix

```r
# Calculate the average on all of these bootstrap samples
avg_star <- apply(X = Sstar, MARGIN = 2, FUN = mean)

# Calculate the IQR on all of these bootstrap samples
iqr_star <- apply(X = Sstar, MARGIN = 2, FUN = IQR)

# Construct histograms for each of these
par(mfrow = c(1, 2))

hist(avg_star, col = adjustcolor("darkgreen", 0.5), xlab = "Average Popularity Score",
    main = "5000 Bootstrap Replicates")
abline(v = mean(spot$popularity), col = "purple", lwd = 2)

hist(iqr_star, col = adjustcolor("darkgreen", 0.5), xlab = "IQR of Popularity Scores",
    main = "5000 Bootstrap Replicates")
abline(v = IQR(spot$popularity), col = "purple", lwd = 2)
```





(d) Calculate naive normal theory, quantile method, and bootstrap-t 90% confidence intervals for the population average. For the bootstrap-t interval use $B = 5000$ and $D = 100$.

```r
bootstrap_t_interval <- function(S, a, confidence, B, D) {
    ## Inputs: S = an n element array containing the variate values in the
    ## sample a = a scalar-valued function that calculates the attribute a()
    ## of interest confidence = a value in (0,1) indicating the confidence
    ## level B = a numeric value representing the outer bootstrap count of
    ## replicates (used to calculate the lower and upper limits) D = a
    ## numeric value representing the inner bootstrap count of replicates
    ## (used to estimate the standard deviation of the sample attribute for
    ## each (outer) bootstrap sample)

    Pstar <- S
    aPstar <- a(Pstar)
    sampleSize <- length(S)
    ## get (outer) bootstrap values
    bVals <- sapply(1:B, FUN = function(b) {
        Sstar <- sample(Pstar, sampleSize, replace = TRUE)
        aSstar <- a(Sstar)
        ## get (inner) bootstrap values to estimate the SD
        Pstarstar <- Sstar
        SD_aSstar <- sd(sapply(1:D, FUN = function(d) {
            Sstarstar <- sample(Pstarstar, sampleSize, replace = TRUE)
            ## return the attribute value
            a(Sstarstar)
        }))
        z <- (aSstar - aPstar)/SD_aSstar
        ## Return the two values
        c(aSstar = aSstar, z = z)
    })
    SDhat <- sd(bVals["aSstar", ])
    zVals <- bVals["z", ]
    ## Now use these zVals to get the lower and upper c values.
    cValues <- quantile(zVals, probs = c((1 - confidence)/2, (confidence +
        1)/2), na.rm = TRUE)
    cLower <- min(cValues)
    cUpper <- max(cValues)
    interval <- c(lower = aPstar - cUpper * SDhat, middle = aPstar, upper = aPstar -
        cLower * SDhat)
    interval
}
```

$\left[a(S) - c \times \widehat{SD}_*[\tilde{a}(S)],\ a(S) + c \times \widehat{SD}_*[\tilde{a}(S)]\right]$

c is determined from $Z \sim N(0,1)$

```
# Naive normal theory
mean(S) + qnorm(0.95) * c(-1, 1) * sd(avg_star)
```

$a(S)$ $\qquad$ $\widehat{SD}_*[\tilde{a}(S)]$

```
## [1] 75.50064 78.01936
```

```
# Quantile method
c(quantile(avg_star, 0.05), quantile(avg_star, 0.95))
```

```
##    5%   95%
## 75.54 78.06
```

determined from Z^*

```
# Bootstrap-t
bootstrap_t_interval(S = S, a = mean, confidence = 0.9, B = 5000, D = 100)
```

```
##    lower   middle    upper
## 75.61863 76.76000 78.15993
```

$\left[a(S) - c_{upper} \times \widehat{SD}_*[\tilde{a}(S)],\ a(S) - c_{lower} \times \widehat{SD}_*[\tilde{a}(S)]\right]$

(e) Calculate naive normal theory, quantile method, and bootstrap-t 90% confidence intervals for the population IQR. For the bootstrap-t interval use $B = 5000$ and $D = 100$.

```
# Naive normal theory
IQR(S) + qnorm(0.95) * c(-1, 1) * sd(iqr_star)
```

```
## [1] 4.16858 7.83142
```

```
# Quantile method
c(quantile(iqr_star, 0.05), quantile(iqr_star, 0.95))
```

```
##   5%  95%
## 3.75 7.50
```

```
# Bootstrap-t
bootstrap_t_interval(S = S, a = IQR, confidence = 0.9, B = 5000, D = 100)
```

```
##    lower   middle    upper
## 4.702215 6.000000 8.485624
```

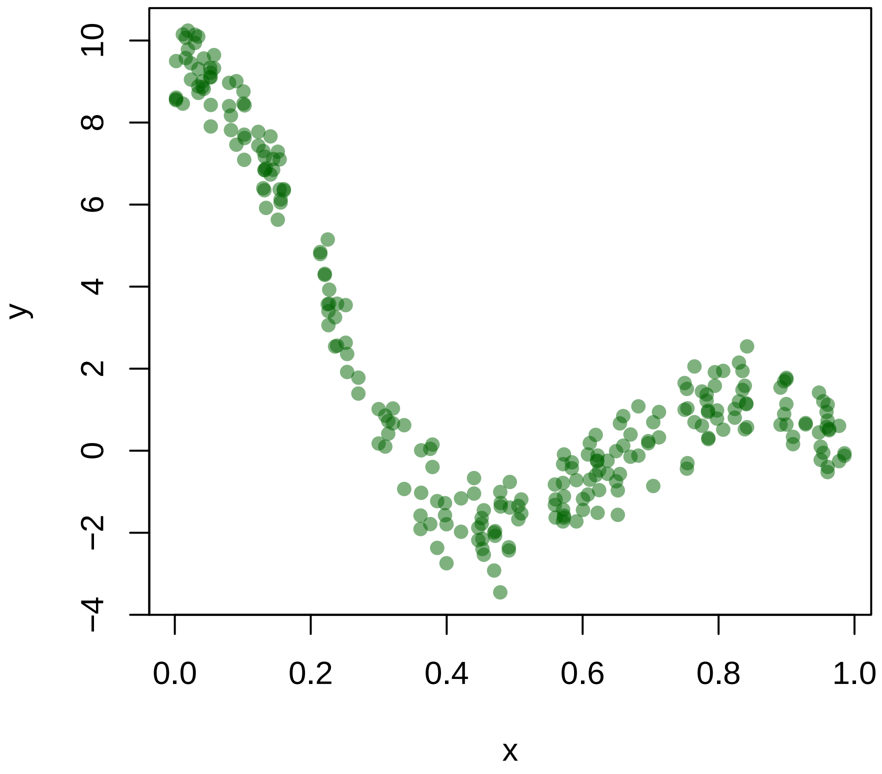
Part II: Predictions and APSE Decomposition

Here we will use some simulated data as our population, from which we will draw a sample, fit various models, and evaluate them on a test set. The data can be found in the `fakedata.csv` file and the relevant variates are `x` and `y`.

$$APSE(P, \tilde{\mu}_S(x)) = \underset{\text{irreducible error}}{Ave_x(Var(y|x))} + \underbrace{Var[\tilde{\mu}_S(x)] + Bias[\tilde{\mu}_S(x)]^2}$$

(a) Load and visualize the data.

```
fake <- read.csv("/Users/nstevens/Dropbox/Teaching/STAT_341/Tutorials/Tutorial 8/fakedata.csv",
    header = TRUE)
plot(fake, pch = 16, col = adjustcolor("darkgreen", 0.5))
```

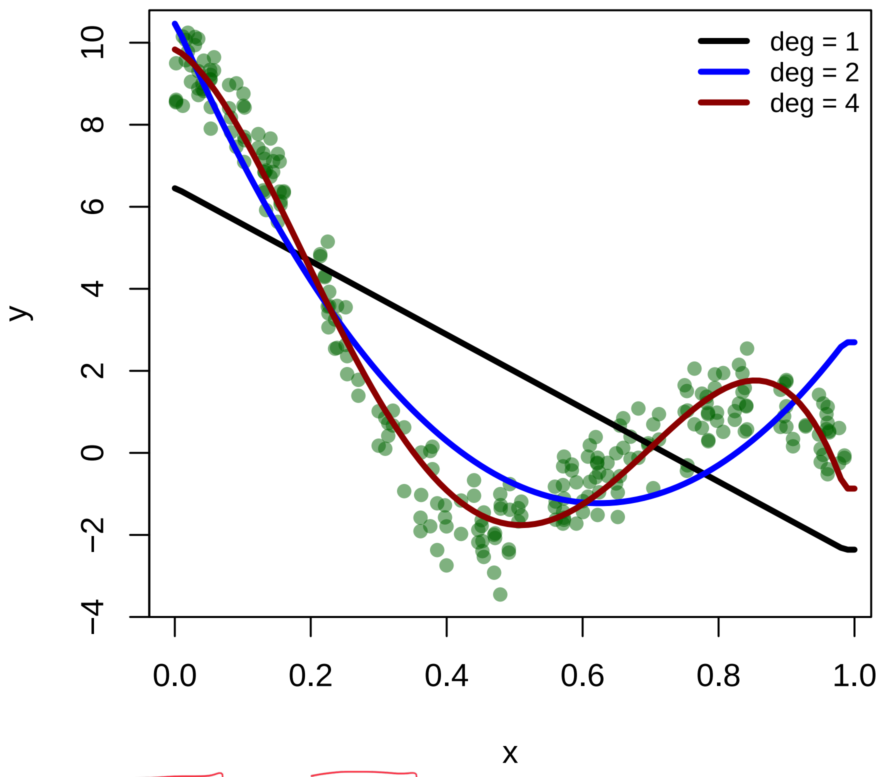


(b) Recreate the plot above, but this time overlay linear, quadratic and quartic polynomials, fitted using all of the data. Make sure the curves have different colours and are distinguished with a legend. **Note:** the getmuhat function will be useful.

```
# Define useful function
getmuhat <- function(sampleXY, complexity = 1){
  formula <- paste0("y ~ ",
                    if (complexity==0) {
                      "1"
                    } else
                      paste0("poly(x, ", complexity, ", raw = FALSE)")
                    #paste0("bs(x, ", complexity, ")")
  )

  fit <- lm(as.formula(formula), data = sampleXY)
  tx = sampleXY$x
  ty = fit$fitted.values

  range.X = range(tx)
  val.rY  = c( mean(ty[tx == range.X[1]]),
               mean(ty[tx == range.X[2]]) )

  ## From this we construct the predictor function
  muhat <- function(x){
    if ("x" %in% names(x)) {
```

```r
      ## x is a dataframe containing the variate named
      ## by xvarname
      newdata <- x
    } else
      ## x is a vector of values that needs to be a data.frame
    { newdata <- data.frame(x = x) }
    ## The prediction
    ##
    val = predict(fit, newdata = newdata)
    val[newdata$x < range.X[1]] = val.rY[1]
    val[newdata$x > range.X[2]] = val.rY[2]
    val
  }
  ## muhat is the function that we need to calculate values
  ## at any x, so we return this function from getmuhat
  muhat
}
```

```r
# Make the plot
plot(fake, pch = 16, col = adjustcolor("darkgreen", 0.5))

# Fit a linear polynomial and add it to the plot
muhat1 <- getmuhat(fake, 1)
curve(muhat1, from = 0, to = 1, add = TRUE, col = "black", lwd = 3)

# Fit a quadratic polynomial and add it to the plot
muhat2 <- getmuhat(fake, 2)
curve(muhat2, from = 0, to = 1, add = TRUE, col = "blue", lwd = 3)

# Fit a quartic polynomial and add it to the plot
muhat4 <- getmuhat(fake, 4)
curve(muhat4, from = 0, to = 1, add = TRUE, col = "darkred", lwd = 3)

# Add a legend
legend("topright", legend = c("deg = 1", "deg = 2", "deg = 4"), col = c("black",
    "blue", "darkred"), lwd = 3, cex = 0.8, bty = "n")
```



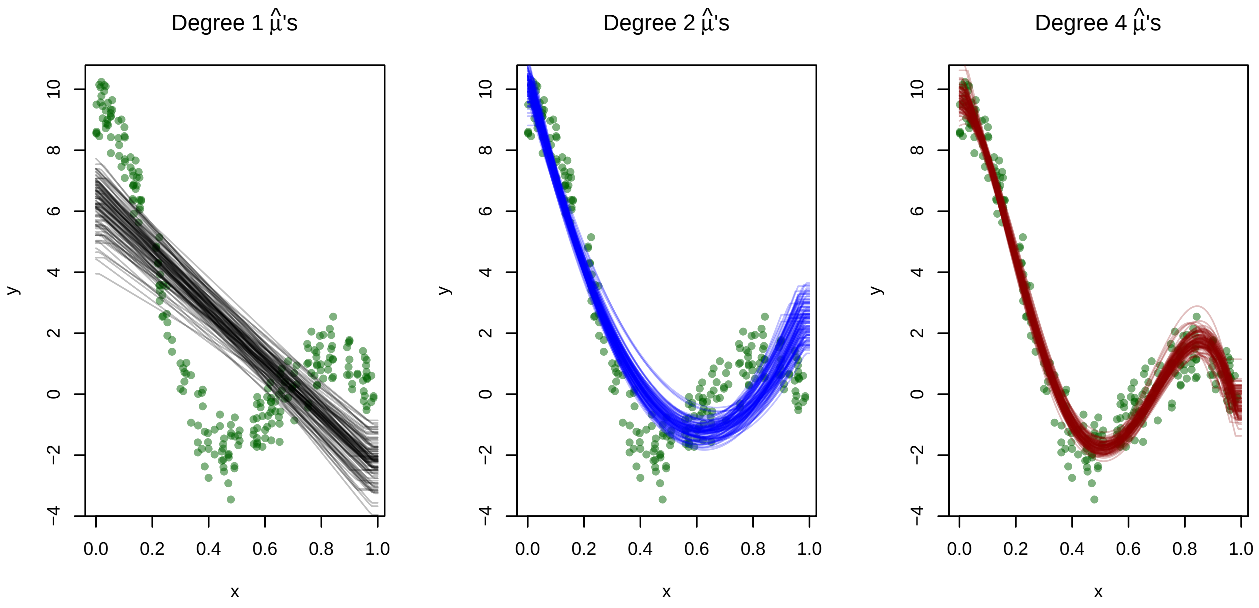
(c) Generate $N_S = 100$ samples of size $n = 40$ and fit polynomials of degree 1, 2 and 4 to every sample. **Note:** the `getSampleComp` and `getXYSample` functions will be useful.

```r
# Define useful functions
getSampleComp <- function(pop, size, replace = FALSE) {
    N <- nrow(as.data.frame(pop))
    samp <- rep(FALSE, N)
    samp[sample(1:N, size, replace = replace)] <- TRUE
    samp
}

getXYSample <- function(xvarname, yvarname, samp, pop) {
    sampData <- pop[samp, c(xvarname, yvarname)]
    names(sampData) <- c("x", "y")
    sampData
}
```

```r
N_S <- 100
n <- 40
set.seed(341)

# Use getSampleComp via lappy to determine which units to include in
# the 100 samples.  samps is a list with 100 elements, each of which is
# a 250-element array, 40 of which are TRUE and 210 of which are FALSE.
# This tells us which units to include in each of the 100 samples
samps <- lapply(1:N_S, FUN = function(i) {
    getSampleComp(fake, n)
})

# getXYsample is applied to fake and each element of samps. The code
# below returns Ssamples which is another 100-element list, but each of
# these elements is a data frame with columns x and y and containing
# just the sample data.
Ssamples <- lapply(samps, FUN = function(Si) {
    getXYSample("x", "y", Si, fake)
})

# Tsamples is the complement of Ssamples; it is a 100-element list,
# where each of the elements is a data frame with columns x and y and
# containing just the test data.
Tsamples <- lapply(samps, FUN = function(Si) {
    getXYSample("x", "y", !Si, fake)
})

# This code applies getmuhat to each element of Ssamples and thereby
# fits a polynomial of the given complexity to every sample and saves
# those results in a list
muhats1 <- lapply(Ssamples, getmuhat, complexity = 1)
muhats2 <- lapply(Ssamples, getmuhat, complexity = 2)
muhats4 <- lapply(Ssamples, getmuhat, complexity = 4)
```

(d) Using `par(mfrow=c(1,3))` plot the data and overlay all the fitted polynomials from part (c) with degree 1 depicted in the left plot, degree 2 depicted on the middle plot, and degree 4 in the right plot. Use colours that are consistent with part (b).

```r
par(mfrow = c(1, 3))

# Plot the data
plot(fake, pch = 16, col = adjustcolor("darkgreen", 0.5), main = bquote("Degree 1" ~
    hat(mu) * "'s"))
# Add the degree 1 curves
for (i in 1:N_S) {
    curveFn <- muhats1[[i]]
    curve(curveFn, from = 0, to = 1, add = TRUE, col = adjustcolor("black",
        0.25))
}

# Plot the data
plot(fake, pch = 16, col = adjustcolor("darkgreen", 0.5), main = bquote("Degree 2" ~
    hat(mu) * "'s"))
# Add the degree 2 curves
for (i in 1:N_S) {
    curveFn <- muhats2[[i]]
    curve(curveFn, from = 0, to = 1, add = TRUE, col = adjustcolor("blue",
        0.25))
}

# Plot the data
plot(fake, pch = 16, col = adjustcolor("darkgreen", 0.5), main = bquote("Degree 4" ~
    hat(mu) * "'s"))
# Add the degree 4 curves
for (i in 1:N_S) {
    curveFn <- muhats4[[i]]
    curve(curveFn, from = 0, to = 1, add = TRUE, col = adjustcolor("darkred",
        0.25))
}
```



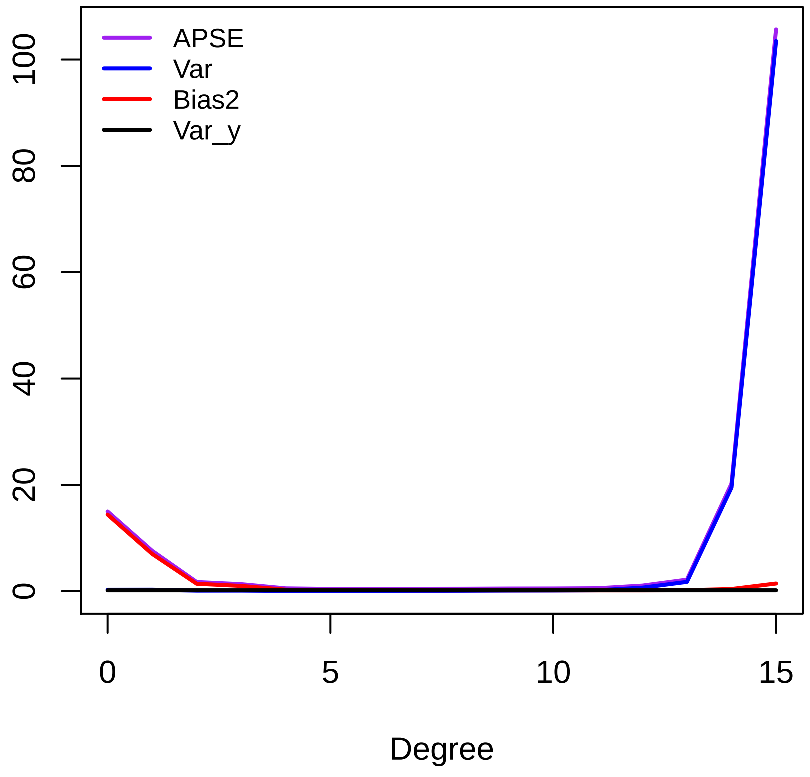
(e) Using the $N_S = 100$ samples of size $n = 40$ generated in part (c), calculate the APSE (and each of its components) for degrees in `0:15`. Summarize the results in tabular and graphical format. **Note:** the `apse_all`, `getmubar` and `getmuFun` functions will be useful.

```
# Define useful functions
apse_all <- function(Ssamples, Tsamples, complexity, mu) {
    ## average over the samples S
    N_S <- length(Ssamples)
    muhats <- lapply(Ssamples, FUN = function(sample) getmuhat(sample,
        complexity))
    ## get the average of these, mubar
    mubar <- getmubar(muhats)

    rowMeans(sapply(1:N_S, FUN = function(j) {
        T_j <- Tsamples[[j]]
        muhat <- muhats[[j]]
        ## Take care of any NAs
        T_j <- na.omit(T_j)
        y <- T_j$y
        x <- T_j$x
        mu_x <- mu(x)
        muhat_x <- muhat(x)
        mubar_x <- mubar(x)

        ## apse average over (x_i,y_i) in a single sample T_j the squares (y -
        ## muhat(x))^2
        apse <- (y - muhat_x)

        ## bias2: average over (x_i,y_i) in a single sample T_j the squares (y -
        ## muhat(x))^2
        bias2 <- (mubar_x - mu_x)
        ## var_mutilde average over (x_i,y_i) in a single sample T_j the squares
        ## (y - muhat(x))^2
        var_mutilde <- (muhat_x - mubar_x)

        ## var_y : average over (x_i,y_i) in a single sample T_j the squares (y
        ## - muhat(x))^2
        var_y <- (y - mu_x)

        ## Put them together and square them
        squares <- rbind(apse, var_mutilde, bias2, var_y)^2

        ## return means
        rowMeans(squares)
    }))
}

getmubar <- function(muhats) {
    function(x) {
        Ans <- sapply(muhats, FUN = function(muhat) {
            muhat(x)
        })
        apply(Ans, MARGIN = 1, FUN = mean)
    }
}

getmuFun <- function(pop, xvarname, yvarname) {
    pop = na.omit(pop[, c(xvarname, yvarname)])

    # rule = 2 means return the nearest y-value when extrapolating, same as
    # above.  ties = mean means that repeated x-values have their y-values
    # averaged, as above.
    muFun = approxfun(pop[, xvarname], pop[, yvarname], rule = 2, ties = mean)
    return(muFun)
}
```

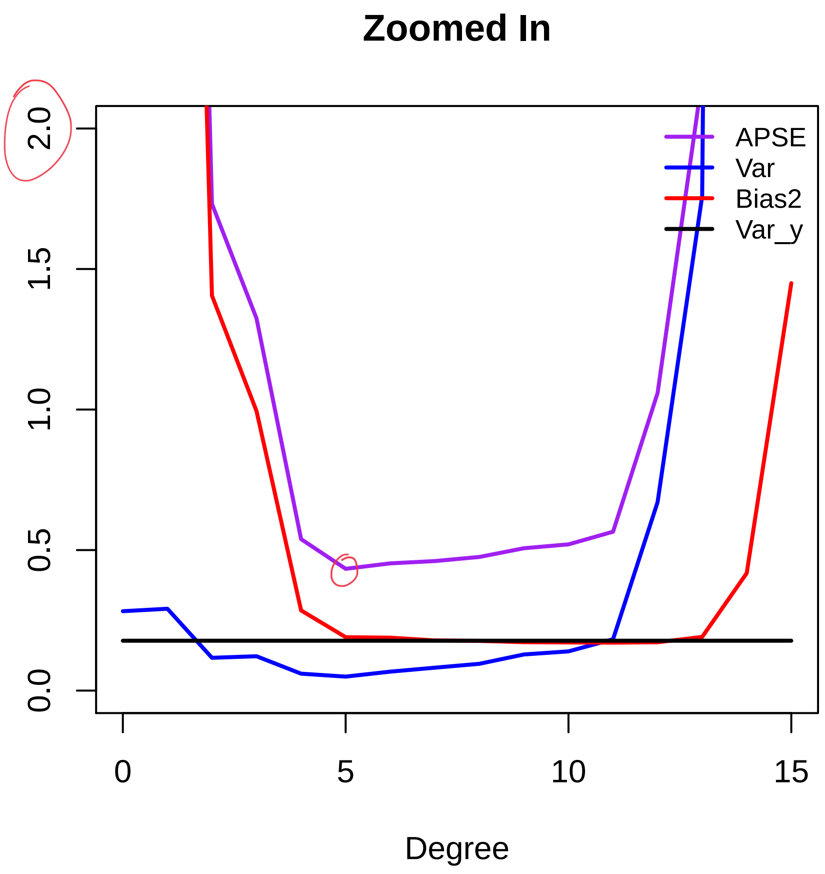
```
muhat <- getmuFun(fake, "x", "y")
degrees <- 0:15
apse_vals <- sapply(degrees, FUN = function(complexity) {
    apse_all(Ssamples, Tsamples, complexity = complexity, mu = muhat)
})

# Print out the results in a table
t(rbind(degrees, apse = round(apse_vals, 5)))
```

```
##       degrees     apse var_mutilde    bias2   var_y
##  [1,]       0 15.00455     0.28249 14.42296 0.17746
##  [2,]       1  7.59809     0.29122  7.02166 0.17746
##  [3,]       2  1.73165     0.11673  1.40530 0.17746
##  [4,]       3  1.32417     0.12218  0.99462 0.17746
##  [5,]       4  0.53903     0.06021  0.28535 0.17746
##  [6,]       5  0.43334     0.04975  0.19005 0.17746
##  [7,]       6  0.45271     0.06744  0.18825 0.17746
##  [8,]       7  0.46081     0.08152  0.17853 0.17746
##  [9,]       8  0.47552     0.09539  0.17685 0.17746
## [10,]       9   0.50650     0.12853  0.17220 0.17746
## [11,]      10   0.52031     0.13963  0.17131 0.17746
## [12,]      11   0.56527     0.18261  0.17050 0.17746
## [13,]      12   1.05900     0.67066  0.17209 0.17746
## [14,]      13   2.17739     1.75961  0.19093 0.17746
## [15,]      14  20.23517    19.49681  0.41786 0.17746
## [16,]      15 105.66131   103.48264  1.44968 0.17746
```

```
# Plot the results in a graph
plot(degrees, apse_vals[1, ], xlab = "Degree", ylab = "", type = "l", ylim = c(0,
    max(apse_vals)), col = "purple", lwd = 2)
lines(degrees, apse_vals[2, ], col = "blue", lwd = 2)
lines(degrees, apse_vals[3, ], col = "red", lwd = 2)
lines(degrees, apse_vals[4, ], col = "black", lwd = 2)
legend("topleft", legend = c("APSE", "Var", "Bias2", "Var_y"), col = c("purple",
    "blue", "red", "black"), lwd = 2, bty = "n", cex = 0.8)
```



```
# Make another 'zoomed-in' graph:
plot(degrees, apse_vals[1, ], xlab = "Degree", ylab = "", type = "l", ylim = c(0,
    2), col = "purple", lwd = 2, main = "Zoomed In")
lines(degrees, apse_vals[2, ], col = "blue", lwd = 2)
lines(degrees, apse_vals[3, ], col = "red", lwd = 2)
lines(degrees, apse_vals[4, ], col = "black", lwd = 2)
legend("topright", legend = c("APSE", "Var", "Bias2", "Var_y"), col = c("purple",
    "blue", "red", "black"), lwd = 2, bty = "n", cex = 0.8)
```

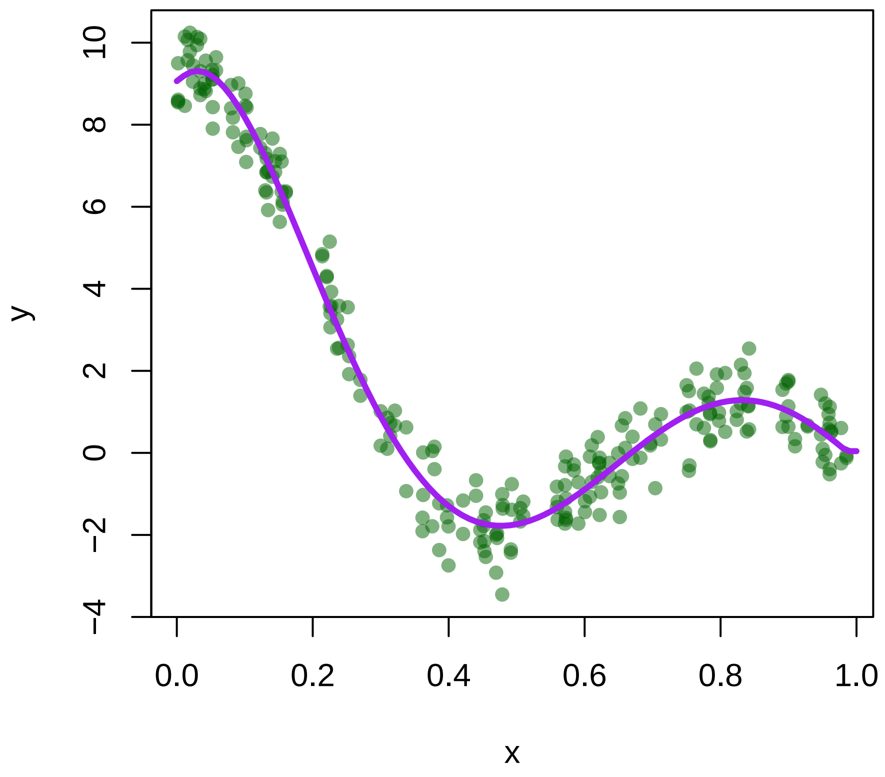


(f) Plot the data again, and this time overlay the best polynomial as determined by your findings in (e).

```r
# Plot the data
plot(fake, pch = 16, col = adjustcolor("darkgreen", 0.5))

# Get the best degree and fit a polynomial with this degree to all of
# the data.
best.deg <- degrees[which.min(apse_vals[1, ])]
print(best.deg)
```

```
## [1] 5
```

```r
muhat.best <- getmuhat(fake, best.deg)

# Add the curve to the plot
curve(muhat.best, from = 0, to = 1, add = TRUE, col = "purple", lwd = 3)
```


y
10
8
6
4
2
0
−2
−4
0.0
0.2
0.4
0.6
0.8
1.0
x