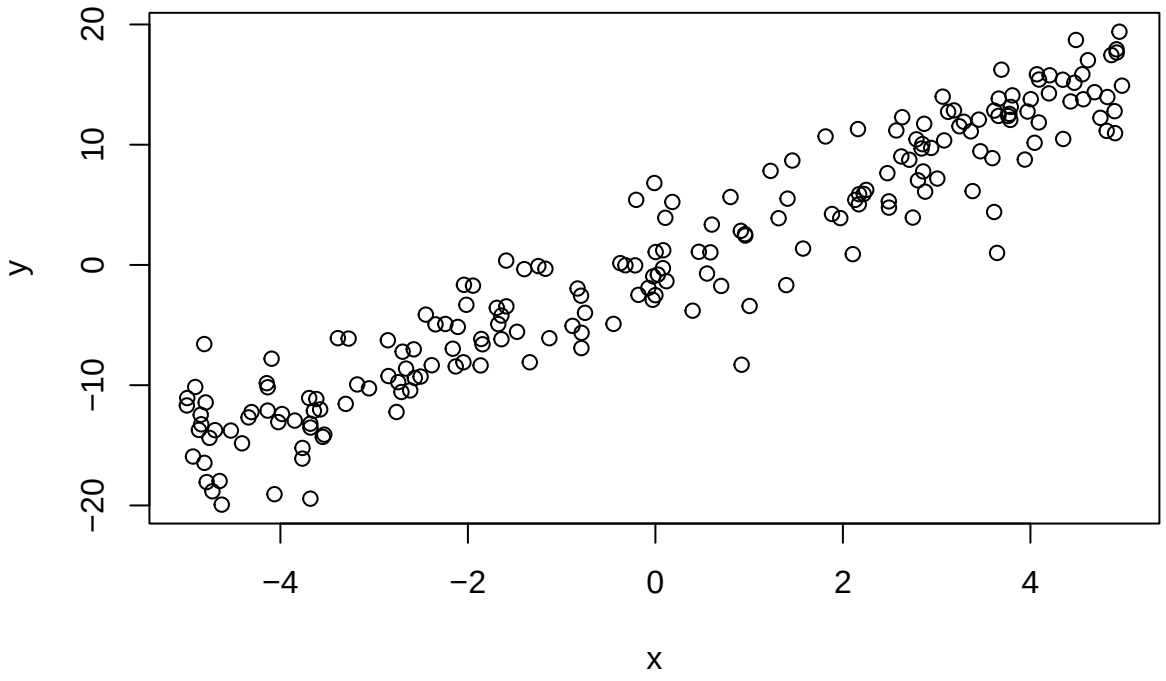
Model Selection Coursework

Student Name: Harleen Gulati, Student Number: 2101550

2024-04-24

Question 1 (a)

```
load('xy.Rdata')
plot(x, y)
```



Question 1 (b)

```
ps <- runif(80, 0, 5) # generate a set of 80 samples of p drawn uniformly from [0,5]
residuals <- rep(0, 80) # each sample of p, the corresponding residual is stored in residuals
for (i in 1:80) {
  ys = ps[i] * x # for each sample of p, find the predicted values of y
  residuals[i] = 1/length(y) * (sum((y-ys)^2)) # then find the corresponding residual sum of squares on
}
```

Question 1 (c)

The mean residual sum of squared across P_1 is $\mu_1 = 33.41458$

```
mu1 = mean(residuals)
print(mu1)
```

```
## [1] 33.57993
```

Question 1 (d)

P_1 has 80 values of ρ and thus we want 20% * 80 = 16 values of ρ. The best 20% of values are the 16 values with the smallest residual sum of squares on the data.

The mean residual sum of squares across P_1^* is 10.26303.

```
sorted = order(residuals) # gets indices of the sorted residuals list (i.e., if the list is 10, 11, 3 t
sorted_best = sorted[1:16] # gets the 16 smallest residuals indices
p1_star = ps[sorted_best] # stores the best 16 parameters corresponding to the smallest residuals (i.e.
residuals_star = residuals[sorted_best] # stores the first 16 smallest residuals

mu_1_star = mean(residuals_star) # calculates the mean residual sum of squares across P1 star
print(mu_1_star)
```

```
## [1] 9.921553
```

Question 1 (e)

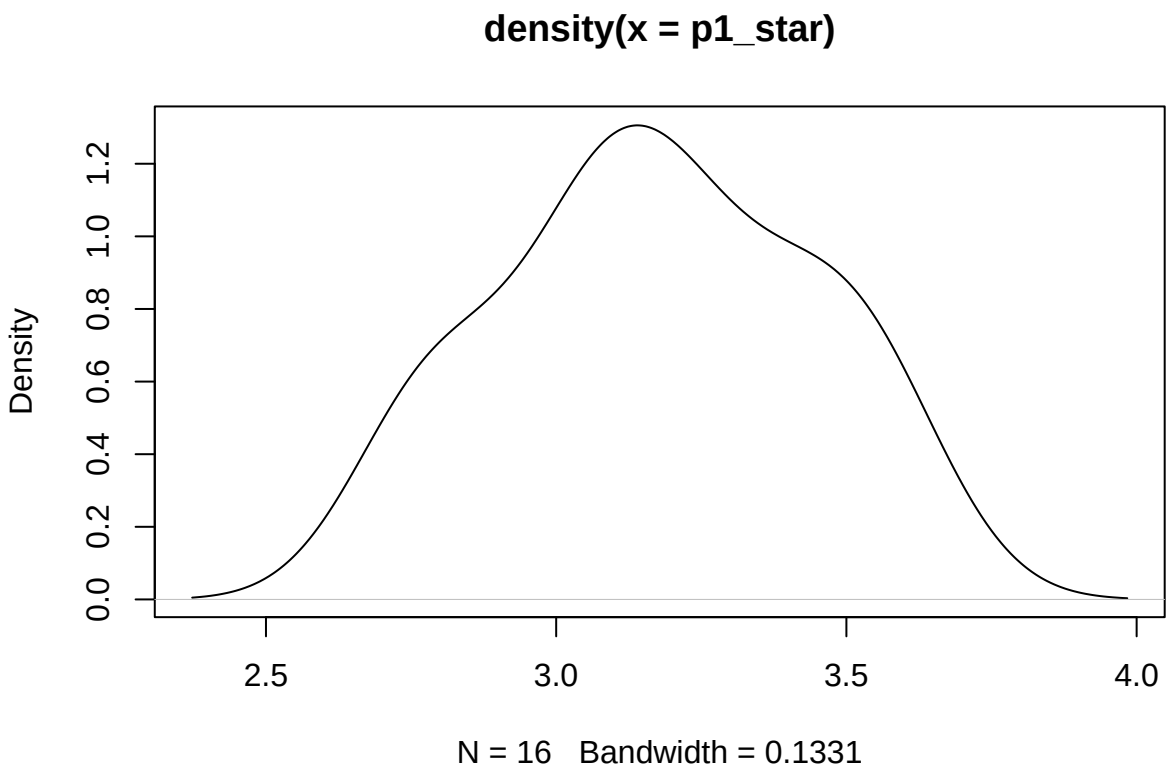
The kernel density estimate a(t) for the distribution of ρ in P_1^* is computed below and has a bandwidth of 0.1625.

```
a <- density(p1_star) # kernel density estimate for p in P1*
print(a)
```

```
##
## Call:
##  density.default(x = p1_star)
##
## Data: p1_star (16 obs.); Bandwidth 'bw' = 0.1331
##
##        x               y
##  Min.   :2.373   Min.   :0.003289
##  1st Qu.:2.776   1st Qu.:0.143867
##  Median :3.178   Median :0.675551
##  Mean   :3.178   Mean   :0.619923
##  3rd Qu.:3.581   3rd Qu.:1.006262
##  Max.   :3.984   Max.   :1.305754
```

Question 1 (f)

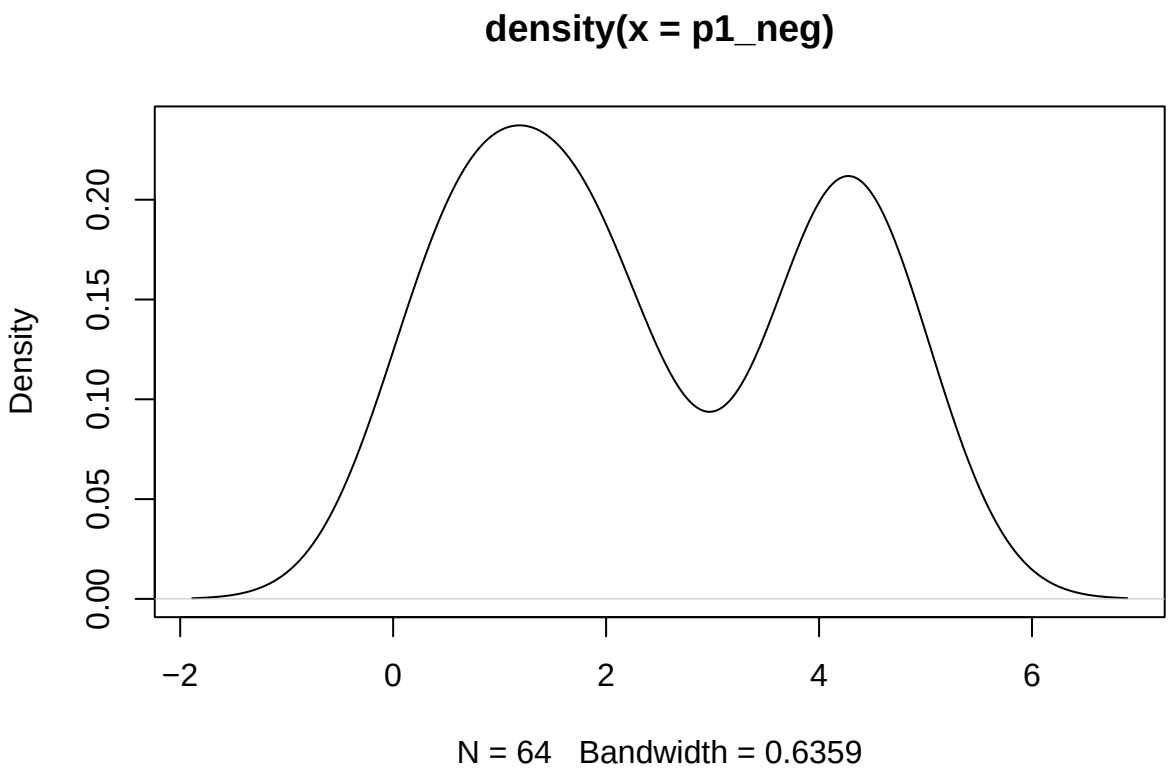
```
plot(a)
```



Question 1 (g)

The density b(t) for the distribution of values of ρ in P_1 P_1^* has a bandwidth of 0.6422

```
sorted = order(residuals) # gets indices of the sorted residuals list (i.e., if the list is 10, 11, 3 t
sorted_worst = sorted[17:80] # gets the worst 80% values indices (i.e. the 80% of values with the large
p1_neg = ps[sorted_worst] # stores the ps corresponding to the worst 80% values
residuals_neg = residuals[sorted_worst] # stores the largest 80% residuals
b <- density(p1_neg) # kernel density estimate for p in P1 \ P1*
plot(b)
```



Question 1 (h)

```
p2 =  sample(a$x, 100, replace = TRUE, prob = a$y) # generates 100 samples from a$x (i.e., the data to
```

Question 1 (i)

The mean residual sum of squares across P_2 is 10.67671

```
residuals = rep(0, 100) # store all the residual sum of squares for each p in P2
for (i in 1:100) {
  ys = p2[i] * x # predicted value of ys for each p in P2
  residuals[i] = 1/length(y) * sum((y-ys)^2) # residual sum of squares for each p in P2
}
print(mean(residuals)) # outputs mean residual sum of squares across P2
```

```
## [1] 10.22927
```

Question 1 (j)

```
rs = rep(0, 100) # stores all the rs (where each r = b(t)^-1 * a(t))
for (i in 1:100) {
  t = p2[i] # gets the rho value in P2
  df_a = approxfun(a) # gets a function to estimate values for a(t)
  at = df_a(t) # uses this function to estimate the a(t)
  df_b = approxfun(b) # function to estimate values for b(t)
  bt = df_b(t) # uses this function to estimate b(t)
  rs[i] = at / bt # calculates r and stores in list of all rs
}
```

Question 1 (k)

The values of ρ giving the best corresponding values of r are the values of ρ which are more likely to be observed under $a(t)$ than under $b(t)$. This is because $a(t)$ modeled the best 20% of values of ρ from P_1 and $b(t)$ modeled the worst 80% of values of ρ from P_1. We want our values of rho to be more likely to be sampled from the distribution modelling the best values of ρ thus we want $a(t) > b(t)$ thus we want $\frac{a(t)}{b(t)}$ to be as large as possible hence we choose the largest 20 values of r.

```
sorted = order(rs) # gets indices of the sorted rs list (i.e., if the list is 10, 11, 3 then sorted ret
sorted_best = sorted[80:100] # gets the best 20 values indices (i.e.  20 values with the largest r)
p2star = p2[sorted_best] # stores the ps corresponding to the best 20 values
rsbest = rs[sorted_best] # stores the largest 20 residuals
```

Question 1 (l)

The mean residual sum of squares across P_2^* is $\mu_2^* = 10.03707$

From previous questions, we have $\mu_1 = 33.41458, \mu_1^* = 10.26303, \mu_2 = 10.67671, \mu_2^* = 10.03707$

When we consider 80 samples of ρ drawn uniformly from [0,5] we get the largest mean residual sum of squares μ_1; when we then filter our samples to only consider the best 20% of values (i.e., the values of ρ with the smallest residual sum of squares) our mean residual sum of squares μ_1^* significantly reduces too. When we then fit a distribution to the best values of ρ and find the mean residual sum of squares from 100 samples of this distribution μ_2, we find it similar to μ_1^* but slightly larger.

When we then filter values from P_2, we observe the mean residual sum of squares μ_2^* smaller compared to μ_2.

Thus, filtering out best values of ρ for both P_1 and P_2 helps to obtain a collection of values wherein the mean residual sum of squares is much smaller.

μ_1 is also significantly larger than μ_2 thus taking values from a distribution of previously chosen best values of ρ will yield a smaller mean residual sum of squares than taking values from a uniform distribution with no/little knowledge of where the best values of ρ lie.

```
residuals = rep(0, 20) # list of all residuals for values of p in P2*
for (i in 1:20) {
  p = p2star[i] # gets p from P2*
  ys = p * x # gets estimates values given p
  residuals[i] = 1/length(y) * sum((y-ys)^2) # gets residuals for p
}
mu2star = mean(residuals) # gets mu2star
print(mu2star)
```

```
## [1] 9.275793
```

Question 1 (m)

The prior distribution for sampling ρ is the uniform [0,5] distribution and the posterior distribution for sampling ρ is the density a(t) calculated in question 1 (e). This is because with no knowledge of where good values of ρ lie, we chose to sample values from the uniform [0,5] distribution and then when we chose the best samples from the uniform [0,5] distribution we gained some knowledge of where good values of ρ lie. We fitted a distribution to these good values of ρ a(t) and thus a(t) is the posterior distribution since it encodes information gained about the 'good values' of ρ after observing samples.

Question 1 (n)

The method requires beginning with a prior distribution to sample initial values of ρ from (which in our case was a uniform [0,5] distribution).

Consider a plot of the residual sum of squares against all the possible values of ρ. Then this plot may have one global minimum, but many local minima. If the prior distribution is chosen close to a local minima which is not the global minimum, then the 'good samples' from this distribution will lie close to the local minima (i.e. values with a smaller residual sum of squares).

Thus, the 'good' distribution fitted to the 'good' samples of the prior will model values closer to the local minima with higher probabillity; when sampling from this 'good' distribution, we are likely to ger values closer to the local minima.

Once we have sampled and chosen values from the 'good' distribution, we iterate the process and so we have a collection of samples close to the local minima and this process repeats (i.e. split into good and bad values, fit distributions, sample from good distribution, decide which values to keep).

Thus, we see iterating through the process could potentially continuously give values close to the local minima and thus we could get 'stuck' within the local minima and be unable to escape, hence iterating this method is not necessarily going to converge to the best fit model if we choose a prior close to a local minima which is not the global minima (of the graph of residual sum of squares against values of ρ).

Question 1 (o)

We find the optimal value of ρ as follows:

We begin with 100 values from the uniform [0,5] distribution and the residual sum of squares of these samples. We then choose the sample minimizing the AIC criterion for a regression model (since the plot in question 1 (a)) suggested a linear relationship between x and y. Choosing the value of ρ minimizing the AIC implies we have chosen the value of ρ from our sample which best balances overfitting and underfitting.

This method gives the optimal value of ρ to be 3.144667.

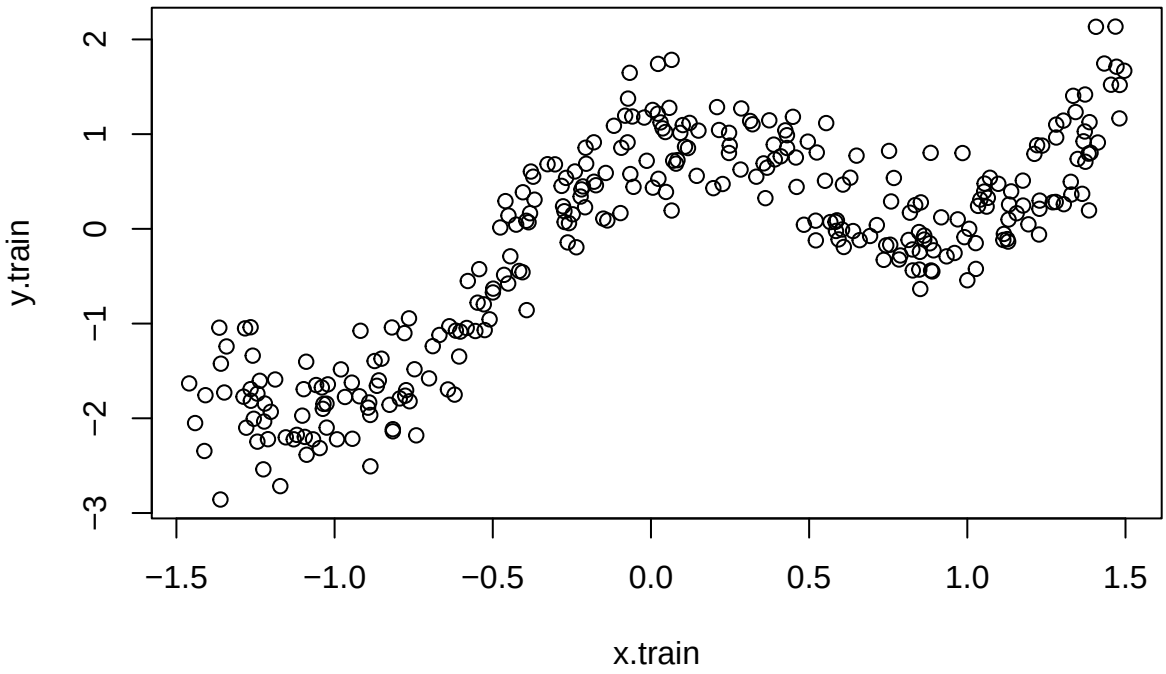
```
ps = runif(100, 0, 5) # generate a set of 100 samples of p drawn uniformly from [0,5]
residuals = rep(0, 100) # stores the residuals of each sample
for (i in 1:100) {
  p = ps[i] # gets the ith sample
  ys = p * x # gets the prediction from this sample
  residuals[i] = 1/length(y) * sum((y-ys)^2) # gets the residual sum of squares from this sample
}

minAIC = 100000000 # set min to arbitrary large number
index = -1 # stores index value of p with smallest AIC
N = length(x) # number of samples we have
p = 1 # model has one parameter since x is 1 dimensional
for (i in 1:100) { # for each value of p
  currAIC = N * log(residuals[i]/N) + 2*p # find the AIC
  if (currAIC < minAIC) { # see if the AIC is the new minimum
    minAIC = currAIC
    index = i
    }
}
print(ps[index])
```

```
## [1] 3.116932
```

Question 2

Question 2 (a)

```
load('curve.RData')
plot(x.train, y.train)
```




```
print(data)
```

```
## function (..., list = character(), package = NULL, lib.loc = NULL,
##     verbose = getOption("verbose"), envir = .GlobalEnv, overwrite = TRUE)
## {
##     fileExt <- function(x) {
##         db <- grepl("\\.[^.]+\\.(gz|bz2|xz)$", x)
##         ans <- sub(".*\\.", "", x)
##         ans[db] <- sub(".*\\.([^.]+\\.)(gz|bz2|xz)$", "\\1\\2",
##             x[db])
##         ans
##     }
##     my_read_table <- function(...) {
##         lcc <- Sys.getlocale("LC_COLLATE")
##         on.exit(Sys.setlocale("LC_COLLATE", lcc))
##         Sys.setlocale("LC_COLLATE", "C")
##         read.table(...)
##     }
##     stopifnot(is.character(list))
##     names <- c(as.character(substitute(list(...)))[-1L]), list)
##     if (!is.null(package)) {
##         if (!is.character(package))
##             stop("'package' must be a character vector or NULL")
##     }
##     paths <- find.package(package, lib.loc, verbose = verbose)
##     if (is.null(lib.loc))
```

```
##         paths <- c(path.package(package, TRUE), if (!length(package)) getwd(),
##             paths)
##     paths <- unique(normalizePath(paths[file.exists(paths)]))
##     paths <- paths[dir.exists(file.path(paths, "data"))]
##     dataExts <- tools:::.make_file_exts("data")
##     if (length(names) == 0L) {
##         db <- matrix(character(), nrow = 0L, ncol = 4L)
##         for (path in paths) {
##             entries <- NULL
##             packageName <- if (file_test("-f", file.path(path,
##                 "DESCRIPTION")))
##                 basename(path)
##             else "."
##             if (file_test("-f", INDEX <- file.path(path, "Meta",
##                 "data.rds"))) {
##                 entries <- readRDS(INDEX)
##             }
##             else {
##                 dataDir <- file.path(path, "data")
##                 entries <- tools::list_files_with_type(dataDir,
##                   "data")
##                 if (length(entries)) {
##                   entries <- unique(tools::file_path_sans_ext(basename(entries)))
##                   entries <- cbind(entries, "")
##                 }
##             }
##             if (NROW(entries)) {
##                 if (is.matrix(entries) && ncol(entries) == 2L)
##                   db <- rbind(db, cbind(packageName, dirname(path),
##                     entries))
##                 else warning(gettextf("data index for package %s is invalid and will be ignored",
##                   sQuote(packageName)), domain = NA, call. = FALSE)
##             }
##         }
##         colnames(db) <- c("Package", "LibPath", "Item", "Title")
##         footer <- if (missing(package))
##             paste0("Use ", sQuote(paste("data(package =", ".packages(all.available = TRUE))")),
##                 "\n", "to list the data sets in all *available* packages.")
##         else NULL
##         y <- list(title = "Data sets", header = NULL, results = db,
##             footer = footer)
##         class(y) <- "packageIQR"
##         return(y)
##     }
##     paths <- file.path(paths, "data")
##     for (name in names) {
##         found <- FALSE
##         for (p in paths) {
##             tmp_env <- if (overwrite)
##                 envir
##             else new.env()
##             if (file_test("-f", file.path(p, "Rdata.rds"))) {
##                 rds <- readRDS(file.path(p, "Rdata.rds"))
##                 if (name %in% names(rds)) {
```

```
##                   found <- TRUE
##                   if (verbose)
##                     message(sprintf("name=%s:\t found in Rdata.rds",
##                       name), domain = NA)
##                   thispkg <- sub(".*/([^/]*)/data$", "\\1", p)
##                   thispkg <- sub("_.*$", "", thispkg)
##                   thispkg <- paste0("package:", thispkg)
##                   objs <- rds[[name]]
##                   lazyLoad(file.path(p, "Rdata"), envir = tmp_env,
##                     filter = function(x) x %in% objs)
##                   break
##                 }
##                 else if (verbose)
##                   message(sprintf("name=%s:\t NOT found in names() of Rdata.rds, i.e.,\n\t%s\n",
##                     name, paste(names(rds), collapse = ",")),
##                     domain = NA)
##             }
##             if (file_test("-f", file.path(p, "Rdata.zip"))) {
##                 warning("zipped data found for package ", sQuote(basename(dirname(p))),
##                   ".\nThat is defunct, so please re-install the package.",
##                   domain = NA)
##                 if (file_test("-f", fp <- file.path(p, "filelist")))
##                   files <- file.path(p, scan(fp, what = "", quiet = TRUE))
##                 else {
##                   warning(gettextf("file 'filelist' is missing for directory %s",
##                     sQuote(p)), domain = NA)
##                   next
##                 }
##             }
##             else {
##                 files <- list.files(p, full.names = TRUE)
##             }
##             files <- files[grep(name, files, fixed = TRUE)]
##             if (length(files) > 1L) {
##                 o <- match(fileExt(files), dataExts, nomatch = 100L)
##                 paths0 <- dirname(files)
##                 paths0 <- factor(paths0, levels = unique(paths0))
##                 files <- files[order(paths0, o)]
##             }
##             if (length(files)) {
##                 for (file in files) {
##                   if (verbose)
##                     message("name=", name, ":\t file= ...", .Platform$file.sep,
##                       basename(file), "::\t", appendLF = FALSE,
##                       domain = NA)
##                   ext <- fileExt(file)
##                   if (basename(file) != paste0(name, ".", ext))
##                     found <- FALSE
##                   else {
##                     found <- TRUE
##                     zfile <- file
##                     zipname <- file.path(dirname(file), "Rdata.zip")
##                     if (file.exists(zipname)) {
##                       Rdatadir <- tempfile("Rdata")
```

```
##                       dir.create(Rdatadir, showWarnings = FALSE)
##                       topic <- basename(file)
##                       rc <- .External(C_unzip, zipname, topic,
##                         Rdatadir, FALSE, TRUE, FALSE, FALSE)
##                       if (rc == 0L)
##                         zfile <- file.path(Rdatadir, topic)
##                     }
##                     if (zfile != file)
##                       on.exit(unlink(zfile))
##                     switch(ext, R = , r = {
##                       library("utils")
##                       sys.source(zfile, chdir = TRUE, envir = tmp_env)
##                     }, RData = , rdata = , rda = load(zfile,
##                       envir = tmp_env), TXT = , txt = , tab = ,
##                       tab.gz = , tab.bz2 = , tab.xz = , txt.gz = ,
##                       txt.bz2 = , txt.xz = assign(name, my_read_table(zfile,
##                         header = TRUE, as.is = FALSE), envir = tmp_env),
##                       CSV = , csv = , csv.gz = , csv.bz2 = ,
##                       csv.xz = assign(name, my_read_table(zfile,
##                         header = TRUE, sep = ";", as.is = FALSE),
##                         envir = tmp_env), found <- FALSE)
##                   }
##                   if (found)
##                     break
##                 }
##                 if (verbose)
##                   message(if (!found)
##                     "*NOT* ", "found", domain = NA)
##             }
##             if (found)
##                 break
##         }
##         if (!found) {
##             warning(gettextf("data set %s not found", sQuote(name)),
##                 domain = NA)
##         }
##         else if (!overwrite) {
##             for (o in ls(envir = tmp_env, all.names = TRUE)) {
##                 if (exists(o, envir = envir, inherits = FALSE))
##                   warning(gettextf("an object named %s already exists and will not be overwritten",
##                     sQuote(o)))
##                 else assign(o, get(o, envir = tmp_env, inherits = FALSE),
##                   envir = envir)
##             }
##             rm(tmp_env)
##         }
##     }
##     invisible(names)
## }
## <bytecode: 0x7fc9e8b33d00>
## <environment: namespace:utils>
```

Question 2 (b)

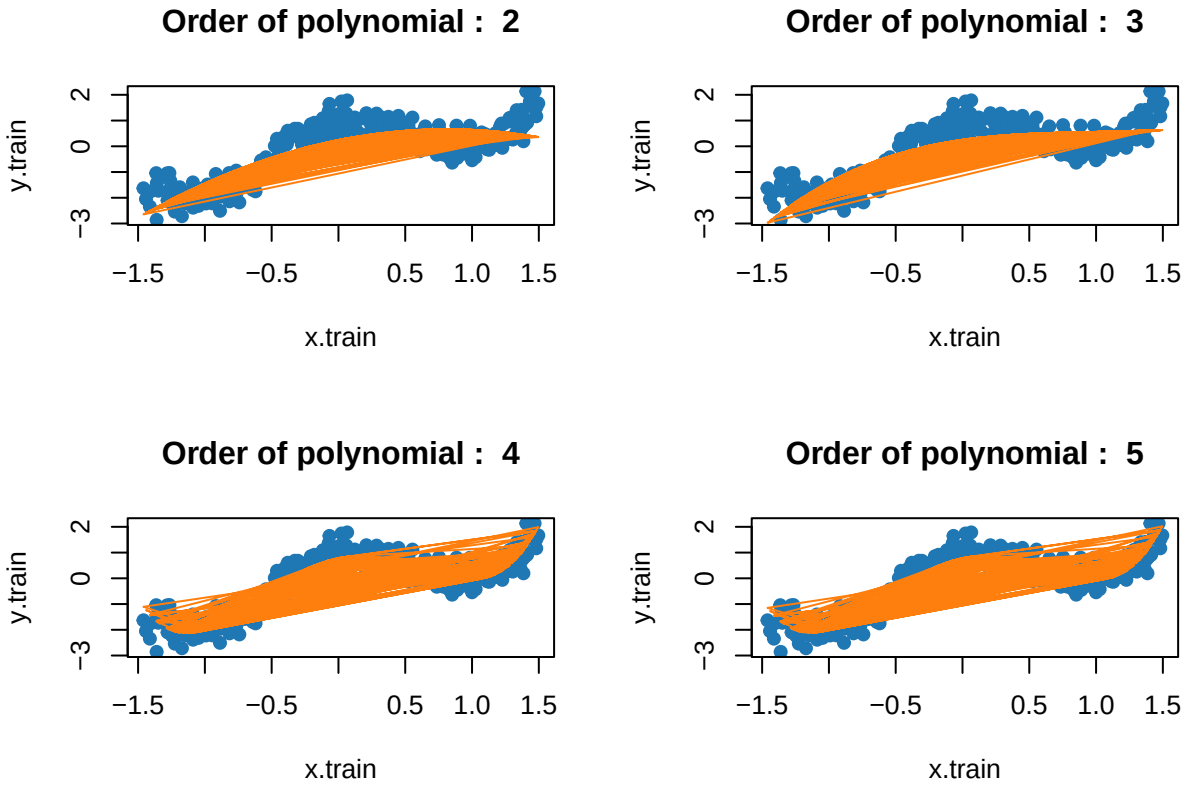
The training error for a degree 2 polynomial is: 0.4106640

The training error for a degree 3 polynomial is: 0.3990116

The training error for a degree 4 polynomial is: 0.1571666

The training error for a degree 5 polynomial is: 0.1570639

```
par(mfrow=c(2,2))
residuals = rep(0, 4) # stores the training error of each polynomial
for (k in 2:5) {
  kth_order_model = lm(y.train ~ poly(x.train, k, raw = TRUE), data=data.frame(x.train, y.train)) # fit
  residuals[k-1] = 1/length(y.train) * sum((y.train - kth_order_model$fitted.values)^2) # finds the tra
  plot(data.frame(x.train, y.train), pch = 19, col = "#1f77b4", main = paste("Order of polynomial : " ,
  lines(x.train, kth_order_model$fitted.values, lty = 1, col = "#ff7f0e", lwd = 1)} # plots the kth ord
```




```
print(residuals)
```

```
## [1] 0.4106640 0.3990116 0.1571666 0.1570639
```

Question 1 (c)

```
cross_validate = function(p, K, x.train, y.train) {
  residuals = rep(0, K)
  set_size = ceiling(length(y.train) / K) # find the ideal size of each fold
  start = 1
  end = set_size
  i = 1
  while (end <= 300) {
    # go through data
```

```
    xtrains = x.train[-(start:end)] # the data points on which we train the model
    ytrains = y.train[-(start:end)]
    xleftout = x.train[start:end] # the data points on which we test the model
    yleftout = y.train[start:end]
    pth_order_model = lm(ytrains ~ poly(xtrains, p, raw = TRUE)) # train the model
    xleftout_df <- data.frame(xtrains = xleftout) # convert testing data point observations to suitable
    ypredicts = predict(pth_order_model, newdata = xleftout_df) # predict testing data points from the
    residuals[i] = 1/length(yleftout) * sum((yleftout - ypredicts)^2) # stores residuals for testing da
    i = i + 1
    start = end + 1 # update start and end to leave out the next block of size set_size
    end = set_size + end
  }
  if (300 %% K != 0) { # do not have equal sized blocks and need to account for last few terms
    end = 300 # set end = 300 in this case and repeat process
    xtrains = x.train[-(start:end)]
    ytrains = y.train[-(start:end)]
    xleftout = x.train[start:end]
    yleftout = y.train[start:end]
    pth_order_model = lm(ytrains ~ poly(xtrains, p, raw = TRUE))
    xleftout_df <- data.frame(xtrains = xleftout)
    ypredicts = predict(pth_order_model, newdata = xleftout_df)
    residuals[i] = 1/length(yleftout) * sum((yleftout - ypredicts)^2)
  }
  return(mean(residuals)) # error estimate for K cross validation on polynomial of size p
}
```

```
K = c(5,6,7,8) # all the values of K
P = c(2,3,4,5) # degrees of all the polynomials
table_errors = matrix(0, nrow = 4, ncol = 4) # table of error estimates

for (p in P) { # for each polynomial
  for (k in K) { # for each value of K
    cross_validation_error = cross_validate(p, k, x.train, y.train) # gets the cross validation error
    table_errors[which(K == k), which(P == p)] = cross_validation_error
  } # stores the error in the table where kth row and pth column represents degree p polynomial with k
}
table_df <- as.data.frame(table_errors) # turns table to dataframe
colnames(table_df) <- paste("Degree", P, sep = "_") # labels column
rownames(table_df) <- paste("K", K, sep = "_") # labels row
print(table_df)
```

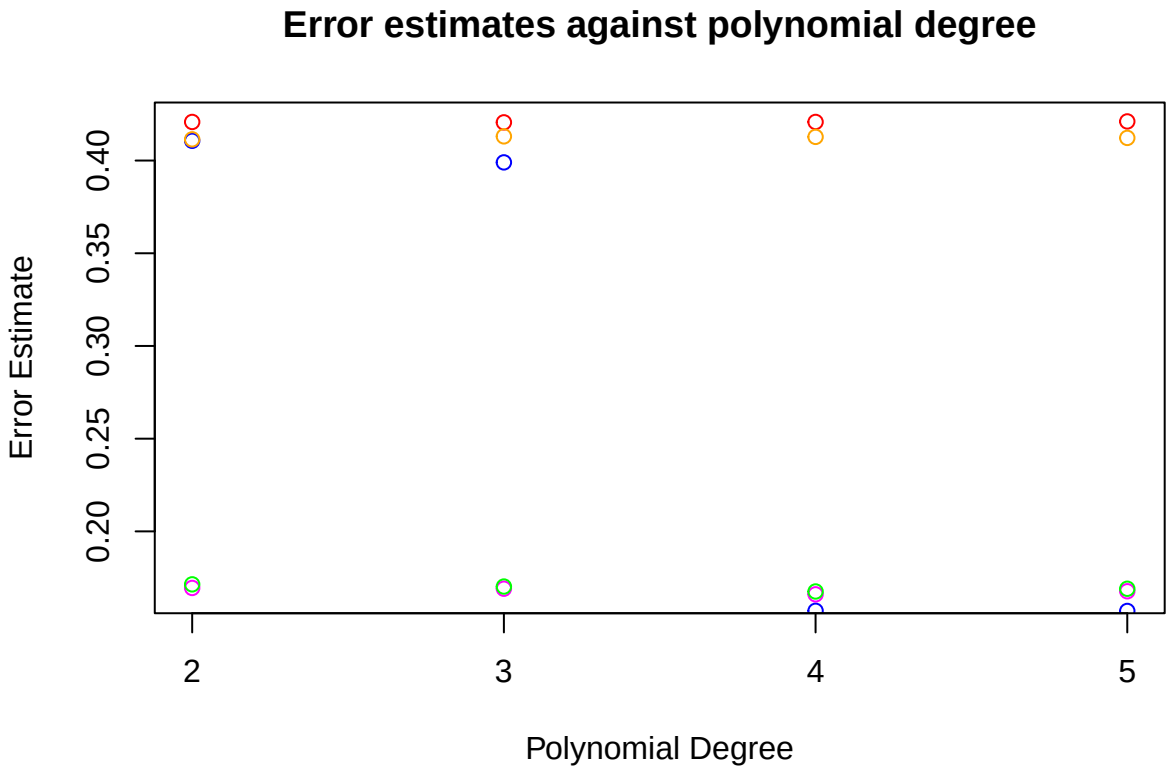
```
##      Degree_2  Degree_3  Degree_4  Degree_5
## K_5 0.4208145 0.4114341 0.1694797 0.1714232
## K_6 0.4205992 0.4130458 0.1690821 0.1702380
## K_7 0.4208147 0.4127805 0.1660482 0.1675805
## K_8 0.4211008 0.4122242 0.1676935 0.1690563
```

Question 2 (d)

Blue dots = training error. Red dots = polynomial degree 1 Orange dots = polynomial degree 2 Magenta dots = polynomial degree 3 Green dots = polynomial degree 4

```
training_error = residuals
```

```r
df = data.frame(training_error, table_df[, 1], table_df[, 2], table_df[, 3], table_df[,4])

plot(seq(2,5,1), df[, 1], xlab="Polynomial Degree", ylab="Error Estimate", main="Error estimates agains

axis(1, at=seq(2,5,1), las=1)

points(seq(2,5,1), df[, 2], col="red")

points(seq(2,5,1), df[, 3], col="orange")

points(seq(2,5,1), df[, 4], col="magenta")

points(seq(2,5,1), df[, 5], col="green")
```




```r
# legend(4.6, 0.42, legend=c("training error", "d=2", "d=3", "d=4", "d=5"), fill=c("blue", "red", "oran
```

Question 2 (e)

We have 4 different $\hat{f}$, each of degree 2, 3, 4 and 5.

Since our values of K = 5,6,7 and 8 are small, we can apply the approximation:

$Var(Err(CV(\hat{f}))) \approx \frac{1}{K} Var(CV_1((\hat{f})))$

Now $CV_1((\hat{f})$ is the error of leaving one block out (but this is a real-value so not sure how we compute the variance).

```r
error_cross_validate = function(p, K, x.train, y.train) {
  residuals = rep(0, K)
  set_size = ceiling(length(y.train) / K) # find the ideal size of each fold
  start = 1
```

```
  end = set_size
  i = 1
  xtrains = x.train[-(start:end)] # the data points on which we train the model
  ytrains = y.train[-(start:end)]
  xleftout = x.train[start:end] # the data points on which we test the model
  yleftout = y.train[start:end]
  pth_order_model = lm(ytrains ~ poly(xtrains, p, raw = TRUE)) # train the model
  xleftout_df <- data.frame(xtrains = xleftout) # convert testing data point observations to suitable f
  ypredicts = predict(pth_order_model, newdata = xleftout_df) # predict testing data points from the mo
  residuals = 1/length(yleftout) * sum((yleftout - ypredicts)^2) # stores residuals for testing data po
}
```

```
ps = c(2,3,4,5)
ks = c(5,6,7,8)
table_errors = matrix(0, nrow = 4, ncol = 4) # table of variance estimates for each polynomial


for (p in ps) {
  for (k in ks) {
    errorvar = 1/k * error_cross_validate(p, k, x.train, y.train) # variance estimate where error_cross
    table_errors[which(K == k), which(P == p)] = errorvar
  }
}
table_df <- as.data.frame(table_errors) # turns table to dataframe
colnames(table_df) <- paste("Degree", P, sep = "_") # labels column
rownames(table_df) <- paste("K", K, sep = "_") # labels row
print(table_df)
```

```
##       Degree_2   Degree_3   Degree_4   Degree_5
## K_5 0.07829130 0.07317643 0.04347754 0.04342417
## K_6 0.07125846 0.06678533 0.03629436 0.03626572
## K_7 0.06259846 0.05751932 0.02843679 0.02864075
## K_8 0.05427669 0.04940459 0.02438278 0.02450062
```

Question 2 (f)

We observe a degree 4 polynomial to have the smallest errors for K=5,6,7 and 8 cross fold validations. Thus, the data best fits to a degree 4 polynomial and hence the variance estimators for a degree 4 polynomial are most likely to be accurate.