HW3

Omair Shafi Ahmed

11/13/2017

Homework 3

Part A

```r
library('ggplot2')
library('modelr')
library('tidyverse')
```

```
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr

## Warning: package 'purrr' was built under R version 3.4.2

## Warning: package 'dplyr' was built under R version 3.4.2

## Conflicts with tidy packages ----------------------------------------------

## filter(): dplyr, stats
## lag():    dplyr, stats
```

Problem 1

Building the Model and Residuals

Fit a model that predicts highway miles per gallon using no more than 3 predictor variables. Use plots to justify your choice of predictor variables. Print the values of the fitted model parameters.

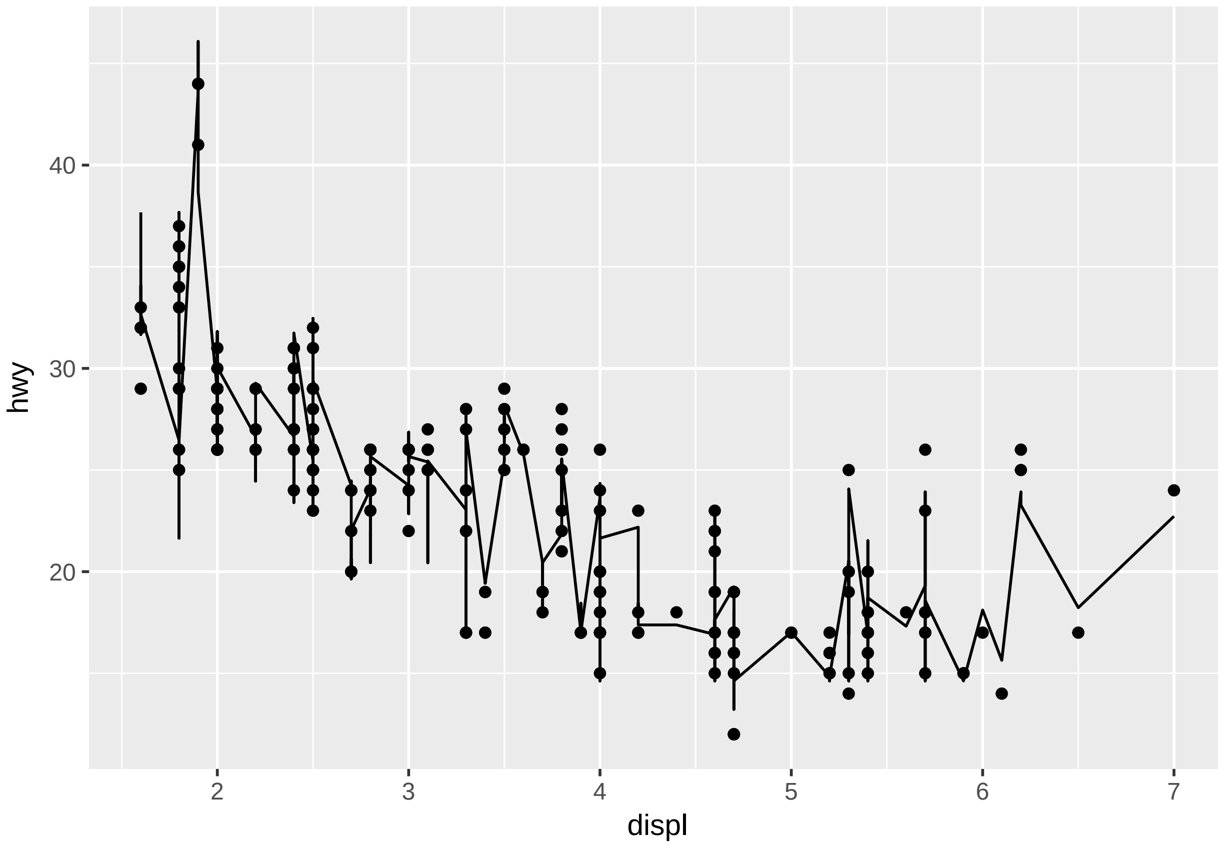
We leverage the variable city mileage as a predictor for highway mileage as, logically, the two should be well correlated. Besides that, the transmission and the drive type dictate the highway mile

```r
model <- lm(hwy ~ drv + cty + trans, data=mpg)

coef(model)
```

```
##     (Intercept)            drvf            drvr             cty
##      1.36069048      2.41276110      2.28852994      1.20132742
##   transauto(l3)   transauto(l4)   transauto(l5)   transauto(l6)
##     -2.00132742      0.04785533      1.06389700      0.65407109
##   transauto(s4)   transauto(s5)   transauto(s6) transmanual(m5)
##      1.07694396      1.54112682      1.60433191      0.26589633
## transmanual(m6)
##      1.05603001
```

```r
mpg %>% add_predictions(model) %>% ggplot(aes(x=displ)) + geom_point(aes(y=hwy)) + geom_line(aes(y=pred)
```



Root Mean Square Error

```
sqrt(mean(resid(model)^2))
```

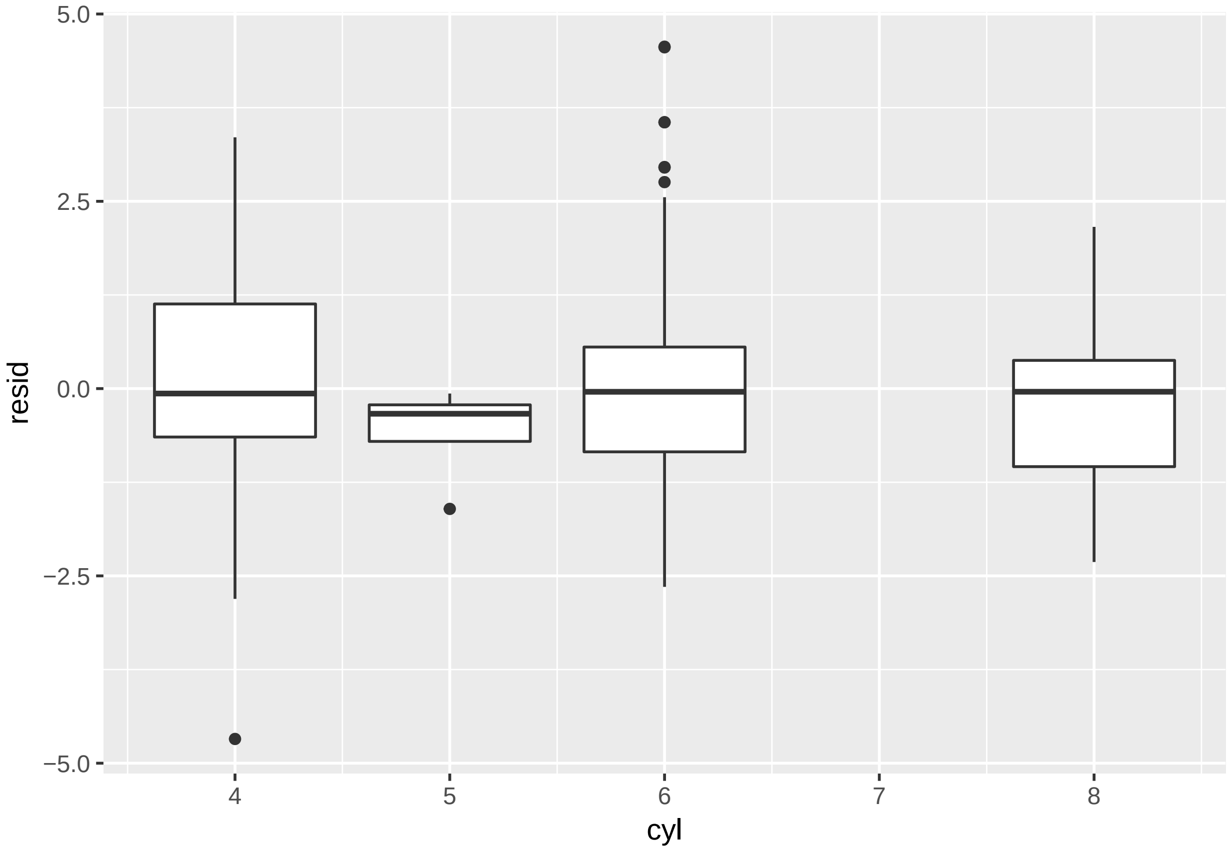
```
## [1] 1.370404
```

Problem 2

Plotting Residual 1

The residual plot appears to have it's variance randomly distributed, with the mean hovering around 0, indicating no pattern in the residuals.

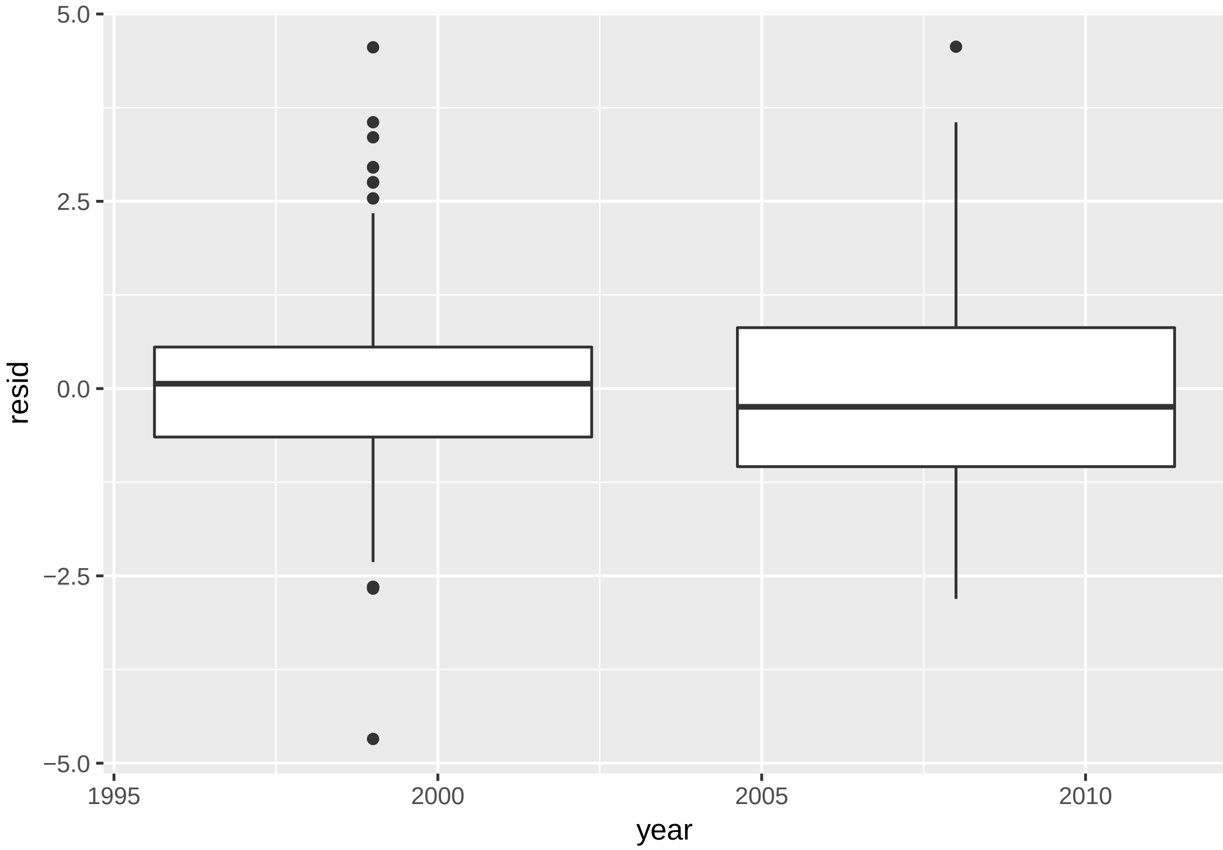
```
mpg %>% add_residuals(model) %>% ggplot(aes(x=cyl, y=resid, group=cyl)) + geom_boxplot()
```



Plotting Residuals 2

The residual plot appears to randomly distributed, with the mean in the proximity of 0, indicating no pattern in the residuals.

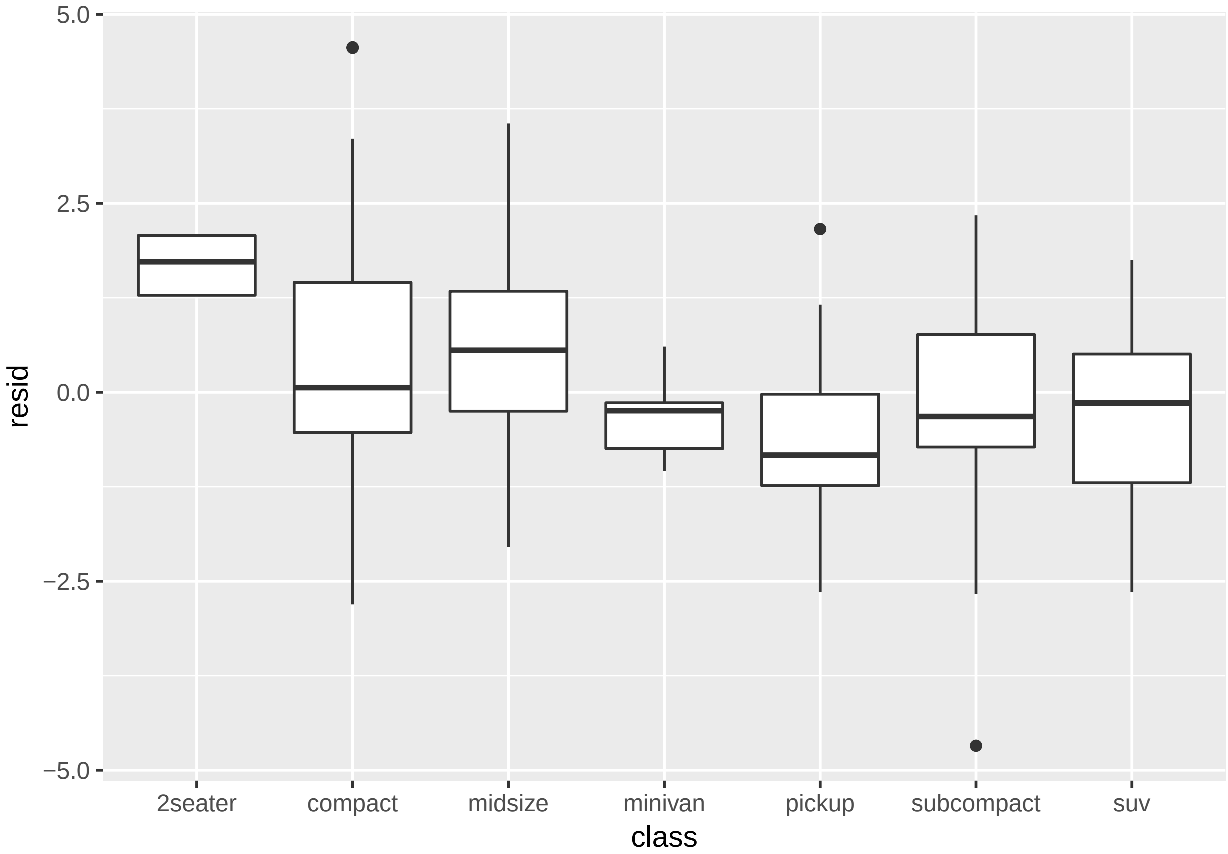
```
mpg %>% add_residuals(model) %>% ggplot(aes(x=year, y=resid, group=year)) + geom_boxplot()
```



Plotting Residuals 3

The residual plot appears to have a patern associated with what looks like the frontal surface area of the vehicle. 2 seaters being lower, have a smaller surface area at it's front, making it more aerodynamically efficient and therefore have a strong positive residual. Pickup trucks, on the other hand, have a larger frontal area making it aerodynamically inefficient, leading to negative residuals more that is not captured by the model.

```
#mpg %>% add_residuals(model) %>% ggplot(aes(x=class, y=resid)) + geom_point(alpha = 1/10)
mpg %>% add_residuals(model) %>% ggplot(aes(x=class, y=resid)) + geom_boxplot()
```



Problem 3

The new model, as dictated by the previous exercise

```
model1 <- lm(hwy ~ class + cty + trans, data=mpg)

coef(model1)
```

```
##     (Intercept)    classcompact    classmidsize    classminivan
##       7.0259749      -1.5271156      -1.0508126      -2.8173989
##     classpickup classsubcompact        classsuv             cty
##      -5.2495842      -1.9980241      -4.6771897       1.1007679
##   transauto(l3)   transauto(l4)   transauto(l5)   transauto(l6)
##      -0.9698445       0.6458718       1.2227407       1.9874560
##   transauto(s4)   transauto(s5)   transauto(s6) transmanual(m5)
##      -0.5386286       1.8045210       0.9088100       0.5445938
## transmanual(m6)
##       0.8521040
```

Part B

Problem 4

Write a function that performs cross-validation for a linear model (fit using lm) and returns the average root-mean-square-error across all folds. The function should take as arguments (1) a formula used to fit the model, (2) a dataset, and (3) the number of folds to use for cross-validation. The function should partition the dataset, fit a model on each training partition, make predictions on each test partition, and return the average root-mean-square-error.

```
library(purrr)

function_to_cross_validate <- function(formula, dataset, number_of_folds){

  folded_data     <- crossv_kfold(dataset, number_of_folds)

  formula_applied <- purrr::map(folded_data$train, ~lm(formula, data=.))

  rmse_test       <- purrr::map2_dbl(formula_applied, folded_data$test, ~rmse(.x, .y))

  return(mean(rmse_test))

}

function_to_cross_validate(hwy ~ drv + cty + trans, mpg, 10)
```

```
## [1] 1.43858
```

Problem 5

Use your function from Problem 4 to compare the models you used from Part A. Report the cross-validated root-mean-square-error for the models from Problems 1 and 3. Which model was more predictive?

```
function_to_cross_validate(hwy ~ class + cty + trans, mpg, 10)
```

```
## [1] 1.244545
```

The newer model has a lower RMSE, making it a better fit than the first model.

Part C

```
insert_ref_groups <- function(x) {
  ref_groups <- xml_root(x) %>% xml_child("d1:referenceableParamGroupList") %>% xml_children()
  ref <- xml_child(x, "d1:referenceableParamGroupRef")
  name <- xml_attr(ref, "ref")
  ref_groups_exist <- xml_attr(ref_groups, "id") %in% name
  if ( any(ref_groups_exist) )
  group <- ref_groups[[which(ref_groups_exist)]]
  for ( g in xml_children(group) )
  xml_add_child(x, g)
  xml_remove(ref)
  x
}


xml_find_by_attribute <- function(x, attr, value) { match <- xml_attr(x, attr) == value
  if ( isTRUE(any(match)) ) {
    x[[which(match)]] } else {
```

```r
    NULL
    }
  }

get_spectrum_data <- function(x, i) {
  spectrum <- x %>% xml_child("d1:run") %>% xml_child("d1:spectrumList") %>%
  xml_child(i)
  spectrum <- insert_ref_groups(spectrum)
  scan <- spectrum %>% xml_child("d1:scanList") %>% xml_child("d1:scan")
  scan <- insert_ref_groups(scan)
  data <- spectrum %>% xml_child("d1:binaryDataArrayList") %>%
  xml_children()
  for ( d in data )
  insert_ref_groups(d)
  data <- lapply(data, xml_children)
  for ( i in seq_along(data) ) {
    if ( !is.null(xml_find_by_attribute(data[[i]], "name", "m/z array")) )
      names(data)[i] <- "mz"
    if ( !is.null(xml_find_by_attribute(data[[i]], "name", "intensity array")) )
      names(data)[i] <- "intensity"
    }
  data$coord <- xml_children(scan)
  data[c("mz", "intensity", "coord")]
}

get_spectra_n <- function(x) {
    x %>%
    xml_child("d1:run") %>% xml_child("d1:spectrumList") %>% xml_attr("count") %>% as.numeric()
}

get_spectra <- function(x) {
    n <- get_spectra_n(x)
    lapply(1:n, function(i) get_spectrum_data(x, i))
}
```

Problem 6

```r
library(xml2)
imzml <- read_xml("/Users/omairs/Documents/Masters/DS 5110/HW3/data/example_files/Example_Continuous.im

parser <- function(x)
{
  spectra_n    <- get_spectra_n(x)

  spectra_info <- get_spectra(x)

  intensity_length <- spectra_info %>%

                      map_dbl(~ xml_find_by_attribute(.$intensity,
                                                      "name", "external array length") %>%

                      xml_attr("value") %>%
```

```
                    as.numeric())
intensity_offset <- spectra_info %>%
                      map_dbl(~ xml_find_by_attribute(.$intensity, "name", "external offset") %>%
                      xml_attr("value") %>%
                        as.numeric())
mz_length <- spectra_info[[1]]$mz %>%
                  xml_find_by_attribute("name", "external array length") %>%
                  xml_attr("value") %>%
                  as.numeric()
mz_offset <- spectra_info[[1]]$mz %>%
              xml_find_by_attribute("name", "external offset") %>%
              xml_attr("value") %>%
              as.numeric()
x_axis <- spectra_info %>%
            map_dbl(~ xml_find_by_attribute(.$coord, "name", "position x") %>%
            xml_attr("value") %>%
            as.numeric())
y_axis <- spectra_info %>%
            map_dbl(~ xml_find_by_attribute(.$coord, "name", "position y") %>%
            xml_attr("value") %>%
            as.numeric())

list(intensity_length = intensity_length,
     intensity_offset = intensity_offset,
     mz_length = mz_length,
     mz_offset = mz_offset,
     x_axis = x_axis,
```

```
       y_axis = y_axis)
}

imz <- parser(imzml)
```

Problem 7

Using the information you parsed in Problem 6 and the base R function readBin, write a function that reads the m/z array and all of the intensity arrays in the "Example_Continuous.ibd" binary file.

```
ibd_file <- "/Users/omairs/Documents/Masters/DS 5110/HW3/data/example_files/Example_Continuous.ibd"

reader <- function(file_name, imz)
  {
      intensity <- map2(imz$intensity_offset, imz$intensity_length,
                        function(offset, length)
                        {
                            f <- file(file_name, "rb")
                            seek(f, offset)
                            iout <- readBin(f, "double", n=length, size=4)
                            close(f)
                            iout
                        }
                      )

      f  <- file(file_name, "rb")

      mz_array <- readBin(f, "double", n=imz[["mz_length"]], size=4)

      close(f)

      list(intensity = intensity, mz_array = mz_array)
  }

output <- reader(ibd_file, imz)
```

Problem 8

Write a constructor for a class that stores the coordinates, m/z array, and intensity arrays that you parsed in Problems 6 and 7. You may use either an S3 or an S4 class.

```
msi_processor <- function (m_z, intensity, x_axis, y_axis) structure(list(m_z=m_z, intensity=simplify2a

msi_output <-  msi_processor(output$m_z,
                     output$intensity,
                     imz$x_axis,
                     imz$y_axis)

#msi_output
```

Problem 9

Write methods to access the coordinates, m/z array, and intensity arrays. Write another method to plot an image of the data for a particular m/z value.

```
access <- function(msi_processor) UseMethod("access", msi_processor)
plot <- function(msi_processor,m_z) UseMethod("plot", msi_processor)

access.msi_processor <- function(obj) {

    coordinates = msi_output$coord

    mz_arr  = msi_output$m_z

    intensity_arr =  msi_output$intensity

    return(list(coordinates, mz_arr, intensity_arr))
}

plot.msi <- function(x, m_z) {

    idx <- which.min(abs(m_z - x$m_z))

    idf <- x$coord

    idf$intensity <- x$intensity[idx,]

    ggplot(idf) + geom_tile(aes(x=x, y=y, fill=intensity)) + scale_y_reverse()
}

#plot(msi_output, m_z = 173.4)
```