Homework 4
STAT 4410/8416 Section 001/002
Fall 2016
Due: November 8, 2016 by midnight

1. **Working with date-time data:** The object myDate contains the date and time when this question was provided to you. Based on this object answer the following questions.

```
library(lubridate)
myDate <- ymd_hms("2015-10-28 19:50:21", tz = "America/Chicago")
myDate

## [1] "2015-10-28 19:50:21 CDT"
```

   (a) Write your codes so that it displays the week day of myDate.
   Answer:

```
weekdays(myDate)

## [1] "Wednesday"
```

   (b) What weekday is it after exactly 100 years from myDate? Show your codes and the answer.
   Answer:

```
myDate.100 <- myDate + years(100)
weekdays(myDate.100)

## [1] "Monday"
```

   (c) Add one month with myDate and display the resulting date time. Explain why the time zone has changed even though you did not ask for time zone change.
   Answer:

```
myDate.1.month <- myDate + months(1)
myDate.1.month

## [1] "2015-11-28 19:50:21 CST"
```

   The time zone changed from Central Daylight Time (CDT) to Central Standard Time (CST). This is because Daylight Saving Time happens in November in most states in the United States.

   (d) Suppose this homework is due on November 8, 2015 by 11.59PM. Compute and display how many minutes you got to complete this homework?

   Answer:

```
due.date <- ymd_hms("2016-11-08 23:59:59")
now.date <- ymd_hms("2016-11-02 21:16:50")
date.diff <- due.date - now.date
minutes.left <- (as.numeric(as.duration(date.diff)) / 60)
```

   I've got 8803.15 minutes left! Better hurry!

2. **Boston hubway data:** This question will explore Boston hubway data. Please carefully answer each question below including your codes and results.

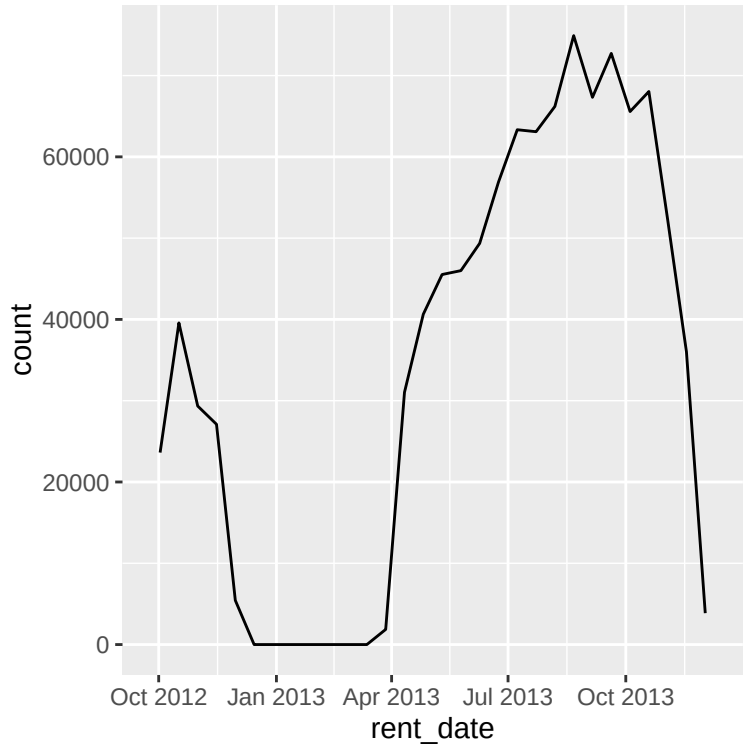
(a) Obtain the compressed data, bicycle-rents.csv.zip, from blackboard and display few data rows.
Answer:

```
bikes <- read.csv('bicycle-rents.csv', stringsAsFactors = F)
bikes$rent_date_time <- ymd_hms(bikes$rent_date, tz = "America/New_York")
bikes$rent_date <- as.Date(ymd_hms(bikes$rent_date, tz = "America/New_York"))
head(bikes)
```

```
##       id  rent_date      rent_date_time
## 1 708191 2012-11-01 2012-10-31 23:57:00
## 2 708190 2012-11-01 2012-10-31 23:54:00
## 3 708189 2012-11-01 2012-10-31 23:52:00
## 4 708187 2012-11-01 2012-10-31 23:52:00
## 5 708186 2012-11-01 2012-10-31 23:50:00
## 6 708185 2012-11-01 2012-10-31 23:39:00
```

(b) For each day, count the number of bikes rented for that date and show the data in a time series plot.
Answer:

```
ggplot(bikes, aes(x=rent_date)) +
  geom_line(stat = "bin")
```



(c) Based on the rent date column, create two new columns weekDay and hourDay which represent week day name and hour of the day respectively. Store the data in myDat and display few records of the data. Hint: For weekday use function wday().
Answer:

```
myDat <- as.data.frame(cbind(wday(bikes$rent_date), hour(bikes$rent_date_time)))
colnames(myDat) <- c('weekDay', 'hourDay')
head(myDat)
```

```
##   weekDay hourDay
## 1       5      23
## 2       5      23
## 3       5      23
## 4       5      23
## 5       5      23
## 6       5      23
```

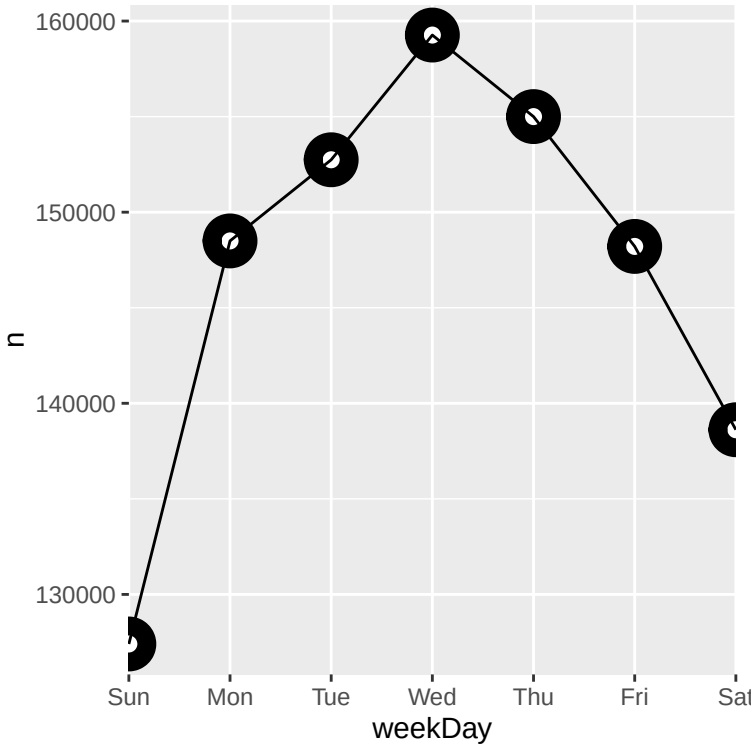
(d) Summarize myDat by weekDay based on the number of rents for each weekDay and store the data in weekDat. Display some data.

```
weekDat <- myDat %>%
  group_by(weekDay) %>%
  summarise(n=n())
weekDat
```

```
## # A tibble: 7 x 2
##   weekDay      n
##     <dbl>  <int>
## 1       1 127406
## 2       2 148496
## 3       3 152742
## 4       4 159268
## 5       5 155001
## 6       6 148209
## 7       7 138617
```

(e) Create a suitable plot of the data you stored in weekDay so that it displays number of bike rents for each week day.

```
ggplot(weekDat, aes(x = weekDay, y = n)) +
  geom_point(shape = 21, colour = "black", fill = "white", size = 3, stroke = 5) +
  geom_line() +
  scale_x_discrete(limit = c("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"))
```

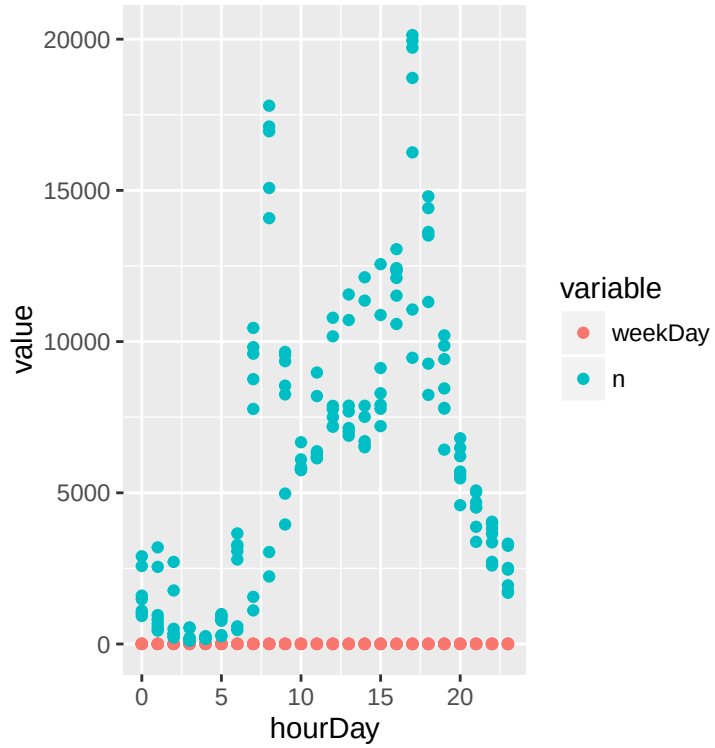


(f) Now we want to investigate what happens in each day. Summarize myDat again but this time by weekDay and hourDay and obtain the number of rents. Store the data in hourDat and Display some data.

```
hourDat <- myDat %>%
  group_by(weekDay, hourDay) %>%
  summarise(n=n())
hourDat <- as.data.frame(hourDat)
test_data_long <- melt(hourDat, id="hourDay")  # convert to long format
```

(g) The dataframe hourDat is now ready for plotting. Generate line plots showing number of bike rents vs hour of the day and colored by weekDay.

```
ggplot(data = test_data_long,
       aes(x = hourDay, y = value, colour = variable)) +
       geom_point()
```



Note: This isn't exactly what I wanted, but I couldn't get the line charts to work.

3. **Creating HTML Page:** In this problem we would like to create a basic HTML page. Please follow the each step below and finally submit your HTML file on the blackboard. Please note that you don't need to answer these questions here in the .Rnw file.

   (a) Open a notepad or any plain text editor. Write down some basic HTML codes as shown in online (year 2014) Lecture 15, slide 6 and modify according to the following questions. Save the file as hw4.html and upload on the blackboard as a separate file.

   (b) Write "What is data science?" in the first header tag, <h1></h1>

   (c) Hw1 solution contains the answer of what is data science. The answer has three paragraphs. Write the three paragraphs of text about data science in three different paragraph tags . You can copy the text from hw1 solution.

   (d) Write"What we learnt from hw1" in second heading under tag <h2></h2>

   (e) Copy all the points we learnt in hw1 solution. List all the points under ordered list tag . Notice that each item of the list should be inside list item tag .

   (f) Now we want to make the text beautiful. For this we would write some CSS codes in between <head></head> tag under <style></style>. For this please refer to online (year 2014) lecture 15 slide 8. First change the fonts of the body tag to Helvetica Neue.

   (g) For the paragraph that contains the definition of data science, give an attribute id='dfn' and in CSS change the color of 'dfn' to white, background-color to olive and font to be bold.

   (h) For other paragraphs, give an attribute class='cls' and in CSS change the color of 'cls' to green.

   (i) Write CSS so that color of h1,h2 becomes orange.

4. **Exploring XML data:** In this problem we will read the xml data. For this we will obtain a xml data called olive oils from the link http://www.ggobi.org/book/data/olive.xml. Please follow the directions in each step and provide your codes and output.

(a) Parse the xml data from the above link and store in a object called `olive`. Obtain the root of the xml file and display its name.

Answer:

```
xmlUrl <- "http://www.ggobi.org/book/data/olive.xml"
olive <- xmlParse(xmlUrl)
root <- xmlRoot(olive)
root.node.name <- xmlName(root)
```

(b) Examine the actual file by going to the link above and identify the path of categorical variables in the xml tree. Use that path to obtain the categorical variable names. Please keep the names, not nick names and store them in `cvNames`. Display `cvNames`.

Answer:

```
attrs <- xpathApply(olive, '//ggobidata/data/variables/categoricalvariable', xmlAttrs)
cvNames <- as.character(unlist(getNodeSet(root, "//ggobidata/data/variables/categoricalvariable
cvNames
```

```
## [1] "region" "area"
```

(c) Now examine the file by going to the link and identify the path of real variables in the xml tree. Use that path to obtain the real variable names. Please keep the names, not nick names and store them in `rvNames`. Display `rvNames`.

Answer:

```
attrs <- xpathApply(olive, '//ggobidata/data/variables/realvariable', xmlAttrs)
rvNames <- as.character(unlist(getNodeSet(root, "//ggobidata/data/variables/realvariable/@name'
rvNames
```

```
## [1] "palmitic"    "palmitoleic" "stearic"     "oleic"       "linoleic"
## [6] "linolenic"   "arachidic"   "eicosenoic"
```

(d) Notice the path for the data in xml file. Use that path to obtain the data and store the data in a data frame called `oliveDat`. Change the column names as you have obtained the column names. Display some data.

Answer:

```
vals <- xpathApply(olive, '//ggobidata/data/records/record', xmlValue)
datValue <- strsplit(gsub('\\n', '', vals), split=" ")
# First row was corrupt so we'll fix it
firstRow <- datValue[1]
datValue <- datValue[2:length(datValue)]
oliveDat <- do.call(rbind.data.frame, datValue)
names(oliveDat) <- c(cvNames, rvNames)
firstRowFixed <- c(unlist(firstRow)[1:7], c("36", "60", "29"))
oliveDat <- rbind(firstRowFixed, oliveDat)
```

```
## Warning in `[<-.factor`(`*tmp*`, ri, value = "7823"):  invalid factor level, NA
generated
```

```
head(oliveDat)
```

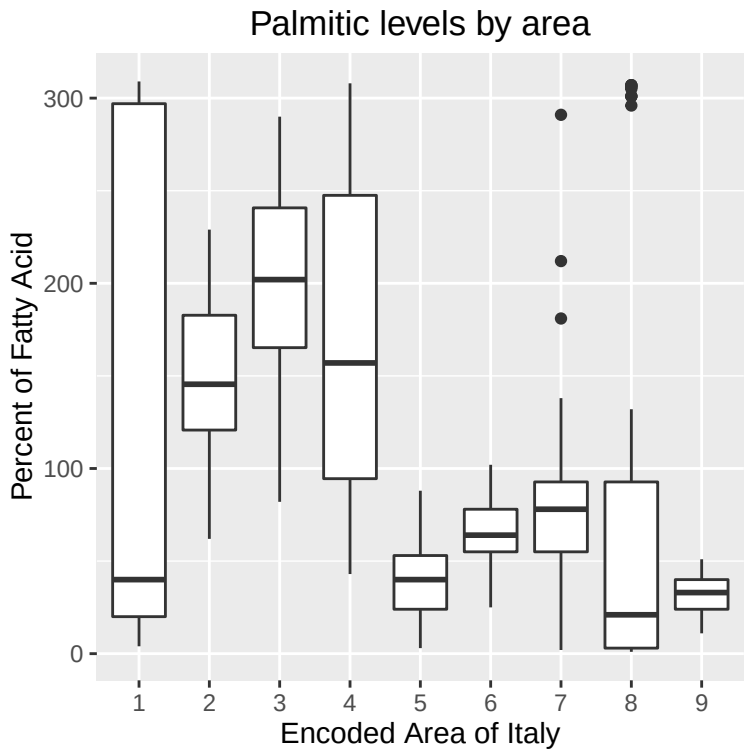
```
##   region area palmitic palmitoleic stearic oleic linoleic linolenic
## 1      1    1     1075          75     226  <NA>      672        36
## 2      1    1     1088          73     224  7709      781        31
## 3      1    1      911          54     246  8113      549        31
## 4      1    1      966          57     240  7952      619        50
```

```
## 5      1    1     1051          67     259  7771      672        50
## 6      1    1      911          49     268  7924      678        51
##   arachidic eicosenoic
## 1        60         29
## 2        61         29
## 3        63         29
## 4        78         35
## 5        80         46
## 6        70         44
```

(e) Generate a plot of your choice to display any feature of `oliveDat` data. Notice that the column names are different fatty acids. The values are % of fatty acids found in the Italian olive oils coming from different regions and areas.

Answer:

```
ggplot(oliveDat, aes(factor(area), as.numeric(palmitic))) +
  geom_boxplot() +
  labs(title="Palmitic levels by area",
       x = "Encoded Area of Italy",
       y = "Percent of Fatty Acid")
```



(f) Explain what these two lines of codes are doing.

```
r <- xmlRoot(olive)
xmlSApply(r[[1]][[2]], xmlGetAttr, "name")
```

Answer:

The first line gets the root element of the XML (which contains all child elements). The next line applies the `xmlGetAttr` function, which gets the xml attribute specified as an argument (which is "name"), to the sub-elements of `r` at [[1]][[2]]. The first offset, [[1]] takes us down to the `<data>`

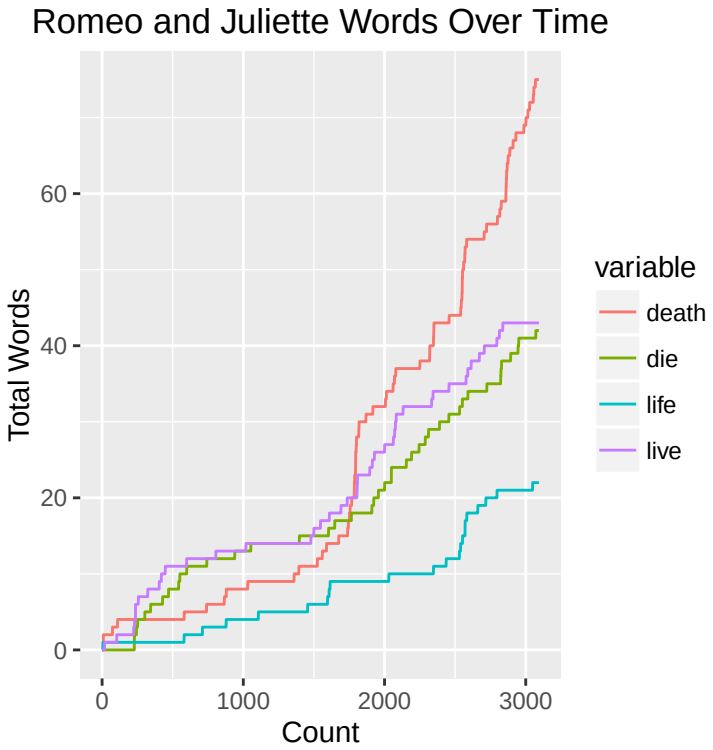
element. The second offset, takes us a level further down, but instead of taking the first element, which is `<description>`, it goes to the next one which is `<variables>`. From there, it applies the `xmlGetAttr` function looking for all the `name`s.

5. **Bonus for undergraduate (3 points) mandatory for graduate students:** The following link contains the complete texts of Romeo and Juliet written by Shakespeare. Read the complete text and generate a plot similar to Romeo and Juliet case study in online(year 2014) lecture 13 (last plot).

   http://shakespeare.mit.edu/romeo_juliet/full.html

   Answer:

```
play <- readLines("http://shakespeare.mit.edu/romeo_juliet/full.html")
speech <- play[str_detect(play, "<A NAME=[0-9.]+")]
speech.text <- trimws(tolower(gsub("</A>(<br>)?", "", gsub("<A NAME=[0-9.]+>", "", speech))))
speech.text.clean <- gsub('[.\'",;:?!]', "", speech.text)

wc <- data.frame(cbind(c(0), c(0), c(0), c(0), c(0)))
colnames(wc) <- c('total.wc', 'word1.wc', 'word2.wc')
total.wc <- 0
word1.wc <- 0
word2.wc <- 0
word3.wc <- 0
word4.wc <- 0

word1 <- 'death'
word2 <- 'die'
word3 <- 'life'
word4 <- 'live'
colnames(wc) <- c('total.wc', word1, word2, word3, word4)
for (i in 1:length(speech.text.clean)) {
  total.wc <- total.wc + length(speech.text.clean[i])

  word1.wc <- word1.wc + length(str_match_all(speech.text.clean[i], word1)[[1]])
  word2.wc <- word2.wc + length(str_match_all(speech.text.clean[i], word2)[[1]])
  word3.wc <- word3.wc + length(str_match_all(speech.text.clean[i], word3)[[1]])
  word4.wc <- word4.wc + length(str_match_all(speech.text.clean[i], word4)[[1]])

  wc <- rbind(wc, c(total.wc, word1.wc, word2.wc, word3.wc, word4.wc))
}

melt.wc <- melt(wc, id.vars = "total.wc")

ggplot(melt.wc, aes(x = total.wc, y=value, color=variable )) +
  geom_line() +
  labs(title="Romeo and Juliette Words Over Time", x = "Count", y = "Total Words")
```



6. **Bonus (2 points) question for all** : GSS data challenge information can be found in the following link. Download the data sets and explore. Provide some plots and numerical summary that creates some interest about this data.

   http://community.amstat.org/governmentstatisticssection/new-item2/new-item

   Answer: I really wish I had time for this!!! :(