HW01 - Solutions

Stat 133, Spring 2018, Prof. Sanchez

Self grade due date: Tue Feb-27 (before midnight)

General Self-Grading Instructions

- Please read this document (answer key).
- You will have to enter your score and comments in the *Assignments Comments* section of the correspoind assignment HW01.
- Enter your own scores and comments for (every part) of every problem in the homework on a simple coarse scale:
  - **0** = Didn't attempt or very very wrong,
  - **2** = Got started and made some progress, but went off in the wrong direction or with no clear direction,
  - **5** = Right direction and got half-way there,
  - **8** = Mostly right but a minor thing missing or wrong,
  - **10** = 100% correct.
- Also, enter your total score (out of an overall score of 110 points).
- If there is a cascading error, use a single deduction (don't double or triple or multiple penalize).
- Note: You must justify every self-grade score with a comment. If you are really confused about how to grade a particular problem, you should post on Piazza. This is not supposed to be a stressful process.
- Your self-grades will be due four days after the homework deadline at 11:59 PM sharp (i.e Tue Feb-06).
- We will accept late self-grades up to a week after the original homework deadline for half credit on the associated homework assignment.
- If you don't enter a proper grade by this deadline, you are giving yourself a zero on that assignment.
- Merely doing the homework is not enough, you must do the homework; turn it in on time; read the solutions; do the self-grade; and turn it in on time. Unless all of these steps are done, you will get a zero for that assignment.

File Structure (10 pts)

After completing the assignment, you should have the following file structure (10 pts). Notice that `first` and `last` correspond to your first and last names):

```
hw01/
   README.md
   imports-85.data
```

```
imports-85-dictionary.md
hw01-first-last.Rmd
hw01-first-last.md
hw01-first-last_files/
   figure-markdown_github/
       ...  # png files
```

It is possible that your directory `hw01/` contains hidden files such as `.Rhistory`, `.RData`, or `.DS_Store`. No need to worry about these extra files.

Ideally, your `README.md` file should have a description of what HW01 is about. However, because no further specifications were included in the HW instructions, some students left this file empty. For future HWs, include some content in the READMEs to describe the assignments.

Data Set downloading (-4 pts)

The instructions to download the data file should NOT be part of your report. Deduct 4 points if your Rmd file includes such commands.

```r
# ------------------------------------------------------------------------
# do NOT include this code in your Rmd file (you'll lose points if you do)
# ------------------------------------------------------------------------

# assembling url (so the text fits on this pdf)
uci <- 'https://archive.ics.uci.edu/ml/machine-learning-databases'
autos <- '/autos/imports-85.data'
url <- paste0(uci, autos)

# download file to your working directory
download.file(url = url, destfile = 'imports-85.data')
```

1) Data Dictionary (10 pts)

Based on the information in the file `imports-85.names`, you will have to create a data dictionary in a separate text file: e.g. `imports-85-dictionary.md`. or `imports-85-dictionary.txt`. The file extension `.md` indicates that the content of the dictionary is written in markdown syntax (don't confuse `.md` with `.Rmd`). By the way, don't use an Rmd file to write this dictionary.

2) Data Import (20 pts)

Because the data file `imports-85.data` is a CSV file, you can use the function `read.csv()` in base R, or the function `read_csv()` from the R package `"readr"`, to import the data in R.

2.1) Importing data frame with `read.csv()` function (10 pts)

In order to import the data with `read.csv()` your code should include:

- a character vector for the names of the columns (passed to `col.names`).
- a character vector for data types (passed to `colClasses`).
- keep in mind that there are several ways to specify the vector of data types. The columns `symboling` and `normalized_losses` can be either integer or real.
- the character used to indicate missing values: `na.strings = '?'`.
- because the file does not have column names, you must use `header = FALSE`.
- After importing the data, your code should also include the output of the function `str()` to see the structure of the data frame.

```
# vector of column names
column_names <- c(
  "symboling",
  "normalized_losses",
  "make",
  "fuel_type",
  "aspiration",
  "num_of_doors",
  "body_style",
  "drive_wheels",
  "engine_location",
  "wheel_base",
  "length",
  "width",
  "height",
  "curb_weight",
  "engine_type",
  "num_of_cylinders",
  "engine_size",
  "fuel_system",
  "bore",
  "stroke",
  "compression_ratio",
  "horsepower",
  "peak_rpm",
  "city_mpg",
  "highway_mpg",
  "price"
)
```

```r
# vector with data types for each column
column_types <- c(
  'integer',  # symboling
  'real',     # normalized_losses
  rep('character', 7), # 'make' to 'engine_location'
  rep('real', 4),  # 'wheel_base' to 'height'
  'integer', # curb_weight
  rep('character', 2),
  'integer',
  'character',
  rep('real', 3), # 'bore' to 'compression_ratio'
  rep('integer', 5)  # 'horsepower' to 'price'
)
```

```r
# Data import with read.csv()
dat1 <- read.csv(
  file = 'imports-85.data',
  header = FALSE,
  na.strings = "?",
  col.names = column_names,
  colClasses = column_types)

str(dat1, vec.len = 1)
```

```
## 'data.frame':    205 obs. of  26 variables:
##  $ symboling        : int  3 3 ...
##  $ normalized_losses: num  NA NA ...
##  $ make             : chr  "alfa-romero" ...
##  $ fuel_type        : chr  "gas" ...
##  $ aspiration       : chr  "std" ...
##  $ num_of_doors     : chr  "two" ...
##  $ body_style       : chr  "convertible" ...
##  $ drive_wheels     : chr  "rwd" ...
##  $ engine_location  : chr  "front" ...
##  $ wheel_base       : num  88.6 88.6 ...
##  $ length           : num  169 ...
##  $ width            : num  64.1 64.1 ...
##  $ height           : num  48.8 48.8 ...
##  $ curb_weight      : int  2548 2548 ...
##  $ engine_type      : chr  "dohc" ...
##  $ num_of_cylinders : chr  "four" ...
##  $ engine_size      : int  130 130 ...
##  $ fuel_system      : chr  "mpfi" ...
##  $ bore             : num  3.47 3.47 ...
##  $ stroke           : num  2.68 2.68 ...
```

```
## $ compression_ratio: num  9 9 ...
## $ horsepower       : int  111 111 ...
## $ peak_rpm         : int  5000 5000 ...
## $ city_mpg         : int  21 21 ...
## $ highway_mpg      : int  27 27 ...
## $ price            : int  13495 16500 ...
```

2.2) Importing data frame with `read_csv()` function (10 pts)

In order to import the data with `read_csv()` your code should include:

- a character vector for the names of the columns (passed to `col_names`).
- a character vector for data types (passed to `col_types`). The columns `symboling` and `normalized_losses` can be either integer or real.
- keep in mind that there are several ways to specify the data types values.
- the character used to indicate missing values: `na = '?'`.
- After importing the data, your code should also include the output of the function `str()` to see the structure of the data frame.

```
# Data import with read_csv()
dat <- read_csv(
  file = 'imports-85.data',
  na = '?',
  col_names = column_names,
  col_types = list(
    "symboling" = col_integer(),
    "normalized_losses" = col_double(),
    "make" = col_character(),
    "fuel_type" = col_character(),
    "aspiration" = col_character(),
    "num_of_doors" = col_character(),
    "body_style" = col_character(),
    "drive_wheels" = col_character(),
    "engine_location" = col_character(),
    "wheel_base" = col_double(),
    "length" = col_double(),
    "width" = col_double(),
    "height" = col_double(),
    "curb_weight" = col_integer(),
    "engine_type" = col_character(),
    "num_of_cylinders" = col_character(),
    "engine_size" = col_integer(),
    "fuel_system" = col_character(),
```

```r
    "bore" = col_double(),
    "stroke" = col_double(),
    "compression_ratio" = col_double(),
    "horsepower" = col_integer(),
    "peak_rpm" = col_integer(),
    "city_mpg" = col_integer(),
    "highway_mpg" = col_integer(),
    "price" = col_integer()
  ))

str(dat, vec.len = 1)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    205 obs. of  26 variables:
##  $ symboling        : int  3 3 ...
##  $ normalized_losses: num  NA NA ...
##  $ make             : chr  "alfa-romero" ...
##  $ fuel_type        : chr  "gas" ...
##  $ aspiration       : chr  "std" ...
##  $ num_of_doors     : chr  "two" ...
##  $ body_style       : chr  "convertible" ...
##  $ drive_wheels     : chr  "rwd" ...
##  $ engine_location  : chr  "front" ...
##  $ wheel_base       : num  88.6 88.6 ...
##  $ length           : num  169 ...
##  $ width            : num  64.1 64.1 ...
##  $ height           : num  48.8 48.8 ...
##  $ curb_weight      : int  2548 2548 ...
##  $ engine_type      : chr  "dohc" ...
##  $ num_of_cylinders : chr  "four" ...
##  $ engine_size      : int  130 130 ...
##  $ fuel_system      : chr  "mpfi" ...
##  $ bore             : num  3.47 3.47 ...
##  $ stroke           : num  2.68 2.68 ...
##  $ compression_ratio: num  9 9 ...
##  $ horsepower       : int  111 111 ...
##  $ peak_rpm         : int  5000 5000 ...
##  $ city_mpg         : int  21 21 ...
##  $ highway_mpg      : int  27 27 ...
##  $ price            : int  13495 16500 ...
##  - attr(*, "spec")=List of 2
##   ..$ cols   :List of 26
##   .. ..$ symboling        : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" ...
##   .. ..$ normalized_losses: list()
```

```
##   .. .. ..- attr(*, "class")= chr  "collector_double" ...
##   .. ..$ make             : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" ...
##   .. ..$ fuel_type        : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" ...
##   .. ..$ aspiration       : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" ...
##   .. ..$ num_of_doors     : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" ...
##   .. ..$ body_style       : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" ...
##   .. ..$ drive_wheels     : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" ...
##   .. ..$ engine_location  : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" ...
##   .. ..$ wheel_base       : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" ...
##   .. ..$ length           : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" ...
##   .. ..$ width            : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" ...
##   .. ..$ height           : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" ...
##   .. ..$ curb_weight      : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" ...
##   .. ..$ engine_type      : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" ...
##   .. ..$ num_of_cylinders : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" ...
##   .. ..$ engine_size      : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" ...
##   .. ..$ fuel_system      : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" ...
##   .. ..$ bore             : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" ...
##   .. ..$ stroke           : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" ...
##   .. ..$ compression_ratio: list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" ...
##   .. ..$ horsepower       : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" ...
##   .. ..$ peak_rpm         : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" ...
##   .. ..$ city_mpg         : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" ...
```

```
##    .. ..$ highway_mpg      : list()
##    .. .. ..- attr(*, "class")= chr  "collector_integer" ...
##    .. ..$ price            : list()
##    .. .. ..- attr(*, "class")= chr  "collector_integer" ...
##    ..$ default: list()
##    .. ..- attr(*, "class")= chr  "collector_guess" ...
##    ..- attr(*, "class")= chr "col_spec"
```

3) Technical Questions about importing data (10 pts)

Answer the following questions (using your own words). You do NOT need to include any commands.

a. If you don't provide a vector of column names, what happens to the column names of the imported data when you simply invoke `read.csv('imports-85.data')`?

`read.csv()` will take the first row in a CSV file and use its values as column names.

b. If you don't provide a vector of column names, what happens to the column names of the imported data when you invoke `read.csv('imports-85.data', header = FALSE)`?

`read.csv()` will name the columns as `V1, V2, V3, ...`.

c. When using the reading table functions, if you don't specify how missing values are codified, what happens to the data type of those columns that contain `'?'`, e.g. `price` or `num_of_doors`?

R will treat the data type of those columns as characters, and then convert them into factors.

d. Say you import `imports-85.data` in two different ways. In the first option you import the data without specifying the data type of each column. In the second option you do specify the data types. You may wonder whether both options return a data frame of the same memory size. You can actually use the function `object.size()` that provides an estimate of the memory that is being used to store an R object. Why is the data frame imported in the second option bigger (in terms of bytes) than the data frame imported in the first option?

While importing the data without specifying the data type of each column, the reading table functions will convert all character columns as factors. Because factors are internally stored as integer vectors, these will tend to occupy less memory.

e. Say the object `dat` is the data frame produced when importing `imports-85.data`. What happens to the data values if you convert `dat` as an R matrix?

Converting `dat` as an R matrix will produce a matrix object of character values. Because `dat` contains some character columns, R will implictly coerce all the values as characters (recall that matrices are atomic structures).

4) Practice base plotting (10 pts)

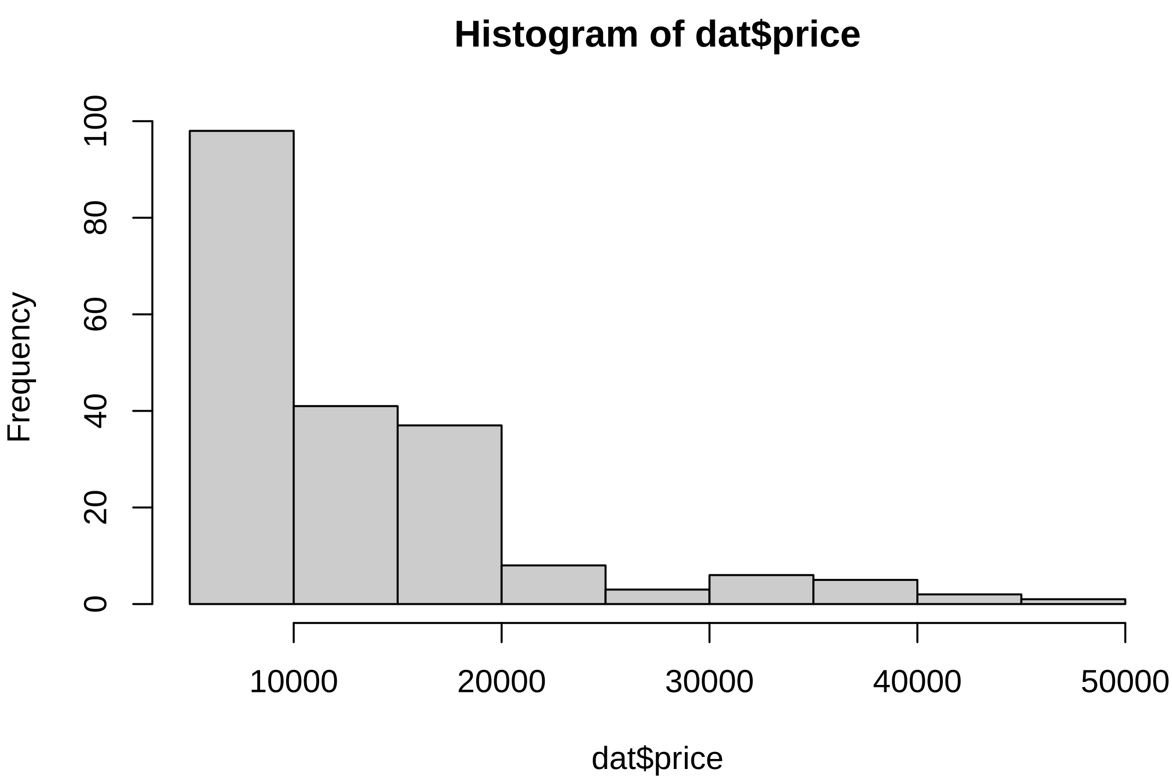
- histogram of `price` with colored bars.
- boxplot of `horsepower` in horizontal orientation.
- barplot of the frequencies of `body_style`, arranged in decreasing order.
- `stars()` plot of vehicles with turbo `aspiration`, using only variables `wheel_base`, `length`, `width`, `height`, and `price`.

The creation of these plots was merely for exploratory purposes, and with the intention that you *played* with base plots (and their parameters). Simply plotting the graphs is NOT enough, your answer should include a (somewhat brief) description. If you don't descriptions, then give half credit.

Keep in mind that in real life, sooner or later, you will have to describe and explain your analysis to a client, a manager, your boss, or any other audience. Even if the graphics are for exploratory purposes, you should jot some notes in your code describing what's going on.

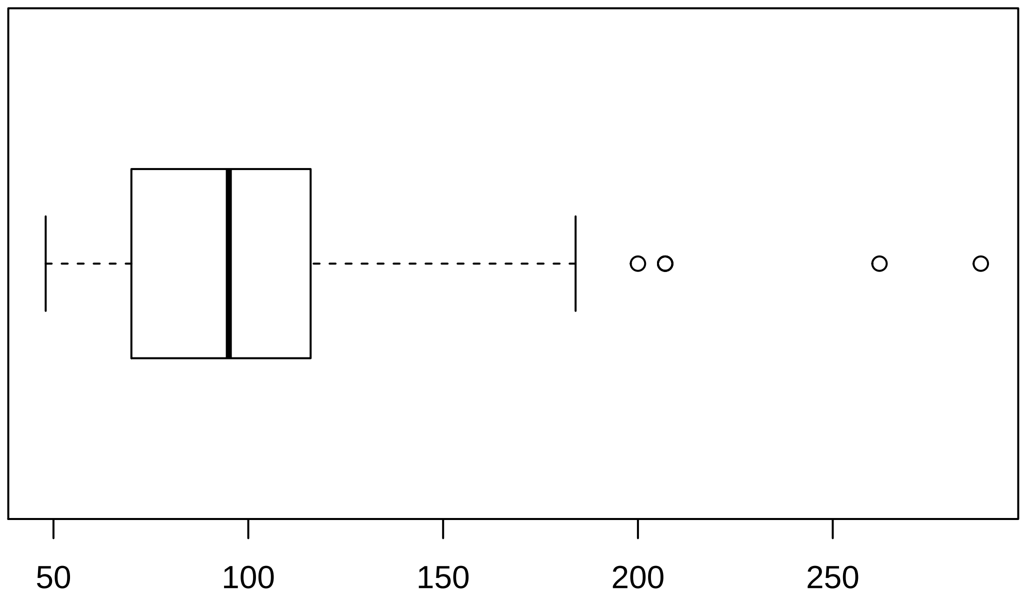
- histogram of `price` with colored bars.

```
hist(dat$price, col = 'gray80')
```



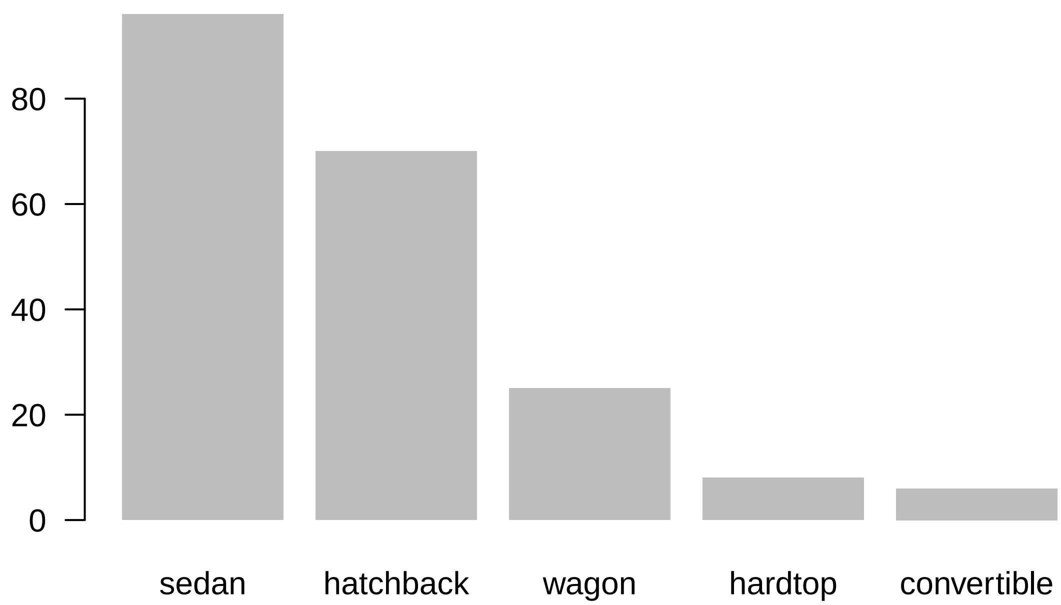
- boxplot of `horsepower` in horizontal orientation.

```
boxplot(dat$horsepower, horizontal = TRUE)
```



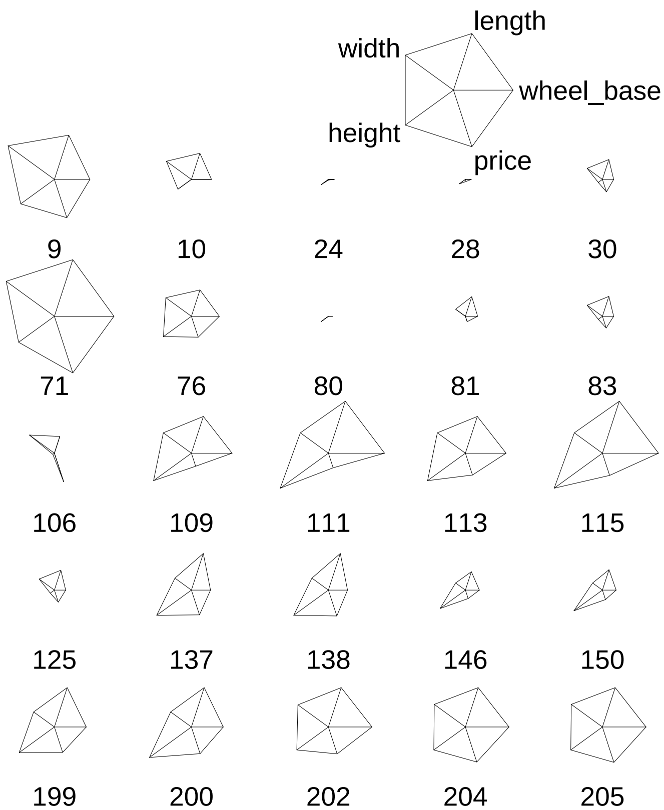
- barplot of the frequencies of `body_style`, arranged in decreasing order.

```
body_style_freqs <- sort(table(dat$body_style), decreasing = TRUE)
barplot(body_style_freqs, las = 1, border = NA)
```



- `stars()` plot of vehicles with turbo `aspiration`, using only variables `wheel_base`, `length`, `width`, `height`, and `price`.

```
turbo_aspiration <- dat$aspiration == 'turbo'
turbo_vars <- c('wheel_base', 'length', 'width', 'height', 'price')
turbo_cars <- dat[turbo_aspiration, turbo_vars]
stars(turbo_cars, labels = rownames(dat)[turbo_aspiration],
      nrow = 5, key.loc = c(9, 13))
```



5) Summaries (10 pts)

Use R code to answer the following questions:

a. What is the mean price of `fuel_type` gas cars? And what is the mean `price` of `fuel_type` diesel cars? (removing missing values)

```
# mean price of fuel-type gas
mean(dat$price[dat$fuel_type == 'gas'], na.rm = TRUE)
```

```
## [1] 12916.41
```

```
# mean price of fuel-type diesel
mean(dat$price[dat$fuel_type == 'diesel'], na.rm = TRUE)
```

```
## [1] 15838.15
```

b. What is the `make` of the car with twelve `num_of_cylinders`?

```
# make of the car with twelve num_of_cylinders
dat$make[dat$num_of_cylinders == 'twelve']
```

```
## [1] "jaguar"
```

c. What is the `make` that has the most diesel cars?

```
# make that has the most diesel cars
diesel_freqs <- table(dat$make[dat$fuel_type == 'diesel'])
most_diesel <- which.max(diesel_freqs)
names(most_diesel)
```

```
## [1] "peugot"
```

d. What is the `price` of the car with the largest amount of `horsepower`?

```
# price of the car with the largest amount of horsepower
max_hp <- max(dat$horsepower, na.rm = TRUE)
dat$price[which(dat$horsepower == max_hp)]
```

```
## [1] NA
```

```
# equivalently
dat$price[which.max(dat$horsepower)]
```

```
## [1] NA
```

e. What is the bottom 10th percentile of `city_mpg`?

```
# bottom 10th percentile of city_mpg
quantile(dat$city_mpg, probs = 0.10)
```

```
## 10%
##  17
```

f. What is the top 10th percentile of `highway_mpg`?

```
# bottom 10th percentile of highway_mpg
quantile(dat$highway_mpg, probs = 0.90)
```

```
## 90%
##  38
```

g. What is the median price of those cars in the bottom 10th percentile of `city_mpg`?

```
# median price of those cars in the bottom 10th percentile of city_mpg
city_mpg_10perc <- dat$city_mpg <= quantile(dat$city_mpg, probs = 0.10)
median(dat$price[city_mpg_10perc], na.rm = TRUE)
```

```
## [1] 32250
```

6) Technical Questions about data frames (10 pts)

Answer the following questions (using your own words). You do NOT need to include any commands.

a. What happens when you use the dollar `$` operator on a data frame, attempting to use the name of a column that does not exist? For example: `dat$xyz` where there is no column named `xyz`.

R will return a `NULL` value

b. Which of the following commands fails to return the vector `mpg` which is a column in the built-in data rfame `mtcars`:
   1. `mtcars$mpg`
   2. `mtcars[ ,1]`
   3. `mtcars[[1]]`
   4. `mtcars[ ,mpg]`
   5. `mtcars[["mpg"]]`
   6. `mtcars$"mpg"`
   7. `mtcars[ ,"mpg"]`

The command `mtcars[ ,mpg]` fails to return the vector `mpg`.

c. Based on your answer for part (b), what is the reason that makes such command to fail?

Because the name of the column `mpg` is not quoted. So R will look for an object `mpg` that does not exist.

d. Can you include an R `list` as a "column" of a data frame? YES or NO, and why.

Keep in mind that a data frame is actually list. So, under some special circumstances, you can include a list as a column of a data frame. The main requirement is that the added list has the same number of elements as the other columns (or that the elements of the added list get recycled to match the length of the other columns).

e. What happens when you apply `as.list()` to a data frame? e.g. `as.list(mtcars)`

When applying `as.list()` to a data frame, R will return a list with as many elements as columns in the data frame.

f. Consider the command: `abc <- as.list(mtcars)`. What function(s) can you use to convert the object `abc` into a data frame?

You can use `data.frame(abc)` or `as.data.frame(abc)` to convert `abc` into a data frame.

7) Correlations of quantitative variables (10 pts)

Except for `symboling` and `normalized_losses`, use the rest of the quantitative variables (both integer and real) to compute a matrix of correlations between such variables. See how to use the function `na.omit()` to create a new data frame with the quantitative variables, that does not contain missing values. Call this data frame `qdat`. *Hint:* see the function `cor()`.

```
# vector of quantitative variable names
quantitative <- c(
  "wheel_base",
  "length",
  "width",
  "height",
  "curb_weight",
  "engine_size",
  "bore",
  "stroke",
  "compression_ratio",
  "horsepower",
  "peak_rpm",
  "city_mpg",
  "highway_mpg",
  "price")

# omit missing values
qdat <- na.omit(dat[ ,quantitative])

# martix of correlations
correlations <- cor(qdat)
round(correlations, 3)
```

```
##                   wheel_base length  width height curb_weight engine_size
## wheel_base             1.000  0.879  0.819  0.593       0.783       0.570
## length                 0.879  1.000  0.858  0.496       0.882       0.687
## width                  0.819  0.858  1.000  0.316       0.867       0.740
## height                 0.593  0.496  0.316  1.000       0.308       0.031
## curb_weight            0.783  0.882  0.867  0.308       1.000       0.858
## engine_size            0.570  0.687  0.740  0.031       0.858       1.000
## bore                   0.498  0.609  0.544  0.189       0.646       0.583
## stroke                 0.172  0.119  0.186 -0.056       0.173       0.212
## compression_ratio      0.248  0.160  0.191  0.261       0.155       0.025
## horsepower             0.376  0.584  0.617 -0.084       0.760       0.843
## peak_rpm              -0.352 -0.281 -0.252 -0.264      -0.279      -0.219
## city_mpg              -0.499 -0.690 -0.647 -0.102      -0.772      -0.711
## highway_mpg           -0.566 -0.719 -0.692 -0.151      -0.813      -0.732
## price                  0.586  0.695  0.754  0.138       0.836       0.889
##                     bore stroke compression_ratio horsepower peak_rpm
## wheel_base         0.498  0.172             0.248      0.376   -0.352
```

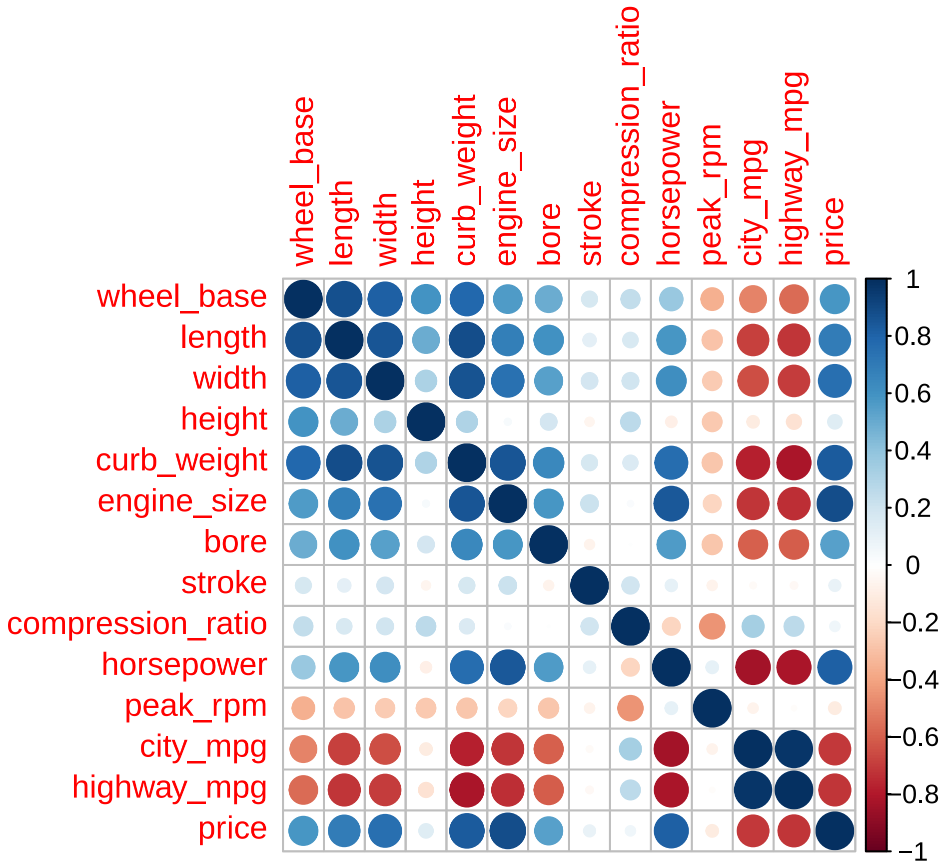
```
## length             0.609  0.119             0.160      0.584   -0.281
## width              0.544  0.186             0.191      0.617   -0.252
## height             0.189 -0.056             0.261     -0.084   -0.264
## curb_weight        0.646  0.173             0.155      0.760   -0.279
## engine_size        0.583  0.212             0.025      0.843   -0.219
## bore               1.000 -0.067             0.003      0.569   -0.278
## stroke            -0.067  1.000             0.200      0.100   -0.068
## compression_ratio  0.003  0.200             1.000     -0.214   -0.445
## horsepower         0.569  0.100            -0.214      1.000    0.106
## peak_rpm          -0.278 -0.068            -0.445      0.106    1.000
## city_mpg          -0.592 -0.028             0.331     -0.834   -0.069
## highway_mpg       -0.600 -0.036             0.268     -0.813   -0.017
## price              0.547  0.094             0.070      0.811   -0.104
##                   city_mpg highway_mpg  price
## wheel_base          -0.499      -0.566  0.586
## length              -0.690      -0.719  0.695
## width               -0.647      -0.692  0.754
## height              -0.102      -0.151  0.138
## curb_weight         -0.772      -0.813  0.836
## engine_size         -0.711      -0.732  0.889
## bore                -0.592      -0.600  0.547
## stroke              -0.028      -0.036  0.094
## compression_ratio    0.331       0.268  0.070
## horsepower          -0.834      -0.813  0.811
## peak_rpm            -0.069      -0.017 -0.104
## city_mpg             1.000       0.972 -0.703
## highway_mpg          0.972       1.000 -0.716
## price               -0.703      -0.716  1.000
```

Read the post *Correlograms* by Xia Liu, available in the file `correlograms-xia-liu`, inside the folder `papers` of the course github repo:

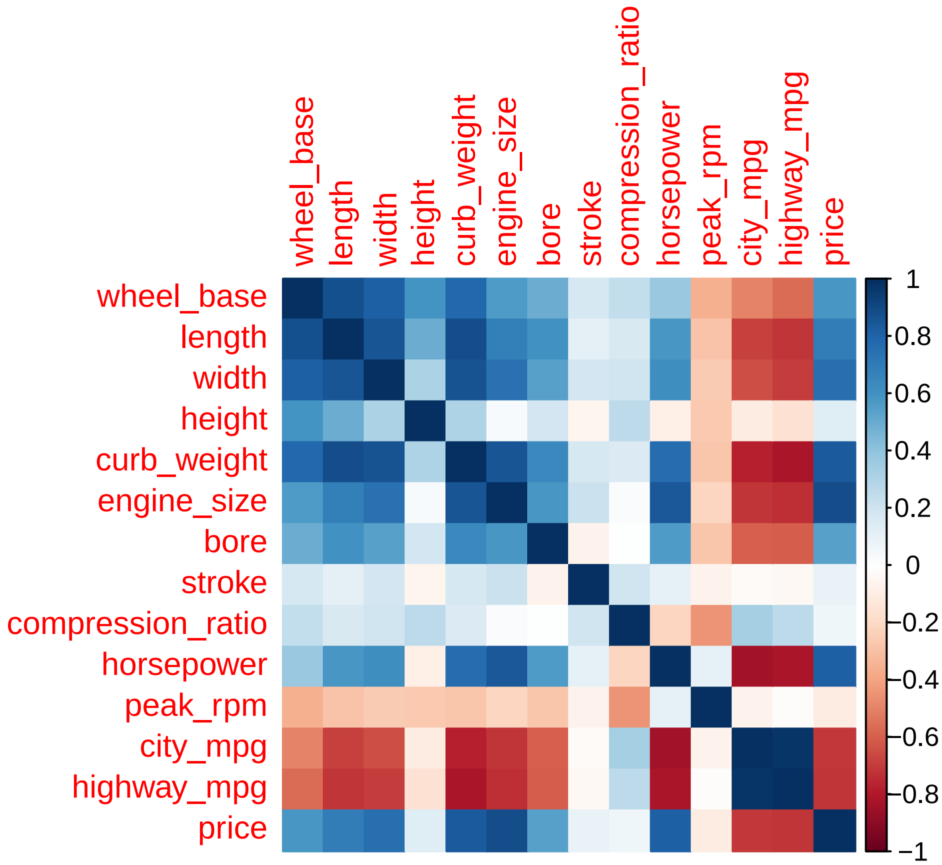
https://github.com/ucb-stat133/stat133-spring-2018/blob/master/papers/correlograms-xia-liu.pdf

Based on the matrix of correlations between the quantitative variables, plot two correlograms, and comment on the patterns and values that you observe.

```
corrplot(correlations, method = 'circle')
```




```
corrplot(correlations, method = 'color')
```



8) Principal Components Analysis (20 pts)

Read the tutorial on Principal Components Analysis (PCA) availabe in the github repository

https://github.com/ucb-stat133/stat133-spring-2018/blob/master/tutorials/06-principal-components.md

8.1) Run PCA (10 pts)

- Use `prcomp()` to perform a principal components analysis on `qdat`; use the argument `scale. = TRUE` to carry out PCA on standardized data.

In theory, your `qdat` object should be the data frame containing the quantitative variables, NOT the correlation matrix of such variables.

However, many students were confused about this, and they have `qdat` defined as the correlation matrix.

As a unique exception, we will accept valid answer of PCA aaplied on the correlation matrix. But keep in mind that the answer key does not show the output based on the correlation data matrix.

```r
pca <- prcomp(qdat, scale. = TRUE)
```

- Examine the eigenvalues and determine the proportion of variation that is "captured" by the first three components.

```r
# table of eigenvalues
eigenvalues <- data.frame(
  eigenvalues = pca$sdev^2,
  proportion = pca$sdev^2 / sum(pca$sdev^2),
  cumulative = cumsum(pca$sdev^2) / sum(pca$sdev^2)
)

eigenvalues
```

```
##    eigenvalues  proportion cumulative
## 1   7.53181553 0.537986823  0.5379868
## 2   2.27923094 0.162802210  0.7007890
## 3   1.21613308 0.086866648  0.7876557
## 4   0.90961519 0.064972514  0.8526282
## 5   0.60894217 0.043495870  0.8961241
## 6   0.41570430 0.029693164  0.9258172
## 7   0.32059895 0.022899925  0.9487172
## 8   0.27014548 0.019296106  0.9680133
## 9   0.12030933 0.008593524  0.9766068
## 10  0.11060092 0.007900066  0.9845068
## 11  0.08158813 0.005827724  0.9903346
## 12  0.06422049 0.004587178  0.9949218
## 13  0.05139667 0.003671191  0.9985929
## 14  0.01969881 0.001407058  1.0000000
```

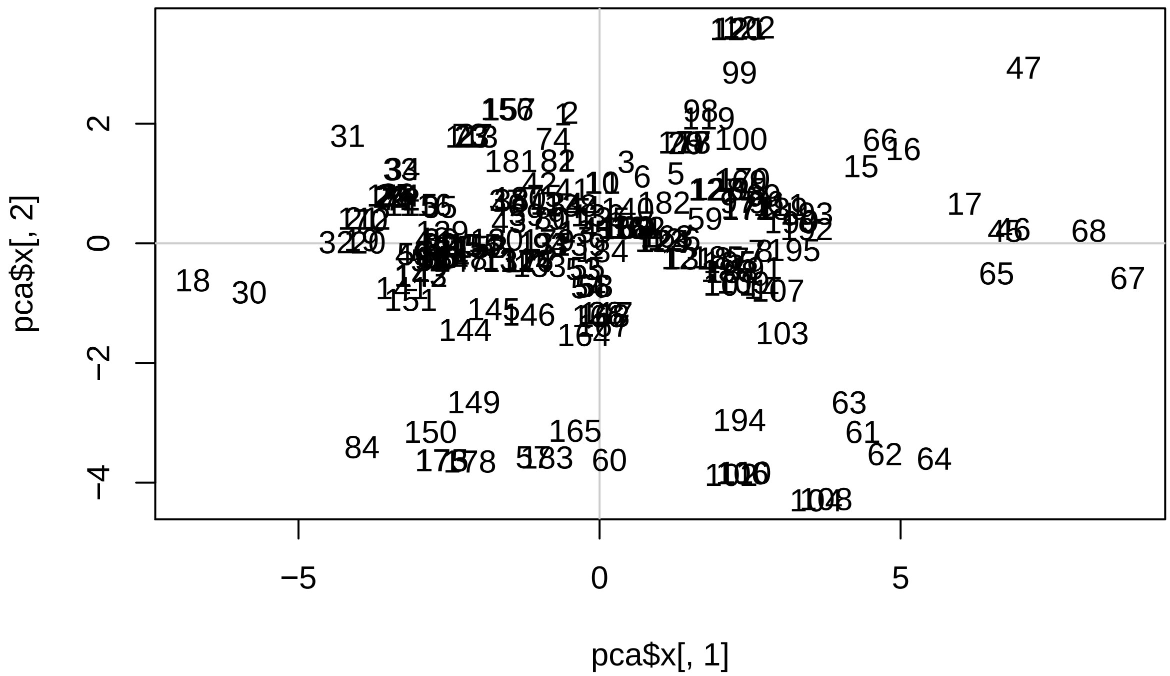
The first three components capture about 78.8 % of the total variation.

This is a very large proportion of variation. Think about it. We are analyzing a data table formed by 14 variables. These variables involve 100% of (multidimensional) variability in the data. However, with the first three components (new synthetic variables) we are able to reproduce almost 80% of the total variation!

8.2) PCA plot of vehicles, and PCA plot of variables (10 pts)

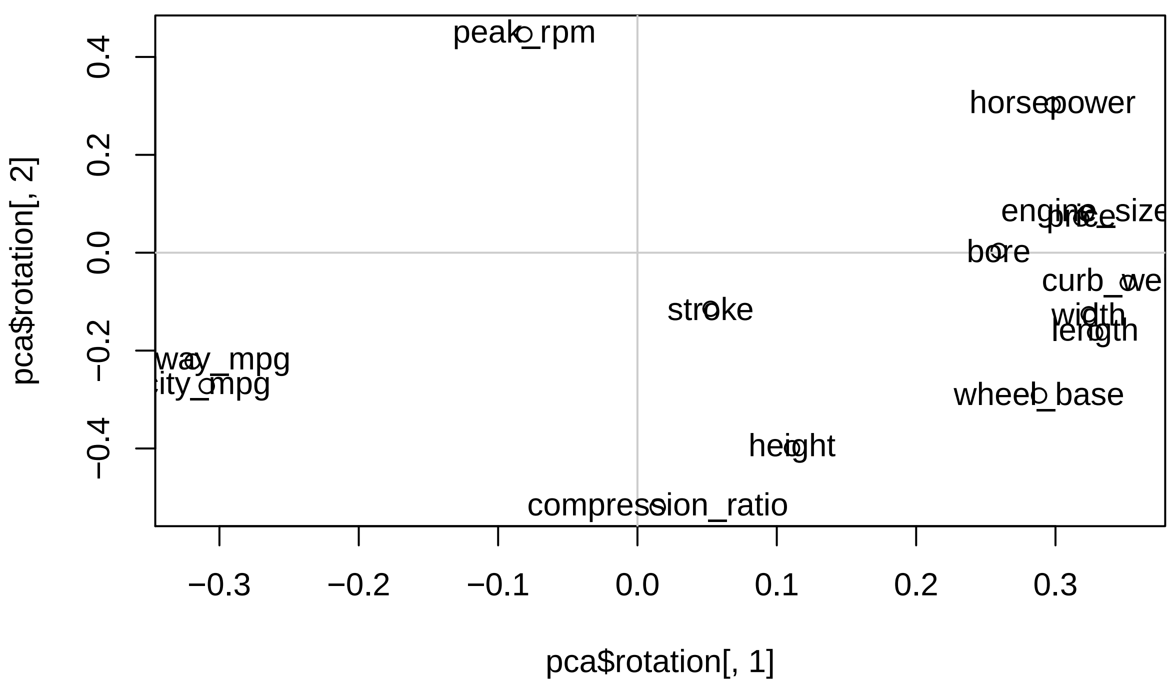
- Use the first two components to graph a scatterplot of the vehicles (do not use "ggplot2" functions).

```r
# plot of variables
plot(pca$x[,1], pca$x[,2], type = 'n')
abline(h = 0, v = 0, col = 'gray80')
text(pca$x[,1], pca$x[,2], labels = 1:nrow(qdat))
```



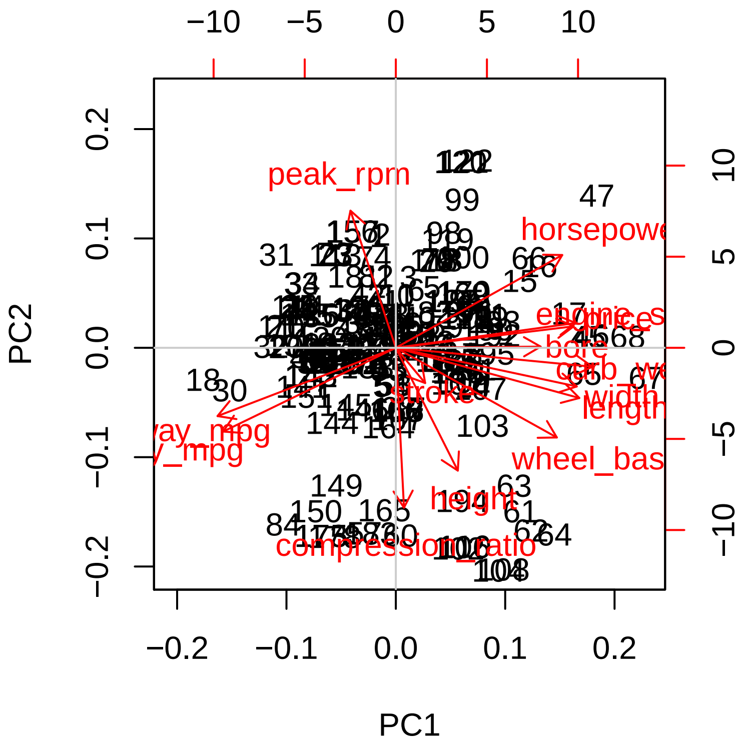
- Use the first two loadings (i.e. eigenvectors) to graph the variables.

```r
# plot of PCs
plot(pca$rotation[,1], pca$rotation[,2])
abline(h = 0, v = 0, col = 'gray80')
text(pca$rotation[,1], pca$rotation[,2], labels = colnames(qdat))
```



- Optionally, you can call `biplot()` of the `"prcomop"` object to get a simultaneous plot of both the vehicles and the variables.

```r
# biplot
biplot(pca)
abline(v = 0, h = 0, col = 'gray80')
```



Unfortunately, the data set does not contain the names of the vehicles, so the interpretation is a bit limited.

The first axis, associated to the first principal component, opposes fuel consumption variables `highway_mpg`, and `city_mpg` against variables `price`, `engine_size`, `curb_weight`, `width`, `length`, and `bore`. We may say that the latter group of variables reflect the *size* of the vehicles: bigger cars, which tend to cost more.

As for the second axis, associated to the second principal component, it opposes `peak_rpm` against `height` and `compression_ratio`.

Looking at the vehicles, a large amount of them tend to be grouped around the center of the scatterplot. As a matter of fact, the center of the scatterplot represents the `average vehicle`. But there is variability and many cars are scattered on all directions: some to the far left (e.g. 18, 30), some to the far right (e.g. 16, 65, 68, 67), some at the top (e.g. 98, 99), some at the bottom (e.g. 165, 60, 57, 183).

Cars that lie on the *East* direction are vehicles which, compared to the rest the cars, tend to be bigger (large values of `engine_size`, `price`, `weight`, etc). Cars 47, 66, 16, 15 lie on the direction of `horsepower`, and are on the opposite directions of `city_mpg` and `highway_mpg` (e.g. cars 18, 30). The more the fuel efficiency of the car, the less horsepowers, and vice versa.