Lab Report 1

your name here

```
# Insert necessary packages
library('tidyverse')
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library('ISLR')
library('gridExtra')
```

```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine
```

Question 1: Linear Regression

```
# Read in the data
spotify <- read.csv('data/spotify_songs.csv')
dim(spotify)
```

```
## [1] 32833    23
```

1.1. (10 pts)

Give basic insights into your numeric variable you have picked as output variable using one categorical variable you selected.

- What are the min / max values and median of the output variable, Y?

```
spotify1_1 = spotify[,c('energy','loudness','tempo','playlist_genre','danceability')]
# min
min(spotify1_1['danceability'])
```

```
## [1] 0
```

```
# max
max(spotify1_1['danceability'])
```

```
## [1] 0.983
```

```
# median
median(spotify1_1$danceability, na.rm=TRUE)
```

```
## [1] 0.672
```

- What is median of the output value among different classes of the categorical variable you picked? You must use `group_by` and `summarize` functions.

```
# median output among different classes
group_by(spotify1_1, playlist_genre) %>%
  summarise(median(danceability))
```

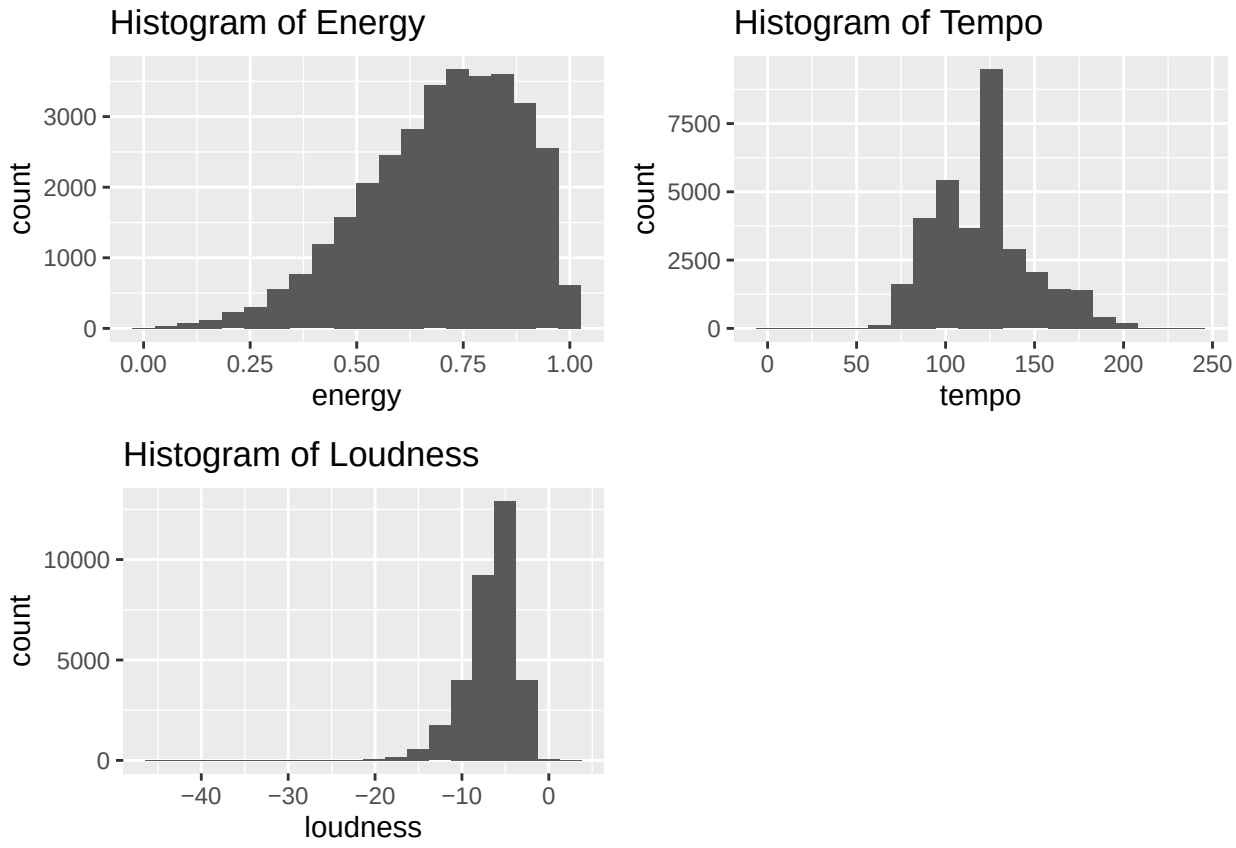
```
## `summarise()` ungrouping output (override with `.groups` argument)

## # A tibble: 6 x 2
##   playlist_genre `median(danceability)`
##   <chr>                           <dbl>
## 1 edm                             0.659
## 2 latin                           0.729
## 3 pop                             0.652
## 4 r&b                             0.689
## 5 rap                             0.737
## 6 rock                            0.523
```

1.2. (10 pts)

Visualize the variables you selected.

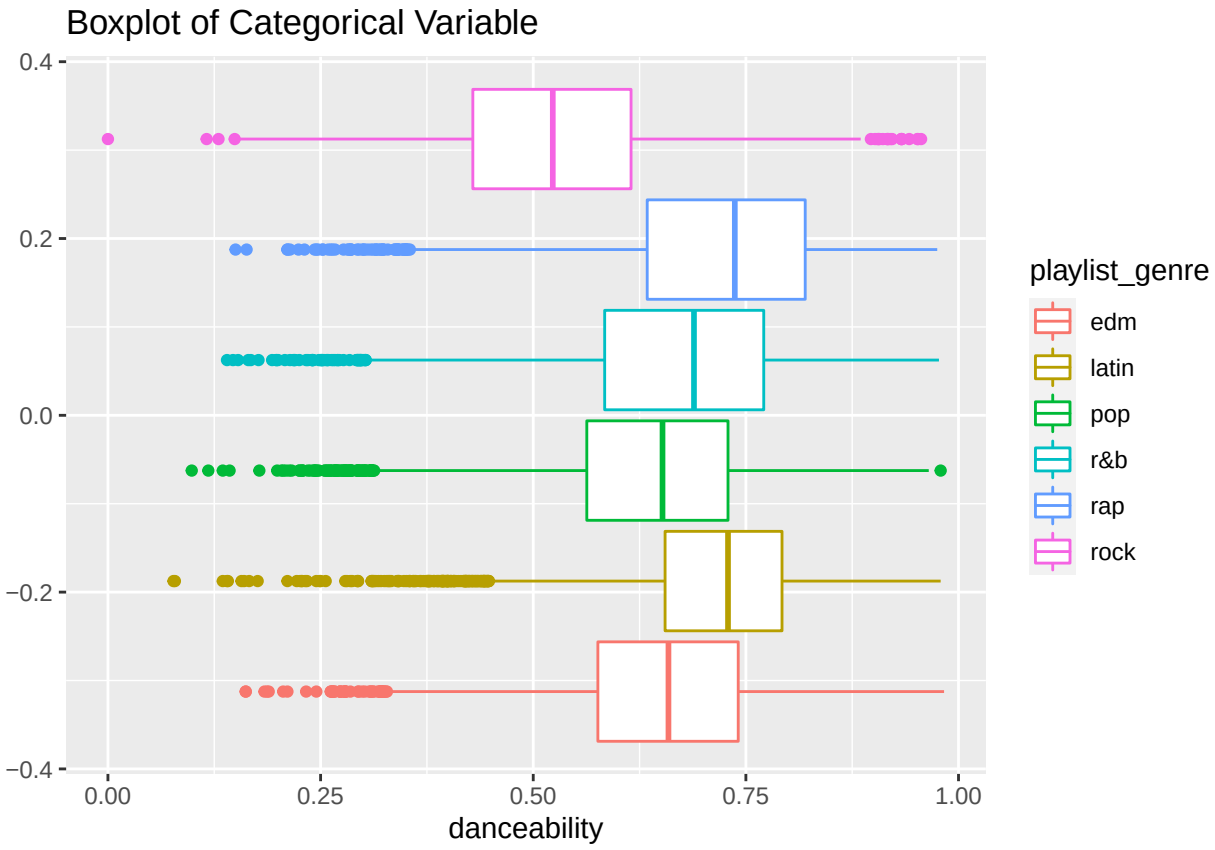
- Draw histogram of the numeric variables you selected.

```r
# Histogram plot of numeric features

energy <- ggplot(data = spotify1_1) +
  geom_histogram(aes(x = energy), bins=20) +
  ggtitle("Histogram of Energy")

loudness <- ggplot(data = spotify1_1) +
  geom_histogram(aes(x = loudness), bins=20) +
  ggtitle("Histogram of Loudness")

tempo <- ggplot(data = spotify1_1) +
  geom_histogram(aes(x = tempo), bins=20) +
  ggtitle("Histogram of Tempo")

grid.arrange(energy, tempo, loudness, ncol=2)
```



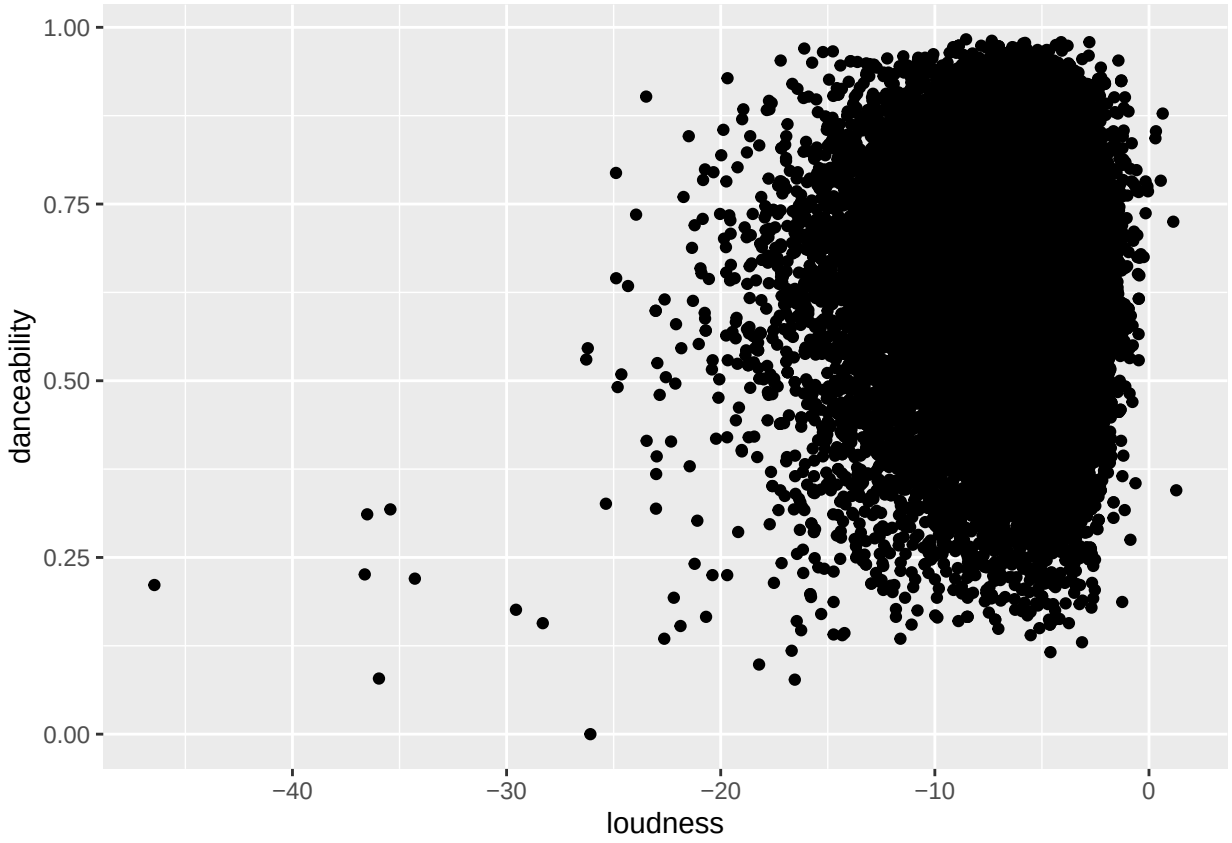
- Draw distribution of the output variable Y with respect to the different classes of your categorical variable. The plot must somehow show the distributional differences among different classes. You can use boxplot, histogram, or other visuals (e.g. density ringes).

```r
# Boxplot of categorical feature classes
ggplot(data = spotify1_1) +
  geom_boxplot(aes(x = danceability, colour=playlist_genre)) +
  ggtitle("Boxplot of Categorical Variable")
```



- Draw scatter plot between one of your numeric inputs and the output variable. Discuss whether the plot indicate a relation, if it is linear, if there are outliers? Feel free to remove the outlier. Feel free to transform the data.

```
# Plot input variable vs output variable
ggplot(data = spotify1_1) +
  geom_point(aes(x = loudness, y=danceability))
```




```
# Remove outliers
spotify1_2<-spotify1_1[(spotify1_1$loudness>-25),]
```

There does not seem to be any linear relation between the input variable (loudness) and the output variable (danceability). In terms of removing outliers, loudness values < -25 were dropped.

1.3. (15 pts)

Using the all dataset, fit a regression:

1. Using the one numeric input variable fit a simple regression model.

- Write down the model.
  - Fit the regression line.
  - Summarize the output.
  - Plot the input and output variables in a scatter plot and add the predicted values as a line.
  - Interpret the results. Is it a good fit? Is your input variable good in explaining the outputs?

2. Using all your input variables, fit a multiple linear regression model

   - Write down the model
   - Fit the regression line and summarize the output
   - Interpret the results. Is it a good fit? Are the input variables good in explaining the outputs?

3. Now, do the same things as you did, but this time add an interaction between one categorical and one numeric variable.

    - Write down the model, fit to the data, summarize and interpret the results.

4. Which model you fit is the best in predicting the output variable? Which one is the second and third best? Rank the models based on their performance.

1.4. (15 pts)

In this section, you will do the same you did in 1.3, but this time you will first split the data into train and test.

- Select seed to fix the random numbers you will generate using `set.seed(...)`.
- Split your data into test and train sets with 20/80 test-train ratio.
- Fit the model to the train set and evaluate the how well the model performed on test set.
- Which model performed the best on test set? Rank the models based ion their performance.
- Is the rank the same as the one you had in 1.3?

Question 2: Gradient Descent Algorithm (By hand)

In case you want to take a picture (screenshot) of your notebook (tablet), you can use the below lines to embed the image to the output PDF file:

```
#knitr::include_graphics('conspiracy.jpg')
```

Question 3. Gradient Descent Algorithm

3.1. Get familiar

You will use horsepower as input variable and miles per gallon (mpg) as output:

1. Plot the scatterplot between `mpg` (Y) and `horsepower` (X).
   - Is the relationship positive or negative? Does mpg increase or reduce as horsepower increases?
   - Is the relationship linear?
2. Plot the scatterplot between `log(mpg)` and `log(horsepower)`.
   - Is the relationship positive or negative?
   - Is the relationship linear?
3. Which of the two versions is better for linear regression?

3.2. Fill in the code

The code below estimates the coefficients of linear regression using gradient descent algorithm. If you are given a single linear regression model;

$$Y = \beta_0 + \beta_1 X$$

where $Y = [Y_1, \ldots, Y_N]^T$ and $X = [X_1, \ldots, X_N]^T$ are output and input vectors containing the observations.

The algorithm estimates the parameter vector $\theta = [\beta_0, \beta_1]$ by starting with an arbitrary θ_0 and adjusting it with the gradient of the loss function as:

$$\theta := \theta + \frac{\alpha}{N} X^T (Y - \theta X)$$

where α is the step size (or learning rate) and $(Y - \theta X)^T X$ is the gradient. At each step it calculates the gradient of the loss and adjusts the parameter set accordingly.

```
GDA <- function(x, y, theta0, alpha = 0.01, epsilon = 1e-8, max_iter=25000){

  # Inputs
  # x      : The input variables (M columns)
  # y      : Output variables    (1 column)
  # theta0 : Initial weight vector (M+1 columns)

  x     <- as.matrix(x)
  y     <- as.matrix(y)
  N     <- nrow(x)
  i     <- 0
  theta <- theta0
  x     <- cbind(1, x) # Adding 1 as first column for intercept
  imprv <- 1e10
  cost  <- (1/(2*N)) * t(x %*% theta - y) %*% (x %*% theta - y)
  delta <- 1
```

```
  while(imprv > epsilon & i < max_iter){
    i <- i + 1
    grad  <- (t(x) %*% (y-x %*% theta))
    theta <- theta + (alpha / N) * grad
    cost  <- append(cost, (1/(2*N)) * t(x %*% theta - y) %*% (x %*% theta - y))
    imprv <- abs(cost[i+1] - cost[i])
    if((cost[i+1] - cost[i]) > 0) stop("Cost is increasing. Try reducing alpha.")
  }
  if (i==max_iter){print(paste0("maximum interation ", max_iter, " was reached"))} else {
    print(paste0("Finished in ", i, " iterations"))
  }

  return(theta)
}

plot_line <- function(theta) {
  ggplot(Auto, aes(x=log(horsepower),y=log(mpg))) +
    geom_point(alpha=.7) +
    geom_abline(slope = theta[2], intercept = theta[1], colour='firebrick') +
    ggtitle(paste0('int: ', round(theta[1],2), ', slope: ', round(theta[2],2)))
}

x <- log(Auto$horsepower)
y <- log(Auto$mpg)
theta0 <- c(1,1)
```

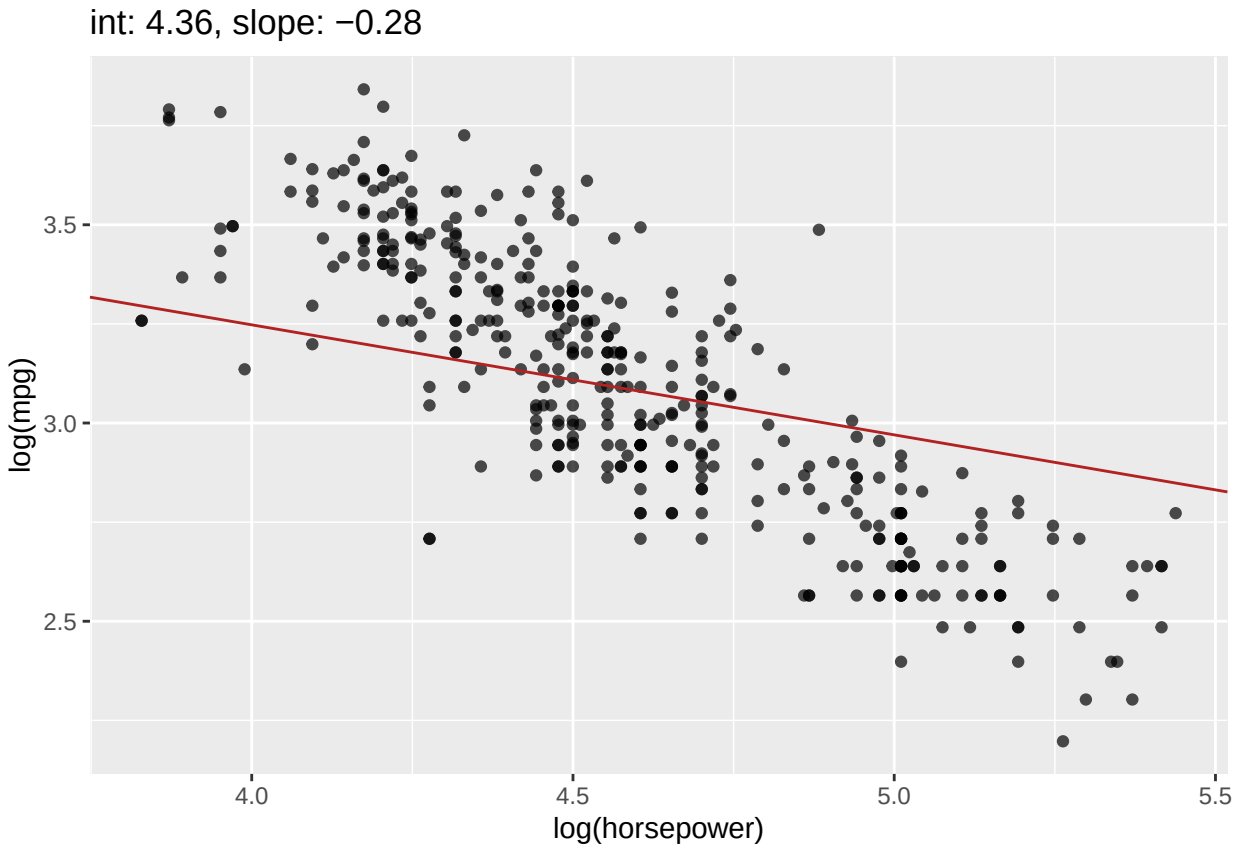
3.3. Run GDA

1. Run the code with the above parameters. How many iterations did it take to estimate the parameters?

```
theta <- GDA(x, y, theta0, alpha = 0.05, epsilon = 1e-5)
```

```
## [1] "Finished in 3193 iterations"
```

```
plot_line(theta)
```



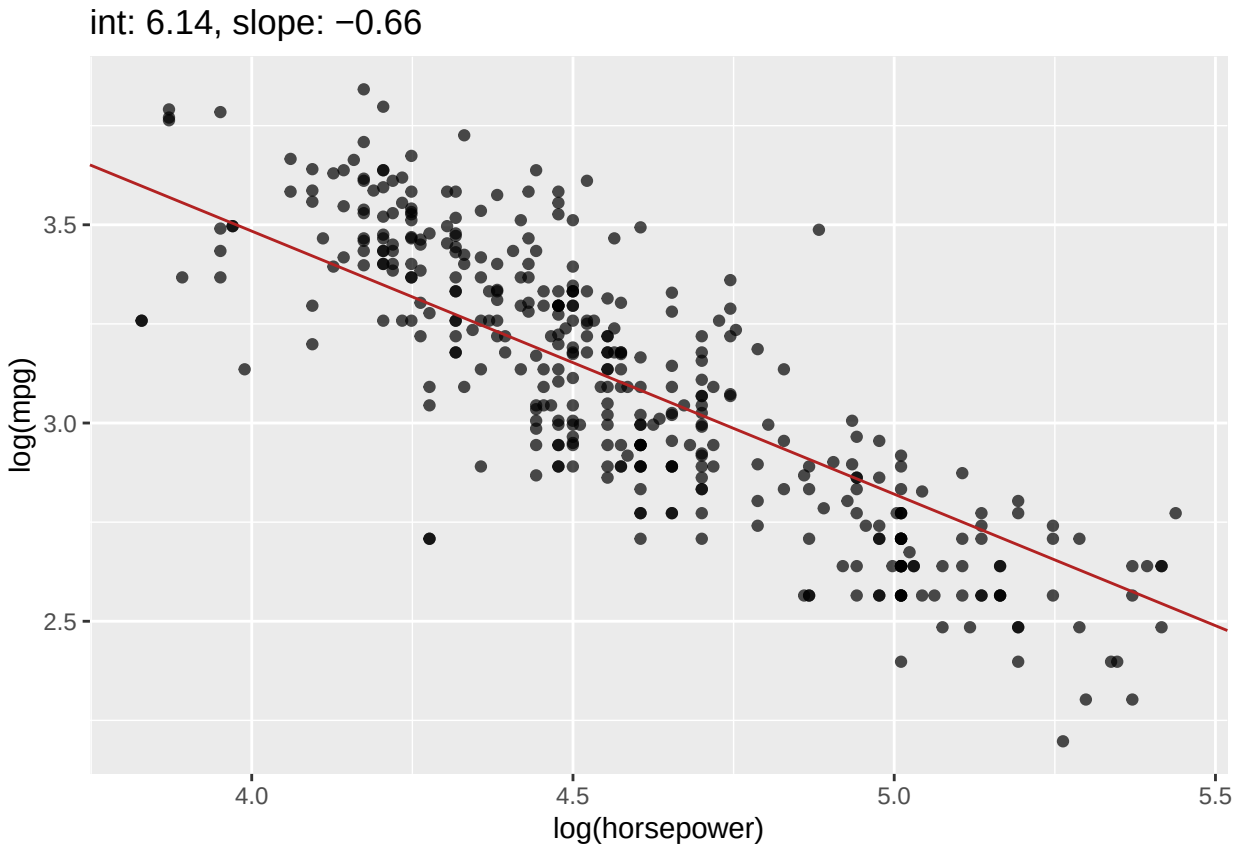
3.3.1) It took 3193 iterations to estimate the parameters.

2. Reduce epsilon to `1e-6`, set `alpha=0.05` run the code.
    - How many iterations did it take to estimate the parameters?
    - Does the result improve? Why or why not?

```
theta <- GDA(x, y, theta0, alpha = 0.05, epsilon = 1e-6)
```

```
## [1] "Finished in 7531 iterations"
```

```
plot_line(theta)
```



3.3.2) It took 7531 iterations to estimate the parameters. The results did improve because the epsilon value used is smaller than the one listed in Question 3.3.1. As the epsilon value is smaller, this means that the Gradient Descent Algorithm code only stops when the difference between the actual value and the estimated value is less than the given epsilon value. The smaller the epsilon value, the smaller the difference has to be, thus a more optimal solution is achieved.

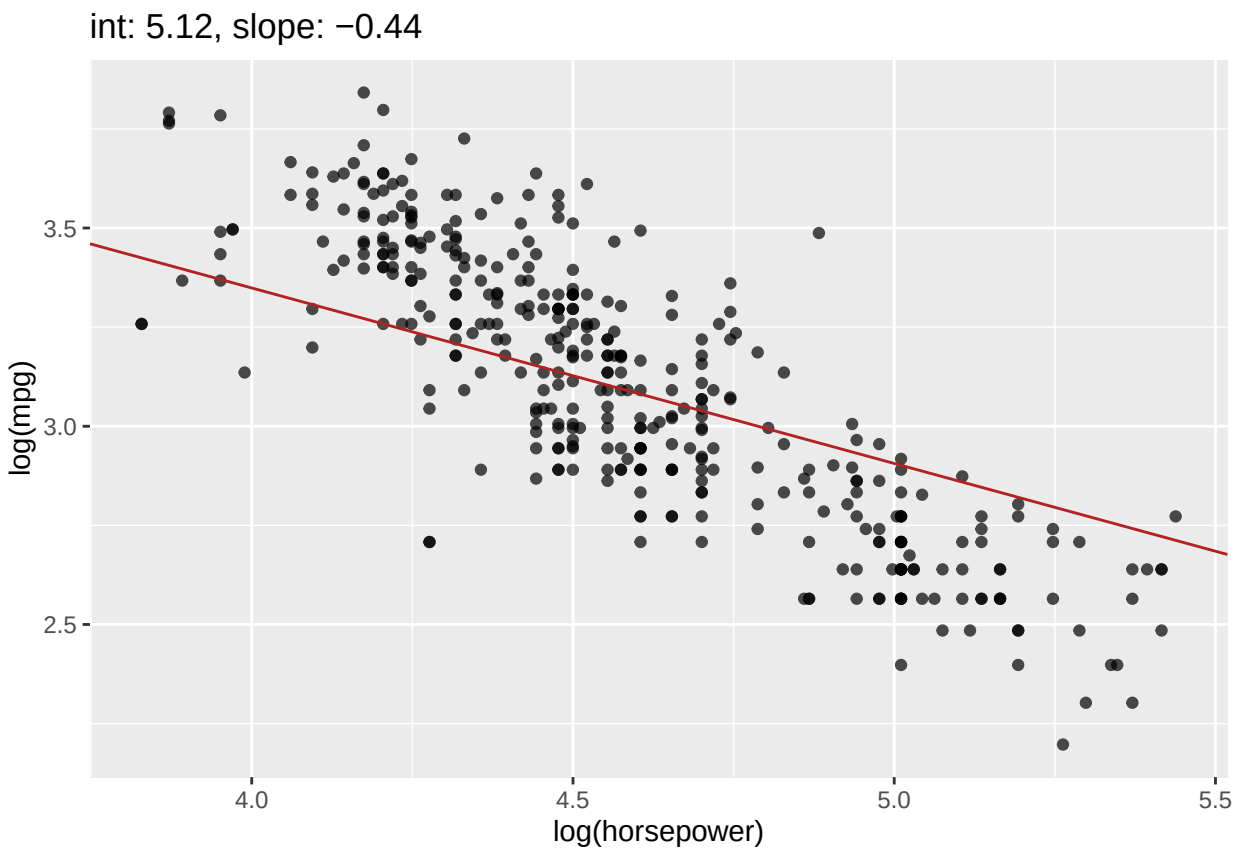
3. Reduce alpha to `alpha=0.01`

   - How many iterations did it take?
   - Did the resulting line change? Why or why not?

```
theta <- GDA(x, y, theta0, alpha = 0.01, epsilon = 1e-6)
```

```
## [1] "Finished in 22490 iterations"
```

```
plot_line(theta)
```



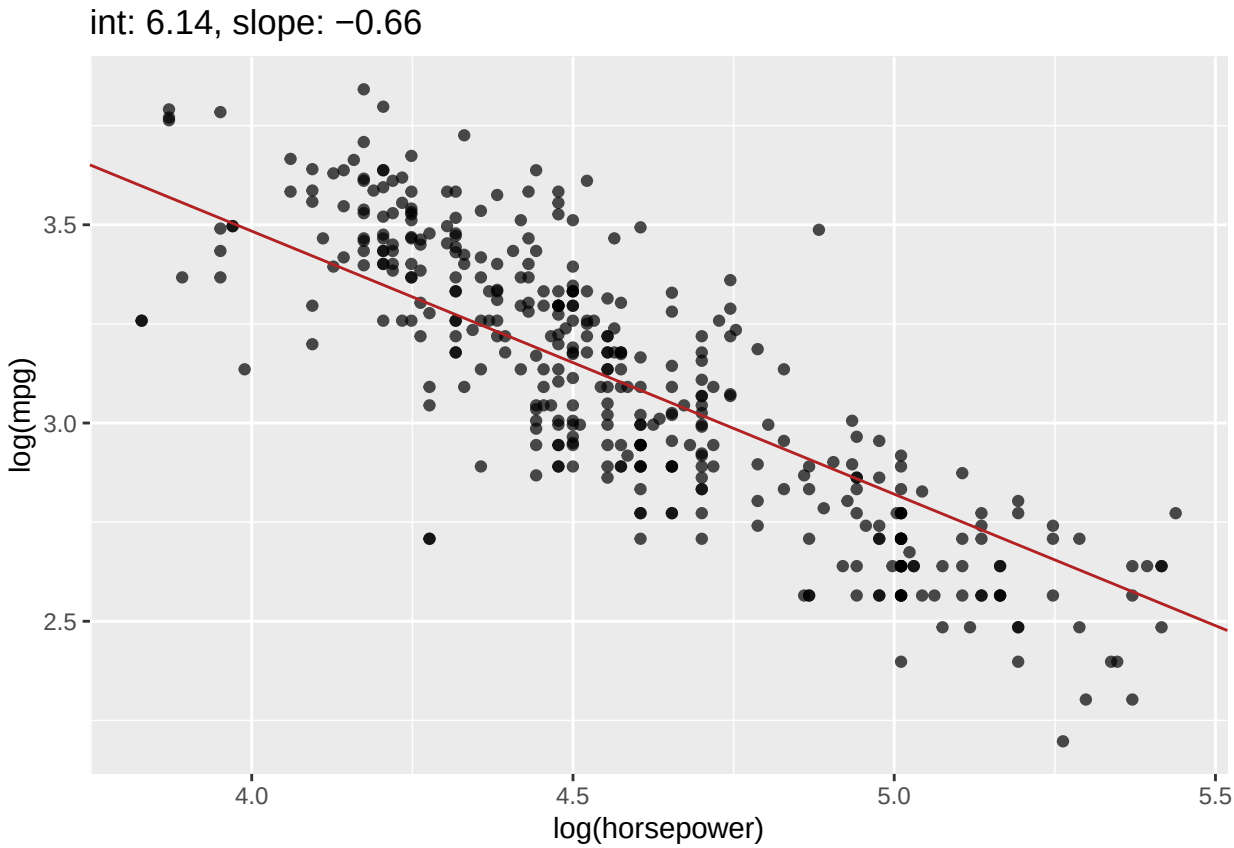
3.3.3) It took 22490 iterations to estimate the parameters. The resulting line did change because the alpha value was decreased from the previous question. This means that it will take a longer time for the solution to converge as the learning rate is smaller now.

4. Set alpha back to `alpha=0.05` and try `theta0=c(1,1)` vs. `theta0=c(1,-1)`:
    - How many iterations did it take? Which is less than the other?
    - Why starting with a negative slope have this effect?

```
theta0 <- c(1,-1)
theta <- GDA(x, y, theta0, alpha = 0.05, epsilon = 1e-6)
```

```
## [1] "Finished in 7265 iterations"
```

```
plot_line(theta)
```



3.3.4) It took 7265 iterations to estimate the parameters with theta0=(1,-1) which was less than the amount of iterations it took theta0=(1,1) (it took 7531 iterations from Question 3.3.2). Starting with a negative slope has this effect because the data has a negative relationship as evident by the plot above. With a negative slope it matches the trend of the data and thus allows for a faster convergence to the optimal solution.

5. Reduce epsilon to `epsilon = 1e-8` and try `alpha=0.01`, `alpha=0.05` and `alpha=0.1`.
    - What effect does alpha have on iterations and resulting fitted line?

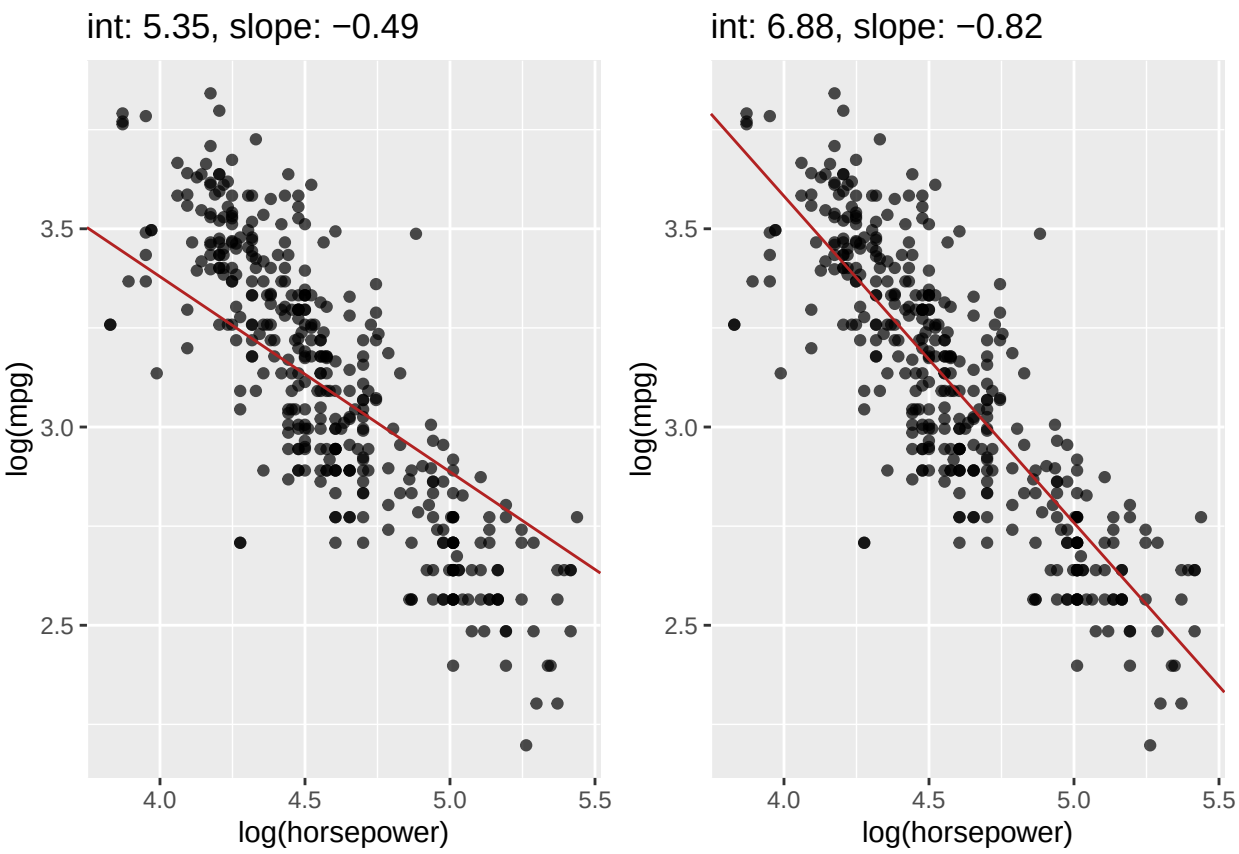
```
theta0 <- c(1,1)
theta <- GDA(x, y, theta0, alpha = 0.01, epsilon = 1e-8)
```

```
## [1] "maximum interation 25000 was reached"
```

```
q_3.3.5_1 <- plot_line(theta)
theta <- GDA(x, y, theta0, alpha = 0.05, epsilon = 1e-8)
```

```
## [1] "Finished in 16207 iterations"
```

```
q_3.3.5_2 <- plot_line(theta)

# setting alpha = 0.1 the cost starts to increase
# theta <- GDA(x, y, theta0, alpha = 0.1, epsilon = 1e-8)

grid.arrange(q_3.3.5_1, q_3.3.5_2, ncol=2)
```



The plot on the left took the maximum amount of iterations (25000), while the plot on the right took 16207 iterations. Using an alpha value of 0.1, the cost started to increase and thus the results could not be computed. The result that alpha has on the iterations is that with a lower alpha value, the learning rate is a lot smaller and thus more iterations are required to reach the optimal solution. A higher alpha value leads to a faster learning rate and less iterations are required to reach the optimal solution. However, too large of an alpha value can lead to a much faster convergence but to a suboptimal solution.

Question 4. BGD vs SGD