Stats 102B - Homework 1

Charles Liu (304804942)

Spring 2020

Modify this file with your answers and responses.

Academic Integrity Statement

By including this statement, I, Charles Liu, declare that all of the work in this assignment is my own original work. At no time did I look at the code of other students nor did I search for code solutions online. I understand that plagiarism on any single part of this assignment will result in a 0 for the entire assignment and that I will be referred to the dean of students.

I did discuss ideas related to the homework with Shirley Mach for Part 2 regarding the plots of fitted and actual values. We merely discussed how it should be displayed and plotted. We exchanged different ideas and different functions we should look into to making it work. At no point did I show another student my code, nor did I look at another student's code.

Part 1: Weighted Least Squares Regression

Task 1A

The gradient of the loss w.r.t. $\mathbf{w}$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{0} - \mathbf{X}^{\mathbf{T}}\mathbf{A}\mathbf{t} + \mathbf{X}^{\mathbf{T}}\mathbf{A}\mathbf{X}\hat{\mathbf{w}} = \mathbf{0}$$

The solution for $\hat{\mathbf{w}}$:

$$\hat{\mathbf{w}} = (\mathbf{X}^{\mathbf{T}}\mathbf{A}\mathbf{X})^{-1}\mathbf{X}^{\mathbf{T}}\mathbf{A}\mathbf{t}$$

Task 1B

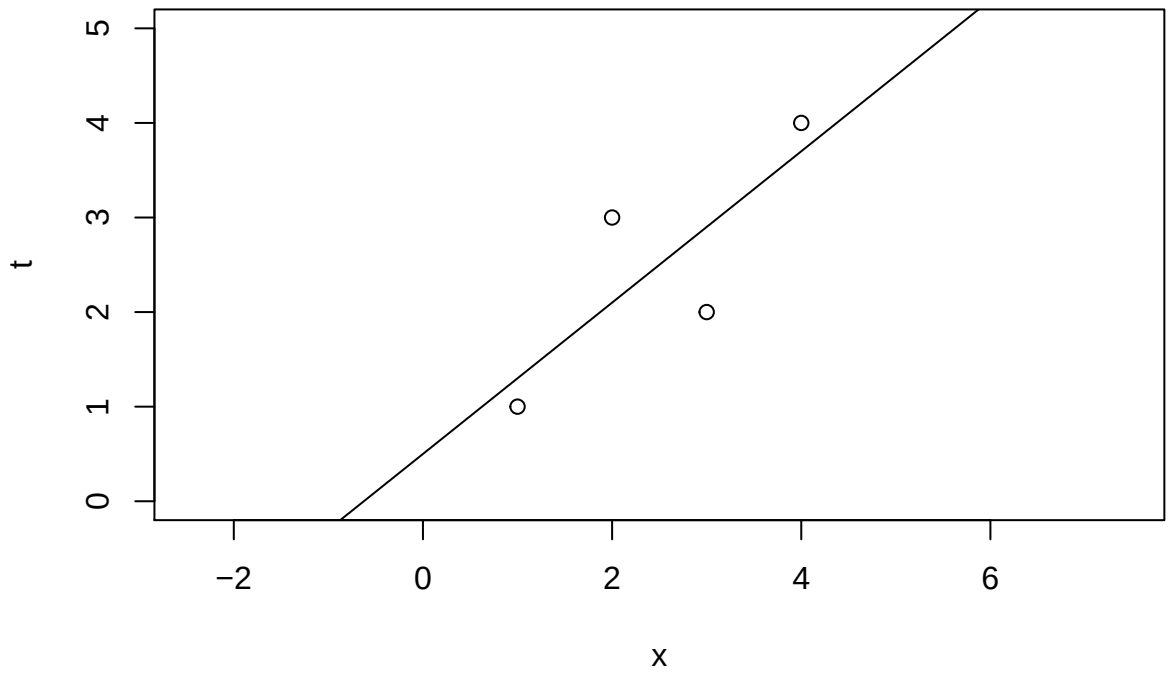
Let's see the effect of altering the values of our weights α.

Toy data:

```
x <- c(1, 2, 3, 4)
t <- c(1, 3, 2, 4)
```

I have already fit a model using `lm()` with the argument `weights`.

```
a1 <- c(1, 1, 1, 1)
model1 <- lm(t ~ x, weights = a1)
plot(x, t, xlim = c(0, 5), ylim = c(0, 5), asp = 1)
abline(model1)
```




```
print(model1$coefficients)
```

```
## (Intercept)           x
##         0.5         0.8
```

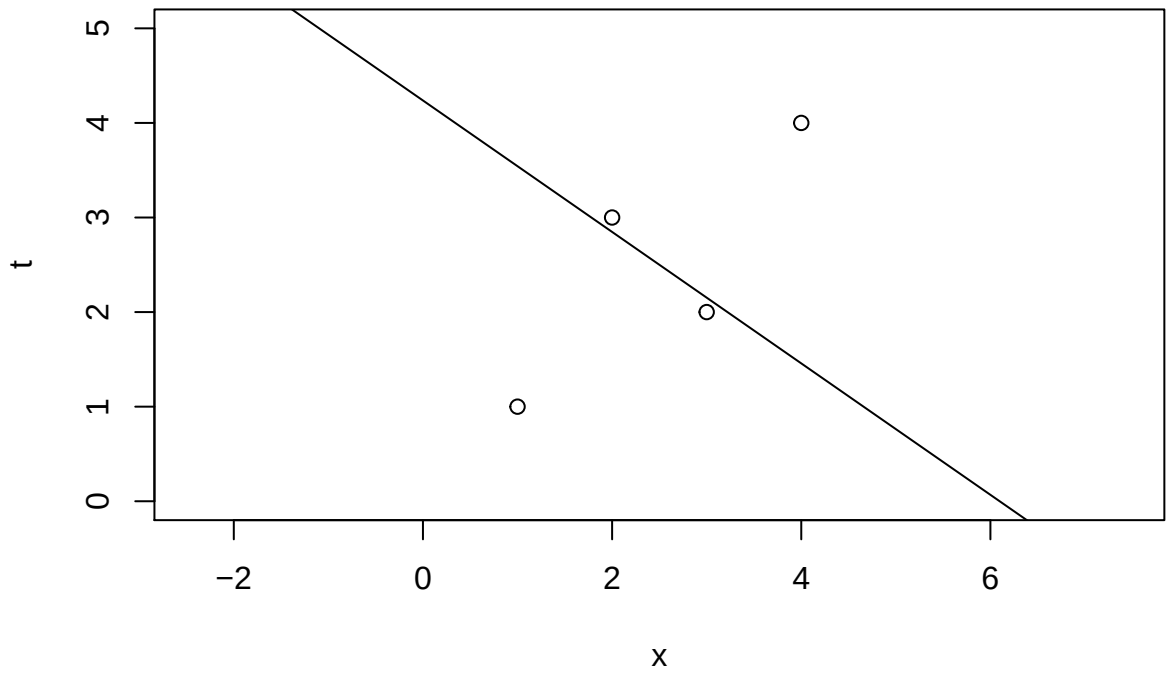
Go ahead and use the matrix operations from your solution from above to find and print the parameter estimates. They should equal the parameter estimates found via `lm()`

```
x1 <- cbind(rep(1, length(x)), x)
w_1B.1hat <- solve(t(x1) %*% diag(a1) %*% x1) %*% (t(x1) %*% diag(a1) %*% t)
w_1B.1hat
```

```
##   [,1]
##    0.5
## x  0.8
```

Now we alter the weights. This vector puts large weight on the two inner points (x=2, x= 3), and small weight on the outer points (x=1, x=4).

```
# large weights on x = 2 & x = 3
a2 <- c(0.1, 5, 5, 0.1)
model2 <- lm(t ~ x, weights = a2)
plot(x,t,xlim = c(0,5), ylim = c(0,5), asp=1)
abline(model2)
```




```
print(model2$coefficients)
```

```
## (Intercept)           x
##   4.2372881  -0.6949153
```

Again, use the matrix operations to find and print the parameter estimates using the provided weights. Compare them against the estimates found via `lm()`. I have plotted the fitted line, comment on the effect of the weights.

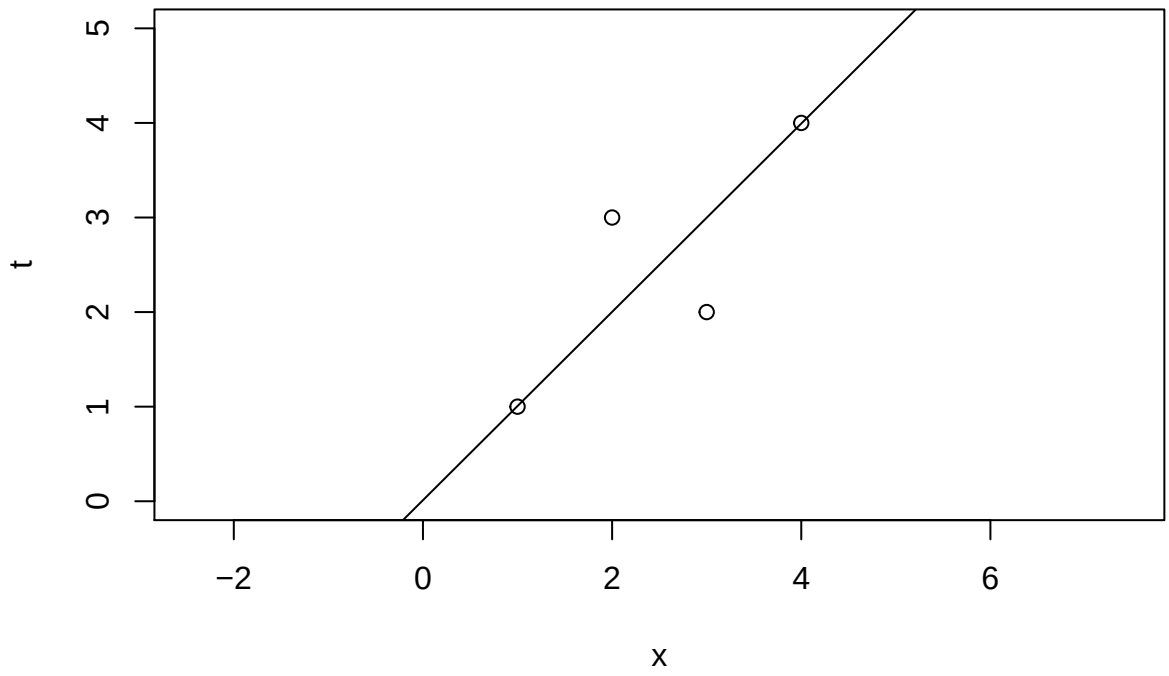
```
x2 <- cbind(rep(1, length(x)), x)
w_1B.2hat <- solve(t(x2) %*% diag(a2) %*% x2) %*% (t(x2) %*% diag(a2) %*% t)
w_1B.2hat
```

```
##         [,1]
##    4.2372881
## x -0.6949153
```

```
# We can see here that the estimatation of the intercept is close to 0.5, but we can see that there slo
```

We alter the weights again. This time large weight are on the two outer points (x=1, x=4), and small weight on the inner points (x=2, x= 3).

```
# large weights on x = 1 & x = 4
a3 <- c(5, 0.1, 0.1, 5)
model3 <- lm(t ~ x, weights = a3)
plot(x, t, xlim = c(0, 5), ylim = c(0, 5), asp = 1)
abline(model3)
```




```
print(model3$coefficients)
```

```
## (Intercept)           x
##  0.01108647  0.99556541
```

Again, use the matrix operations to find and print the parameter estimates using the provided weights. Compare them against the estimates found via `lm()`. Look at the fitted line and comment on the effect of the weights.

```
x3 <- cbind(rep(1, length(x)), x)
w_1B.3hat <- solve(t(x3) %*% diag(a3) %*% x3) %*% (t(x3) %*% diag(a3) %*% t)
w_1B.3hat
```

```
##         [,1]
##   0.01108647
## x 0.99556541
```

```
# We can see here that the estimatation of the intercept is no longer that close to 0.5, but we can see
```

- Explain Weighted Least Squares regression. What effect would putting a very large weight, say 1000, on a point have on the regression line? What effect would putting a weight of 0 on a point have on the regression line?

The Weighted Least Squares regression is used when the OLS Regression's assumption of constant variance in errors (homoscedasticity) is violated. The more weight you put on a point/points, the closer you are to the actual estimation. Keep in mind that you can also cause other problems to arise from using this method as well. It also targets specific areas that might be computational expensive to calculate. The WLS Regression targets those areas and puts heavier emphasis in those areas. It is the only method that can be used when you have vary quality of data points. If you were to have a very large weight (say maybe of 1000) on a point,

you may cause your model to be overfitting, resulting in inaccurate conclusions. Similarly, if you have very small weight on a point, you may have your model be underfitting, resulting in also an inaccurate conclusion.

Part 2: OLS Matrix Notation

Task 2

The Chirot dataset which covers the 1907 Romanian Peasant Revolt:

```
library(carData)
data(Chirot)
chirot_mat <- as.matrix(Chirot)
```

Already done for you: extract the commerce column and create our **X** matrix.

```
t <- chirot_mat[, 1, drop = FALSE]  # response, keep as a matrix
x <- chirot_mat[, 2, drop = FALSE]  # commerce column, keep as matrix
X <- cbind(1, x)
colnames(X) <- c('1','commerce')
head(X)
```

```
##   1 commerce
## 1 1     13.8
## 2 1     20.4
## 3 1     27.6
## 4 1     18.6
## 5 1     17.2
## 6 1     21.5
```

- Use `lm()` to fit the rebellion intensity to matrix X (which has columns for a constant and commercialization). Make sure you only calculate the coefficient for the intercept once.

```
m2.1 <- lm(t ~ X)
m2.1$coefficients
```

```
## (Intercept)          X1   Xcommerce
##  -2.8058274          NA   0.1066466
```

- Using only matrix operations, calculate and show the coefficient estimates. Verify that they match the estimates from `lm()`.

```
w_2.1hat <- solve( t(X) %*% X) %*% t(X) %*% matrix(t)
w_2.1hat # They match!
```

```
##                [,1]
## 1        -2.8058274
## commerce  0.1066466
```

- Create another matrix (call it X_ct, the ct is for commerce and tradition) with three columns: a constant, variable commerce, variable tradition. Print the head of this matrix.

```
t <- chirot_mat[, 1, drop = FALSE]  # response, keep as a matrix
a <- chirot_mat[, 2, drop = FALSE]  # commerce column, keep as matrix
b <- chirot_mat[, 3, drop = FALSE]  # tradition column, keep as matrix
X_ct <- cbind(1, a, b)
colnames(X_ct) <- c('1','commerce', 'tradition')
head(X_ct)
```

```
##   1 commerce tradition
```

```
## 1 1     13.8      86.2
## 2 1     20.4      86.7
## 3 1     27.6      79.3
## 4 1     18.6      90.1
## 5 1     17.2      84.5
## 6 1     21.5      81.5
```

- Using matrix operations, calculate and show the coefficient estimates of the model with the variables commerce and tradition.

```
w_2.2hat <- solve( t(X_ct) %*% X_ct) %*% t(X_ct) %*% matrix(t)
w_2.2hat
```

```
##                   [,1]
## 1         -12.61148785
## commerce    0.09522408
## tradition   0.11992396
```

- Create another matrix (call it X_all) with all of the x variables (plus a constant). Print the head of this matrix. Using matrix operations, calculate and show the coefficient estimates of the model with all the variables.

```
t <- chirot_mat[, 1, drop = FALSE]  # response, keep as a matrix
a <- chirot_mat[, 2, drop = FALSE]  # commerce column, keep as matrix
b <- chirot_mat[, 3, drop = FALSE]  # tradition column, keep as matrix
c <- chirot_mat[, 4, drop = FALSE]  # midpeasant column, keep as matrix
d <- chirot_mat[, 5, drop = FALSE]  # inequality column, keep as matrix
X_all <- cbind(1, a, b, c, d)
colnames(X_all) <- c('1','commerce', 'tradition', 'midpeasant', 'inequality')
head(X_all)
```

```
##   1 commerce tradition midpeasant inequality
## 1 1     13.8      86.2        6.2       0.60
## 2 1     20.4      86.7        2.9       0.72
## 3 1     27.6      79.3       16.9       0.66
## 4 1     18.6      90.1        3.4       0.74
## 5 1     17.2      84.5        9.0       0.70
## 6 1     21.5      81.5        5.2       0.60
```

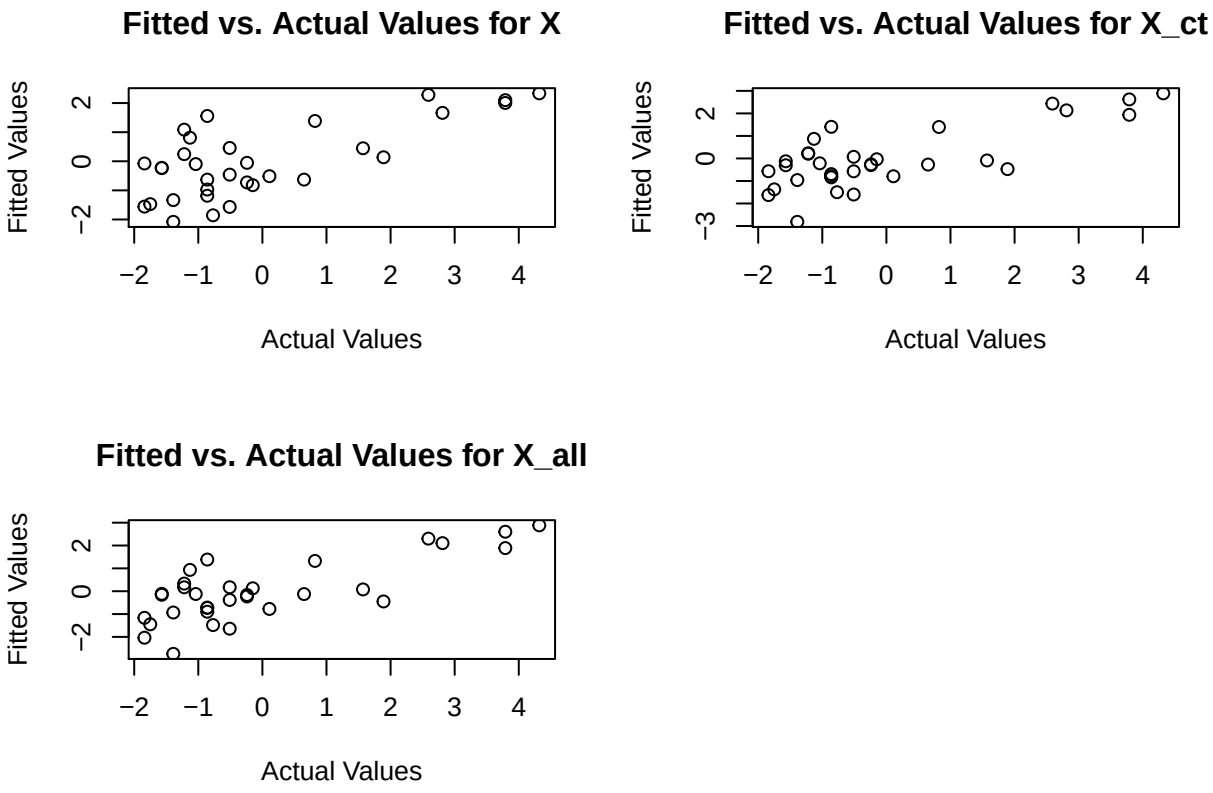
```
w_2.3hat <- solve( t(X_all) %*% X_all) %*% t(X_all) %*% matrix(t)
w_2.3hat
```

```
##                     [,1]
## 1          -12.919017967
## commerce     0.091140456
## tradition    0.116786574
## midpeasant  -0.003341668
## inequality   1.137969619
```

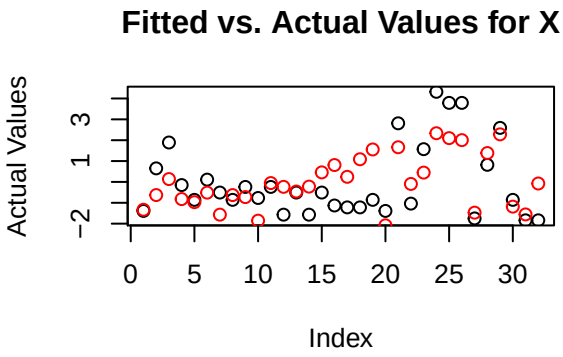
- Using matrix operations, calculate the fitted values for all three models. (No need to print out the fitted values.) Create plots of the fitted values vs the actual values for each model (there will be three plots). Be sure to provide a title for each plot.

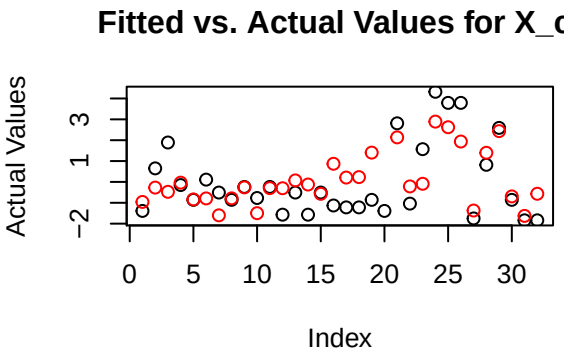
```
# Our Fitted Values
t_2.1hat <- X %*% w_2.1hat
t_2.2hat <- X_ct %*% w_2.2hat
t_2.3hat <- X_all %*% w_2.3hat
```

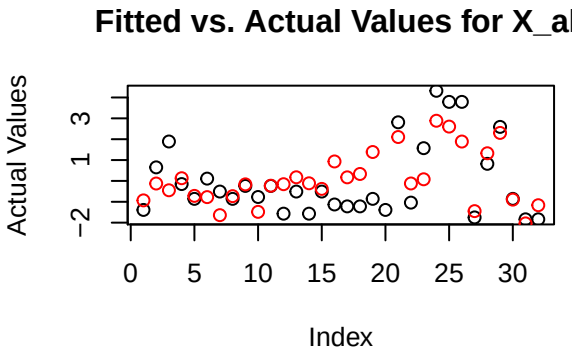
```r
# I used two ways of showing the Fitted vs Actual Values but still used 3 plots:
par(mfrow = c(2, 2))
plot(t, t_2.1hat, main = "Fitted vs. Actual Values for X",
     xlab = "Actual Values", ylab = "Fitted Values")
plot(t, t_2.2hat, main = "Fitted vs. Actual Values for X_ct",
     xlab = "Actual Values", ylab = "Fitted Values")
plot(t, t_2.3hat, main = "Fitted vs. Actual Values for X_all",
     xlab = "Actual Values", ylab = "Fitted Values")

# Black points are Actual Values
# Red points are Fitted Values
par(mfrow = c(2, 2))
```




```r
plot(t, main = "Fitted vs. Actual Values for X", ylab = "Actual Values")
points(t_2.1hat, col = "red")
plot(t,main = "Fitted vs. Actual Values for X_ct", ylab = "Actual Values")
points(t_2.2hat, col = "red")
plot(t,main = "Fitted vs. Actual Values for X_all", ylab = "Actual Values")
points(t_2.3hat, col = "red")
```







- Now that you have calculated the columns of fitted values, find the Residual Sum of Squares and the R-squared values of the three models. Which model has the smallest RSS? (RSS is the sum of (actual - fitted)^2. R-sq is the correlation between actual and fitted, squared.)

```
RSS_X <- sum((t - t_2.1hat)^2)
RSS_X_ct <- sum((t - t_2.2hat)^2)
RSS_X_all <- sum((t - t_2.3hat)^2)

R2_X <- cor(t, t_2.1hat)^2
R2_X_ct <- cor(t, t_2.2hat)^2
R2_X_all <- cor(t, t_2.3hat)^2

RSS_X
```

```
## [1] 47.5052
```

```
RSS_X_ct
```

```
## [1] 41.43137
```

```
RSS_X_all
```

```
## [1] 40.62534
```

```
R2_X
```

```
##                [,1]
## intensity 0.5131222
```

```
R2_X_ct
```

```
##                [,1]
## intensity 0.5753725
```

```
R2_X_all
```

```
##                [,1]
## intensity 0.5836334
```

The smallest RSS is RSS_X_all, where it has all the variables tested. It would be the 3rd model with the lowest RSS.

Part 3: Cross-Validation

The `ironslag` data set, and the linear model predicting the results of the chemical test based on the result from the easier magnetic test.

```
library(DAAG)
```

```
## Warning: package 'DAAG' was built under R version 3.6.3
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 3.6.3
```

```
x <- seq(10,40, .1)
```

Task 3A

Create a plot showing the fitted line for each of the following models:

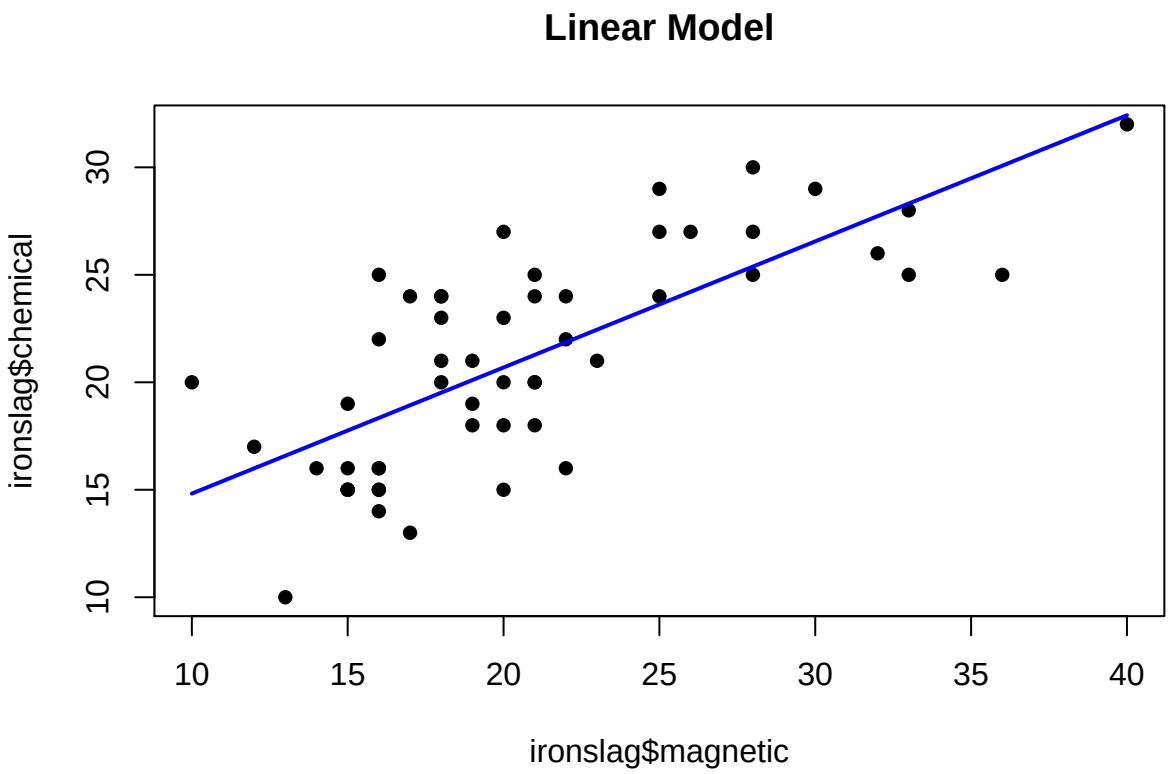
Linear: $y_n = w_0 + w_1 x_n + \epsilon_n$.

Quadratic: $y_n = w_0 + w_1 x_n + w_2 x_n^2 + \epsilon_n$

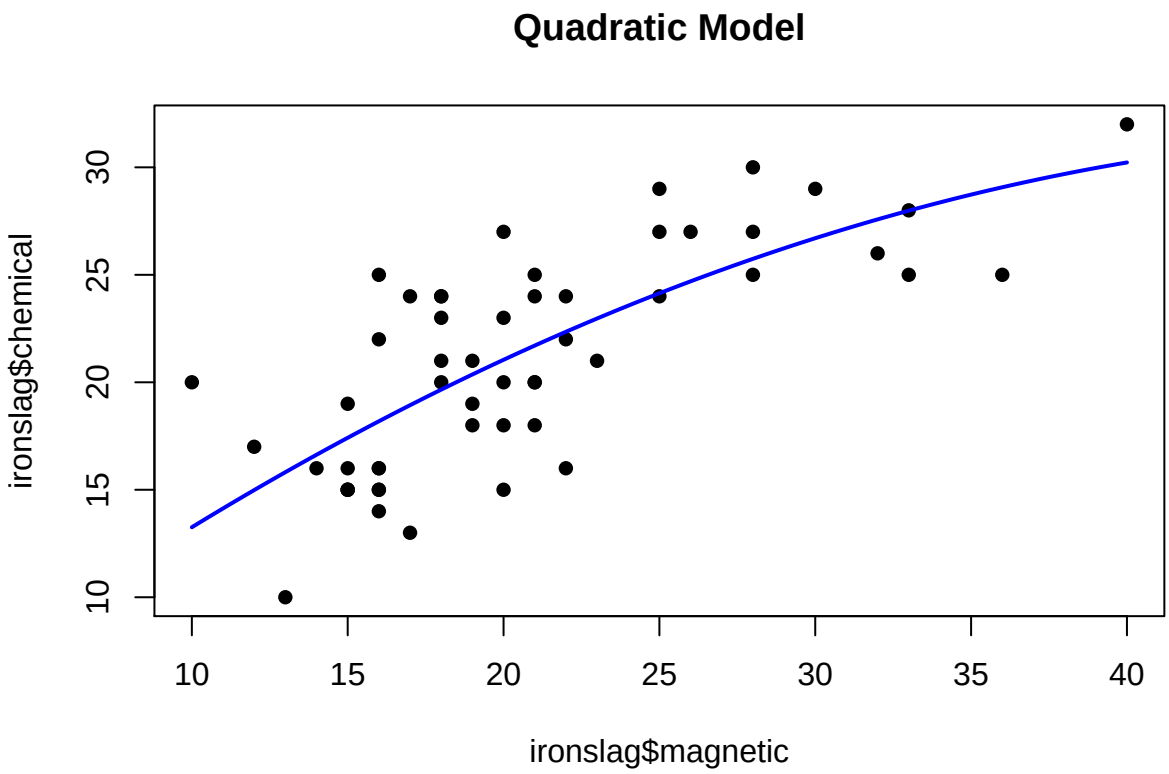
Exponential: $\log(y_n) = w_0 + w_1 x_n + \epsilon_n$, equivalent to $y_n = \exp(w_0 + w_1 x_n + \epsilon_n)$

log-log: $\log(y_n) = w_0 + w_1 \log(x_n) + \epsilon_n$

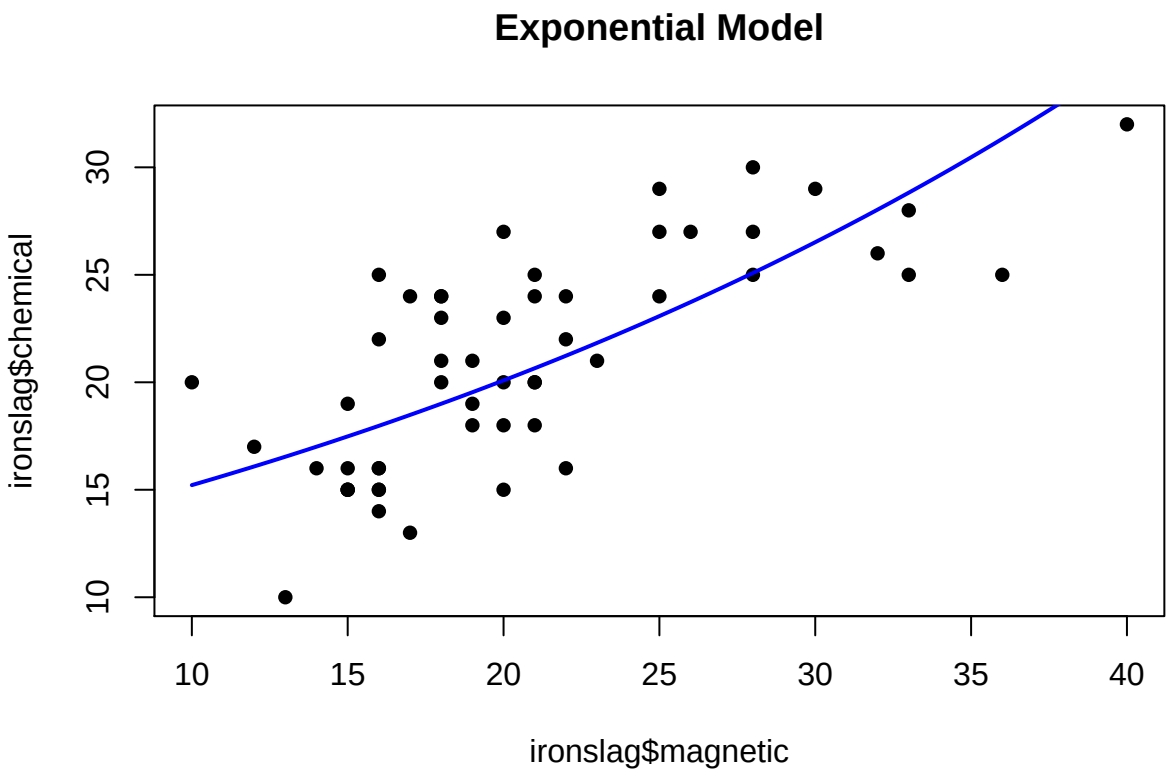
```
# Standard Linear Model
L1 <- lm(chemical ~ magnetic, data = ironslag)
plot(ironslag$magnetic, ironslag$chemical, main = "Linear Model", pch = 16)
yhat1 <- L1$coef[1] + L1$coef[2] * x
lines(x, yhat1, lwd = 2, col = "blue")
```



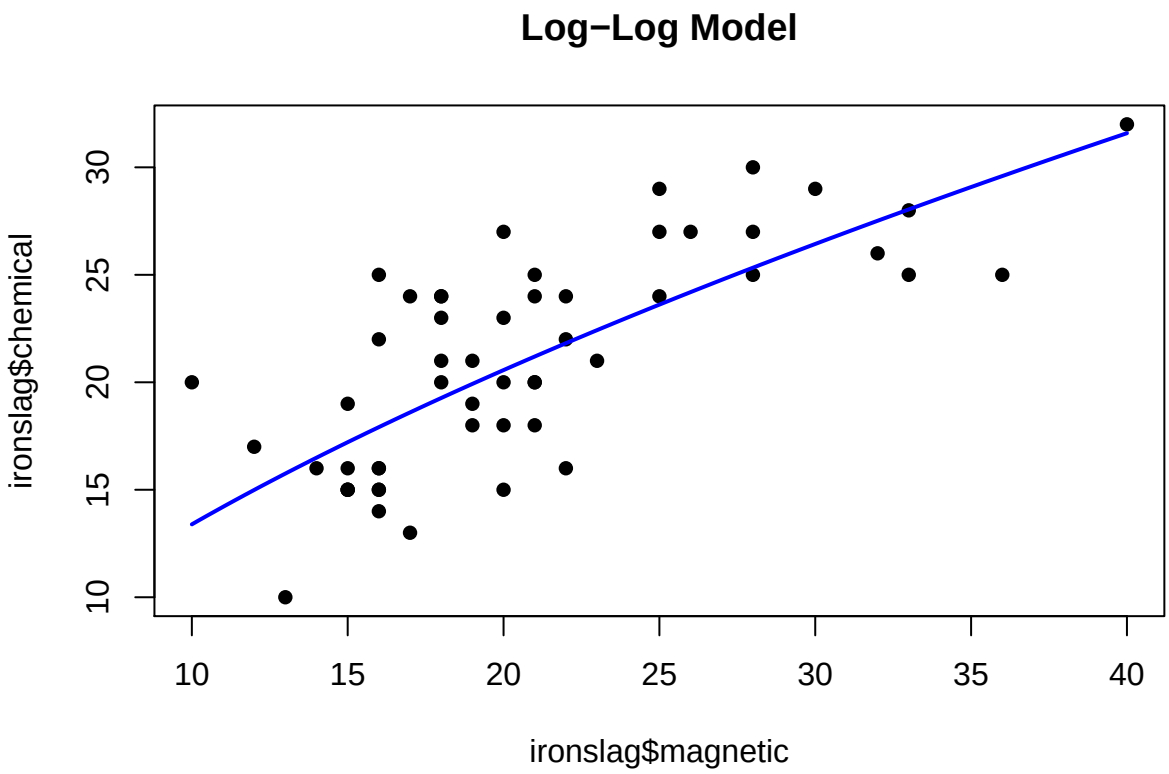

```
# Quadratic Model
L2 <- lm(chemical ~ magnetic + I(magnetic^2), data = ironslag)
plot(ironslag$magnetic, ironslag$chemical, main = "Quadratic Model", pch = 16)
yhat2 <- L2$coef[1] + L2$coef[2] * x + L2$coefficients[3]*x^2 # must take power to 2
lines(x, yhat2, lwd = 2, col = "blue")
```




```
# Exponential Model
L3 <- lm(log(chemical) ~ magnetic, data = ironslag)
plot(ironslag$magnetic, ironslag$chemical, main = "Exponential Model", pch = 16)
yhat3 <- L3$coef[1] + L3$coef[2] * x
lines(x, exp(yhat3), lwd = 2, col = "blue") # must take exp() of our y_hat
```




```r
# Log-Log Model
L4 <- lm(log(chemical) ~ log(magnetic), data = ironslag)
plot(ironslag$magnetic, ironslag$chemical, main = "Log-Log Model", pch = 16)
yhat4 <- L4$coef[1] + L4$coef[2] * log(x) # must take log() of our Predictor
lines(x, exp(yhat4), lwd = 2, col = "blue") # must take exp() of our y_hat
```



Task 3B: Leave-one-out Cross validation

Code leave-one-out cross validation. Follow the guide and hints provided.

```
error_model1 <- rep(NA, nrow(ironslag))
error_model2 <- rep(NA, nrow(ironslag))
error_model3 <- rep(NA, nrow(ironslag))
error_model4 <- rep(NA, nrow(ironslag))

for(i in seq_len(nrow(ironslag))){
  valid <- ironslag[i, ]
  train <- ironslag[-i, ]
  # m1 is our Linear Model
  m1 <- lm(chemical ~ magnetic, data = train)
  fitted1 <- predict(m1, valid)
  error_model1[i] <- (valid[, 1] - fitted1)^2
  # m2 is our Quadratic Model
  m2 <- lm(chemical ~ magnetic + I(magnetic^2), data = train)
  fitted2 <- predict(m2, valid)
  error_model2[i] <- (valid[, 1] - fitted2)^2
  # m3 is our Exponential Model
  m3 <- lm(log(chemical) ~ magnetic, data = train)
  fitted3 <- predict(m3, valid)
  error_model3[i] <- (valid[, 1] - exp(fitted3))^2 # have to take exp() of fitted
  # m4 is our Log-Log Model
  m4 <- lm(log(chemical) ~ log(magnetic), data = train)
```

```
  fitted4 <- predict(m4, valid)
  error_model4[i] <- (valid[, 1] - exp(fitted4))^2 # have to take exp() of fitted
}

mean(error_model1)
```

```
## [1] 12.44774
```

```
mean(error_model2)
```

```
## [1] 13.02726
```

```
mean(error_model3)
```

```
## [1] 13.58675
```

```
mean(error_model4)
```

```
## [1] 12.47131
```

Compare the sizes of the cross validation error to help you decide which model does the best job of predicting the test cases.

Model1 does the best job at predicting the test cases as it has the lowest MSE of 12.44774.

Task 3C: Cross-validation with R

Use `glm()` to fit a model and `cv.glm()` to perform LOOCV.

```
library(boot)
```

```
## Warning: package 'boot' was built under R version 3.6.3
```

```
##
## Attaching package: 'boot'
```

```
## The following object is masked from 'package:lattice':
##
##     melanoma
```

```
gL1 <- glm(chemical ~ magnetic, data = ironslag)
loocv1 <- cv.glm(ironslag, gL1)$delta[1]

gL2 <- glm(chemical ~ magnetic + I(magnetic^2), data = ironslag)
loocv2 <- cv.glm(ironslag, gL2)$delta[1]

# At this point, we have to start using the cost() function.
# Create our cost() function:
cost <- function(y, yhat) (exp(y) - exp(yhat))^2

gL3 <- glm(log(chemical) ~ magnetic, data = ironslag)
loocv3 <- cv.glm(ironslag, gL3, cost)$delta[1]

gL4 <- glm(log(chemical) ~ log(magnetic), data = ironslag)
loocv4 <- cv.glm(ironslag, gL4, cost)$delta[1]

loocv1
```

```
## [1] 12.44774
```

```
loocv2
```

```
## [1] 13.02726
```

```
loocv3
```

```
## [1] 13.58675
```

```
loocv4
```

```
## [1] 12.47131
```

Your LOOCV estimates from `cv.glm()` should match your estimates when you coded your algorithm from scratch.

Based on your Cross Validation scores, which model seems to be the best?

Model1 does the best job at predicting the test cases as it has the lowest LOOCV of 12.44774.

Task 3D: Revisit Chirot

Use `glm()` to fit the three models to the Chirot data.

- model1: intensity ~ commerce
- model2: intensity ~ commerce + tradition
- model3: intensity ~ all other variables

Use `cv.glm()` to calculate the LOOCV scores to compare models. Which model would you select?

```
attach(Chirot)

# model1 --> X
gL5 <- glm(intensity ~ commerce, data = Chirot)
loocv5 <- cv.glm(Chirot, gL5)$delta[1]

# model2 --> X_ct
gL6 <- glm(intensity ~ commerce + tradition, data = Chirot)
loocv6 <- cv.glm(Chirot, gL6)$delta[1]

# model3 --> X_all
gL7 <- glm(intensity ~ commerce + tradition + midpeasant + inequality,
           data = Chirot)
loocv7 <- cv.glm(Chirot, gL7)$delta[1]

loocv5
```

```
## [1] 1.714944
```

```
loocv6
```

```
## [1] 1.591027
```

```
loocv7
```

```
## [1] 1.693952
```

I would choose model X_ct (or *Model2* in this case) because it has the lowest LOOCV of 1.591027.

Part 4: Using R's `optim()` function

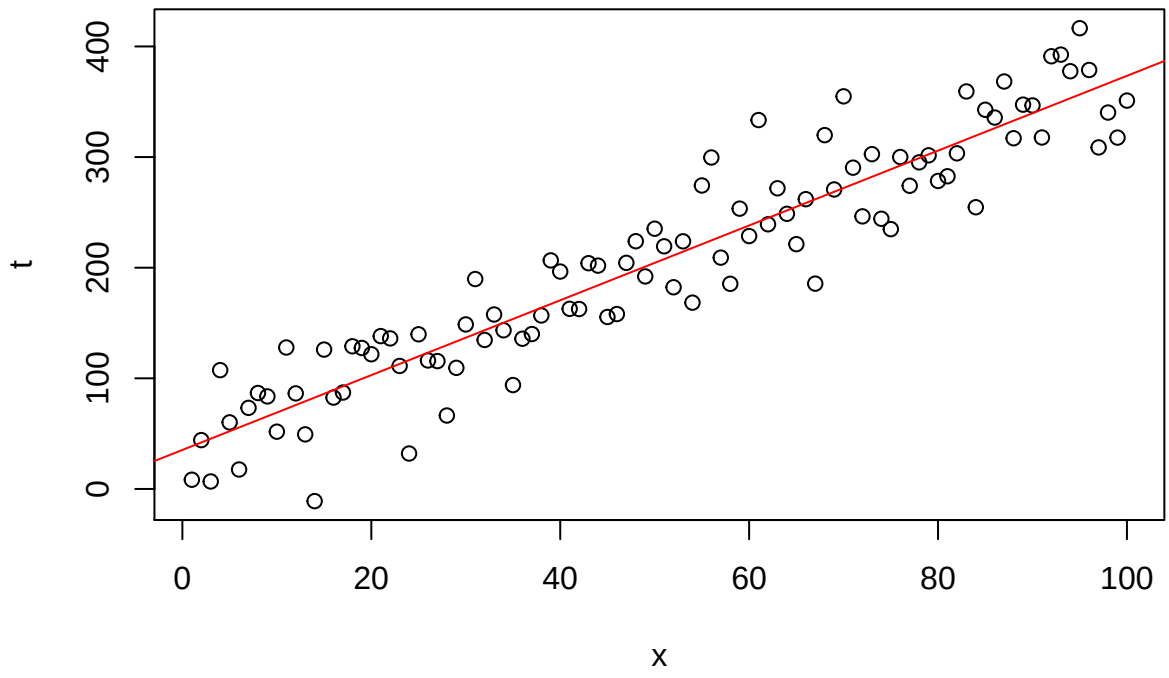
Synthetic data. Intercept of the population is 30. Slope of the population is 3.4.

Based on the provided sample, the OLS estimates are 35.266 and 3.382 for intercept and slope.

```
x <- 1:100
set.seed(1)
e <- rnorm(100, 0, 40)
t <- 30 + 3.4 * x + e
coefs <- coef(lm(t ~ x))
print(coefs)  # if you use set.seed(1), this should be 35.266629, 3.381958
```

```
## (Intercept)           x
##   35.266629    3.381958
```

```
plot(x, t)
abline(coef = coefs, col = "red")
```



Task 4: Use `optim()` to minimize the sum of squared residuals in linear regression

Write your loss function as a function of the vector `par` that will return a single value: the sum of squared residuals.

```
loss <- function(par) {
  t_hat <- par[1] + par[2]*x
  rss <- sum((t - t_hat)^2)
  return(rss)
}
```

The following chunk runs optim and returns the results. The arguments it takes in is the initial values of `par`, and the function you wrote which it will try to minimize. I have specified the method 'BFGS' which

uses a version of gradient descent as the numeric algorithm to optimize the function.

```
results <- optim(par = c(0,1), fn = loss, method = 'BFGS')
results$par # They match!
```

```
## [1] 35.266629  3.381958
```

Look at the results and the values of the parameters that minimize the loss function, they should match the coefficients estimated by `lm()`.