Statistical Analysis, MSCA 31007, Lecture 4

Hope Foster-Reyes

November 6, 2016

Contents

Analysis of Residuals of a Linear Model

Understand analysis of residuals of the estimated linear model.

Notes

- *Style Guide:* https://google.github.io/styleguide/Rguide.xml
- *Packages required: note*
- *Files required: ResidualAnalysisProjectData_1.csv; store in RStudio project directory*

1 Data

Import and plot the sample data.

```
# Import linear model ('lm') data
sample.data <- read.csv("ResidualAnalysisProjectData_1.csv")
```

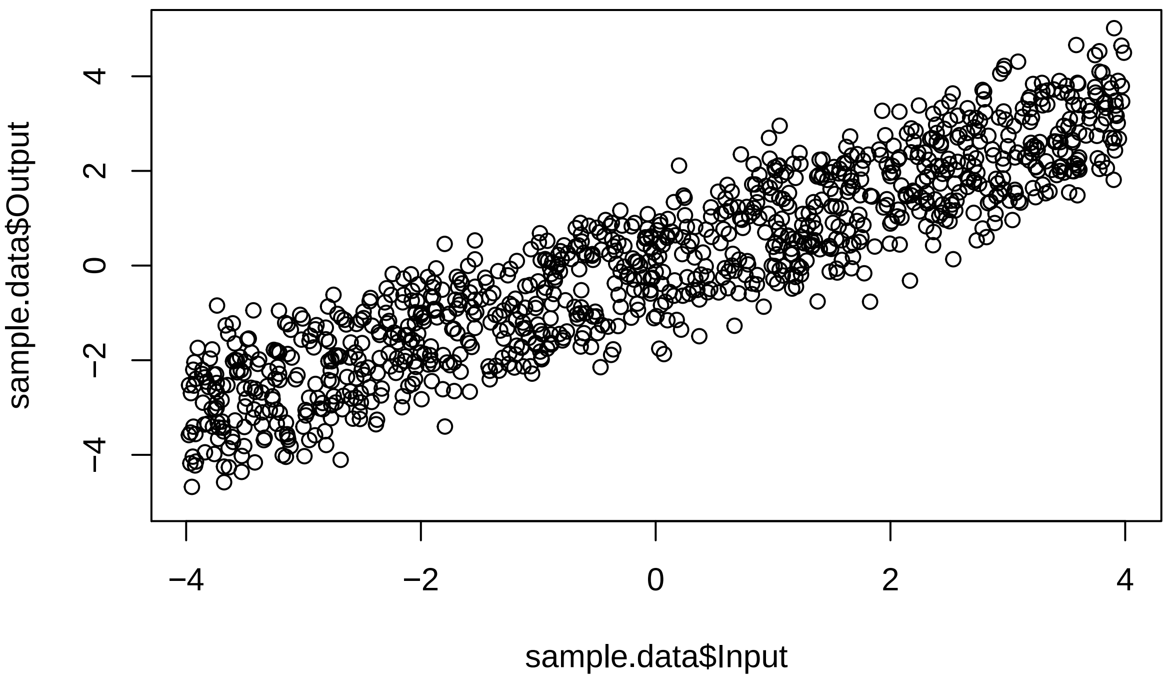
Look at the sample in the file LinearModelCase1.csv. The first rows and the X-Y plot are:

```
head(sample.data)
```

```
##         Input    Output
## 1  3.6664327  2.747905
## 2 -2.5194424 -3.242035
## 3  0.6475581  1.559734
## 4  2.4439621  1.292082
## 5  1.9921334  1.958417
## 6  1.7534556  2.049381
```

Plot the data.

```
plot(sample.data$Input, sample.data$Output, ylim = c(-5, 5))
```



2 Fitting the Linear Model

Estimate a linear model using function `lm()`. Examine the output of the function.

```
lm.estimate <- lm(Output ~ Input, data = sample.data)
names(lm.estimate)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

2.1 Object `lm()`

Explore the elements of the object `lm`:

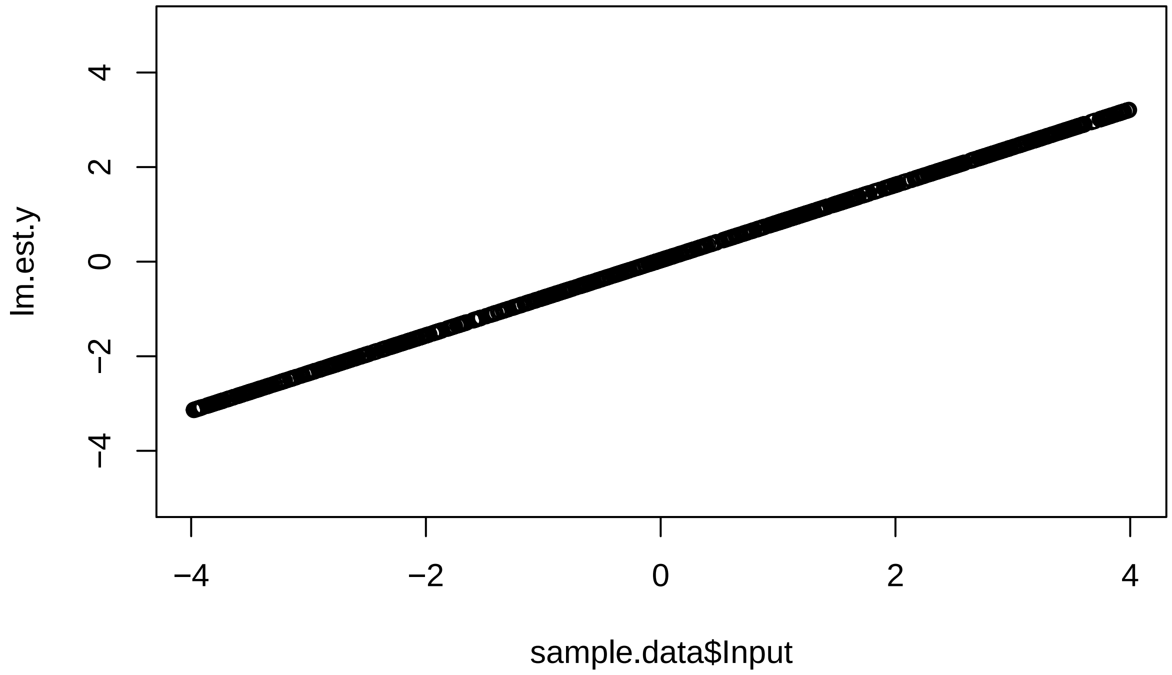
1. Coefficients

The coefficients are our estimate of the slope and intercept of the estimated linear model. We can output our results from our `lm.estimate` object. If we plot these as slope and intercept (while maintaining the same axis), we can compare our plot to the plot above of the data provided:

```
lm.estimate$coefficients
```

```
## (Intercept)       Input
##  0.03160231  0.79627673
```

```
lm.est.b <- lm.estimate$coefficients[1]
lm.est.a <- lm.estimate$coefficients[2]

lm.est.y <- lm.est.a * sample.data$Input + lm.est.b
plot(sample.data$Input, lm.est.y, ylim = c(-5, 5))
```



2. Residuals (make a plot). How are the residuals calculated?

We calculate the residuals by subtracting our predicted linear Y values from the actual Y values in our sample, giving us the "error" or randomness in Y. We will refer to these residuals as Epsilon or 'ε as per the following equation:

$Y = aX + b + \varepsilon$

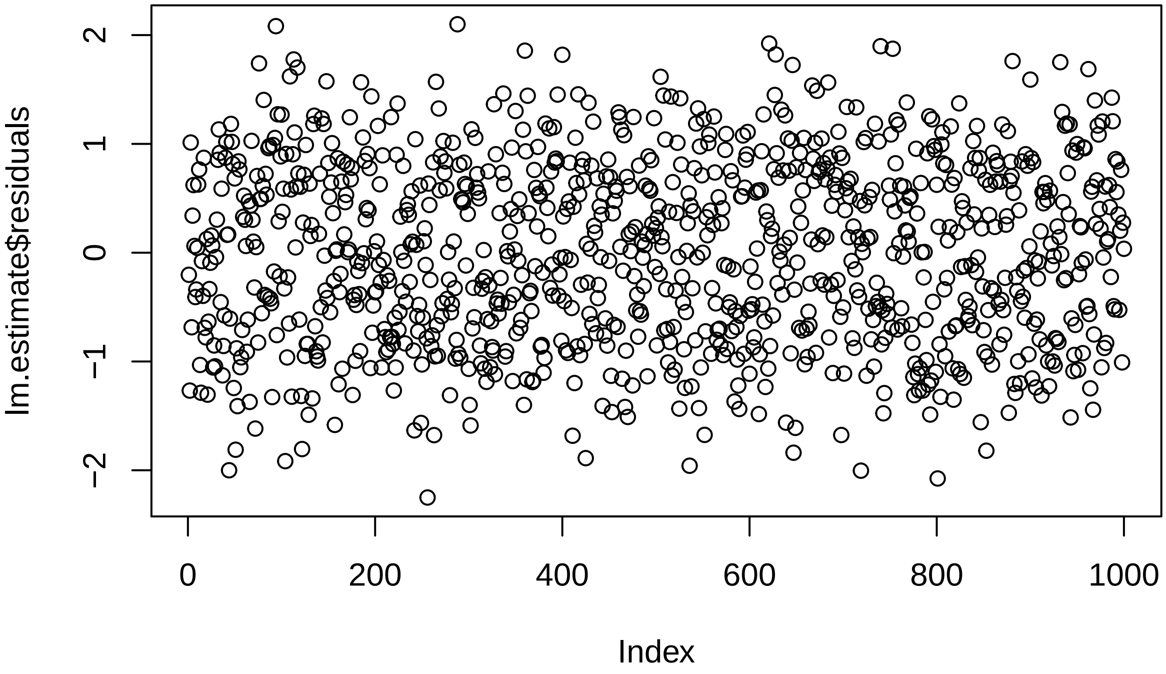
```
lm.est.eps <- sample.data$Output - lm.est.y
# The manual calculation and lm() output are nearly identical
(mean(lm.estimate$residuals - lm.est.eps))
```

```
## [1] -1.802421e-16
```

```
(sum(lm.estimate$residuals - lm.est.eps))
```

```
## [1] -1.802421e-13
```

```
plot(lm.estimate$residuals)
```



3. Investigate: What are `fitted.values` in our `lm()` output?

We might suspect, from the name, that "fitted values" refers to some sort of estimate on Y. Considering that the goal of `lm()` is to fit our data to a linear model, could it be that the fitted values are the pure values of Y from the linear relationship, without residuals? Let's test this.

```
head(lm.estimate$fitted.values)
```

```
##          1          2          3          4          5          6 
##  2.9510974 -1.9745711  0.5472377  1.9776724  1.6178918  1.4278382
```

```
head(lm.est.y)
```

```
## [1]  2.9510974 -1.9745711  0.5472377  1.9776724  1.6178918  1.4278382
```

```
(all.equal(as.vector(lm.estimate$fitted.values), lm.est.y))
```

```
## [1] TRUE
```

We have confirmed that the `lm()` output of `fitted.values` represents the sum of the intercept coefficient and the input (or predictor) variable multiplied by the slope coefficient, in other words our resultant output or response variable of our linear function based on our estimated coefficients, without residuals.

2. The `summary()` Output

Look at the summary.

```
(lm.est.summary <- summary(lm.estimate))
```

```
##
## Call:
## lm(formula = Output ~ Input, data = sample.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.25025 -0.68362  0.01354  0.66505  2.09946
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.03160    0.02653   1.191    0.234
## Input        0.79628    0.01138  69.993   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8389 on 998 degrees of freedom
## Multiple R-squared:  0.8308, Adjusted R-squared:  0.8306
## F-statistic:  4899 on 1 and 998 DF,  p-value: < 2.2e-16
```

Let's interpret the summary.

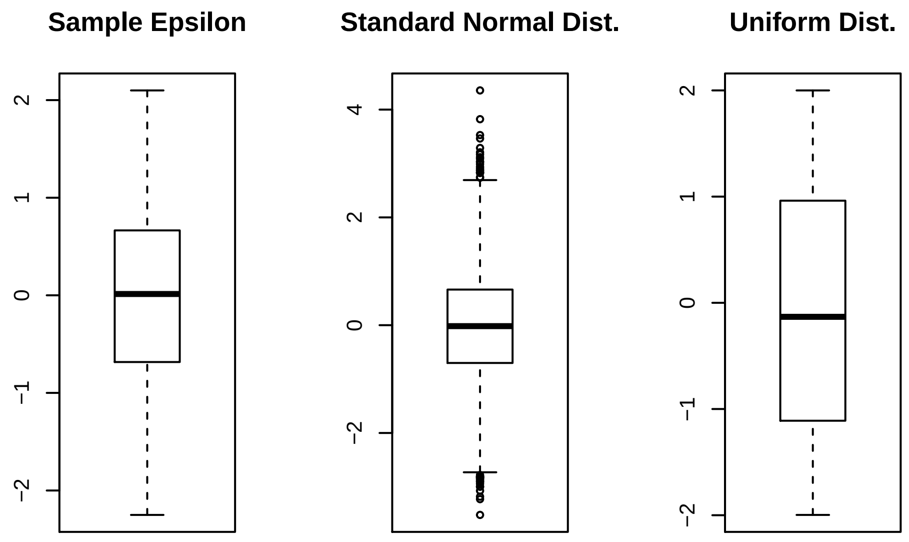
Call

The `Call` section simply reminds us of our call to the function, which provides the data and specifies which variable is our independent (predictor or input variable) and which is our dependent (response or output variable). Note that we denote this by first listing our dependent variable, followed by a tilde symbol, and finally our independent variable.

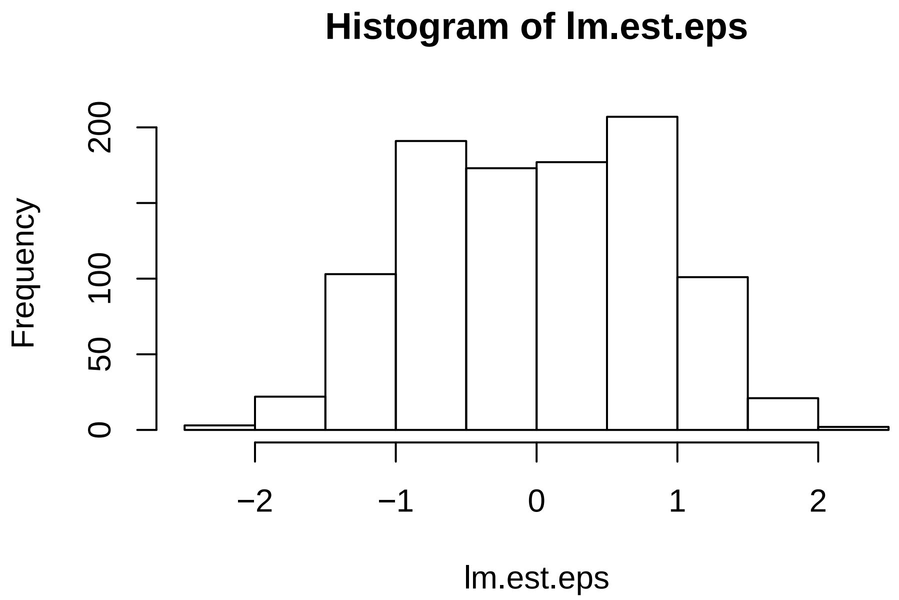
Residuals

This is followed by the `Residuals` section which provides the five number summary of the residuals. What may be interesting to us here is that the summary is nearly symmetrical, and the quartiles seem to divide the data nearly evenly, as opposed to the 1Q and 3Q values being much closer to the Median as you might expect in a normal distribution. A quick boxplot confirms that the shape, while not uniform, appears to be more uniform than a normal distribution:

```
par(mfrow=c(1,3))
boxplot(lm.est.eps, main="Sample Epsilon")
boxplot(rnorm(5000), main="Standard Normal Dist.")
boxplot(runif(500, -2, 2), main="Uniform Dist.")
```



```r
par(mfrow=c(1,1))

hist(lm.est.eps)
```



Coefficients

Next, the `Coefficients` section lists our estimated linear coefficients. The first will always be our Intercept (which, essentially, is a coefficient to the number 1), followed by the coefficients of our remaining predictor variables. In this case there is only one predictor variable, which we labeled Input. Our Input coefficient, is positive, leading to a positive correlation between Input (X) and Output (Y), which is also demonstrated by our plots above.

The `Pr(>|t|)` column of the Coefficients section tells us the probability of obtaining our data if the null hypothesis were true, in this case if the coefficients were actually zero. In our case the p-value of a or our Input coefficient is quite low and statistically significant. We can be less confident of our intercept, with a p-value of 0.234.

Model

It is worth noting that R documentation states that the term "residual standard error" that we see in our `summary` output is a misnomer, and the proper term is residual standard deviation.

Our summary output includes this calculation of the residual standard deviation, or σ as described below, as well as R-squared (also notated as r^2 or ρ^2) which is our "coefficient of determination" or "squared correlation coefficient".

Whereas the correlation coefficient, "r" measures the strength and the direction of a linear relationship between two variables, the square of the correlation coefficient, "r squared", which we see in our summary, measures the proportion of our data that is explained by the linear relationship. Thus r-squared can be considered one estimate of the "predictive power" of our model.

In this case roughly 83% of the variation in our Output is explained by our linear model of:

$Output = 0.796 \cdot Input + 0.032$

The p-value of the model itself is found in the F-statistic section. Our p-value is quite low indicating statistical significance in our linear model.

Thus our model explains roughtly 83% of our variation and has a statistically significant relationship between Input and Output, however our intercept estimate is not statistically significant. In other words, we are quite confident of the slope of our line, yet we're not so sure about its intercept.

The wide gap in p-value between slope and intercept is worth further investigation.

What is `summary(lm.est)$sigma`?

```
names(lm.est.summary)
```

```
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

```
lm.est.summary$sigma
```

```
## [1] 0.838893
```

```
lm.est.summary$sigma^2
```

```
## [1] 0.7037415
```

In this case "sigma" refers to the standard deviation of our residuals, also known as the residual standard deviation. It describes the amount that the residuals vary or spread from their mean, which is theoretically zero, and hence the amount that our output variable varies or spreads from a pure linear relationship with our input variable(s). For a simple Gaussian residual, this would be the σ value in the below equation:

$Y = aX + b + \varepsilon$

$\varepsilon \sim Norm(0, \sigma)$

And finally we'll test calculation of sigma, the residual standard deviation:

Check how `sigma` is calculated in the summary object by reproducing the square of it:

1. Using `var()`

```
sample.n <- nrow(sample.data)
lm.sigma.sq.byvar <- var(lm.estimate$residuals)

# Convert degrees of freedom
lm.sigma.sq.byvar = lm.sigma.sq.byvar * (sample.n - 1) / lm.estimate$df.residual
```

2. Using only `sum()`

```
lm.eps.deviation <- lm.estimate$residuals - mean(lm.estimate$residuals)
lm.sigma.sq.bysum <- sum(lm.eps.deviation^2) / lm.estimate$df.residual
```

Compare the two calculations with the `summary` output:

```
c("By Var()" = lm.sigma.sq.byvar,
  "By Sum()" = lm.sigma.sq.bysum,
  "From Model" = lm.est.summary$sigma^2)
```

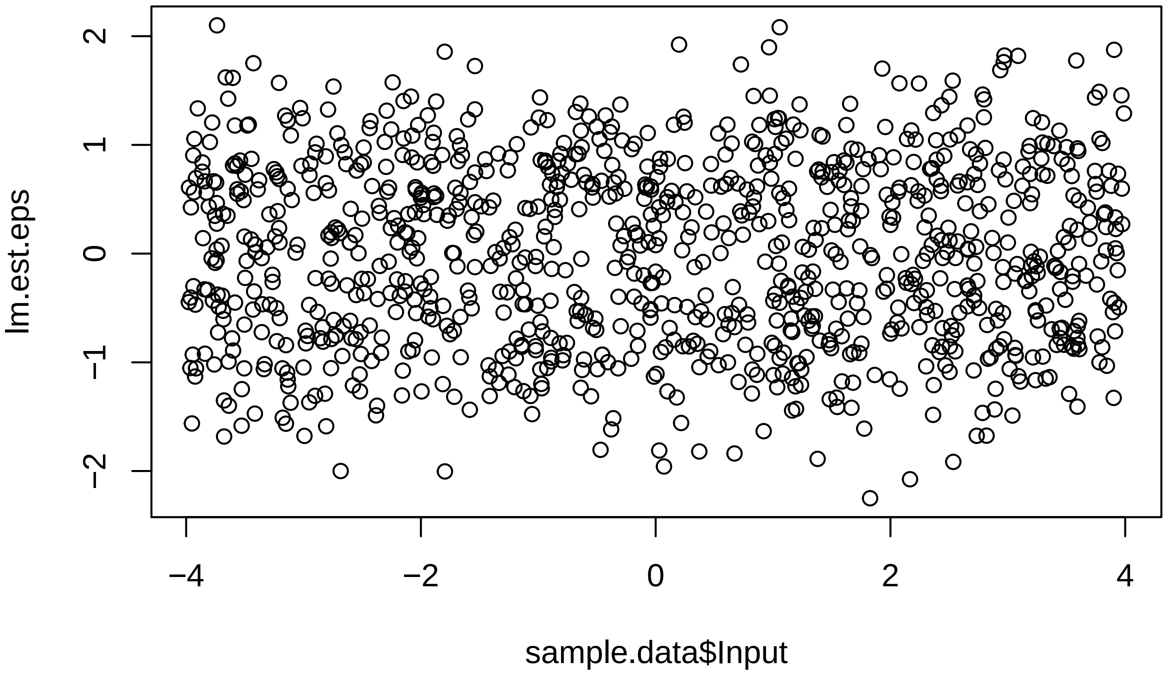
```
##   By Var()   By Sum() From Model
##  0.7037415  0.7037415  0.7037415
```

3 Analysis of Residuals

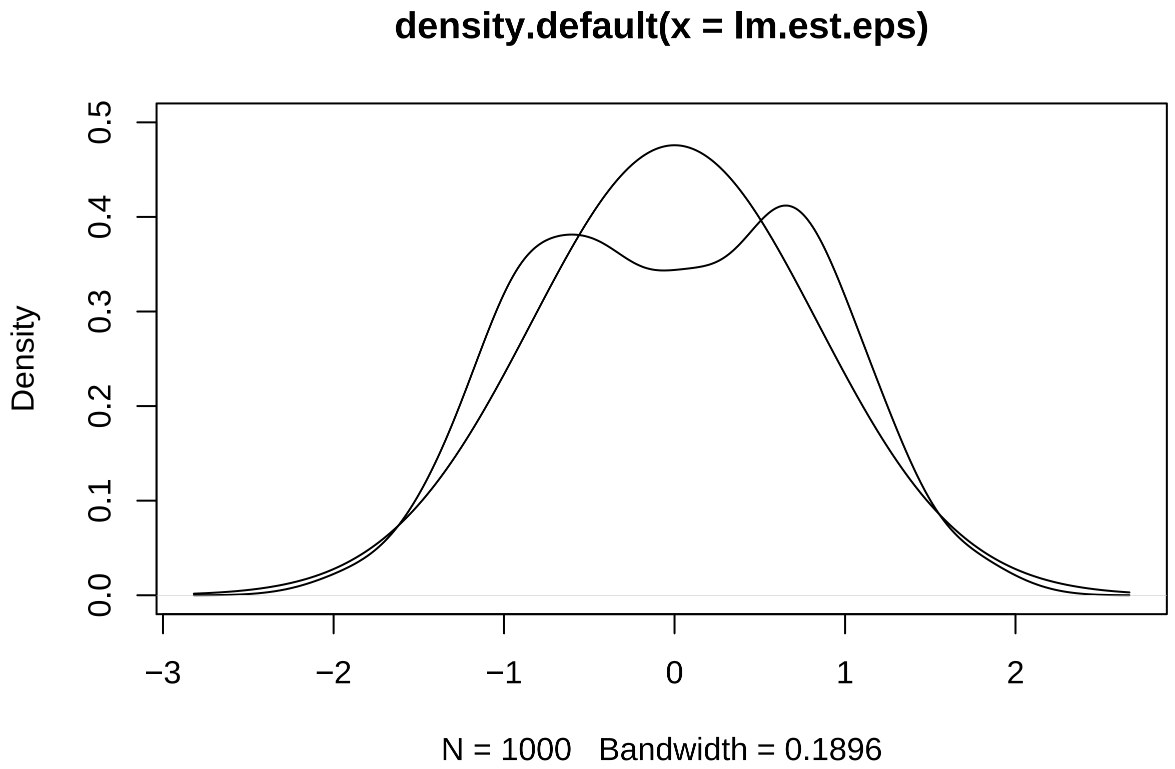
3.1 Residuals of the model

Observe the residuals, plot them against the input. Also plot their probability density in comparison with the normal density.

```
lm.est.eps <- lm.estimate$residuals
plot(sample.data$Input, lm.est.eps)
```



```
lm.est.eps.density <- density(lm.est.eps)
plot(lm.est.eps.density, ylim = c(0, 0.5))
lines(lm.est.eps.density$x,
      dnorm(lm.est.eps.density$x, mean = mean(lm.est.eps), sd = sd(lm.est.eps)))
```



What do you conclude from the analysis of residuals?

An observation of the density of the residuals is multi-modal and seems to indicate that we may be looking at two different normal distributions, rather than one.

3.2 Clustering the Sample

Calculate the mean values of negative residuals and positive residuals.

```
c(Left.Mean = mean(lm.est.eps[lm.est.eps < 0]),
  Right.Mean = mean(lm.est.eps[lm.est.eps > 0]))
```

```
##  Left.Mean Right.Mean
## -0.7241664  0.7013580
```

Separate the given sample into 2 subsamples:

- One in which the residuals are below zero
- One in which they are above zero

Create a selection variable which estimates switching between the two subsamples (a value of 1 in this variable corresponds to the positive residual case, and a value of 0 corresponds to the negative residual case).

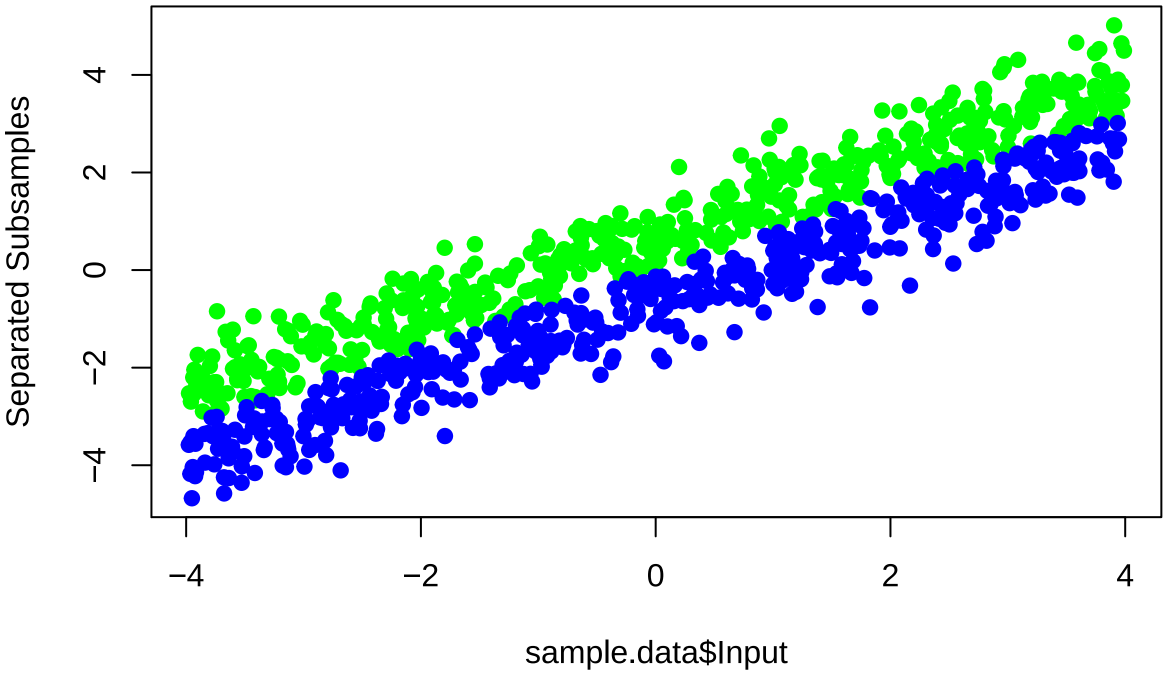
```
select.seq.unscrambled <- as.integer(lm.est.eps > 0)
head(select.seq.unscrambled, 30)
```

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

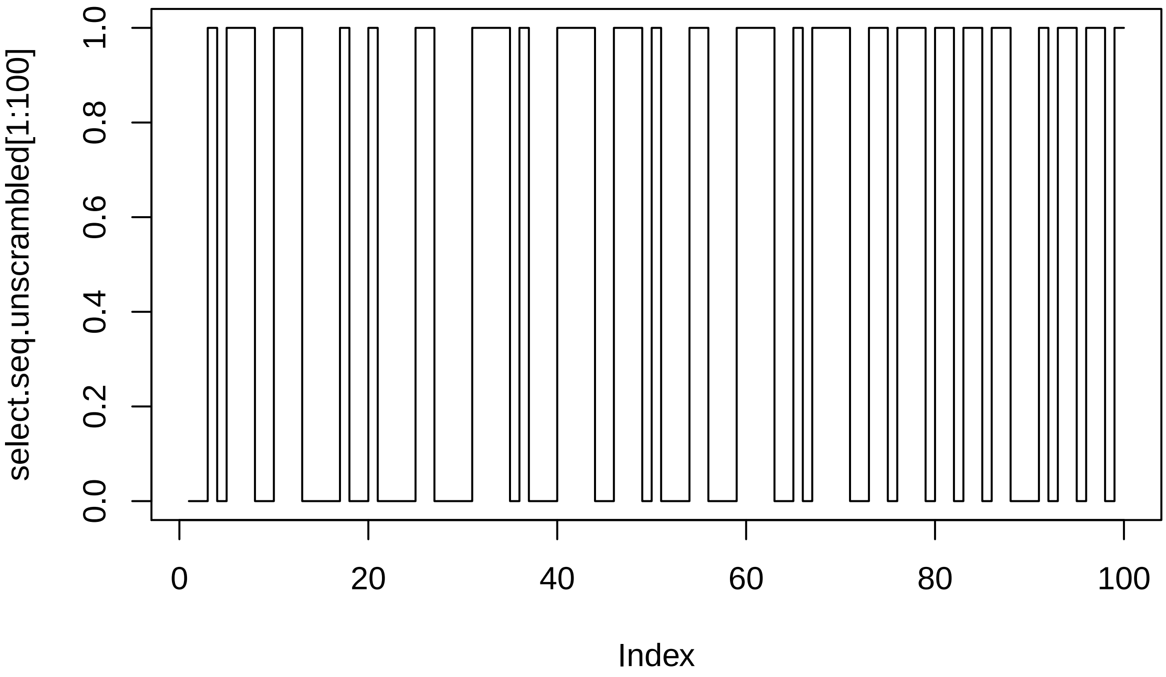
```r
ChoosePos <- function(j) {
  vapply(seq_along(select.seq.unscrambled),
         function(i) {switch(select.seq.unscrambled[i] + 1, NA, sample.data[i, j])},
         numeric(1))
}

ChooseNeg <- function(j) {
  vapply(seq_along(select.seq.unscrambled),
         function(i) {switch(select.seq.unscrambled[i] + 1, sample.data[i, j], NA)},
         numeric(1))
}

lm.subset1 <- sapply(seq_along(sample.data), ChoosePos)
lm.subset2 <- sapply(seq_along(sample.data), ChooseNeg)

head(cbind(lm.subset1, lm.subset2), 30)
```

```
##             [,1]      [,2]       [,3]        [,4]
##  [1,]         NA        NA  3.6664327  2.74790517
##  [2,]         NA        NA -2.5194424 -3.24203530
##  [3,]  0.6475581  1.559734         NA          NA
##  [4,]         NA        NA  2.4439621  1.29208230
##  [5,]  1.9921334  1.958417         NA          NA
##  [6,]  1.7534556  2.049381         NA          NA
##  [7,]  2.7300053  2.267323         NA          NA
##  [8,]         NA        NA  1.2366129  0.60842281
##  [9,]         NA        NA  1.7351840  1.07506483
## [10,]  2.6600869  2.193584         NA          NA
## [11,]  2.5722176  2.706334         NA          NA
## [12,]  1.5666576  2.043858         NA          NA
## [13,]         NA        NA  3.8438847  2.06073032
## [14,]         NA        NA  0.8196281 -0.60317801
## [15,]         NA        NA  0.4030093  0.27503423
## [16,]         NA        NA -0.3165287 -0.61742911
## [17,]  1.1915423  1.852328         NA          NA
## [18,]         NA        NA  3.4420387  2.08164193
## [19,]         NA        NA -2.8572507 -3.02171399
## [20,]  1.3629237  1.242134         NA          NA
## [21,]         NA        NA -2.1617141 -2.99443098
## [22,]         NA        NA  0.7875045  0.02474258
## [23,]         NA        NA -3.8181210 -3.34300048
## [24,]         NA        NA  1.0497982  0.77485532
## [25,] -3.2411387 -2.391492         NA          NA
## [26,]  1.7421789  1.485163         NA          NA
## [27,]         NA        NA -3.9158641 -4.14178218
## [28,]         NA        NA  2.9313577  1.51307707
## [29,]         NA        NA  0.3721062 -0.71655061
## [30,]         NA        NA  2.5544887  2.02524424
```

```r
# Plot the two clusters
matplot(sample.data$Input, cbind(lm.subset1[, 2], lm.subset2[,2]),
        type = "p", col = c("green", "blue"), pch = 19, ylab = "Separated Subsamples")
```



```
plot(select.seq.unscrambled[1:100], type = "s")
```



We've divided our data into two separate sets based on a guess (an "assumption") that:

- Rather than looking at a *single* population whose linear correlation between predictor and response behaves according to certain linear coefficients and a Gaussian residual with certain parameters,
- We are instead looking at *two* populations each of whose linear correlation between predictor and response behaves according to *a different set* of linear coefficients and residual different parameters.

In other words, we have moved to analyzing our data set according to the following two equations:

Subsample 1 (green dots): $Y_1 = a_1 X_1 + b_1 + \varepsilon_1$

Subsample 1 (blue dots): $Y_2 = a_2 X_2 + b_2 + \varepsilon_2$

3.3 Confusion Matrix

The problem we have, of course, is that even if we are correct that we are indeed observing two different populations, we cannot know which of our data sets belongs to which population.

Our sequence above is the map of our "guesses". We created `select.seq.unscrambled` and assigned it the following values:

- 1 = Subsample 1
- 0 = Subsample 2

Now, this is a simulation, so let's say that hypothetically we had access to the training data set and we knew in fact far every data point whether it was in Subsample 1 or the Subsample 2.

There is a common device used to compare our estimated sequence, `select.seq.unscrambled` and the true selection sequence, let's say this was called `select.seq.true` and was mapped as above. This device is called the confusion matrix.

To create a confusion matrix in this case, we would call the function `confusion.Matrix` from the library `caret` as follows:

`cm <- confusionMatrix(select.seq.unscrambled, select.seq.true)$table`

Let's hard code a hypothetical output of the above function:

```
cm <- as.table(matrix(c(450, 42, 50, 458), ncol = 2, byrow = T,
               dimnames = list(c("Pred.T", "Pred.F"),
                               c("Act.T", "Act.F"))))
cm
```

```
##        Act.T Act.F
## Pred.T   450    42
## Pred.F    50   458
```

The the elements $C(Predicted, Actual)$ of the resultant table are:

- Number of True Negatives: $C(0, 0)$
- Number of True Positives: $C(1, 1)$
- Number of False Negatives: $C(0, 1)$
- Number of False Positives: $C(1, 0)$

Now let's demonstrate how to calculate conditional probabilities based on our confusion matrix.

There are several characteristics of prediction quality with following definitions:

- Accuracy: $P(Prediction\ correct)$
- Sensitivity: $P(Pred = 1|Act = 1)$
- Specificity: $P(Pred = 0|Act = 0)$
- Balanced Accuracy: $\frac{Sensitivity+Specificity}{2}$

Calculate accuracy, sensitivity, specificity and balanced accuracy for the confusion table above.

```
cm.accuracy <- (cm["Pred.T", "Act.T"] + cm["Pred.F", "Act.F"]) / 1000
cm.sensitivity <- cm["Pred.T", "Act.T"] / sum(cm[,"Act.T"])
cm.specificity <- cm["Pred.F", "Act.F"] / sum(cm[, "Act.F"])
cm.balanced <- (cm.sensitivity + cm.specificity) / 2
(cm.out <- c(Accuracy = cm.accuracy,
             Sensitivity = cm.sensitivity,
             Specificity = cm.specificity,
             Balanced = cm.balanced))
```

```
##    Accuracy Sensitivity Specificity    Balanced
##       0.908       0.900       0.916       0.908
```

4 Estimating models for subsamples

4.1 Fitting models

Now estimate the linear models from the subsamples.

```
lm.subset1.estimate <- lm(Output ~ Input,
                          data = data.frame(Input = lm.subset1[,1], Output = lm.subset1[,2]))
lm.subset2.estimate <- lm(Output ~ Input,
                          data = data.frame(Input = lm.subset2[,1], Output = lm.subset2[,2]))
lm.subset1.est.summary <- summary(lm.subset1.estimate)
lm.subset2.est.summary <- summary(lm.subset2.estimate)

lm.subset1.est.summary$coefficients
```

```
##              Estimate  Std. Error  t value      Pr(>|t|)
## (Intercept) 0.7331544 0.019479048 37.63810 8.716159e-149
## Input       0.8012446 0.008346118 96.00207  0.000000e+00
```

```
lm.subset1.est.summary$sigma
```

```
## [1] 0.4389739
```

```
lm.subset1.est.summary$df
```

```
## [1]   2 506   2
```

```
lm.subset1.est.summary$r.squared
```

```
## [1] 0.9479552
```

```
lm.subset1.est.summary$adj.r.squared
```

```
## [1] 0.9478524
```

```
lm.subset2.est.summary$coefficients
```

```
##               Estimate  Std. Error   t value      Pr(>|t|)
## (Intercept) -0.6941222 0.020008001 -34.69223 4.708656e-134
## Input        0.8107406 0.008586561  94.41971 1.557398e-316
```

```
lm.subset2.est.summary$sigma
```

```
## [1] 0.4433244
```

```
lm.subset2.est.summary$df
```

```
## [1]   2 490   2
```

```
lm.subset2.est.summary$r.squared
```

```
## [1] 0.9479005
```

```
lm.subset2.est.summary$adj.r.squared
```

```
## [1] 0.9477942
```

And write a summary comparing the fit of the subsamples to the whole sample:

- For the sigma parameters:

```
c(lm.est.summary$sigma,
  lm.subset1.est.summary$sigma,
  lm.subset2.est.summary$sigma)
```

```
## [1] 0.8388930 0.4389739 0.4433244
```

- For the ρ^2:

```
c(lm.est.summary$r.squared,
  lm.subset1.est.summary$r.squared,
  lm.subset2.est.summary$r.squared)
```

```
## [1] 0.8307611 0.9479552 0.9479005
```

- For the F-statistics:

```
rbind(LinearModel=lm.est.summary$fstatistic,
      LinearModelPositive=lm.subset1.est.summary$fstatistic,
      LinearModelNegative=lm.subset2.est.summary$fstatistic)
```

```
##                         value numdf dendf
## LinearModel          4898.989     1   998
## LinearModelPositive 9216.397     1   506
## LinearModelNegative 8915.082     1   490
```

Here is how we can calculate p-values of F-test using cumulative probability function of F-distribution:

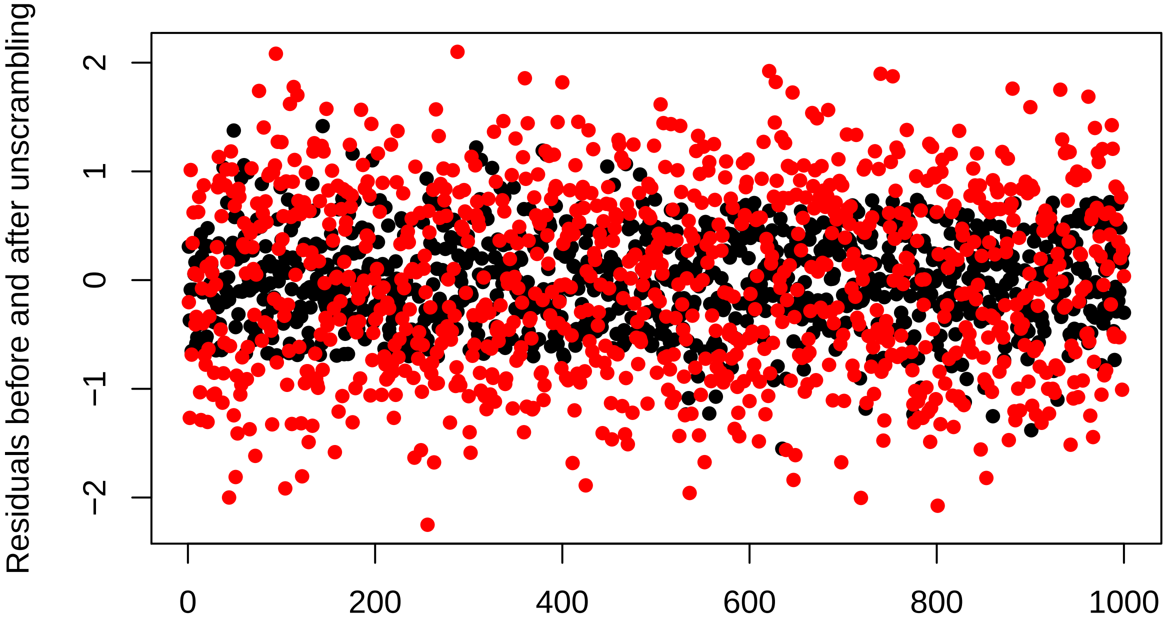
```
lm.est.pf <- pf(lm.est.summary$fstatistic[1],
                lm.est.summary$fstatistic[2],
                lm.est.summary$fstatistic[3],lower.tail = FALSE)
lm.subset1.est.pf <- pf(lm.subset1.est.summary$fstatistic[1],
                        lm.subset1.est.summary$fstatistic[2],
                        lm.subset1.est.summary$fstatistic[3],lower.tail = FALSE)
lm.subset2.est.pf <- pf(lm.subset2.est.summary$fstatistic[1],
                        lm.subset2.est.summary$fstatistic[2],
                        lm.subset2.est.summary$fstatistic[3],lower.tail = FALSE)
c(LinearModel = lm.est.pf,
  LinearModelPositive = lm.subset1.est.pf,
  LinearModelNegative = lm.subset2.est.pf)
```

```
##         LinearModel.value LinearModelPositive.value
##              0.000000e+00              0.000000e+00
## LinearModelNegative.value
##             1.557398e-316
```

The numbers may not look exactly the same as in `summary()` because of the precision limitation.

Compare the combined residuals of the two separated models with the residuals of our original estimated model.

```
# Plot residuals
matplot(cbind(MixedModelResiduals=c(lm.subset1.est.summary$residuals,
                                    lm.subset2.est.summary$residuals),
              SingleModelResiduals=lm.est.summary$residuals),
        type="p", pch=16, ylab="Residuals before and after unscrambling")
```



```r
# Estimate standard deviations
apply(cbind(MixedModelResiduals=c(lm.subset1.est.summary$residuals,
                                   lm.subset2.est.summary$residuals),
            SingleModelResiduals=lm.est.summary$residuals),2,sd)
```

```
##  MixedModelResiduals SingleModelResiduals
##            0.4404568            0.8384730
```

And let's do a final output table to make the data easier to compare:

```r
t00.2 <- round(lm.estimate$coefficients[1], 3)
t00.3 <- round(lm.subset1.estimate$coefficients[1], 3)
t00.4 <- round(lm.subset2.estimate$coefficients[1], 3)
t01.2 <- round(lm.estimate$coefficients[2], 3)
t01.3 <- round(lm.subset1.estimate$coefficients[2], 3)
t01.4 <- round(lm.subset1.estimate$coefficients[2], 3)
t1.2 <- round(lm.est.summary$sigma, 3)
t1.3 <- round(lm.subset1.est.summary$sigma, 3)
t1.4 <- round(lm.subset2.est.summary$sigma, 3)
t2.2 <- round(lm.est.summary$r.squared, 3)
t2.3 <- round(lm.subset1.est.summary$r.squared, 3)
t2.4 <- round(lm.subset2.est.summary$r.squared, 3)
t3.2 <- round(lm.est.summary$coefficients[1,4], 3)
t3.3 <- round(lm.subset1.est.summary$coefficients[1,4], 3)
t3.4 <- round(lm.subset2.est.summary$coefficients[1,4], 3)
t4.2 <- round(lm.est.summary$coefficients[2,4], 3)
t4.3 <- round(lm.subset1.est.summary$coefficients[2,4], 3)
t4.4 <- round(lm.subset2.est.summary$coefficients[2,4], 3)
```

```
t5.2 <- round(lm.est.summary$fstatistic[1])
t5.3 <- round(lm.subset1.est.summary$fstatistic[1])
t5.4 <- round(lm.subset2.est.summary$fstatistic[1])
```

Estimate	Linear Model	Linear Model Positive	Linear Model Negative
(Intercept) estimate	0.032	0.733	-0.694
Input estimate	0.796	0.801	0.801
σ_ε	0.839	0.439	0.443
ρ^2	0.831	0.948	0.948
(Intercept) p-value	0.234	0	0
Input p-value	0	0	0
F-statistic	4899	9216	8915
F p-value	0	0	0

What is the difference between the quality of fit?

Slope Coefficient P-Value

The utility of a linear model is demonstrated in both cases, the single model and the mixed model. We see this in the extremely low p-value of our slope coefficient. The p-value in this case tests against the null hypothesis that the slope of the line is actually 0. If the slope of the line were truly zero, then our Output or response values would not vary predictably based on our Input or response values, since for a 0 sloped line y values would be constant with respect to x. Our extremely low p-values in both models suggest that the data in our sample would be extremely unlikely if the true distribution had a zero slope. Therefore we reject the null hypothesis that the data has no slope, and accept the likelihood that a linear model with our estimated slope(s) is useful in predicting our response variables.

F Statistic P-Value

This is also confirmed by the F-statistic p-value, which for both models is statistically significant. Unlike the t-tests of individual coefficients, the F-test can assess multiple coefficients simultaneously. The null hypothesis of the F-test is an intercept-only test, in other words testing whether *all* slope coefficients are zero in the true distribution. In this simple linear regression example, since we have only one slope coefficient (by virtue of having only one predictor variable) our F-statistic p-value and our Input coefficient p-value are the same test. Each of these show as zero in our rounded table above, but a quick glance back at the `lm()` output and F-statistic calculations for our Negative Model show that these are the same at 8915.081724.

Similarities vs. Differences

So, we come to a conclusion that regardless of a single model or mixed models, a linear model has utility in representing our data and that it is highly unlikely that the Output variable of our true population does not vary according to some multiple (our slope coefficient) of our Input variable. Bearing in mind that – assuming there is randomness in our population – it is still possible that the relationship we see in the charts and our software has found in the data could **still** be a random alignment within the sample we happened to draw (That would be magical wouldn't it? But still possible and we should always keep this in mind.), we can move on to our most important question:

What is the difference between the two, in terms of quality of fit?

Since the difference is not found in our slope or F-statistic p-values, we move to the intercept p-value, r-squared statistic, and F-statistic. Each of these show discernable differences.

F-Statistic

The F-Statistic in simple linear regression is the ratio of the variance explained by the model and the residual or unexplained variance. As discussed above the p-value of our F-Statistics is extremely low in both the single model and mixed model solutions, but as we can see in the table above our F-statistic is roughly twice as large in our two subsets (Positive and Negative) as our F-statistic in our single model. Since we've split the data in half, essentially halving our variation, this makes sense.

We can predict that if instead a single model, rather than a mixed model, were more accurate, the subsets would not demonstrate a twice improved F-statistic. Intuitively we can envision what this would look like: if the Output values were, in the true population, normally distributed about a single line, splitting that line in half as we have done in our mixed model would produce two sets of data whose values were clustered around the "bottom" in one and clustered about the "top" in the other. Thus their means would be pulled away from their center and their variability increased, reducing the predictive ability of the model and the ratio of the F-statistic.

So, our higher F-statistics in the mixed model subsets tells us that these subsets do a better job of explaining the variation than a single model, intuitvely this makes sense as well, and we conclude that we are looking at two populations with the same slope (multiplier of Input to produce Output) but different intercepts (mean value) – that is two different parallel lines on an xy-plot.

Intercept P-Value

Since our mixed model is based on the same slope for each subset but a differing intercept, it makes sense also that the intercept p-value would justify our use of the mixed model solution. This is a clear difference. For a single model, our intercept estimate is not significant. For our two subsets, it is.

R-squared

Our r-squared value is linked to our F-statistic, since a low p-value for our overall F-test leads us to conclude that the R-squared value is significantly different from zero. In this case we are comparing our single model with a mixed model, we can observe the difference in r-squared between the single model and our subsets. Here we see that r-squared is larger in our subsets, indicating that the portion of the variation explained by the subsets is greater than that defined by a single model.

What is the difference between the two estimated models?

What have we proved with our demonstration? As discussed above, all of our models are based on simple linear regression, and in all three cases the use of linear regression and the slope calculated in our tests demonstrate statistical significance.

But only through noting the insignificant intercept estimate and visually examininng our residuals were we able to detect that two populations rather than one were seemingly represented in our data.

The two new models we located differ from eachother and from the single model in *intercept estimate* only, the slope is the same for all three models.

Try to guess how the model data were simulated and with what parameters?

Based on an examination of our `lm()` results above, we guess that the Output data was simulated with a slope of 0.8 for both, and a intercept of ± 0.7. We'll also guess a standard deviation for our residuals of 0.45. So we ascribe to our two models the following equations:

$Output1 = 0.8 \cdot Input + 0.7 + \varepsilon$

$Output2 = 0.8 \cdot Input - 0.7 + \varepsilon$

$\varepsilon \sim Norm(0, 0.45)$

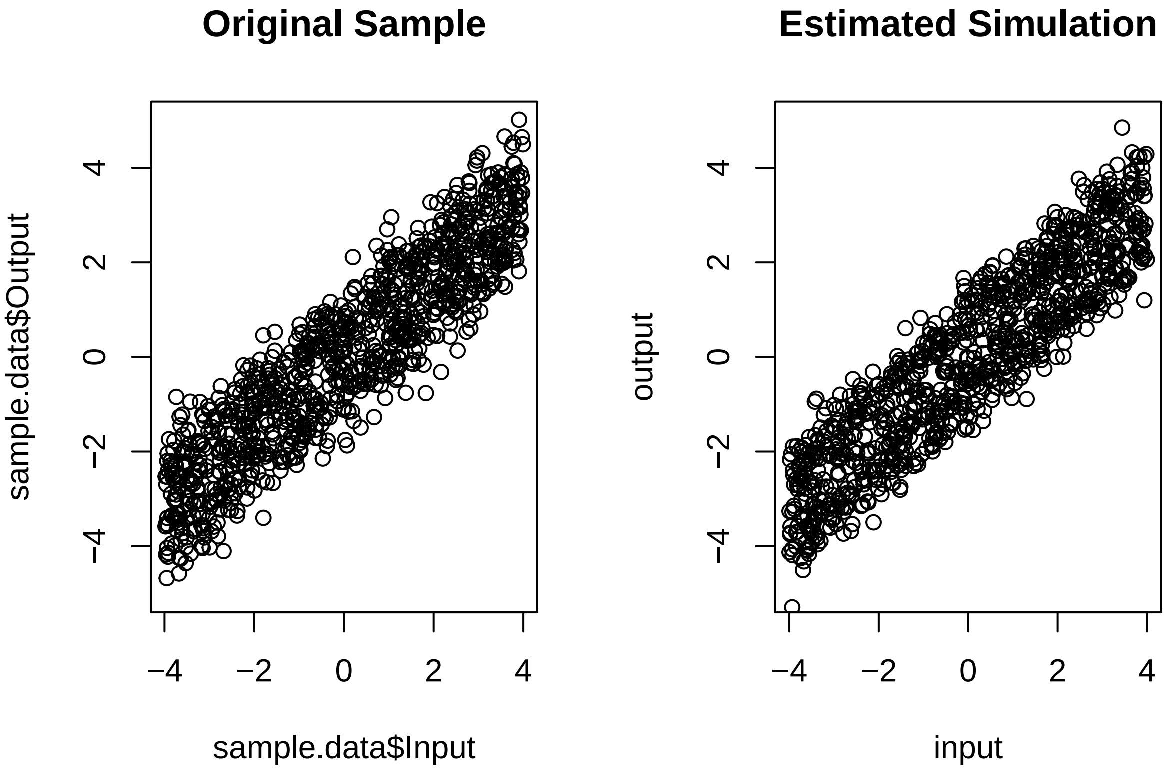
An inspection of the Input variable leads us to believe that it was simulated as following:

$Input \sim Unif(-4, 4)$

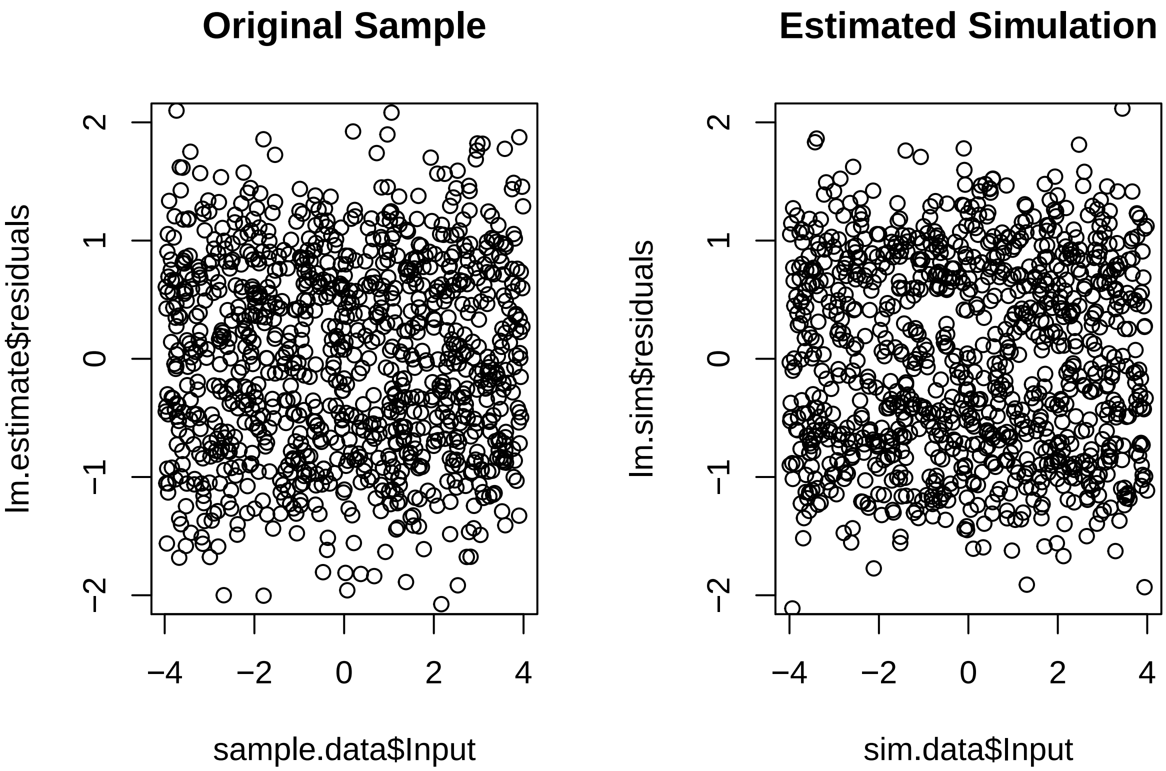
Let's assume that our selection variable simply alternated between the two evenly. We can now reproduce our estimate of the simulation, plot, and compare.

As seen below our guess at the parameters of the original simulation of our sample data may not be 100% accurate, but they clearly demonstrate similar characteristics to the original sample from `ResidualAnalysisProjectData_1.csv`.

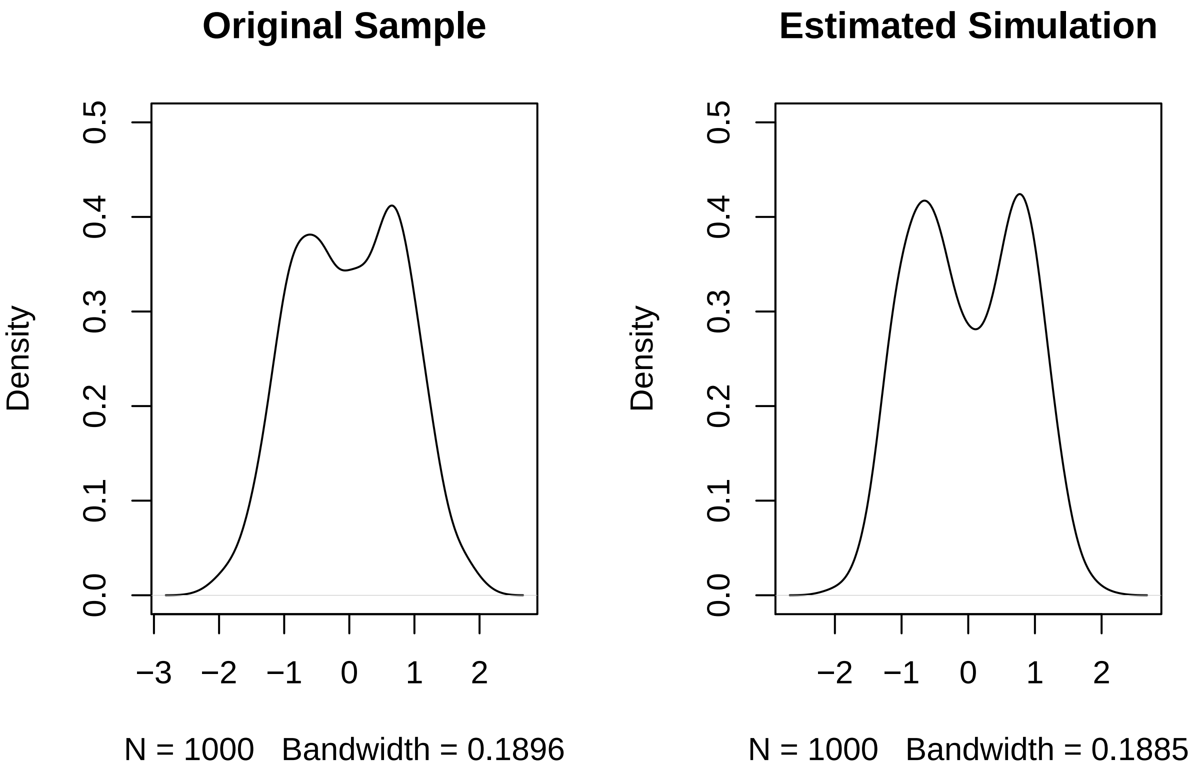
```
input <- runif(1000, -4, 4)
output.a <- 0.8
subset1.b <- 0.7
subset2.b <- -0.7
output.eps <- 0.45

select.seq <- rep(c(0,1), 500)
output.fit <- rep(NA, 1000)
output.fit[select.seq == TRUE] <- output.a * input[select.seq == TRUE] + subset1.b
output.fit[select.seq == FALSE] <- output.a * input[select.seq == FALSE] + subset2.b
output <- output.fit + rnorm(1000, mean = 0, sd = output.eps)

sim.data <- data.frame(Input = input, Output = output)
lm.sim <- lm(Output ~ Input, sim.data)
lm.sim.summary <- summary(lm.sim)

title.orig <- "Original Sample"
title.est <- "Estimated Simulation"
par(mfrow=c(1,2))
plot(sample.data$Input, sample.data$Output, ylim = c(-5, 5), main = title.orig)
plot(input, output, ylim = c(-5, 5), main = title.est)
```



```
plot(sample.data$Input, lm.estimate$residuals, ylim = c(-2,2), main = title.orig)
plot(sim.data$Input, lm.sim$residuals, ylim = c(-2,2), main = title.est)
```



```
plot(density(lm.estimate$residuals), ylim = c(0, 0.5), main = title.orig)
plot(density(lm.sim$residuals), ylim = c(0, 0.5), main = title.est)
```



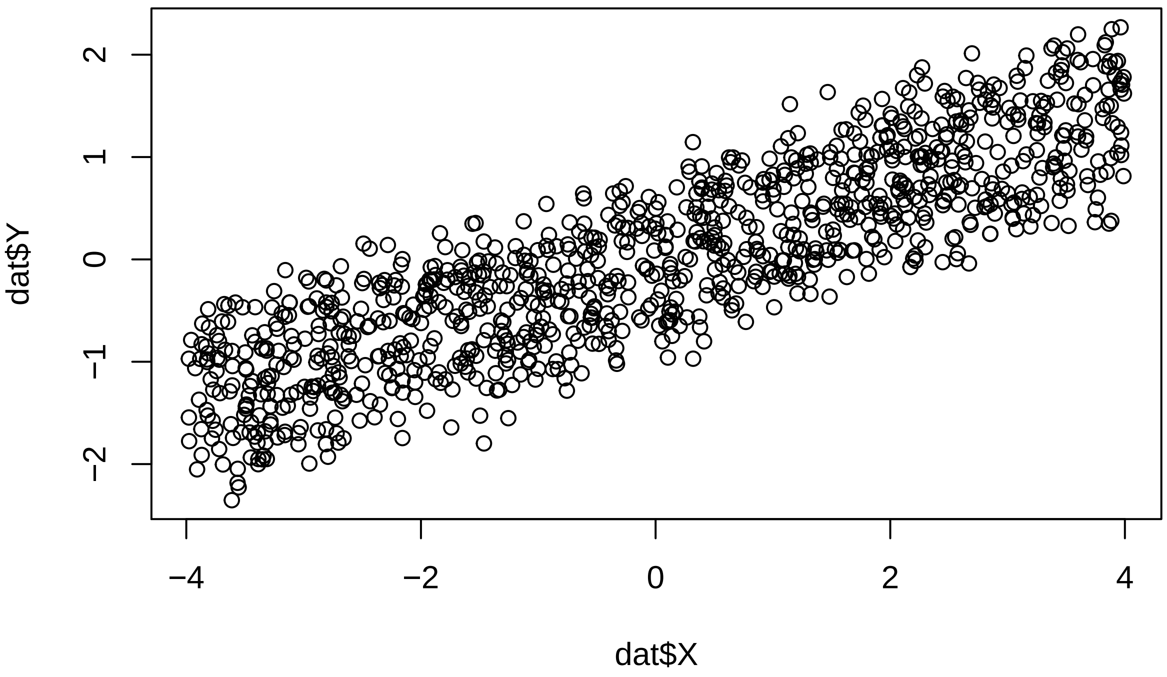
```
par(mfrow=c(1,1))
```

Test

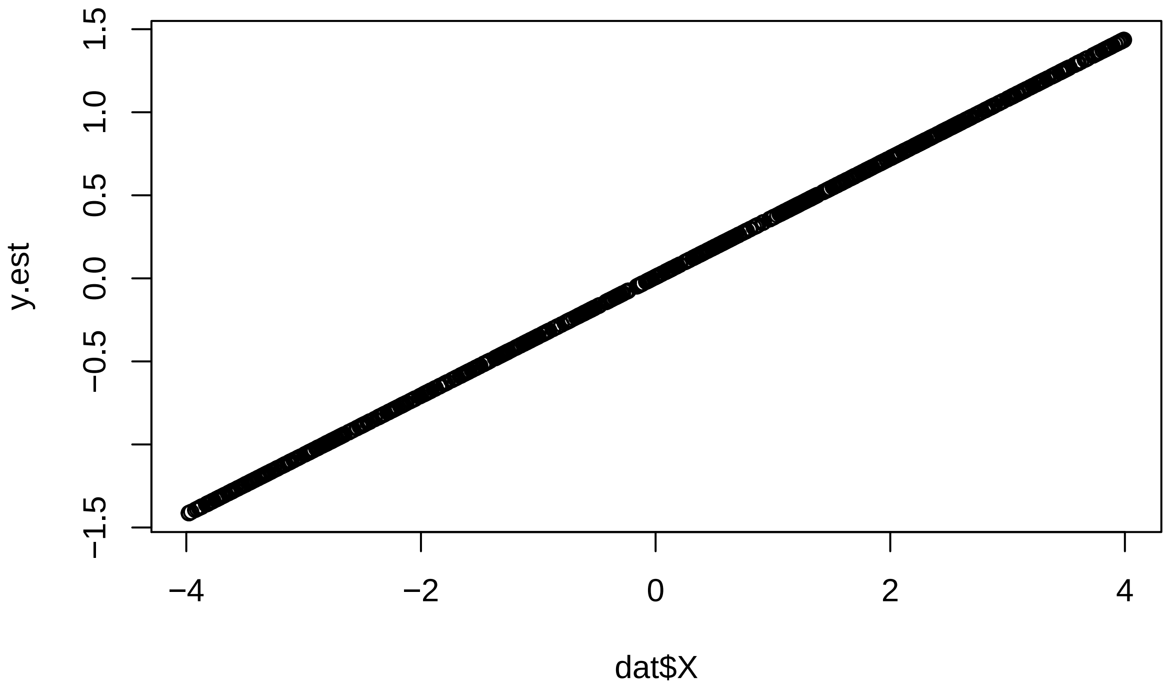
```
dat <- read.table('Week4_Test_Sample.csv', header=TRUE)

test.fit <- lm(Y~X, dat)
test.fit.summary <- summary(test.fit)
test.fit.summary
```

```
##
## Call:
## lm(formula = Y ~ X, data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.28390 -0.40175  0.00108  0.40025  1.10136
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.008739   0.015421   0.567    0.571
## X           0.357452   0.006718  53.208   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4876 on 998 degrees of freedom
## Multiple R-squared:  0.7394, Adjusted R-squared:  0.7391
## F-statistic:  2831 on 1 and 998 DF,  p-value: < 2.2e-16
```

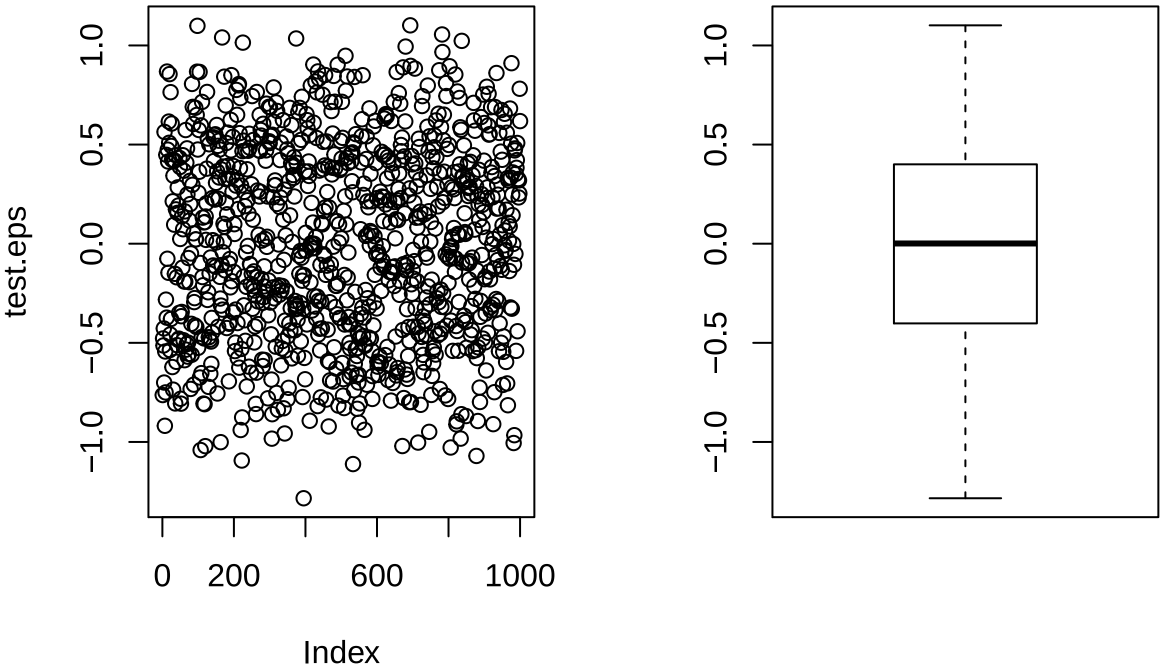
```
plot(dat$X, dat$Y)
```



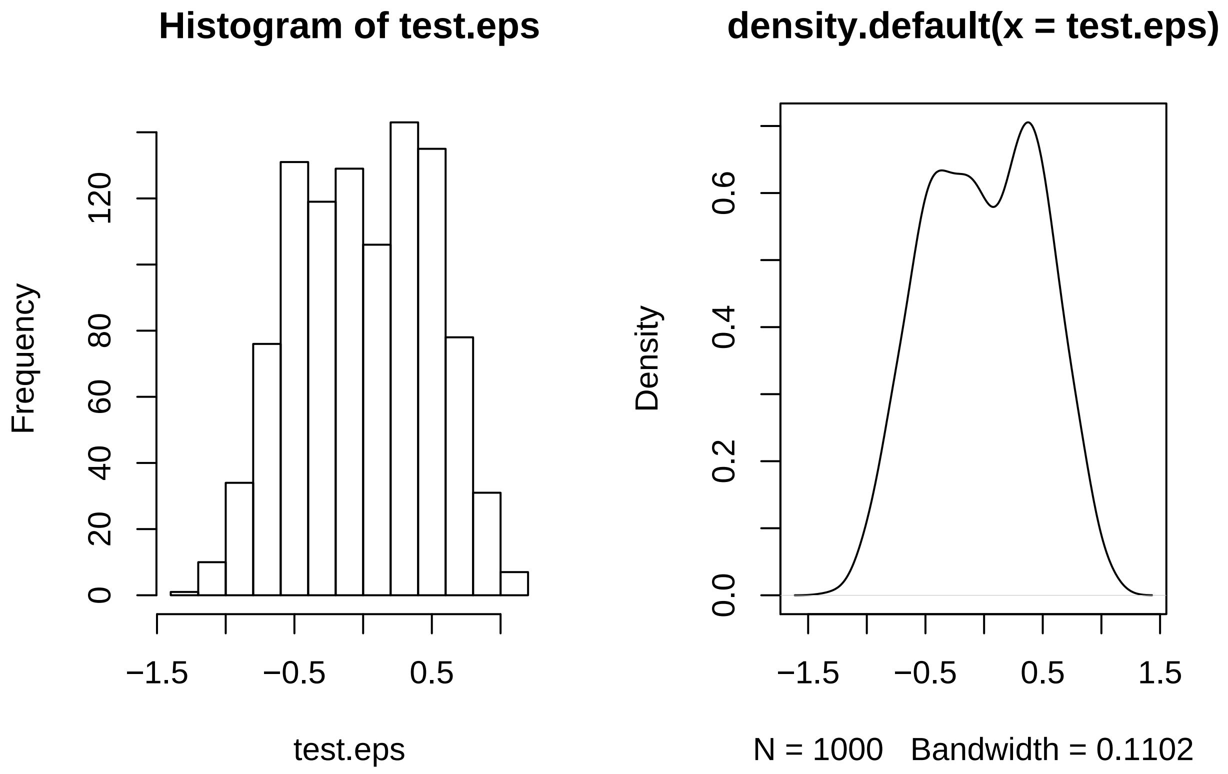
```
y.est <- dat$X * coef(test.fit)[2] + coef(test.fit)[1]
plot(dat$X, y.est)
```



```
test.eps <- dat$Y - y.est
par(mfrow=c(1,2))
plot(test.eps)
boxplot(test.eps)
```



```
hist(test.eps)
eps.density <- density(test.eps)
plot(eps.density)
```



```
par(mfrow=c(1,1))

# Cluster
c(Left.Mean = mean(test.eps[test.eps < 0]),
  Right.Mean = mean(test.eps[test.eps > 0]))
```

```
##  Left.Mean Right.Mean
## -0.4153683  0.4153683
```

```
select.seq.unscrambled <- as.integer(test.eps > 0)
head(select.seq.unscrambled, 30)
```

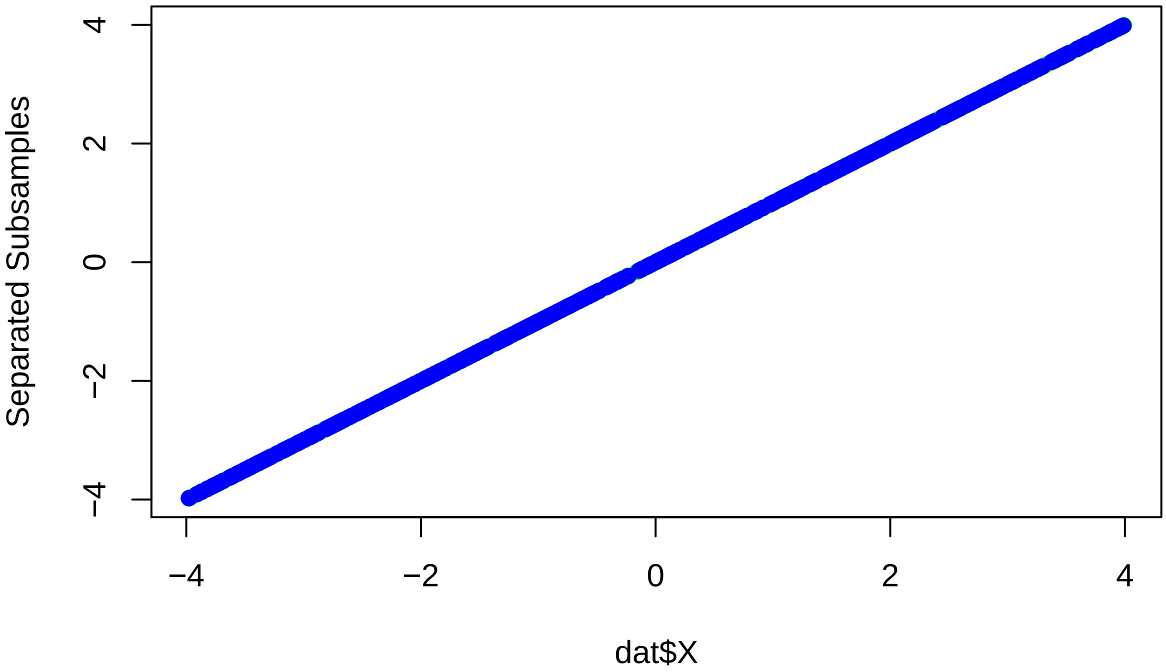
```
##  [1] 0 0 0 0 0 1 0 0 0 0 1 0 1 0 1 1 0 1 1 1 0 0 1 0 1 1 1 0 1 0
```

```
ChoosePos <- function(j) {
  vapply(seq_along(select.seq.unscrambled),
         function(i) {switch(select.seq.unscrambled[i] + 1, NA, dat[i, j])},
         numeric(1))
}

ChooseNeg <- function(j) {
  vapply(seq_along(select.seq.unscrambled),
         function(i) {switch(select.seq.unscrambled[i] + 1, dat[i, j], NA)},
         numeric(1))
}

lm.subset1 <- sapply(seq_along(dat), ChoosePos)
```

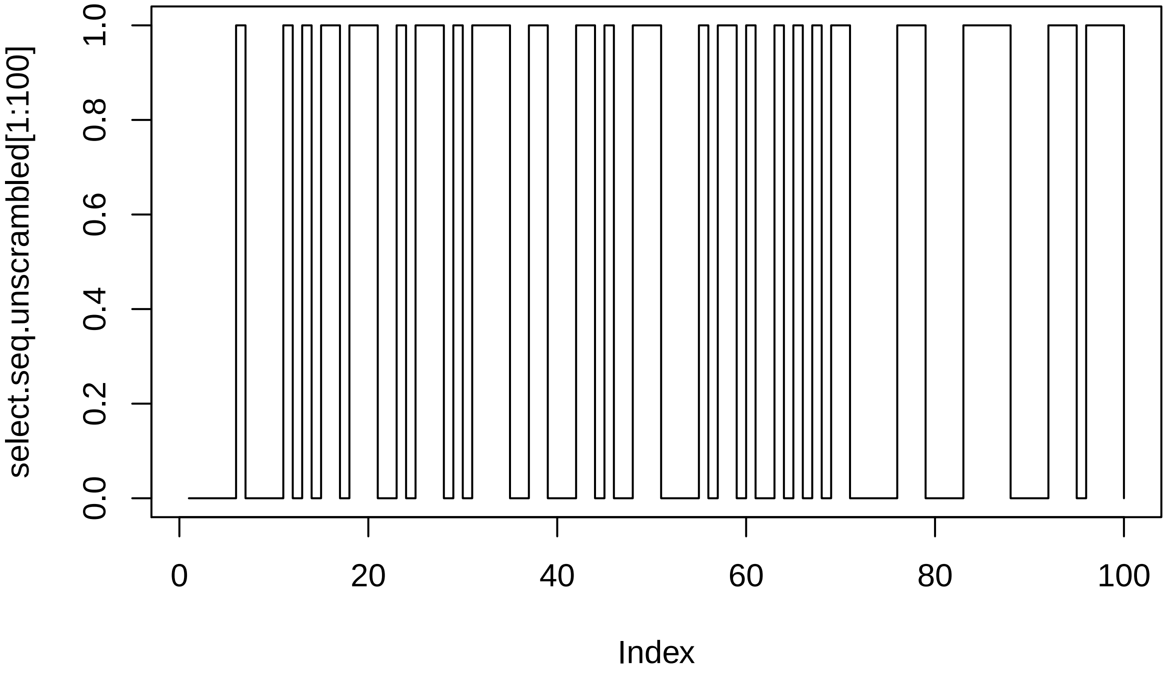
```r
lm.subset2 <- sapply(seq_along(dat), ChooseNeg)

head(cbind(lm.subset1, lm.subset2), 30)
```

```
##              [,1]       [,2]        [,3]        [,4]
##  [1,]          NA         NA -0.17203655  1.63007615
##  [2,]          NA         NA  0.50425867  2.72350071
##  [3,]          NA         NA -0.84243853 -0.94508648
##  [4,]          NA         NA -0.80014421 -1.06713872
##  [5,]          NA         NA -0.55850922  0.37229503
##  [6,] -0.5445710 -3.1252106          NA          NA
##  [7,]          NA         NA -2.18334020 -3.56286508
##  [8,]          NA         NA -0.05142797  1.35239455
##  [9,]          NA         NA -1.25618558 -1.44029303
## [10,]          NA         NA -0.45975629 -0.52151178
## [11,]  0.6691585  0.5931595          NA          NA
## [12,]          NA         NA -1.32337397 -2.68356245
## [13,] -0.4370878 -3.6763861          NA          NA
## [14,]          NA         NA -0.09584977 -0.08020687
## [15,]  0.6755839  0.5385551          NA          NA
## [16,] -0.1281226 -1.5432964          NA          NA
## [17,]          NA         NA -1.10342708 -2.70152878
## [18,]  1.9572353  3.7273420          NA          NA
## [19,] -0.1526437 -1.8753789          NA          NA
## [20,]  0.6433269 -0.6178240          NA          NA
## [21,]          NA         NA  0.82712474  3.79015820
## [22,]          NA         NA -0.38642141  0.17514152
## [23,]  0.6438216 -0.3617344          NA          NA
## [24,]          NA         NA -0.56807348 -0.55170202
## [25,]  1.4564981  2.6708037          NA          NA
## [26,]  1.8499391  3.4555036          NA          NA
## [27,] -0.3967423 -2.3187048          NA          NA
## [28,]          NA         NA  0.18559056  2.23545576
## [29,]  0.2891007  0.1863342          NA          NA
## [30,]          NA         NA -1.69923778 -2.71890770
```

```r
# Plot the two clusters
matplot(dat$X, cbind(lm.subset1[, 2], lm.subset2[, 2]),
        type = "p", col = c("green", "blue"), pch = 19, ylab = "Separated Subsamples")
```



```
plot(select.seq.unscrambled[1:100], type = "s")
```



```r
res <- list(Unscrambled.Selection.Sequence =  select.seq.unscrambled)
write.table(res, file = 'result.csv', row.names = F)
```