R Notebook

Code

Hide

```
setwd("C:/Users/Veeru/Desktop/ML/linear regression")
getwd()
```

```
[1] "C:/Users/Veeru/Desktop/ML/linear regression"
```

Hide

```
train2 <- read.csv("TrainData_Group2.csv",header=TRUE, sep = ",")
str(train2)
```

```
'data.frame':   1000 obs. of  6 variables:
 $ X1: num  -1.57628 -0.00496 2.04462 -0.76652 -0.09431 ...
 $ X2: num  -2.5 -1.72 -4.11 -2.21 -1.94 ...
 $ X3: num  0.4467 -0.0263 3.5557 -0.8501 1.3209 ...
 $ X4: num  1.34 -0.0789 10.6671 -2.5502 3.9627 ...
 $ X5: num  -2.28 -4.39 -2.9 3.73 -4.58 ...
 $ Y : num  7.778 0.552 -8.703 3.313 1.878 ...
```

saving the training dataset in “df1”. structure of df1 shows we have 1000 rows and 6 variables. linear model will be constructed using the df1 dataset as trainin set for predicting Y varibles values in testing dataset.

Hide

```
sum(is.na(train2))
```

```
[1] 0
```

checking for NA’s in the dataset

Hide

```
library(ggplot2)
library(cowplot)
```

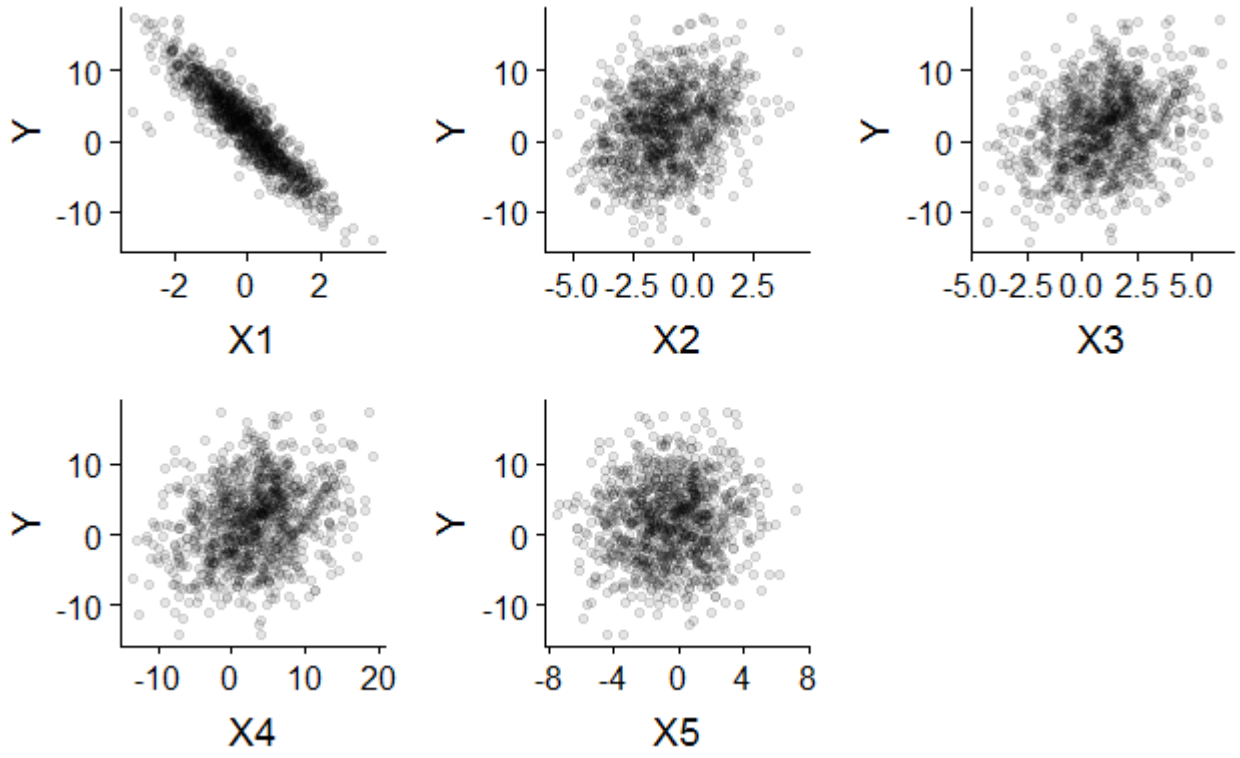
```
Attaching package: <U+393C><U+3E31>cowplot<U+393C><U+3E32>

The following object is masked from <U+393C><U+3E31>package:ggplot2<U+393C><U+3
E32>:

    ggsave
```

Hide

```
plot1 <- ggplot(train2, aes(X1, Y)) + geom_point(alpha=0.1)
plot2 <- ggplot(train2, aes(X2, Y)) + geom_point(alpha=0.1)
plot3 <- ggplot(train2, aes(X3, Y)) + geom_point(alpha=0.1)
plot4 <- ggplot(train2, aes(X4, Y)) + geom_point(alpha=0.1)
plot5 <- ggplot(train2, aes(X5, Y)) + geom_point(alpha=0.1)
cowplot::plot_grid(plot1,plot2,plot3,plot4,plot5)
```



from above plots it is visible that only X1 variable shares linear relationship. Rest all variable from X2 to X5 are scattered all over.

Hide

```
x <- as.matrix(train2[-ncol(train2)])
int <- rep(1, nrow(x))
x <- cbind(int, x)
y  <- train2$Y
#checking for x matrix variables
colnames(x)
```

```
[1] "int" "X1"  "X2"  "X3"  "X4"  "X5"
```

Hide

```
# checking y is a vector having 1000 observations
length(y)
```

```
[1] 1000
```

“x” and “y” has to be in matrix form for matrix multiplication “%*%"

“x” matrix has now only variables from X1 to X5 of “train2”

“y” vector has only y variable from “train2”

The linear regression equation is y=B0 + B1x1 +B2x2 + B3X3 + B4x4 + B5x5. Betas are the co-efficients, one for each variable in matrix “x”, thus i have B1,B2,B3,B4,B5 computed from below formula and B0 is the intercept. These parameters are the linear relationship between response variable which is “y” and predictor variables which are from “X1” to “X5”.

Hide

```
betas <- t(x) %*% y %*% solve(t(x) %*% x)
```

```
Error in t(x) %*% y %*% solve(t(x) %*% x) : non-conformable arguments
```

as per http://faculty.cas.usf.edu/mbrannick/regression/Reg2IV.html (http://faculty.cas.usf.edu/mbrannick/regression/Reg2IV.html) website for calculating multivariable co-efficients:

generic equation for computing betas:

(summation of x.y)/(summation of x^2)

The rule for matrix multiplication is that the number of columns of first matrix should match with number of rows of the second matrix, without which the error as above will be generated.

breaking down the equation:

A.) summation of x.y:

which is t(x) %*% y, The dimension of matrix x is (1000,6) i.e rows,columns and that of “y” is (1000,1), the resulting matrix out of their product is not possible. If transposed the matrix “x” dimension changes to (6,1000). The product of the marices t(x).y then produces the matrix of dimension (6,1). The product within the respective elements of matrices also add up and that is where summation happens.

B.) summation of x^2:

which is solve(t(x) %x% x), solve() function is to find the values of co-effecients in linear or quadratic equation and also to find the inverse of given matrix. Mathematically inverse of an interger is equal to 1/the same integer eg: inverse of 10 is 1/10 or 10^-1.

since we want summation of x^2 as denominator to compute betas, i will be using solve() to produce the inverse of x^2 matrix.

x.x is x^2 and for matrix multiplication, transpose of matrix is considered. solve(t(x) %x% x) produces matrix of dimension (6,1000)(1000,6) = (6,6)

As per the generic betas formula, dim(A.)*dim(B.) =(6,1).(6,6), since the columns of A. part is not same to rows of B. part the matrix multiplication is not possible. But if the part A. is transposed then the matrix multiplication is possible.

Hide

```
betas <- t(t(x) %*% y) %*% solve(t(x) %*% x)
colnames(betas) <- c("B0", "B1", "B2", "B3", "B4", "B5")
```

Changing the names of columns for better underestanding.

Hide

```
linearmodel <- lm(Y~.,data=train2)
```

comparing beta values as that of generated by lm

Hide

```
comparison <- data.frame(B = t(betas), lm = linearmodel$coefficients)
print(comparison)
```

	B <dbl>	lm <dbl>
B0	1.900014674	1.900014674
B1	-4.868649672	-4.868649672
B2	0.887624287	0.887624287
B3	0.019080442	0.019080442
B4	0.261369554	0.261369554
B5	-0.002523345	-0.002523345

6 rows

the coefficients are same

Hide

```
B0 <- mean(y)-(betas[,2]*mean(x[,2]))-(betas[,3]*mean(x[,3]))-(betas[,4]*mean(x
[,4]))-(betas[,5]*mean(x[,4]))-(betas[,5]*mean(x[,5]))
B0_values <- data.frame(B0[1],linearmodel$coefficients[1],betas[,1])
row.names(B0_values) <- NULL
B0_values
```

B0.1. <dbl>	linearmodel.coefficients.1. <dbl>	betas...1. <dbl>
1.64118	1.900015	1.900015

1 row

Calculating B0 using the formula B0 = mean(y)-B1.mean(X1)-B2.mean(X2)-B3.mean(X3) -B4.mean(X4)-B5.mean(X5) whish is explained in the same website link above. B0 calculated above does not match with the intercept value produced by linear model neither matches with betas matrix's variable "B0". The comparison is as shown in table above.

Hide

```
y_model_train <- rep(0,nrow(train2))
for (i in 1:nrow(train2)){
  y_model_train[i] <- betas[1] + betas[2]*train2$X1[i] + betas[3]*train2$X2[i]
+ betas[4]*train2$X3[i] + betas[5]*train2$X4[i] + betas[6]*train2$X5[i]
}
str(y_model_train)
```

```
 num [1:1000] 7.718 0.391 -8.839 2.978 1.707 ...
```

The calculation above represents the linear regression equation is y=B0 + B1x1 +B2x2 + B3X3 + B4x4 + B5x5. The beta values computed above are same as statistical R's lm() function output. These beta values are used to predict the "Y" variable of the same train2 dataset. The intension is that the actual "Y" variable's values will be compared with "y_model_train" dataset and the accuracy of beta values are determined by computing R square.

Hide

```
res2 <- train2$Y - y_model_train
ss_res2 <- sum(res2**2)
ss_tot2 <- sum((y-mean(y))**2)
r_sq2 <- 1-(ss_res2/ss_tot2)
r_sq2
```

```
[1] 0.9722856
```

finding the residuals “res2” by subtracting the predicted values from the actual values from train2$y and y_model,these are the errors made by fitting a line through the data points.

R square value should be close to one for best fit of the linear model.

Hide

```
summary(linearmodel)
```

```
Call:
lm(formula = Y ~ ., data = train2)

Residuals:
     Min       1Q   Median       3Q      Max 
-11.7646  -0.1172   0.0739   0.2541   0.9141 

Coefficients:
             Estimate Std. Error  t value Pr(>|t|)    
(Intercept)  1.900015   0.039649   47.921  < 2e-16 ***
X1          -4.868650   0.028346 -171.757  < 2e-16 ***
X2           0.887624   0.018732   47.385  < 2e-16 ***
X3           0.019080   0.231589    0.082 0.934354    
X4           0.261370   0.077505    3.372 0.000774 ***
X5          -0.002523   0.012079   -0.209 0.834563    
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.9246 on 994 degrees of freedom
Multiple R-squared:  0.9723,    Adjusted R-squared:  0.9721 
F-statistic:  6974 on 5 and 994 DF,  p-value: < 2.2e-16
```

Summary(linear model) shows the co-efficients and also the R square

Hide

```
comparison_rsq <- data.frame(r_sq2, summary(linearmodel)[8])
comparison_rsq
```

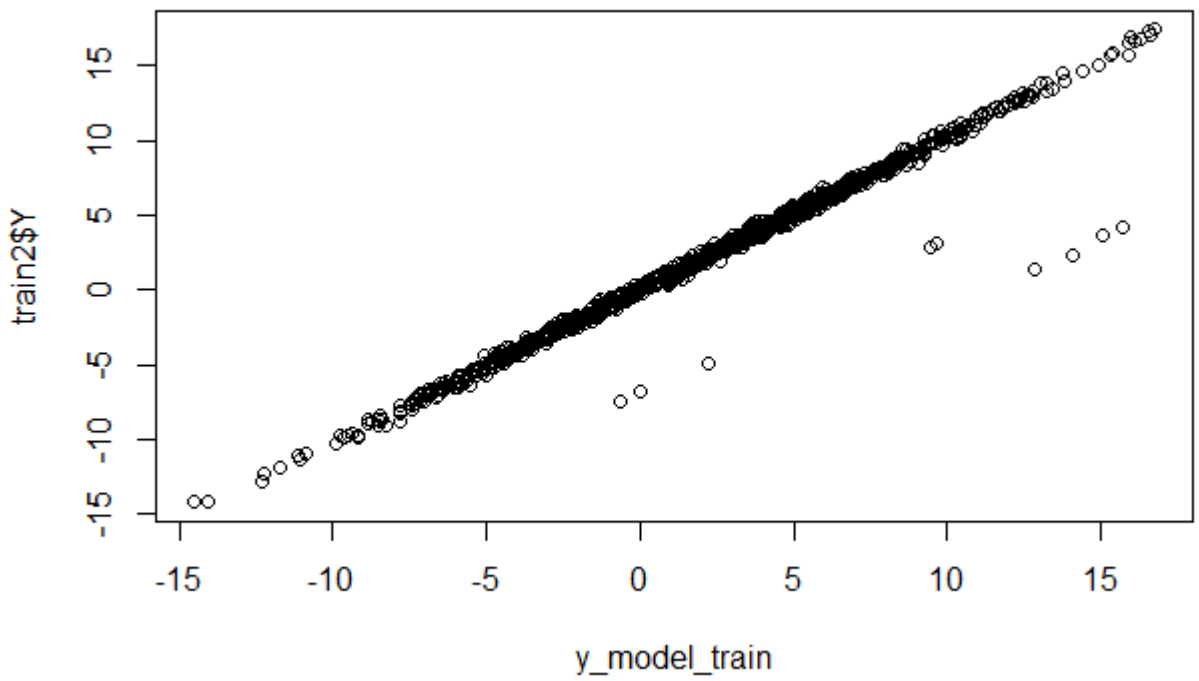
r_sq2 <dbl>	r.squared <dbl>
0.9722856	0.9722856

1 row

r_sq2 is the r-sq calculated without using of statistical linear code of lm() unlike r.squared. Both the values computed are same and thus prove that the calculations are accurate and since rsq is close to 1, the linear model fits well. Thus, the same beta values will be used to predict the “Y” variable of “test2” dataset.

Hide

```
plot(y_model_train,train2$Y)
```



There’s a strong correlation between the model’s predictions and its actual results.

Hide

```
test2 <- read.csv("TestData_Group2.csv",header=TRUE, sep = ",")
str(test2)
```

```
'data.frame':   250 obs. of  5 variables:
 $ X1: num  -1.12 0.465 -1.581 2.039 -0.958 ...
 $ X2: num  -0.565 -0.144 -2.833 0.472 -0.359 ...
 $ X3: num  2.614 -1.943 6.066 0.567 0.102 ...
 $ X4: num  7.841 -5.829 18.198 1.7 0.305 ...
 $ X5: num  -5.709 1.394 -4.858 -1.652 -0.229 ...
```

Hide

```
sum(is.na(test2))
```

```
[1] 0
```

checking for NAN values if any.

Hide

```
nrow(test2)
```

```
[1] 250
```

The co-efficient values B0…B5 are calculated using training dataset, these values will be used with testing dataset to predict unknown values.Here the known values are the predictors, which is variables from X1 to X5 of testing dataset. “Y” is to be predicted by using the formula of Y=B0 + m1.X1 + m2.X2 +…+m5.X5. B0 is the y-intercept and “B1” to “B5” are the slopes. The formula above is written within for loop, so that each observation is calculated at every iteration and the predictions are stored accordingly in y_model.

Hide

```
y_model_test2 <- rep(0,nrow(test2))
for (i in 1:nrow(test2)){
  y_model_test2[i] <- betas[1] + betas[2]*test2$X1[i] + betas[3]*test2$X2[i] +
betas[4]*test2$X3[i] + betas[5]*test2$X4[i] + betas[6]*test2$X5[i]
}
y_model_test2
```

```
  [1]   8.9628235  -2.0538743  11.9675058  -7.1489466   6.3278963   8.4246690
1.3845898  -3.1165829
  [9] -11.5206788   5.9220421  -4.1825748   6.7279760   9.7697923  -2.2680118
-1.4675852  -4.1467232
 [17]   2.3124711  -0.5084914   1.2189552  -5.4507730   0.8833957   5.712139
3   8.6040158   3.9251731
 [25]   0.1408654  -0.2421535   1.8021988   6.9059413  -2.9899869   5.884516
2   5.7882127   0.4549209
 [33]   3.2173945   5.9488461  -8.4008063   1.9848157  -6.7620305  -4.245266
0   7.7259151  12.0157987
 [41]   0.8247840  -1.8840672  -0.8711188  -1.1562441  -0.1452258   5.605493
0   1.4691334  -0.7798185
 [49]   1.4174556  -2.7867247   4.8941814   2.4092869   7.0503349  -1.719229
1   1.2299401   5.5069650
 [57]   6.7198039   2.5493399  10.1465339  -5.6978424   1.2328360  -3.530642
4   1.3276670   2.9222687
 [65]   6.0555213   1.2892617   8.9218406   2.1986354   7.7021057  11.4782211
-8.6420307   5.4662976
 [73]   2.0243397  -1.5490605  -4.6810593   2.2895932   6.3986308  -2.394160
8   2.4381676   5.4335757
 [81]   1.1861843   7.8253730  -6.1254821   0.9058400  -4.8816110  -1.515915
2   0.8516775  14.0527168
 [89]  -1.9256379  -2.5163342   3.4102851   4.0670767   3.1036132   3.8173435
-1.5175565  -5.0231241
 [97] -10.8637146  10.3559799   9.1630922   5.8654150   1.5176632  -2.218765
1   1.3361065   7.1109243
[105]  -1.6239507  12.1596346   6.7832804  11.3834591  -0.4546509   2.905984
9   1.0848315   4.2061291
[113]   3.9861728   9.1567017   8.6065420 -10.0029244  -4.7722902  10.087456
5   4.3534147   4.2983081
[121]  -3.6117746   8.4581259   9.1672271  -1.4491255  -5.9895705  -0.640046
5   1.6834150   1.6098954
[129]   4.3366032   8.4634119  10.1288949   8.2186574  -6.0339900   0.668749
3   8.3500355  -4.2433700
[137]   0.2357259   8.3052500   9.1734571  14.9625218   0.8012166  -8.254983
2   2.3731231  -1.2792824
[145]   1.5248094  -6.9797877   7.1709825   7.6326944  -2.7137285   0.4119719
-0.2933123   0.5291898
[153]   0.4340023  -1.9192758   7.4290332  -0.3601979  -2.4970357 -16.2858807
-9.4098849  -0.5815011
[161]  -2.5793753  -2.2344229   3.7404356  -2.9959033   7.5521691   3.7522216
-4.0379238  -7.3478515
[169]   1.1469178  -9.2096826   6.7292191  -0.3057282   7.7205880  13.451959
2   7.9903481   7.7360708
[177]  -1.3272805   0.1684622   1.6361308  -9.0487713  -1.7606154   4.094911
7   4.1120162  -3.1113174
[185]   9.4085134  -0.6381390  -1.0111716  -7.5650826   0.3512714  -0.9912848
-9.8170853  -1.0971449
```

```
[193]   5.6418668   7.7160595   4.9546324   3.2676418   6.3410522   0.777757
4   0.7223251   6.6714634
[201]   3.1937460  -5.4311835  -1.1579731  -5.6983220  -7.1598780   5.828029
2   8.2290724   2.2181048
[209]  -4.7836858   3.0959064  -7.7626934  -6.3192193   2.6989860   4.673404
2   3.0751224   2.8220806
[217]   1.7187242  -3.6349062  -2.9840927  10.5054540  -3.8378719  -1.906755
0   6.3914991   0.1393172
[225]   7.2144018  -0.5998237  -6.9292021  -4.4566012  -0.4334702   3.1013454
-6.0489774  -2.2465411
[233]   5.2265747   3.4440658   5.5480500   7.1839697   6.0795985  -3.049238
6   5.4162937   2.0621714
[241]   1.6179636  -5.2382801   0.9001987  12.3557382   9.3136253   0.738792
8   3.2148443  -0.6281552
[249]   3.3014468  10.2528753
```

Hide

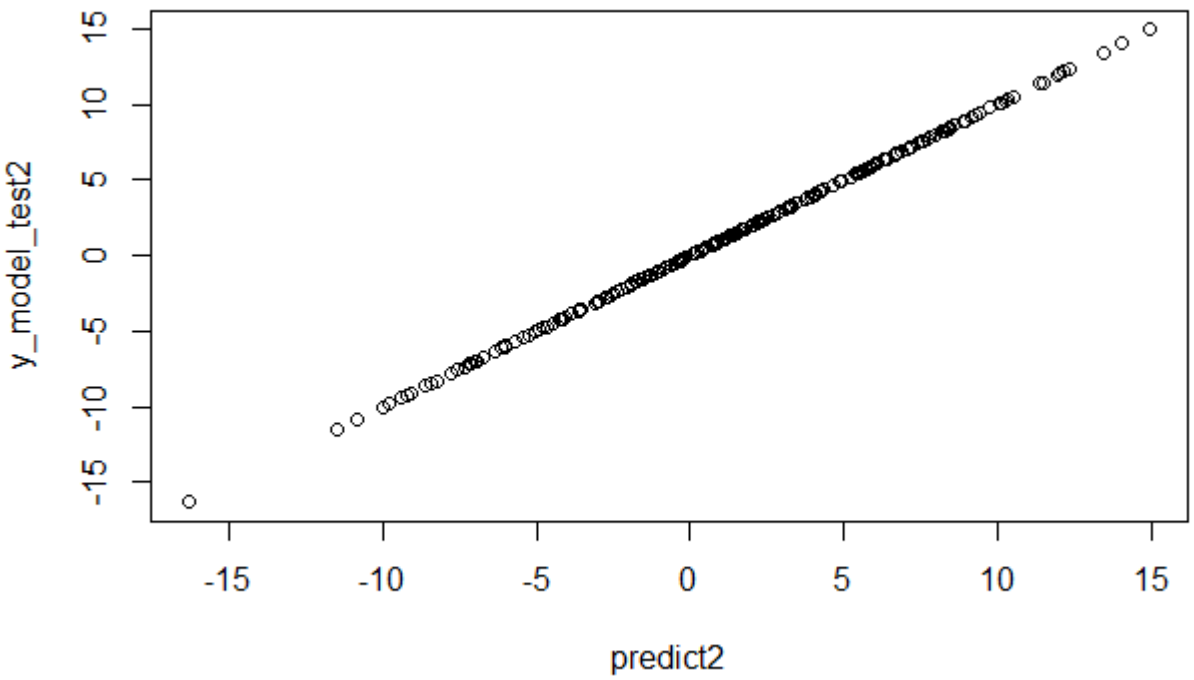
```
predict2 <- predict(linearmodel,test2)
str(predict2)
```

```
 Named num [1:250] 8.96 -2.05 11.97 -7.15 6.33 ...
 - attr(*, "names")= chr [1:250] "1" "2" "3" "4" ...
```

test2 dataset do not have y variable values, by using predict() function y variable's values are predicted by initially training the "linear model" with train2 dataset and then using this model to predict y variable of test2 dataset.

Hide

```
plot(predict2,y_model_test2)
```

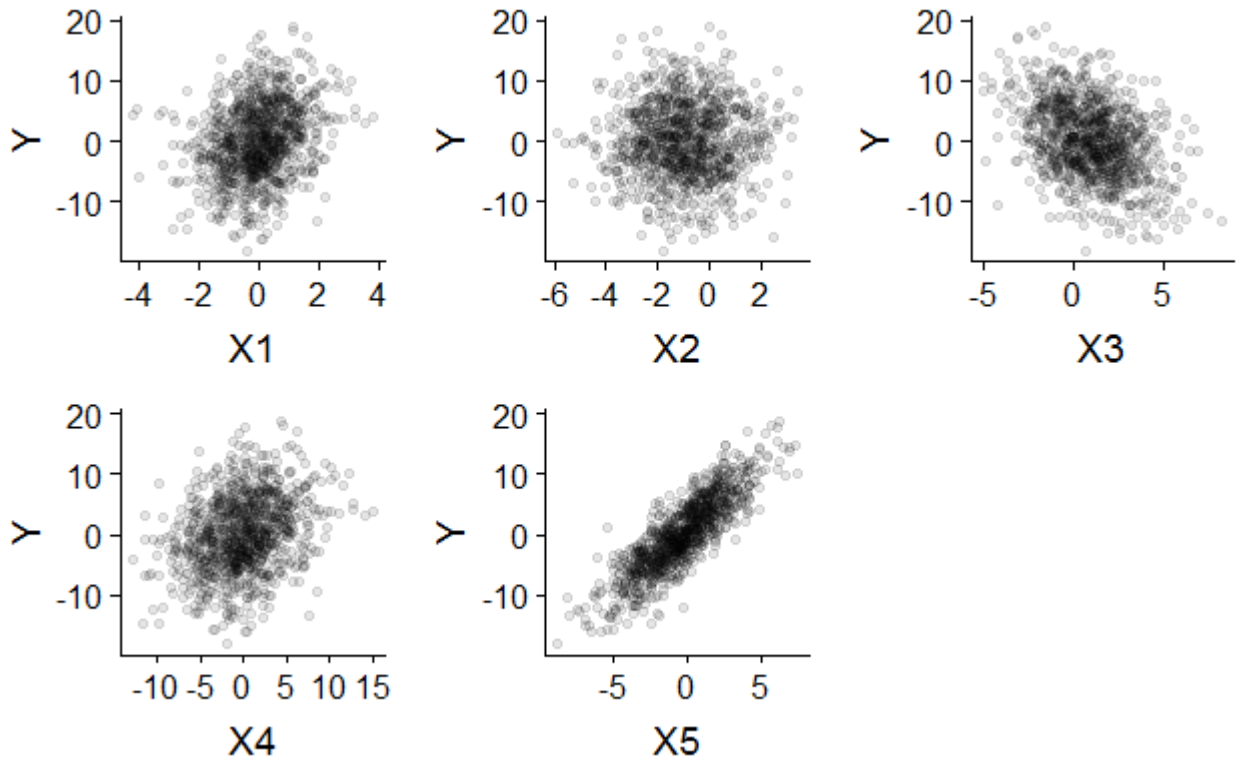


the plot above shows that the predicted “Y” variable is same as “Y” variable computed using basic R statistical function.

Putting all together for constructing linear model and prediction of “y” for train_group1 and test_group1 dataset.

Hide

```
# initializing the required libraries
library(ggplot2)
library(cowplot)
# loading the training_group1 and testing_group2 dataset
train1 <- read.csv("TrainData_Group1.csv",header=TRUE, sep = ",")
test1 <- read.csv("TestData_Group1.csv",header=TRUE, sep = ",")
# plotting Y vs X variables to look at their linear relationship
plota <- ggplot(train1, aes(X1, Y)) + geom_point(alpha=0.1)
plotb <- ggplot(train1, aes(X2, Y)) + geom_point(alpha=0.1)
plotc <- ggplot(train1, aes(X3, Y)) + geom_point(alpha=0.1)
plotd <- ggplot(train1, aes(X4, Y)) + geom_point(alpha=0.1)
plote <- ggplot(train1, aes(X5, Y)) + geom_point(alpha=0.1)
plots <- cowplot::plot_grid(plota,plotb,plotc,plotd,plote)
print(plots)
```



Hide

```
# creating X and Y matrix
x <- as.matrix(train1[-ncol(train1)])
int <- rep(1, nrow(x))
x <- cbind(int, x)
y  <- train1$Y
# calculating Betas/ Co-efficients and then linear model of ML algorithm and th
en comparing betas from both the methods
betas <- t(t(x) %*% y) %*% solve(t(x) %*% x)
colnames(betas) <- c("B0", "B1", "B2", "B3", "B4", "B5")
linearmodel <- lm(Y~.,data=train1)
comparison <- data.frame(B = t(betas), lm = linearmodel$coefficients)
print(comparison)
```

	B <dbl>	lm <dbl>
B0	2.127945088	2.127945088
B1	0.010831195	0.010831195
B2	-0.003680646	-0.003680646
B3	-1.305743458	-1.305743458
B4	0.397638392	0.397638392

	B <dbl>	lm <dbl>
B5	2.084182729	2.084182729

6 rows

Hide

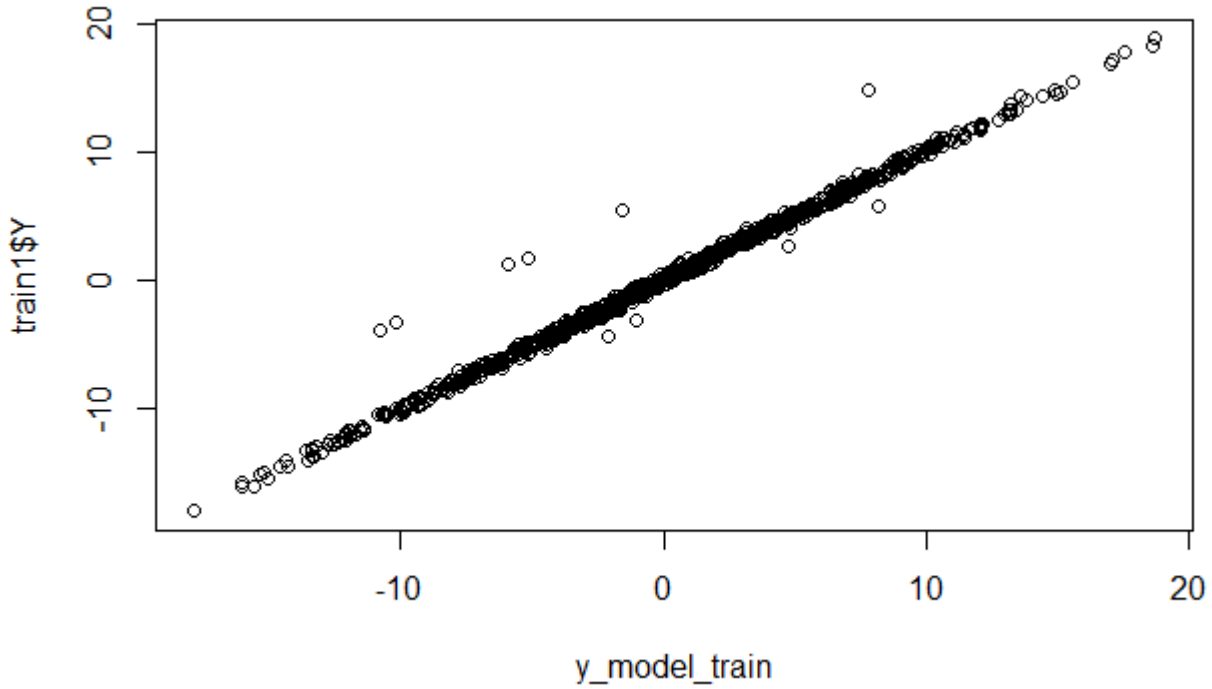
```
# The newly predicted "y" variable is compared with actual "y" variable of trai
n1 dataset.
y_model_train <- rep(0,nrow(train1))
for (i in 1:nrow(train1)){
  y_model_train[i] <- betas[1] + betas[2]*train1$X1[i] + betas[3]*train1$X2[i]
+ betas[4]*train1$X3[i] + betas[5]*train1$X4[i] + betas[6]*train1$X5[i]
}
res1 <- train1$Y - y_model_train
ss_res1 <- sum(res1**2)
ss_tot1 <- sum((y-mean(y))**2)
r_sq1 <- 1-(ss_res1/ss_tot1)
comparison_rsq <- data.frame(r_sq1, summary(linearmodel)[8])
print(comparison_rsq)
```

r_sq1 <dbl>	r.squared <dbl>
0.9899022	0.9899022

1 row

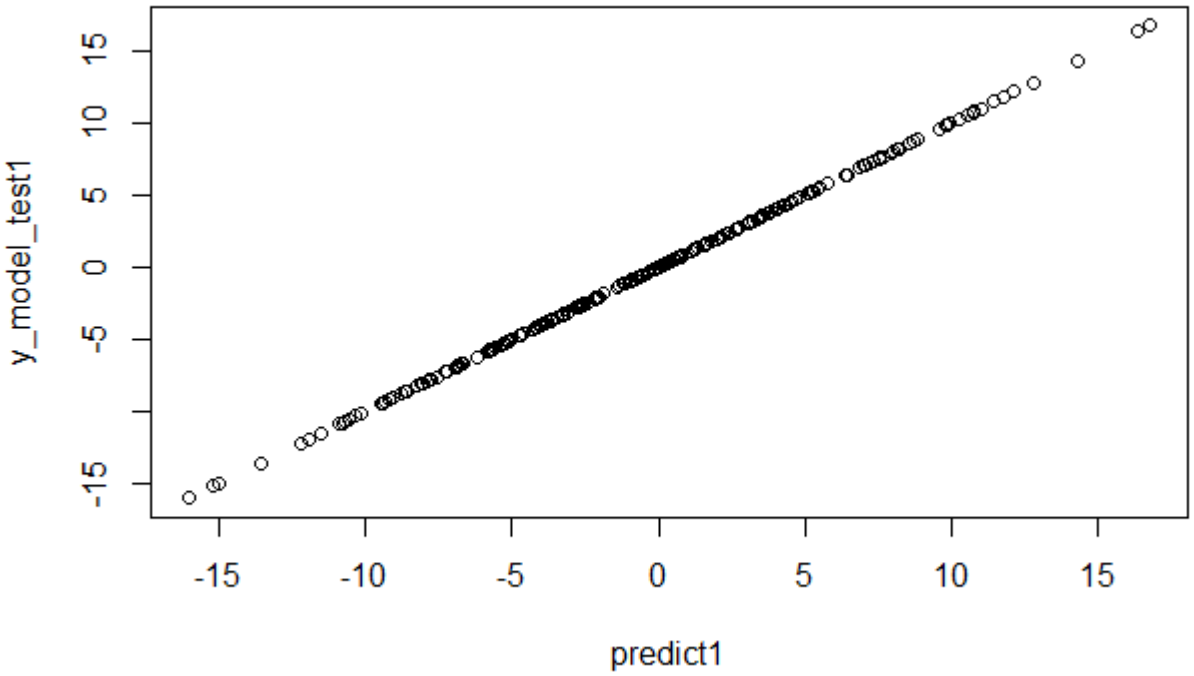
Hide

```
plot1 <- plot(y_model_train,train1$Y)
```



Hide

```
# using beta values to predict "y" variable for test1 dataset.
y_model_test1 <- rep(0,nrow(test1))
for (i in 1:nrow(test1)){
  y_model_test1[i] <- betas[1] + betas[2]*test1$X1[i] + betas[3]*test1$X2[i] +
betas[4]*test1$X3[i] + betas[5]*test1$X4[i] + betas[6]*test1$X5[i]
}
predict1 <- predict(linearmodel,test1)
plot_pred <- plot(predict1,y_model_test1)
```



Reference links: https://datascienceplus.com/linear-regression-from-scratch-in-r/ (https://datascienceplus.com/linear-regression-from-scratch-in-r/) algorithm https://www.youtube.com/watch?v=NUXdtN1W1FE (https://www.youtube.com/watch?v=NUXdtN1W1FE) linear regression https://www.youtube.com/watch?v=J_LnPL3Qg70&t=408s (https://www.youtube.com/watch?v=J_LnPL3Qg70&t=408s) python LR