LAB Manual

PART A

(PART A : TO BE REFFERED BY STUDENTS)

**Experiment No.04**

**A.1 Aim:**

To investigate and practice the Linear Regression method.

* Use R functions for Linear Regression (Ordinary Least Squares – OLS)
* Predict the dependent variables based on the model
* Investigate different statistical parameter tests that measure the effectiveness of the model

**A.2 Prerequisite:**

Understanding of Statistics and basic function of R Studio. Should know Linear Regression algorithm well.

**A.3 Outcome:**

**After successful completion of this experiment students will be able to**

1. Apply appropriate analytic techniques and tools to analyze big data, create statistical models and identify insights leading to actionable results.
2. Mine given data set using data mining tool.

**A.4 Theory**

1. **Prepare working environment for the Lab and load data files**
2. Set the working directory to where we have stored the data. On the console window type:
3. **Use random number generators to create data for the OLS Model:**

1. Run the “runif” function in R which generates random deviates within the specified minimum and maximum range.

**x <- runif(100, 0, 10)**

This generates 100 random values for “x” in the range 0 to 10.

2. Create the dependent variable “y” with the “beta” values as 5 and 6 and the “sigma” = 1 (generated with the “rnorm” function, random generation for the normal distribution with mean =0 and SD= 1.)

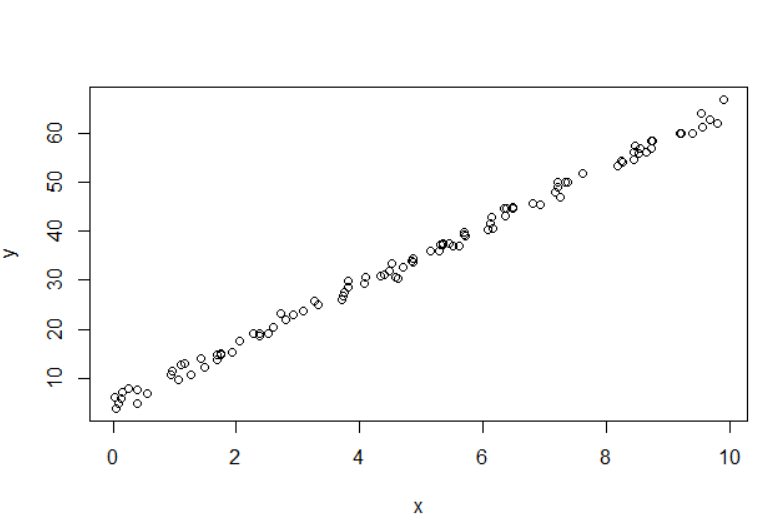
**y <- 5 + 6\*x + rnorm(100)**

3. Plot it

**plot(x,y)**

Review the results in the graphics window





1. **Generate the OLS model using R function “lm”:**

An OLS Model is generated with an R function call “lm”.  
You can learn about “lm” with the following command on the console:

**?lm**

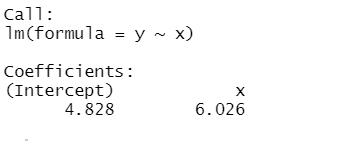
1. Generate an OLS Model using the following command:

**d <- lm(y ~ x)**

2. Use the following command to display the structure of the object “d” created with the function call “lm”

**str(d)**

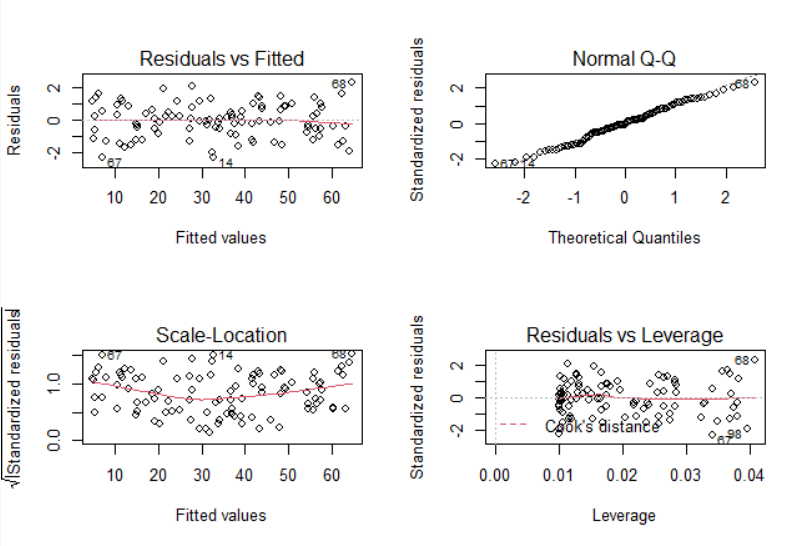
1. You can see the details of the fitted model. What are the values of the coefficients (Beta) in the model?

**

1. **Print and visualize the results and review the plots generated:**
2. Get the compact results of the model with the following command: **print(d)**
3. Visualize the model with the command:

**par(mfrow=c(2,2))**

**plot(d)**



The explanation of the plots are as follows:

**Residuals vs. Fitted:** you want to make sure that the errors are evenly distributed over the entire range of fitted values; if the errors are markedly larger (or biased either positively or negatively) in some range of the data, this is evidence that this model may not be entirely appropriate for the problem.

**Q-Q plot:** tests whether or not the errors are in fact distributed approximately normally (as the model formulation assumes). If they are, the Q-Q plot will be along the x=y line. If they aren't, the model may still be adequate, but perhaps a more robust modeling method is suitable. Also, the usual diagnostics (R-squared, t-tests for significance) will not be valid.

**Scale-Location:** a similar idea to Residuals v. Fitted; you want to make sure that the variance (or its stand-in, "scale") is approximately constant over the range of fitted values.

**Residuals vs. Leverage:** used for identifying potential outliers and "influential" points. Points that are far from the centroid of the scatterplot in the x direction (high leverage) are influential, in the sense that they may have disproportionate influence on the fit (that doesn't mean they are wrong, necessarily). Points that are far from the centroid in the y direction (large residuals) are potential outliers.

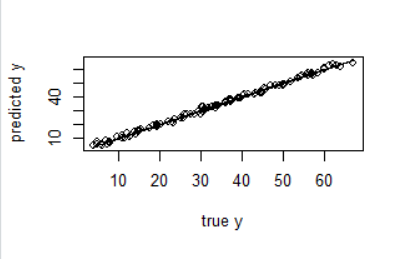
1. Here are some examples of plots that may be a little more intuitive, Type in the following:

**> ypred <- predict(d)**

**> par(mfrow=c(1,1))**

**> plot(y,y, type="l", xlab="true y", ylab="predicted y")**

**> points(y, ypred)**



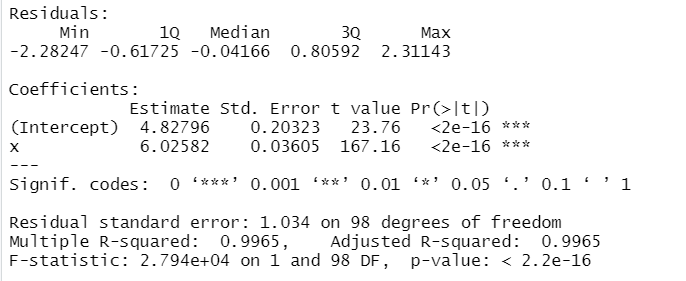
Review the results in the graphics window. The plot of predicted vs. true outcome can be seen there. The plot should be near the x=y line. Where it does not run along the x=y line indicates where the model tends to over-predict or under-predict. You can also use this plot to identify ranges where the errors are especially large. This information is similar to the Residuals vs. Fitted plot, but perhaps is more intuitive to the layperson.

Note: The “predict” function requires the variables to be named exactly as in the fitted model.

1. **Generate Summary Outputs:**
2. For more detailed results type:

**d1 <- summary(d)**

**print(d1)**

****

Read the explanations given below from the summary output and note the values from the output on the console for each statistic detailed:

**coefficients :** the estimated value of each coefficient, along with the standard error. coefficient +/- 2\*std.error is useful as a quick measure of confidence interval around the estimate.

**t-value:** coefficient/std.error, or how tight an estimate this is (compared to 0). If the "true" coefficient is zero (meaning this variable has no effect on the outcome), t-value is "small".

**Pr(>|t|):** the probability of observing this t-value if the coefficient is actually zero. You want this probability to be small. How small depends on the significance desired. Standard significances are given by the significance codes. So, for example "\*\*\*" means that the probability that this coefficient is really zero is negligible.

**R-squared:** A goodness of fit measure: the square of the correlation between predicted response and the true response. You want it close to 1. Adjusted R-squared compensates for the fact that having more parameters tends to increase R-squared. Since we only have one variable here, the two are the same.

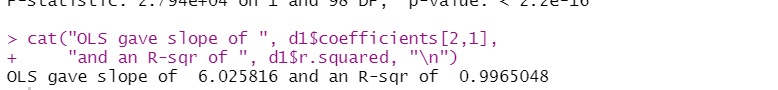
**F-statistic and p-value.** Used to determine if this model is actually doing better than just guessing the mean value of y as the prediction (the "null model"). If the linear model is really just estimating the same as the null model, then the F-statistic should be about 1. The p-value is the probability of seeing an F-statistic this large, if the true value is 1. Obviously, you want this value to be very small.

1. Type in the following command:

**> cat("OLS gave slope of ", d1$coefficients[2,1],**

**"and an R-sqr of ", d1$r.squared, "\n")**

Note the result you see on the console in the space below:



1. **Introduce a slight non-linearity and test the model:**
2. Create a “training” data set  
   First the training set in which we will introduce a slight non-linearity.

**> x1 <- runif(100)**

**> # introduce a slight nonlinearity**

**> y1 = 5 + 6\*x1 + 0.1\*x1\*x1 + rnorm(100)**

2. Generate the model

**> m <- lm(y1 ~ x1)**

1. Create the test data set

**> x2 <- runif(100)**

**> y2 = 5 + 6\*x2 + 0.1\*x2\*x2 + rnorm(100)**

1. Repeat steps 5 and 6 for model “m” and compare what you observe with the "ideal" in the earlier steps.
2. What's the R2 on the model?

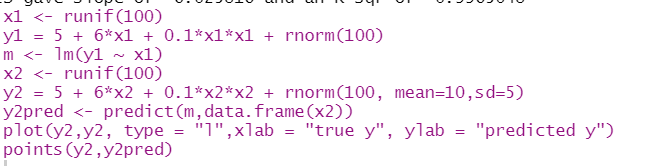
Notice (from the ypred v. y plot) that the model tends to under-predict for higher values of y.

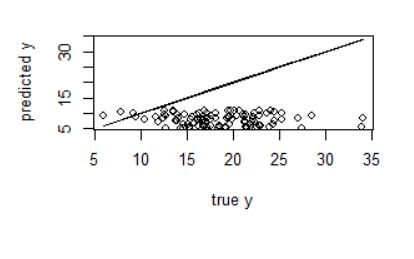
6. Use the predict function on the test data we generated:

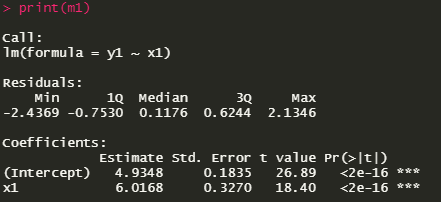
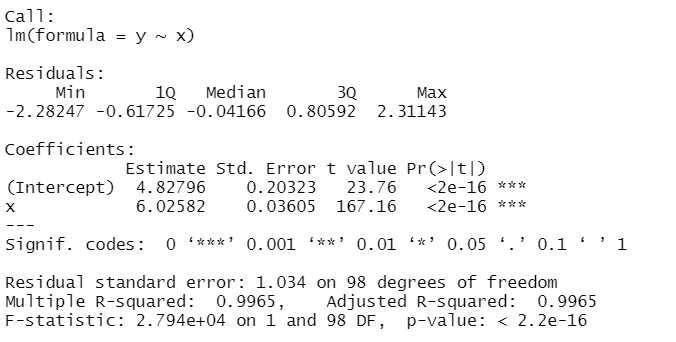
**y2pred <- predict(m,data.frame(x2))**

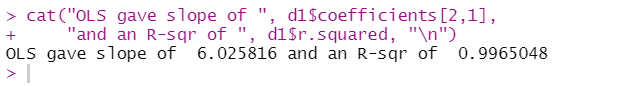
1. compare y2pred with the true outcomes y2
2. What did you observe from the comparison?

Case 1: MEAN = 5 , STANDARD DEVIATION = 2 in the testing data

****

****

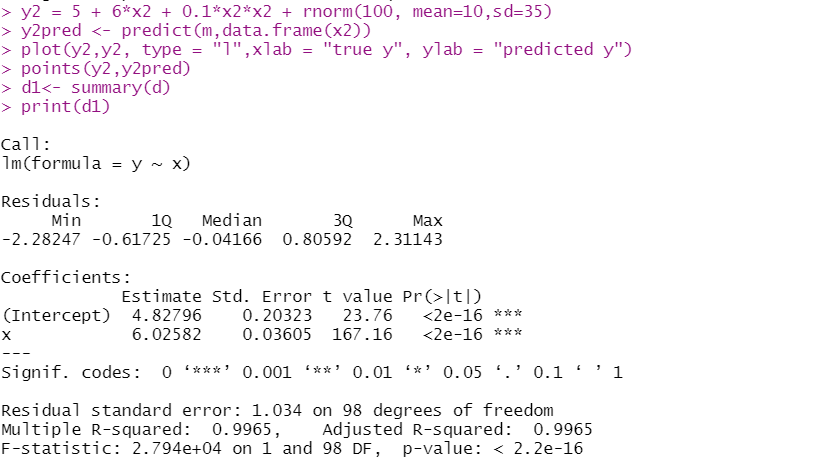
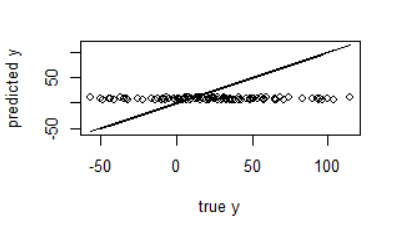
**** 

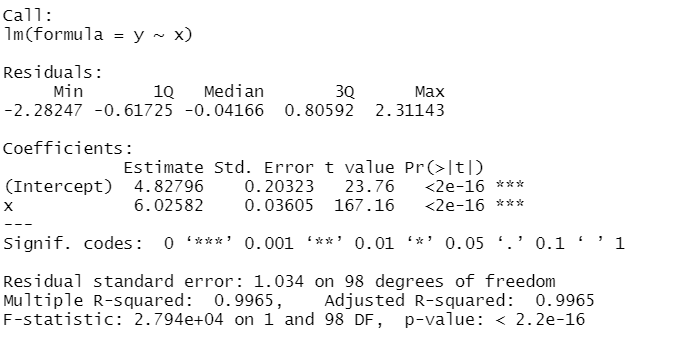


**R square is 0.9965048.** This Sems true as earlier we had fit the model on a linear model where linear regression is best suited for. Here we have use a squared function model ie y is related to x as a 2nd degree polynomial equation. We have fit the model to a polynomial training data where the model will try to fit a straight line to a data distribution which is not linearly dependent but will obviously give a larger residual error. This explains the R square value to be lower as the R square value determines how good the model is fitted to the train data.

In the graph between true y and pred y, ideally it should follow that straight line but the points are scattered indicating lots of wrong prediction. However I have not provided any parameters in the rnorm() in train data therefore the mean is 0 and standard deviation is 1 so we can say that the distribution is still uniform and the values wont vary much. In case 3, we will see the results of these parameters in the train data.

Case 2: MEAN = 5 , STANDARD DEVIATION = 35 in the testing data

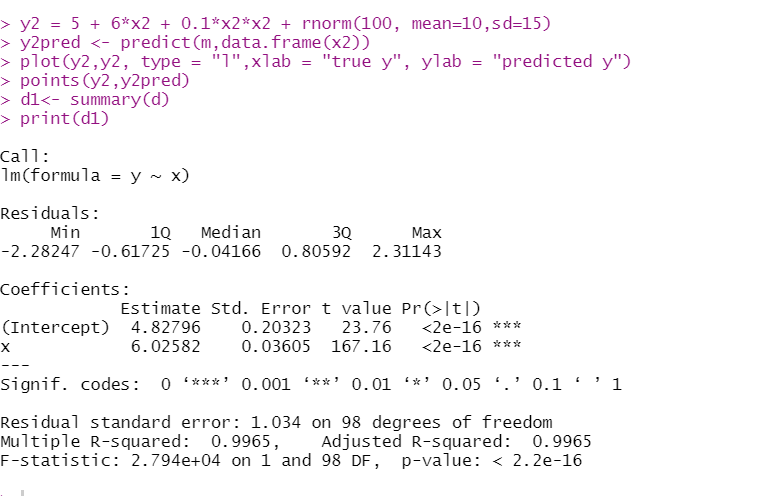
 

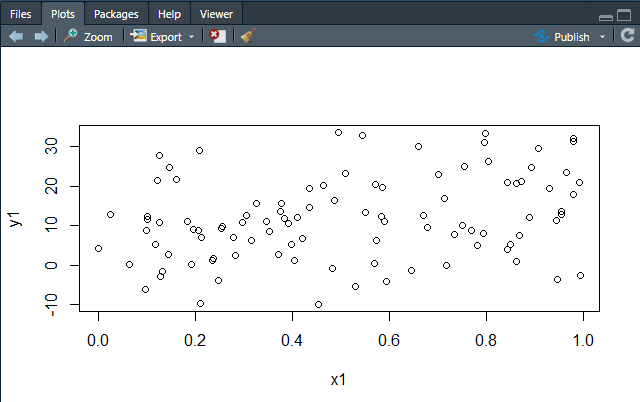


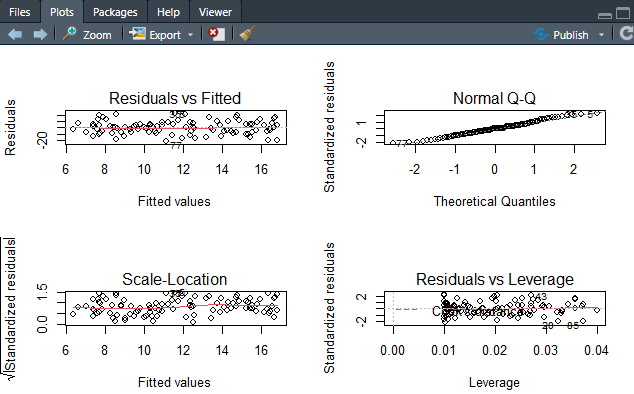
I have varied the standard deviation in testing data but the training data is kept same. However I re ran the training data line so the data is different from case 1 but still gives similar **R square of 0.9965** which means the model we trained on is very similar. However change in the parameters in testing data will affect the distribution even more now as the standard deviation is 25. So a lot more variation in testing data’s true Y values and so lot of error in predictions.

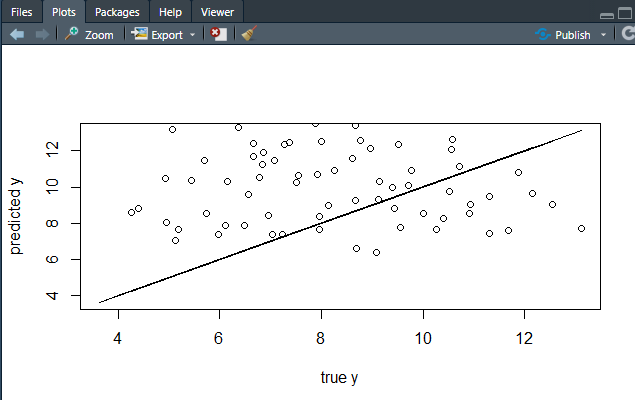
This is seen in the graph between true and predicted y where for every testing data’s X , it outputs a predicted y around 0

Case 3: MEAN = 5 , STANDARD DEVIATION = 10 in the training data









In this case, we add mean and standard deviation to the train data and no additional parameters in test data. Due to the standard deviation 10 , more variation in values are added to the training data’s Y target values. As a result , the model fails to learn a proper pattern in the training data leading to the extremely low **R square value 0.0877**. The model is extremely unconfident about the pattern it learnt and will lead to extreme errors in predicted data. In the graph between true y and predicted y, the points are pretty much random about the straight line indicating high errors. Also note the significance codes of slope and intercept are low – 2 stars (ie poor confidence over the model it trained for the training data which showed lots of variation)

**CONCLUSION::**

**These 3 cases are classic examples where training and testing data are of different distribution. This problem should be avoided at all costs.**