**Dez pressupostos da Regressão Linear com o código do R.**

http://r-statistics.co/Assumptions-of-Linear-Regression.html

**Assumptions of Linear Regression**

Building a linear regression model is only half of the work. In order to actually be usable in practice, the model should conform to the assumptions of linear regression.

**Assumption 1**

***The regression model is linear in parameters***

An example of model equation that is *linear in parameters*   
*Y* = *a* + (*β*1\**X*1) + (*β*2\**X*22)

Though, the X2 is raised to power 2, the equation is still linear in beta parameters. So the assumption is satisfied in this case.

**Assumption 2**

***The mean of residuals is zero***

**How to check?**

Check the mean of the residuals. If it zero (or very close), then this assumption is held true for that model. This is default unless you explicitly make amends, such as setting the intercept term to zero.

mod <- lm(dist ~ speed, data=cars)

mean(mod$residuals)

#=> 2.442491e-17

Since the mean of residuals is approximately zero, this assumption holds true for this model.

**Assumption 3**

***Homoscedasticity of residuals or equal variance***

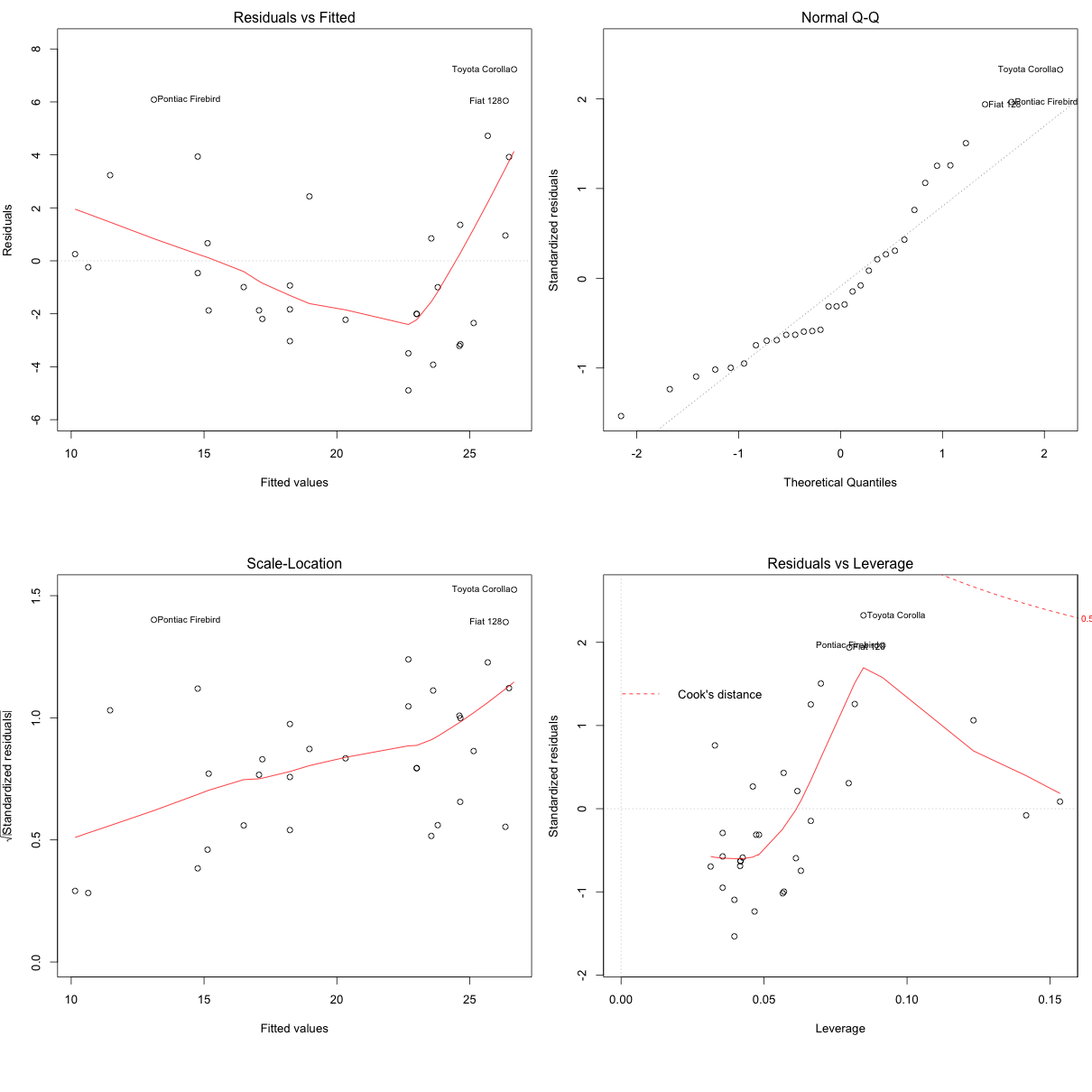
**How to check?**

Once the regression model is built, set par(mfrow=c(2, 2)), then, plot the model using plot(lm.mod). This produces four plots. The *top-left* and *bottom-left* plots shows how the *residuals* vary as the *fitted* values increase.

par(mfrow=c(2,2)) # set 2 rows and 2 column plot layout

mod\_1 <- lm(mpg ~ disp, data=mtcars) # linear model

plot(mod\_1)

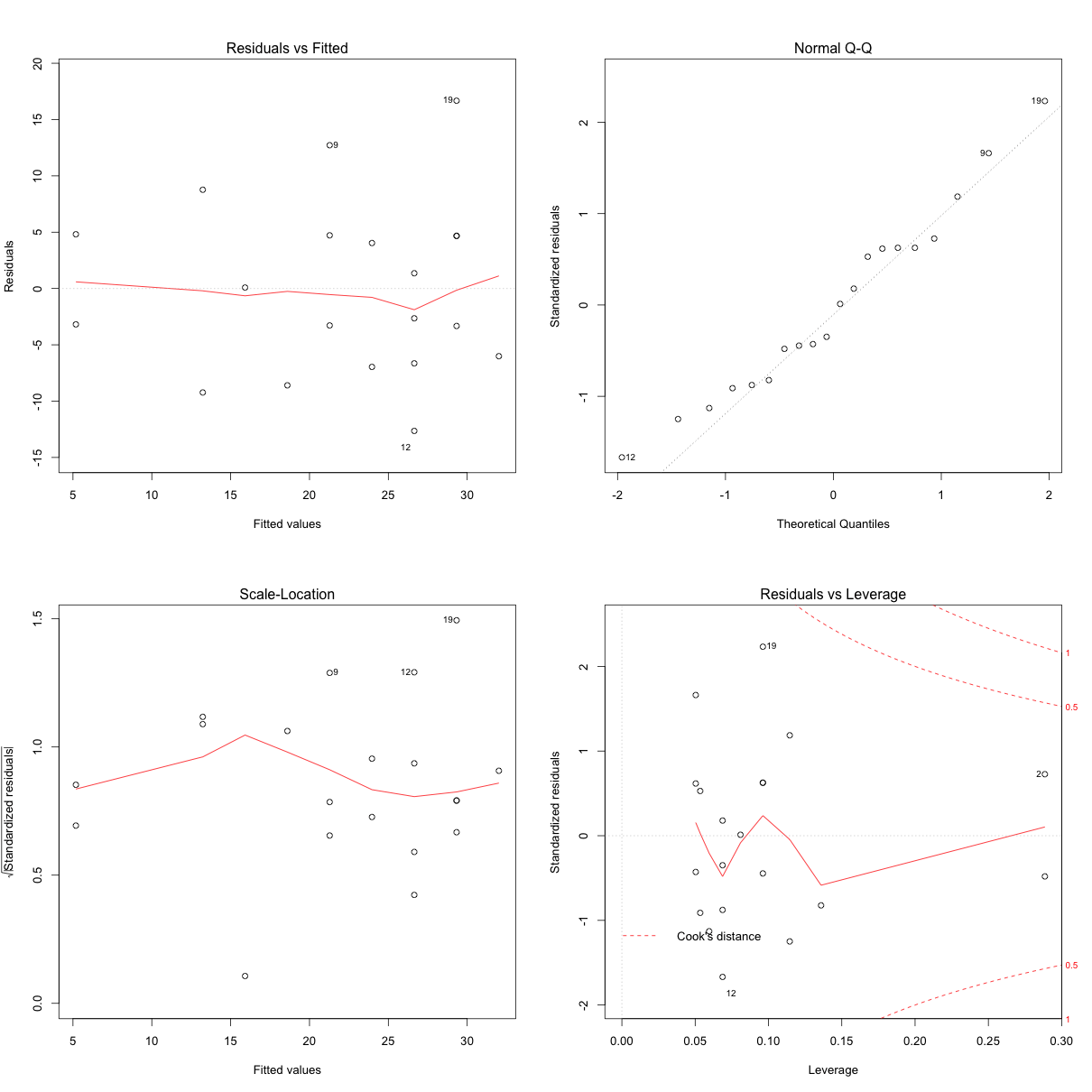


From the first plot (top-left), as the fitted values along x increase, the residuals decrease and then increase. This pattern is indicated by the red line, which should be approximately flat if the disturbances are homoscedastic. The plot on the bottom left also checks this, and is more convenient as the disturbance term in Y axis is standardized.

In this case, there is a definite pattern noticed. So, there is heteroscedasticity. Lets check this on a different model.

mod <- lm(dist ~ speed, data=cars[1:20, ]) # linear model

plot(mod)



Now, the points appear random and the line looks pretty flat, with no increasing or decreasing trend. So, the condition of homoscedasticity can be accepted.

**Assumption 4**

***No autocorrelation of residuals***

This is applicable especially for time series data. Autocorrelation is the correlation of a time Series with lags of itself. When the residuals are autocorrelated, it means that the current value is dependent of the previous (historic) values and that there is a definite unexplained pattern in the Y variable that shows up in the disturbances.

**How to check for autocorrelation?**

Below, are 3 ways you could check for autocorrelation of residuals.

**Using acf plot**

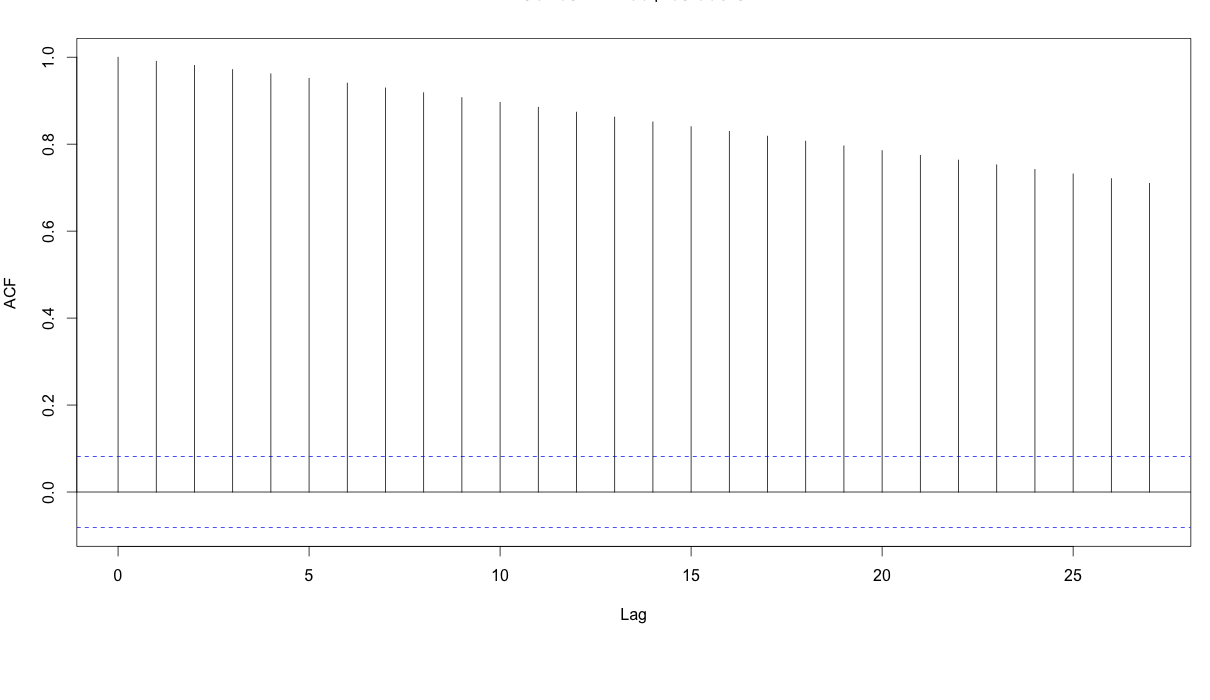
# Method 1: Visualise with acf plot

library(ggplot2)

data(economics)

lmMod <- lm(pce ~ pop, data=economics)

acf(lmMod$residuals) # highly autocorrelated from the picture.



The X axis corresponds to the lags of the residual, increasing in steps of 1. The very first line (to the left) shows the correlation of residual with itself (Lag0), therefore, it will always be equal to 1.

If the residuals were not autocorrelated, the correlation (Y-axis) from the immediate next line onwards will drop to a near zero value below the dashed blue line (significance level). Clearly, this is not the case here. So we can conclude that the residuals are autocorrelated.

**Using runs test**

# Method 2: Runs test to test for randomness

lawstat::runs.test(lmMod$residuals)

#=> Runs Test - Two sided

#=> data: lmMod$residuals

#=> Standardized Runs Statistic = -23.812, p-value < 2.2e-16

With a p-value < 2.2e-16, we reject the null hypothesis that it is random. This means there is a definite pattern in the residuals.

**Using Durbin-Watson test.**

# Method 3: Durbin-Watson test

lmtest::dwtest(lmMod)

#=> Durbin-Watson test

#=> data: lmMod

#=> DW = 0.0021559, p-value < 2.2e-16

#=> alternative hypothesis: true autocorrelation is greater than 0

So, dwtest also confirms our finding.

**How to rectify?**

Add lag1 of residual as an X variable to the original model. This can be conveniently done using the slide function in DataCombine package.

library(DataCombine)

econ\_data <- data.frame(economics, resid\_mod1=lmMod$residuals)

econ\_data\_1 <- slide(econ\_data, Var="resid\_mod1", NewVar = "lag1", slideBy = -1)

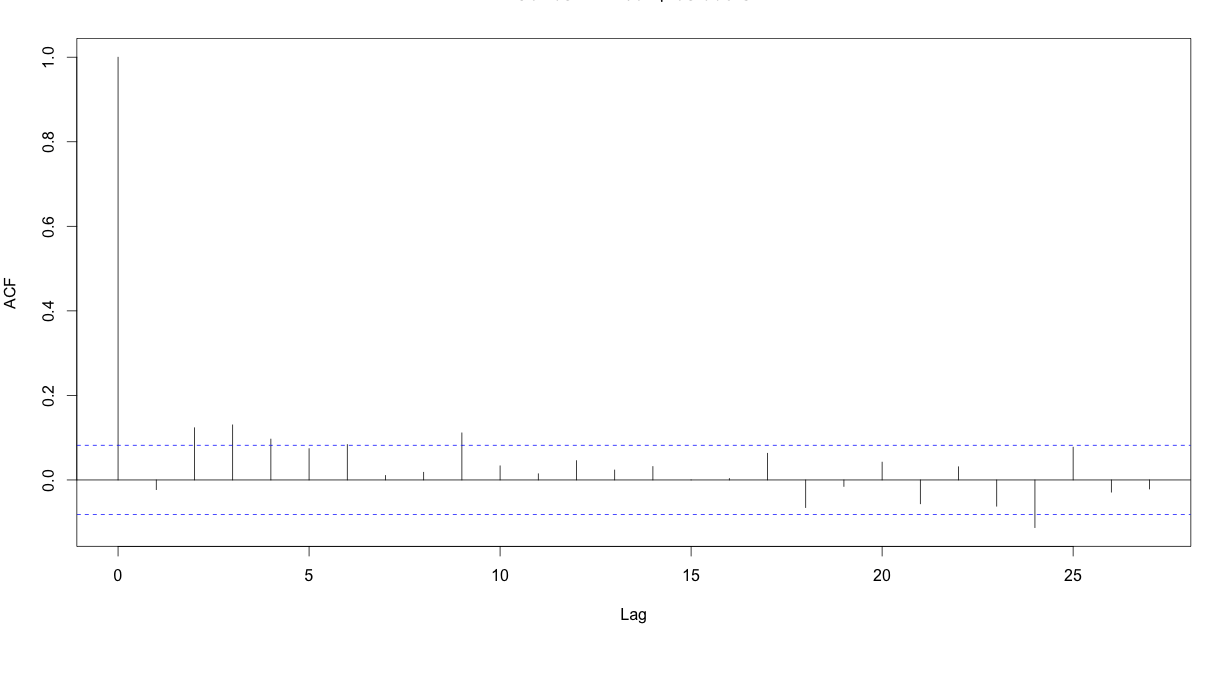
econ\_data\_2 <- na.omit(econ\_data\_1)

lmMod2 <- lm(pce ~ pop + lag1, data=econ\_data\_2)

Lets check if the problem of autocorrelation of residuals is taken care of using this method.

**Check 1**

acf(lmMod2$residuals)



Unlike the acf plot of lmMod, the correlation values drop below the dashed blue line from lag1 itself. So autocorrelation can’t be confirmed.

**Check 2**

runs.test(lmMod2$residuals) # runs test

#=> Runs Test - Two sided

#=>

#=> data: lmMod2$residuals

#=> Standardized Runs Statistic = 0.96176, p-value = 0.3362

p-value = 0.3362. Can’t reject null hypothesis that it is random. With a p-value = 0.3362, we cannot reject the null hypothesis. Therefore we can safely assume that residuals are not autocorrelated.

**Check 3**

lmtest::dwtest(lmMod2)

#=> Durbin-Watson test

#=>

#=> data: lmMod2

#=> DW = 2.0431, p-value = 0.6672

#=> alternative hypothesis: true autocorrelation is greater than 0

With a high p value of 0.667, we cannot reject the null hypothesis that true autocorrelation is zero. So the assumption that residuals should not be autocorrelated is satisfied by this model.

If, even after adding lag1 as an X variable, does not satisfy the assumption of autocorrelation of residuals, you might want to try adding lag2, or be creative in making meaningful derived explanatory variables or interaction terms. This is more like art than an algorithm.

**Assumption 5**

***The X variables and residuals are uncorrelated***

**How to check?**

Do a correlation test on the X variable and the residuals.

mod.lm <- lm(dist ~ speed, data=cars)

cor.test(cars$speed, mod.lm$residuals) # do correlation test

#=> Pearson's product-moment correlation

#=>

#=> data: cars$speed and mod.lm$residuals

#=> t = -8.1225e-17, df = 48, p-value = 1

#=> alternative hypothesis: true correlation is not equal to 0

#=> 95 percent confidence interval:

#=> -0.2783477 0.2783477

#=> sample estimates:

#=> cor

#=> -1.172376e-17

p-value is high, so null hypothesis that true correlation is 0 can’t be rejected. So, the assumption holds true for this model.

**Assumption 6**

***The number of observations must be greater than number of Xs***

This can be directly observed by looking at the data.

**Assumption 7**

***The variability in X values is positive***

This means the X values in a given sample must not all be the same (or even nearly the same).

**How to check?**

var(cars$speed)

#=> [1] 27.95918

The variance in the X variable above is much larger than 0. So, this assumption is satisfied.

**Assumption 8**

***The regression model is correctly specified***

This means that if the Y and X variable has an inverse relationship, the model equation should be specified appropriately:

*Y*=*β*1+*β*2∗(1*X*)

**Assumption 9**

***No perfect multicollinearity***

There is no perfect linear relationship between explanatory variables.

**How to check?**

Using Variance Inflation factor (VIF). But, What is VIF?

VIF is a metric computed for every *X* variable that goes into a linear model. If the VIF of a variable is high, it means the information in that variable is already explained by other *X* variables present in the given model, which means, more redundant is that variable. So, lower the VIF (<2) the better. VIF for a *X* var is calculated as:

*VIF*=1(1−*Rsq*)

where, *Rsq* is the *Rsq* term for the model with given *X* as response against all other *Xs* that went into the model as predictors.

Practically, if two of the *X*′*s* have high correlation, they will likely have high VIFs. Generally, VIF for an *X* variable should be less than 4 in order to be accepted as not causing multi-collinearity. The cutoff is kept as low as 2, if you want to be strict about your *X* variables.

library(car)

mod2 <- lm(mpg ~ ., data=mtcars)

vif(mod2)

# cyl disp hp drat wt qsec vs am gear carb

# 15.373833 21.620241 9.832037 3.374620 15.164887 7.527958 4.965873 4.648487 5.357452 7.908747

**How to rectify?**

Two ways:

1. Either iteratively remove the *X* var with the highest VIF or,
2. See correlation between all variables and keep only one of all highly correlated pairs.

library(corrplot)

corrplot(cor(mtcars[, -1]))

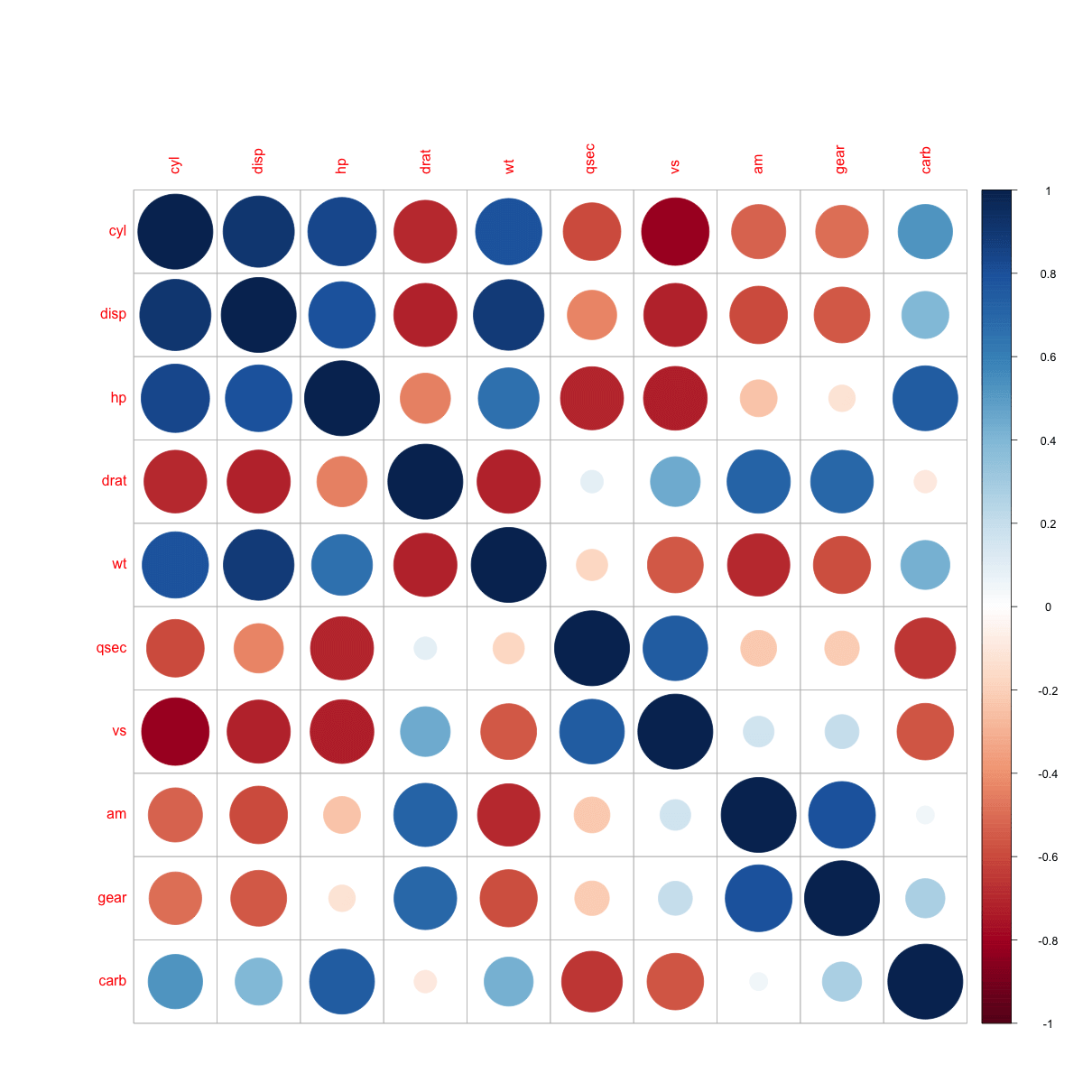
#=> Interpreted from below plot.

#=> Correlated pairs:

#=> - disp, cyl, hp, wt

#=> - gear, am

#=> - hp, carb



mod <- lm(mpg ~ cyl + gear, data=mtcars)

vif(mod)

#=> cyl gear

#=> 1.320551 1.320551

The convention is, the VIF should not go more than 4 for any of the X variables. That means we are not letting the *RSq* of any of the *Xs* (the model that was built with that X as a response variable and the remaining *Xs* are predictors) to go more than 75%. => 1/(1-0.75) => 1/0.25 => 4.

**Assumption 10**

***Normality of residuals***

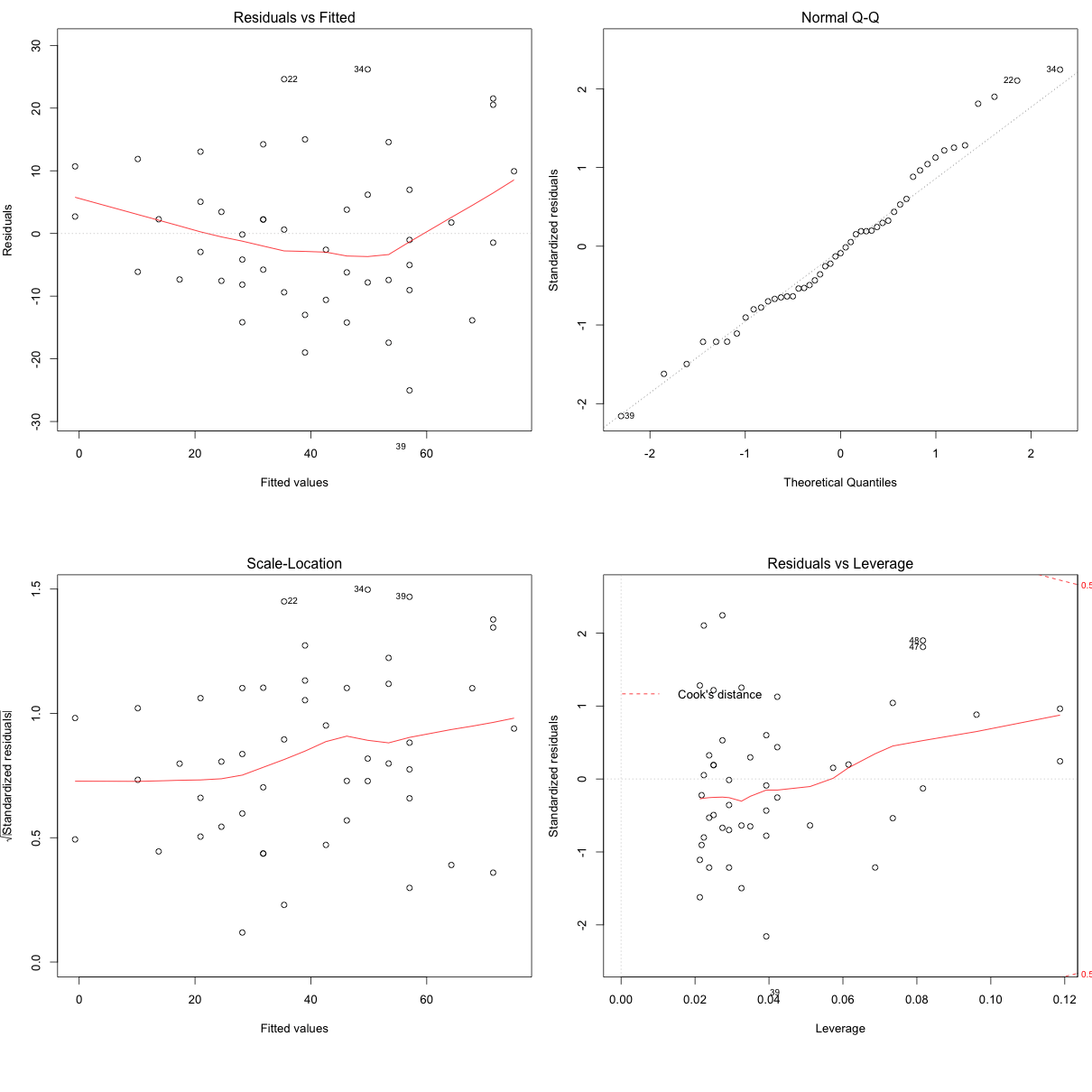
The residuals should be normally distributed. If the maximum likelihood method (not OLS) is used to compute the estimates, this also implies the *Y* and the *Xs* are also normally distributed.

This can be visually checked using the qqnorm() plot (top right plot).

par(mfrow=c(2,2))

mod <- lm(dist ~ speed, data=cars)

plot(mod)



The qqnorm() plot in top-right evaluates this assumption. If points lie exactly on the line, it is perfectly normal distribution. However, some deviation is to be expected, particularly near the ends (note the upper right), but the deviations should be small, even lesser that they are here.

**Check Assumptions Automatically**

The gvlma() function from gvlma offers a way to check the important assumptions on a given linear model.

par(mfrow=c(2,2)) # draw 4 plots in same window

mod <- lm(dist ~ speed, data=cars)

gvlma::gvlma(mod)

#=> Value p-value Decision

#=> Global Stat 15.801 0.003298 Assumptions NOT satisfied!

#=> Skewness 6.528 0.010621 Assumptions NOT satisfied!

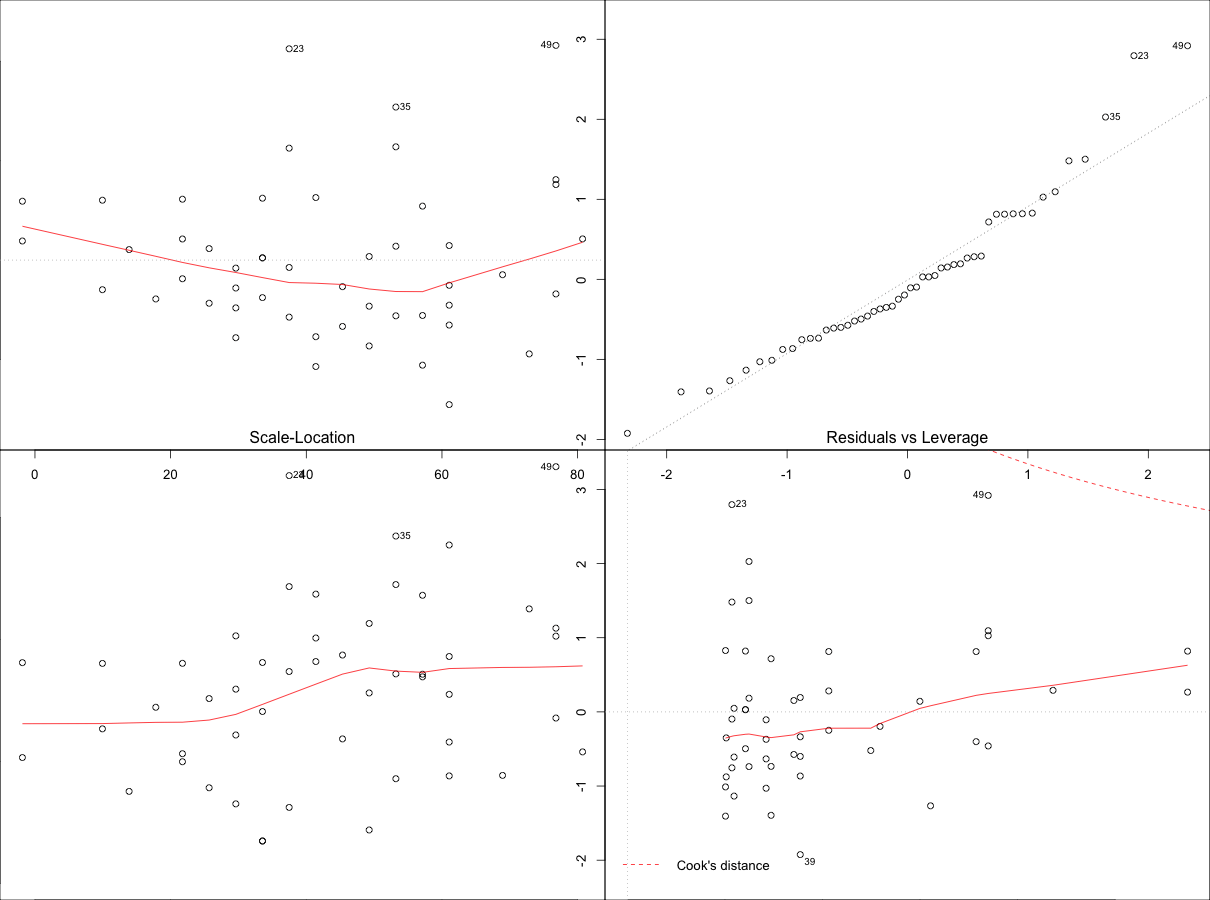
#=> Kurtosis 1.661 0.197449 Assumptions acceptable.

#=> Link Function 2.329 0.126998 Assumptions acceptable.

#=> Heteroscedasticity 5.283 0.021530 Assumptions NOT satisfied!

plot(mod)

Three of the assumptions are not satisfied. This is probably because we have only 50 data points in the data and having even 2 or 3 outliers can impact the quality of the model. So the immediate approach to address this is to remove those outliers and re-build the model. Take a look at the diagnostic plot below to arrive at your own conclusion.



From the above plot the data points: 23, 35 and 49 are marked as outliers. Lets remove them from the data and re-build the model.

mod <- lm(dist ~ speed, data=cars[-c(23, 35, 49), ])

gvlma::gvlma(mod)

#=> Value p-value Decision

#=> Global Stat 7.5910 0.10776 Assumptions acceptable.

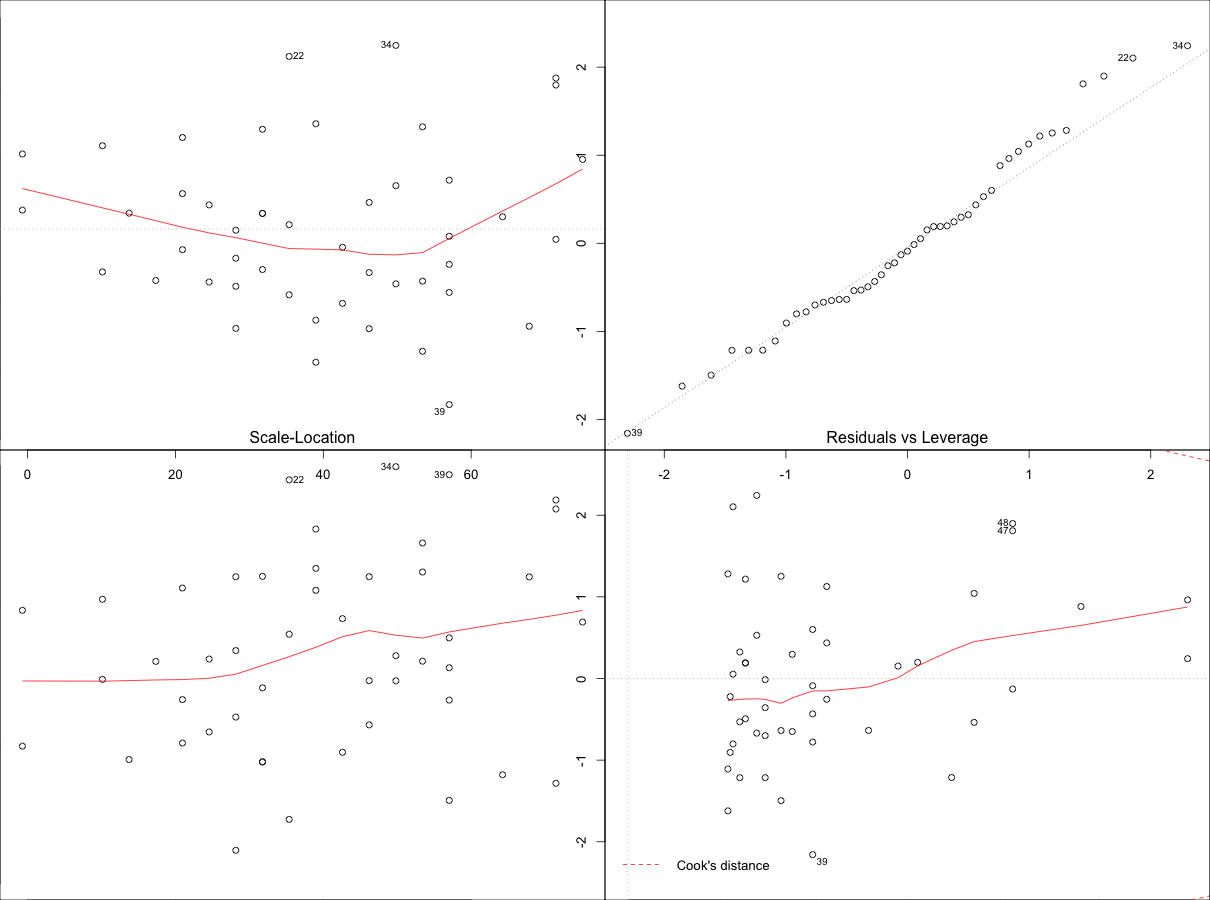
#=> Skewness 0.8129 0.36725 Assumptions acceptable.

#=> Kurtosis 0.2210 0.63831 Assumptions acceptable.

#=> Link Function 3.2239 0.07257 Assumptions acceptable.

#=> Heteroscedasticity 3.3332 0.06789 Assumptions acceptable.

#=> plot(mod)



Though the changes look minor, it is more closer to conforming with the assumptions. There is one more thing left to be explained. That is, the plot in the bottom right. It is the plot of standardized residuals against the leverage. Leverage is a measure of how much each data point influences the regression. The plot also contours values of [Cook’s distance](https://en.wikipedia.org/wiki/Cook%27s_distance), which reflects how much the fitted values would change if a point was deleted.

A point far from the centroid with a large residual can severely distort the regression. For a good regression model, the red smoothed line should stay close to the mid-line and no point should have a large cook’s distance (i.e. should not have too much influence on the model.)

#############################################################################

http://r-statistics.co/Logistic-Regression-With-R.html

**Logistic Regression**

If linear regression serves to predict continuous Y variables, logistic regression is used for binary classification.

If we use linear regression to model a dichotomous variable (as *Y*), the resulting model might not restrict the predicted *Ys* within 0 and 1. Besides, other [assumptions of linear regression](http://r-statistics.co/Assumptions-of-Linear-Regression.html) such as normality of errors may get violated. So instead, we model the log odds of the event *ln*(*P*1−*P*)

, where, *P* is the probability of event.

*Zi*=*ln*(*Pi*1−*Pi*)=*β*0+*β*1*x*1+..+*βnxn*

The above equation can be modeled using the glm() by setting the family argument to "binomial". But we are more interested in the probability of the event, than the log odds of the event. So, the predicted values from the above model, i.e. the log odds of the event, can be converted to probability of event as follows:

*Pi*=1−(11+*ezi*)

This conversion is achieved using the plogis() function, as shown below when we [build logit models and predict](http://r-statistics.co/Logistic-Regression-With-R.html#Build%20Logit%20Models%20and%20Predict).

**Example Problem**

Lets try and predict if an individual will earn more than $50K using logistic regression based on demographic variables available in the [adult data](http://rstatistics.net/wp-content/uploads/2015/09/adult.csv). In this process, we will:

1. Import the data
2. Check for class bias
3. Create training and test samples
4. Compute information value to find out important variables
5. Build logit models and predict on test data
6. Do model diagnostics

**Import data**

inputData <- read.csv("http://rstatistics.net/wp-content/uploads/2015/09/adult.csv")

head(inputData)

#=> AGE WORKCLASS FNLWGT EDUCATION EDUCATIONNUM MARITALSTATUS

#=> 1 39 State-gov 77516 Bachelors 13 Never-married

#=> 2 50 Self-emp-not-inc 83311 Bachelors 13 Married-civ-spouse

#=> 3 38 Private 215646 HS-grad 9 Divorced

#=> 4 53 Private 234721 11th 7 Married-civ-spouse

#=> 5 28 Private 338409 Bachelors 13 Married-civ-spouse

#=> 6 37 Private 284582 Masters 14 Married-civ-spouse

# OCCUPATION RELATIONSHIP RACE SEX CAPITALGAIN CAPITALLOSS

#=> 1 Adm-clerical Not-in-family White Male 2174 0

#=> 2 Exec-managerial Husband White Male 0 0

#=> 3 Handlers-cleaners Not-in-family White Male 0 0

#=> 4 Handlers-cleaners Husband Black Male 0 0

#=> 5 Prof-specialty Wife Black Female 0 0

#=> 6 Exec-managerial Wife White Female 0 0

# HOURSPERWEEK NATIVECOUNTRY ABOVE50K

#=> 1 40 United-States 0

#=> 2 13 United-States 0

#=> 3 40 United-States 0

#=> 4 40 United-States 0

#=> 5 40 Cuba 0

#=> 6 40 United-States 0

**Check Class bias**

Ideally, the proportion of events and non-events in the Y variable should approximately be the same. So, lets first check the proportion of classes in the dependent variable ABOVE50K.

table(inputData$ABOVE50K)

# 0 1

# 24720 7841

Clearly, there is a class bias, a condition observed when the proportion of events is much smaller than proportion of non-events. So we must sample the observations in approximately equal proportions to get better models.

**Create Training and Test Samples**

One way to address the problem of class bias is to draw the 0’s and 1’s for the trainingData (development sample) in equal proportions. In doing so, we will put rest of the inputData not included for training into testData (validation sample). As a result, the size of development sample will be smaller that validation, which is okay, because, there are large number of observations (>10K).

# Create Training Data

input\_ones <- inputData[which(inputData$ABOVE50K == 1), ] # all 1's

input\_zeros <- inputData[which(inputData$ABOVE50K == 0), ] # all 0's

set.seed(100) # for repeatability of samples

input\_ones\_training\_rows <- sample(1:nrow(input\_ones), 0.7\*nrow(input\_ones)) # 1's for training

input\_zeros\_training\_rows <- sample(1:nrow(input\_zeros), 0.7\*nrow(input\_ones)) # 0's for training. Pick as many 0's as 1's

training\_ones <- input\_ones[input\_ones\_training\_rows, ]

training\_zeros <- input\_zeros[input\_zeros\_training\_rows, ]

trainingData <- rbind(training\_ones, training\_zeros) # row bind the 1's and 0's

# Create Test Data

test\_ones <- input\_ones[-input\_ones\_training\_rows, ]

test\_zeros <- input\_zeros[-input\_zeros\_training\_rows, ]

testData <- rbind(test\_ones, test\_zeros) # row bind the 1's and 0's

Next it is desirable to find the [information value](http://r-statistics.co/Variable-Selection-and-Importance-With-R.html#7.%20Information%20value%20and%20Weight%20of%20evidence) of variables to get an idea of how valuable they are in explaining the dependent variable (ABOVE50K).

**Create WOE for categorical variables (optional)**

Optionally, we can create WOE equivalents for all categorical variables. This is only an optional step, for simplicity, this step is NOT run for this analysis.

for(factor\_var in factor\_vars){

inputData[[factor\_var]] <- WOE(X=inputData[, factor\_var], Y=inputData$ABOVE50K)

}

head(inputData)

#> AGE WORKCLASS FNLWGT EDUCATION EDUCATIONNUM MARITALSTATUS OCCUPATION

#> 1 39 0.1608547 77516 0.7974104 13 -1.8846680 -0.713645

#> 2 50 0.2254209 83311 0.7974104 13 0.9348331 1.084280

#> 3 38 -0.1278453 215646 -0.5201257 9 -1.0030638 -1.555142

#> 4 53 -0.1278453 234721 -1.7805021 7 0.9348331 -1.555142

#> 5 28 -0.1278453 338409 0.7974104 13 0.9348331 0.943671

#> 6 37 -0.1278453 284582 1.3690863 14 0.9348331 1.084280

#> RELATIONSHIP RACE SEX CAPITALGAIN CAPITALLOSS HOURSPERWEEK

#> 1 -1.015318 0.08064715 0.3281187 2174 0 40

#> 2 0.941801 0.08064715 0.3281187 0 0 13

#> 3 -1.015318 0.08064715 0.3281187 0 0 40

#> 4 0.941801 -0.80794676 0.3281187 0 0 40

#> 5 1.048674 -0.80794676 -0.9480165 0 0 40

#> 6 1.048674 0.08064715 -0.9480165 0 0 40

#> NATIVECOUNTRY ABOVE50K

#> 1 0.02538318 0

#> 2 0.02538318 0

#> 3 0.02538318 0

#> 4 0.02538318 0

#> 5 0.11671564 0

#> 6 0.02538318 0

**Compute Information Values**

The smbinning::smbinning function converts a continuous variable into a categorical variable using recursive partitioning. We will first convert them to categorical variables and then, capture the information values for all variables in iv\_df

library(smbinning)

# segregate continuous and factor variables

factor\_vars <- c ("WORKCLASS", "EDUCATION", "MARITALSTATUS", "OCCUPATION", "RELATIONSHIP", "RACE", "SEX", "NATIVECOUNTRY")

continuous\_vars <- c("AGE", "FNLWGT","EDUCATIONNUM", "HOURSPERWEEK", "CAPITALGAIN", "CAPITALLOSS")

iv\_df <- data.frame(VARS=c(factor\_vars, continuous\_vars), IV=numeric(14)) # init for IV results

# compute IV for categoricals

for(factor\_var in factor\_vars){

smb <- smbinning.factor(trainingData, y="ABOVE50K", x=factor\_var) # WOE table

if(class(smb) != "character"){ # heck if some error occured

iv\_df[iv\_df$VARS == factor\_var, "IV"] <- smb$iv

}

}

# compute IV for continuous vars

for(continuous\_var in continuous\_vars){

smb <- smbinning(trainingData, y="ABOVE50K", x=continuous\_var) # WOE table

if(class(smb) != "character"){ # any error while calculating scores.

iv\_df[iv\_df$VARS == continuous\_var, "IV"] <- smb$iv

}

}

iv\_df <- iv\_df[order(-iv\_df$IV), ] # sort

iv\_df

#> VARS IV

#> RELATIONSHIP 1.5739

#> MARITALSTATUS 1.3356

#> AGE 1.1748

#> CAPITALGAIN 0.8389

#> OCCUPATION 0.8259

#> EDUCATIONNUM 0.7776

#> EDUCATION 0.7774

#> HOURSPERWEEK 0.4682

#> SEX 0.3087

#> WORKCLASS 0.1633

#> CAPITALLOSS 0.1507

#> NATIVECOUNTRY 0.0815

#> RACE 0.0607

#> FNLWGT 0.0000

**Build Logit Models and Predict**

logitMod <- glm(ABOVE50K ~ RELATIONSHIP + AGE + CAPITALGAIN + OCCUPATION + EDUCATIONNUM, data=trainingData, family=binomial(link="logit"))

predicted <- plogis(predict(logitMod, testData)) # predicted scores

# or

predicted <- predict(logitMod, testData, type="response") # predicted scores

A quick note about the plogis function: The glm() procedure with family="binomial" will build the logistic regression model on the given formula. When we use the predict function on this model, it will predict the log(odds) of the Y variable. This is not what we ultimately want because, the predicted values may not lie within the 0 and 1 range as expected. So, to convert it into prediction probability scores that is bound between 0 and 1, we use the plogis().

**Decide on optimal prediction probability cutoff for the model**

The default cutoff prediction probability score is 0.5 or the ratio of 1’s and 0’s in the training data. But sometimes, tuning the probability cutoff can improve the accuracy in both the development and validation samples. The InformationValue::optimalCutoff function provides ways to find the optimal cutoff to improve the prediction of 1’s, 0’s, both 1’s and 0’s and o reduce the misclassification error. Lets compute the optimal score that minimizes the misclassification error for the above model.

library(InformationValue)

optCutOff <- optimalCutoff(testData$ABOVE50K, predicted)[1]

#=> 0.71

**Model Diagnostics**

The summary(logitMod) gives the beta coefficients, Standard error, z Value and p Value. If your model had categorical variables with multiple levels, you will find a row-entry for each category of that variable. That is because, each individual category is considered as an independent binary variable by the glm(). In this case it is ok if few of the categories in a multi-category variable don’t turn out to be significant in the model (i.e. p Value turns out greater than significance level of 0.5).

summary(logitMod)

#> Call:

#> glm(formula = ABOVE50K ~ RELATIONSHIP + AGE + CAPITALGAIN + OCCUPATION +

#> EDUCATIONNUM, family = "binomial", data = trainingData)

#>

#> Deviance Residuals:

#> Min 1Q Median 3Q Max

#> -3.8380 -0.5319 -0.0073 0.6267 3.2847

#>

#> Coefficients:

#> Estimate Std. Error z value Pr(>|z|)

#> (Intercept) -4.57657130 0.24641856 -18.572 < 0.0000000000000002 \*\*\*

#> RELATIONSHIP Not-in-family -2.27712854 0.07205131 -31.604 < 0.0000000000000002 \*\*\*

#> RELATIONSHIP Other-relative -2.72926866 0.27075521 -10.080 < 0.0000000000000002 \*\*\*

#> RELATIONSHIP Own-child -3.56051255 0.17892546 -19.899 < 0.0000000000000002 \*\*\*

#> ...

#> ...

#> Null deviance: 15216.0 on 10975 degrees of freedom

#> Residual deviance: 8740.9 on 10953 degrees of freedom

#> AIC: 8786.9

#>

#> Number of Fisher Scoring iterations: 8

**VIF**

Like in case of linear regression, we should check for multicollinearity in the model. As seen below, all X variables in the model have VIF well below 4.

vif(logitMod)

#> GVIF Df GVIF^(1/(2\*Df))

#> RELATIONSHIP 1.340895 5 1.029768

#> AGE 1.119782 1 1.058198

#> CAPITALGAIN 1.023506 1 1.011685

#> OCCUPATION 1.733194 14 1.019836

#> EDUCATIONNUM 1.454267 1 1.205930

**Misclassification Error**

Misclassification error is the percentage mismatch of predcited vs actuals, irrespective of 1’s or 0’s. The lower the misclassification error, the better is your model.

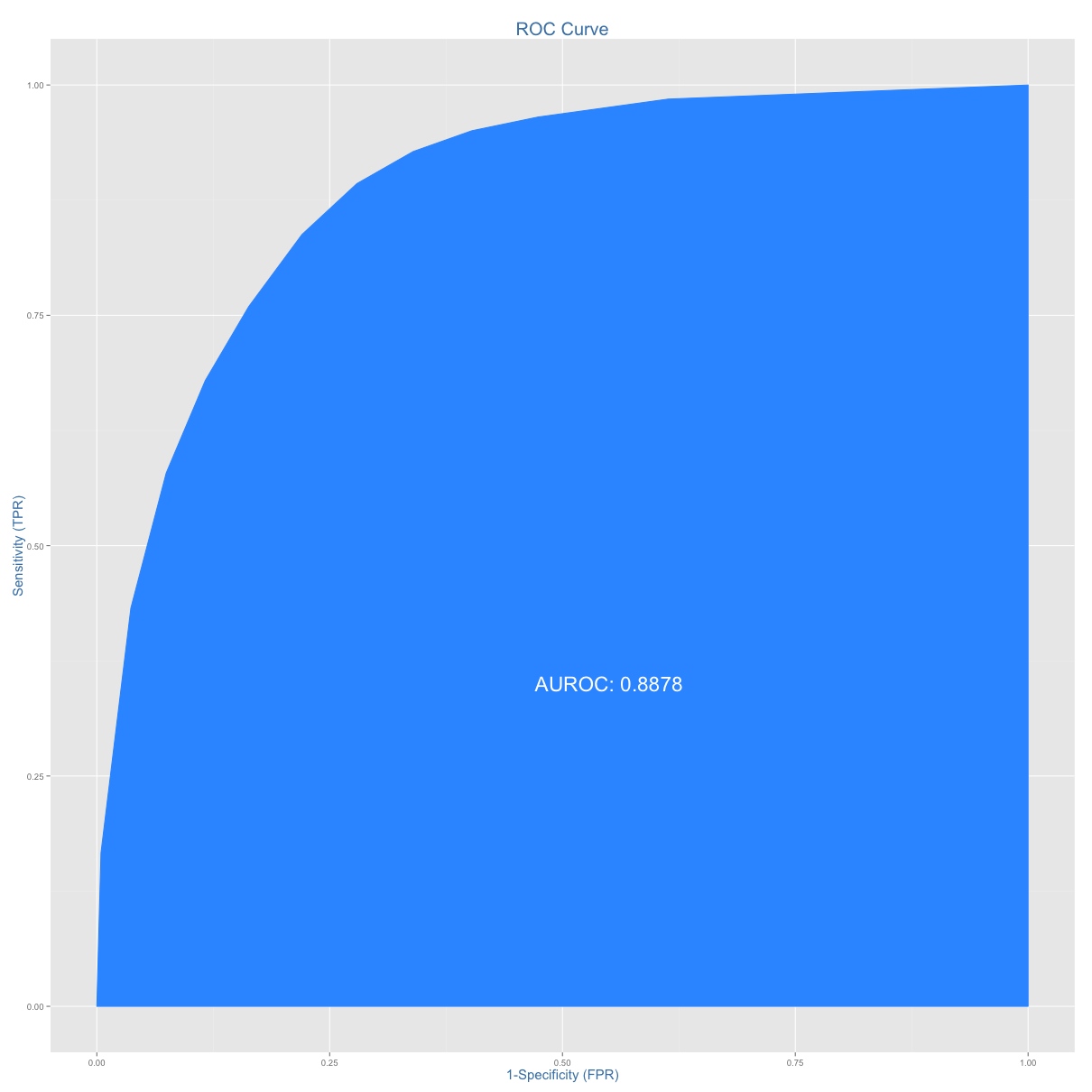
misClassError(testData$ABOVE50K, predicted, threshold = optCutOff)

#=> 0.0899

**ROC**

Receiver Operating Characteristics Curve traces the percentage of true positives accurately predicted by a given logit model as the prediction probability cutoff is lowered from 1 to 0. For a good model, as the cutoff is lowered, it should mark more of actual 1’s as positives and lesser of actual 0’s as 1’s. So for a good model, the curve should rise steeply, indicating that the TPR (Y-Axis) increases faster than the FPR (X-Axis) as the cutoff score decreases. Greater the area under the ROC curve, better the predictive ability of the model.

plotROC(testData$ABOVE50K, predicted)



The above model has area under ROC curve 88.78%, which is pretty good.

**Concordance**

Ideally, the model-calculated-probability-scores of all actual Positive’s, (aka Ones) should be greater than the model-calculated-probability-scores of ALL the Negatives (aka Zeroes). Such a model is said to be perfectly concordant and a highly reliable one. This phenomenon can be measured by Concordance and Discordance.

In simpler words, of all combinations of 1-0 pairs (actuals), *Concordance* is the percentage of pairs, whose scores of actual positive’s are greater than the scores of actual negative’s. For a perfect model, this will be 100%. So, the higher the concordance, the better is the quality of model.

Concordance(testData$ABOVE50K, predicted)

#> 0.8915

The above model with a concordance of 89.2% is indeed a good quality model.

**Specificity and Sensitivity**

Sensitivity (or True Positive Rate) is the percentage of 1’s (actuals) correctly predicted by the model, while, specificity is the percentage of 0’s (actuals) correctly predicted. Specificity can also be calculated as 1 − *False* *Positive* *Rate*.

*Sensitivity*=# *Actual* 1′*s* *and* *Predicted* *as* 1′*s*# *of* *Actual* 1′*s*

*Specificity*=# *Actual* 0′*s* *and* *Predicted* *as* 0′*s*# *of* *Actual* 0′*s*

sensitivity(testData$ABOVE50K, predicted, threshold = optCutOff)

#> 0.3089

specificity(testData$ABOVE50K, predicted, threshold = optCutOff)

#> 0.9836

The above numbers are calculated on the validation sample that was not used for training the model. So, a truth detection rate of 31% on test data is good.

**Confusion Matrix**

confusionMatrix(testData$ABOVE50K, predicted, threshold = optCutOff)

# The columns are actuals, while rows are predicteds.

#> 0 1

#> 0 18918 1626

#> 1 314 727

**InformationValue**

The functions in [InformationValue](https://cran.r-project.org/web/packages/InformationValue/) package are broadly divided in following categories:

**1. Diagnostics of predicted probability scores**

**2. Performance analysis**

**3. Functions that aid accuracy improvement**

First, lets define the meaning of the various terms used in this document.

**How to install**

install.packages("InformationValue") # For stable CRAN version

devtools::install\_github("InformationValue") # For latest dev version.

**Definitions of functions**

**Sensitivity**, a.k.a *True Positive Rate* is the proportion of the events (ones) that a model predicted correctly as events, for a given prediction probability cut-off.

**Specificity**, a.k.a \* 1 - False Positive Rate\* is the proportion of the non-events (zeros) that a model predicted correctly as non-events, for a given prediction probability cut-off.

**False Positive Rate** is the proportion of non-events (zeros) that were predicted as events (ones)

**False Negative Rate** is the proportion of events (ones) that were predicted as non-events (zeros)

**Mis-classification error** is the proportion of observations (both events and non-event) that were not predicted correctly.

**Concordance** is the percentage of *all-possible-pairs-of-predicted Ones and Zeros* where the scores of actual ones are greater than the scores of actual zeros. It represents the predictive power of a binary classification model.

**Weights of Evidence (WOE)** provides a method of recoding the categorical x variable to continuous variables. For each category of a categorical variable, the **WOE** is calculated as:

*WOE*=*ln*(*perc* *good* *of* *all* *goodsperc* *bad* *of* *all* *bads*)

In above formula, *goods* is synonymous with *ones*, *events*, *positives* or *responders* and *bads* is synonymous with *zeros*, *non-events*, *negatives* or *non-responders*.

**Information Value (IV)** is a measure of the predictive capability of a categorical x variable to accurately predict the goods and bads. For each category of x, information value is computed as:

*IV* = (*perc* *good* *of* *all* *goods*−*perc* *bad* *of* *all* *bads*) \* *WOE*

The total IV of a variable is the sum of IV’s of its categories. Here is what the values of IV mean according to Siddiqi (2006):

* Less than 0.02, then the predictor is not useful for modeling (separating the Goods from the Bads)
* 0.02 to 0.1, then the predictor has only a weak relationship.
* 0.1 to 0.3, then the predictor has a medium strength relationship.
* 0.3 or higher, then the predictor has a strong relationship.

Here is a sample MS Excel file that shows how to [calculate WOE and Information Value](http://r-statistics.co/IVCalc.xlsx).

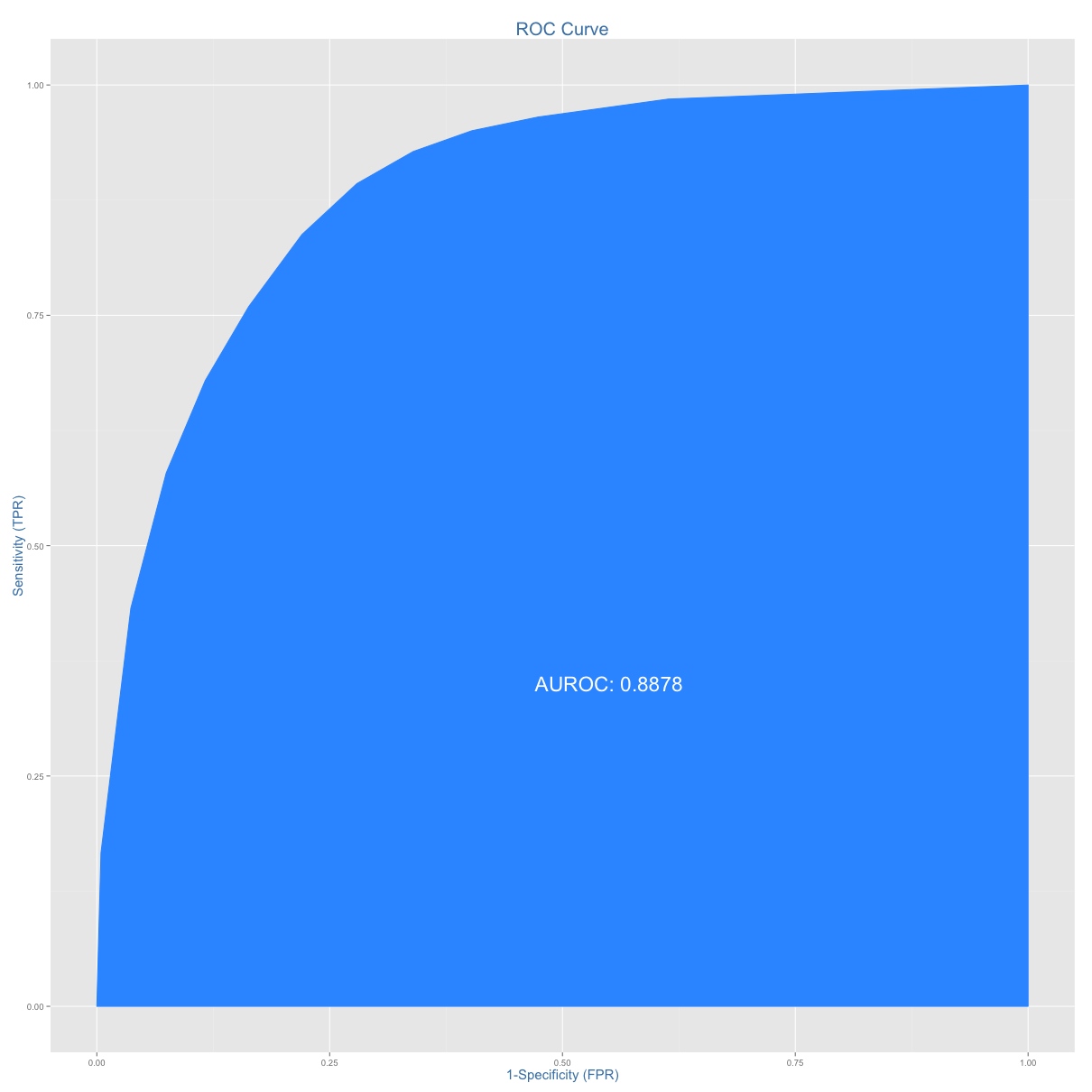
**KS Statistic** or *Kolmogorov-Smirnov statistic* is the maximum difference between the cumulative true positive and cumulative false positive rate. It is often used as the deciding metric to judge the efficacy of models in credit scoring. The higher the ks\_stat, the more efficient is the model at capturing the responders (Ones). This should not be confused with the ks.test function.

**1.1 plotROC**

The plotROC uses the ggplot2 framework to create the ROC curve and prints the AUROC inside. It comes with an option to display the change points of the prediction probability scores on the graph if you set the Show.labels = T.

data('ActualsAndScores')

plotROC(actuals=ActualsAndScores$Actuals, predictedScores=ActualsAndScores$PredictedScores)



You can also get the sensitivity matrix used to make the plot by turning on returnSensitivityMat = TRUE.

sensMat <- plotROC(actuals=ActualsAndScores$Actuals, predictedScores=ActualsAndScores$PredictedScores, returnSensitivityMat = TRUE)

**1.2. sensitivity or recall**

Sensitivity, also considered as the ‘True Positive Rate’ or ‘recall’ is the proportion of ‘Events’ (or ‘Ones’) correctly predicted by the model, for a given prediction probability cutoff score. The default cutoff score for the specificity function unless specified by the threshold argument is taken as 0.5.

sensitivity(actuals = ActualsAndScores$Actuals, predictedScores = ActualsAndScores$PredictedScores)

#> [1] 1

If the objective of your problem is to maximise the ability of your model to detect the ‘Events’ (or ‘Ones’), even at the cost of wrongly predicting the non-events (‘Zeros’) as an event (‘One’), then you could set the threshold as determined by the optimalCutoff() with optimiseFor='Ones'.

**NOTE**: This may not be the best example, because we are able to achieve the maximum sensitivity of 1 with the default cutoff of 0.5. However, I am showing this example just to understand how this could be implemented in real projects.

max\_sens\_cutoff <- optimalCutoff(actuals=ActualsAndScores$Actuals, predictedScores = ActualsAndScores$PredictedScores, optimiseFor='Ones') # determine cutoff to maximise sensitivity.

print(max\_sens\_cutoff) # This would be cut-off score that achieved maximum sensitivity.

#> [1] 0.5531893

sensitivity(actuals = ActualsAndScores$Actuals, predictedScores = ActualsAndScores$PredictedScores, threshold=max\_sens\_cutoff)

#> [1] 1

**1.3. specificity**

For a given probability score cutoff (threshold), specificity computes what proportion of the total non-events (zeros) were predicted accurately. It can alo be computed as 1 - False Positive Rate. If unless specified, the default threshold value is set as 0.5, which means, the values of ActualsAndScores$PredictedScores above 0.5 is considered as events (Ones).

specificity(actuals=ActualsAndScores$Actuals, predictedScores=ActualsAndScores$PredictedScores)

#> [1] 0.1411765

If you wish to know what proportion of non-events could be detected by lowering the threshold:

specificity(actuals=ActualsAndScores$Actuals, predictedScores=ActualsAndScores$PredictedScores, threshold = 0.35)

#> [1] 0.01176471

**1.4. precision**

For a given probability score cutoff (threshold), precision or ‘positive predictive value’ computes the proportion of the total events (ones) out of the total that were predicted to be events (ones).

precision(actuals=ActualsAndScores$Actuals, predictedScores=ActualsAndScores$PredictedScores)

#> [1] 0.5379747

**1.5. npv**

For a given probability score cutoff (threshold), npv or ‘negative predictive value’ computes the proportion of the total non-events (zeros) out of the total that were predicted to be non-events (zeros).

npv(actuals=ActualsAndScores$Actuals, predictedScores=ActualsAndScores$PredictedScores)

#> [1] 1

**1.6. youdensIndex**

Youden’s J Index (Youden 1950), calculated as   
*J* = *Sensitivity* + *Specificity* − 1  
represents the proportions of correctly predicted observations for both the events (Ones) and nonevents (Zeros). It is particularly useful if you want a single measure that accounts for both *false-positive* and *false-negative* rates

youdensIndex(actuals=ActualsAndScores$Actuals, predictedScores=ActualsAndScores$PredictedScores)

#> [1] 0.1411765

**2.1. misClassError**

Mis-Classification Error is the proportion of all events that were incorrectly classified, for a given probability cutoff score.

misClassError(actuals=ActualsAndScores$Actuals, predictedScores=ActualsAndScores$PredictedScores, threshold=0.5)

#> [1] 0.4294

**2.2. Concordance**

Concordance is the percentage of predicted probability scores where the scores of actual positive’s are greater than the scores of actual negative’s. It is calculated by taking into account the scores of all possible pairs of *Ones* and *Zeros*. If the concordance of a model is 100%, it means that, by tweaking the prediction probability cutoff, we could accurately predict all of the events and non-events.

Concordance(actuals=ActualsAndScores$Actuals, predictedScores=ActualsAndScores$PredictedScores)

#> $Concordance

#> [1] 0.8730796

#>

#> $Discordance

#> [1] 0.1269204

#>

#> $Tied

#> [1] 0

#>

#> $Pairs

#> [1] 7225

**2.3. somersD**

somersD computes how many more concordant than discordant pairs exist divided by the total number of pairs. Larger the Somers D value, better model’s predictive ability.

*SomersD*=*ConcordantPairs*−*DiscordantPairsTotalPairs*

somersD(actuals=ActualsAndScores$Actuals, predictedScores=ActualsAndScores$PredictedScores)

#> [1] 0.7461592

**2.4. ks\_stat**

ks\_stat computes the kolmogorov-smirnov statistic that is widely used in credit scoring to determine the efficacy of binary classification models. The higher the ks\_stat more effective is the model at capturing the responders.

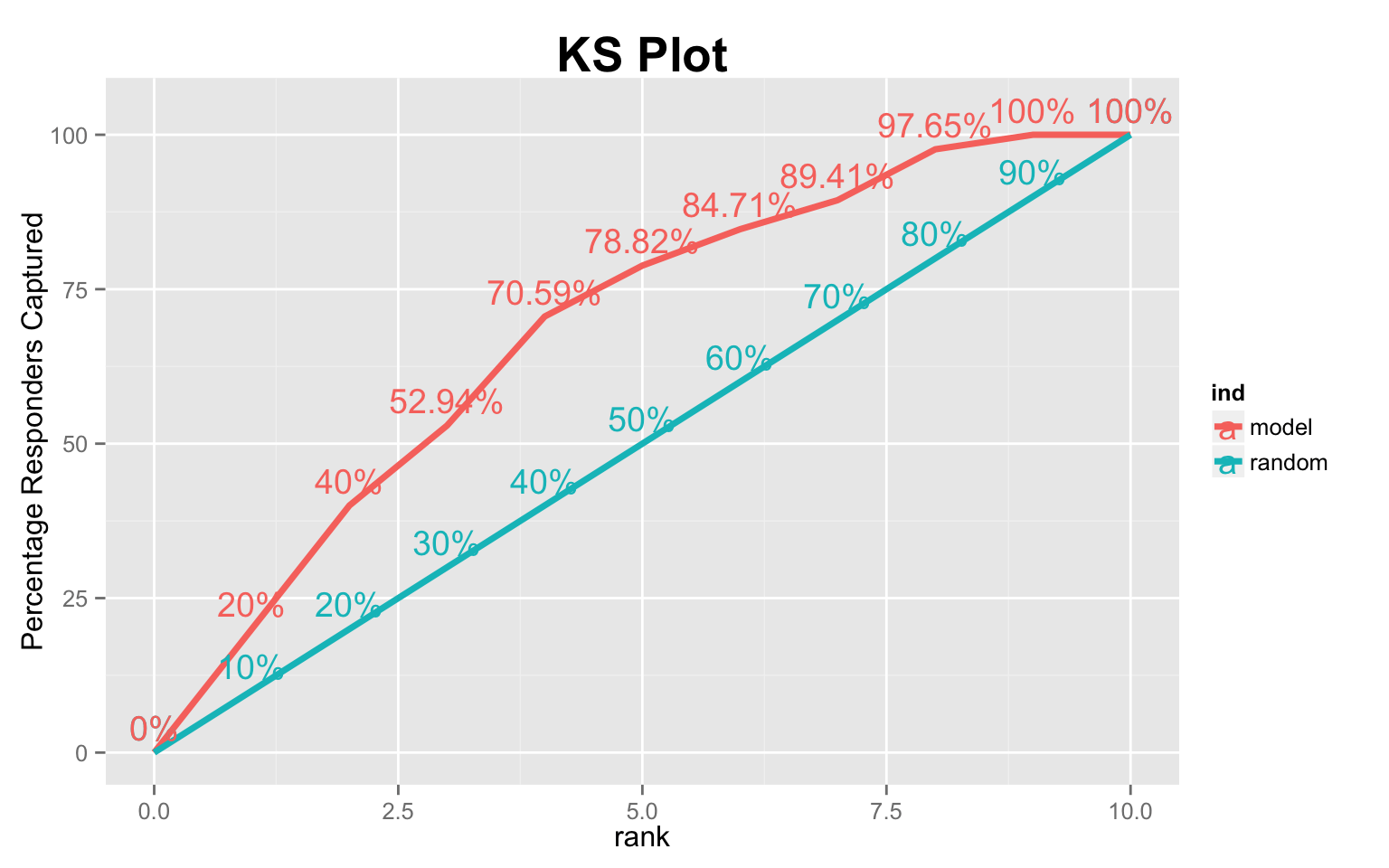
ks\_stat(actuals=ActualsAndScores$Actuals, predictedScores=ActualsAndScores$PredictedScores)

#> [1] 0.6118

**2.5. ks\_plot**

ks\_plot plots the lift is capturing the responders (Ones) against the the random case where we don’t use the model. The more curvier (higher) the model curve, the better is your model.

> ks\_plot(actuals=ActualsAndScores$Actuals, predictedScores=ActualsAndScores$PredictedScores)



**How to interpret this plot?**

This plot aims to answer the question: *Which part of the population should I target for my marketing campaign and what conversion rate can I expect?* Now, lets understand how it is computed and what those numbers mean.

After computing the prediction probability scores from a given logit model, the datapoints are sorted in descending order of prediction probability scores. This set of data points is split into 10 groups (or ranks) as marked on the X-axis, such that, the group ranked 1 contains the top 10% of datapoints with highest prediction probability scores, group ranked 2 containing the next 10% and so on.

The **‘random’** line in above chart corresponds to the case of capturing the responders (‘Ones’) by random selection, i.e., when you don’t have any model (generated probability scores) at disposal. While the **‘model’** line represents the case of capturing the responders if you go by the model generated probability scores, where you begin by targeting datapoint with highest probability scores. In simpler terms, it represents the proportion of total responders you can expect to capture as you keep targeting the data point starting from the bucket rank 1 through 10, in that order. So, now you know which part of the population to target and what is the expected conversion rate.

For example, from the above chart for instance, by targeting first 40% of the population, the model will be able to capture 70.59% of total responders(Ones), while without the model, you can expect to capture only 40% of responders by random targeting.

**3.1. optimalCutoff**

optimalCutoff determines the optimal threshold for prediction probability score based on your specific problem objectives. By adjusting the argument optimiseFor as follows, you can find the optimal cutoff that: 1. Ones: maximizes detection of events or ones 2. Zeros: maximizes detection of non-events or zeros 3. Both: control both *false positive rate* and *false negative rate* by maximizing the Youden’s J Index. 4. misclasserror: minimizes misclassification error (default)

> optimalCutoff(actuals = ActualsAndScores$Actuals, predictedScores = ActualsAndScores$PredictedScores) # returns cutoff that gives minimum misclassification error.

> optimalCutoff(actuals = ActualsAndScores$Actuals, predictedScores = ActualsAndScores$PredictedScores,

+ optimiseFor = "Both") # returns cutoff that gives maximum of Youden's J Index

> # > [1] 0.6431893

By setting the returnDiagnostics=TRUE you can get the sensitivityTable that shows the FPR, TPR, YOUDENSINDEX, SPECIFICITY, MISCLASSERROR for various values of cutoff.

> sens\_table <- optimalCutoff(actuals = ActualsAndScores$Actuals, predictedScores = ActualsAndScores$PredictedScores,

+ optimiseFor = "Both", returnDiagnostics = TRUE)$sensitivityTable

**3.2. WOE**

Computes the Weights Of Evidence (WOE) for each group of a given categorical X and binary response Y.

WOE(X=SimData$X.Cat, Y=SimData$Y.Binary)

**3.3. WOETable**

Generates the WOE table showing the percentage goods, bads, WOE and IV for each category of X. WOE for a given category of X is computed as:

*WOE*=*ln*(*perc*.*Goodperc*.*Bad*)

options(scipen = 999, digits = 2)

WOETable(X=SimData$X.Cat, Y=SimData$Y.Binary)

| **CAT** | **GOODS** | **BADS** | **TOTAL** | **PCT\_G** | **PCT\_B** | **WOE** | **IV** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Group1 | 179 | 1500 | 1679 | 0.0246488571 | 0.0659108885 | -0.9835731 | 0.0405842251 |
| Group2 | 346 | 525 | 871 | 0.0476452768 | 0.0230688110 | 0.7253020 | 0.0178253591 |
| Group3 | 560 | 1354 | 1914 | 0.0771137428 | 0.0594955620 | 0.2593798 | 0.0045698000 |
| Group4 | 6 | 6 | 6 | 0.0008262187 | 0.0002636436 | 1.1422615 | 0.0006426079 |
| Group5 | 4595 | 16369 | 20964 | 0.6327458001 | 0.7192635557 | -0.1281591 | 0.0110880366 |
| Group6 | 577 | 461 | 1038 | 0.0794546957 | 0.0202566131 | 1.3667057 | 0.0809063559 |
| Group7 | 658 | 1670 | 2328 | 0.0906086478 | 0.0733807892 | 0.2108875 | 0.0036331398 |
| Group8 | 327 | 859 | 1186 | 0.0450289177 | 0.0377449688 | 0.1764527 | 0.0012852725 |
| Group9 | 14 | 14 | 14 | 0.0019278436 | 0.0006151683 | 1.1422615 | 0.0014994184 |

**3.4. IV**

Compute the information value of a given categorical X (Factor) and binary Y (numeric) response. The information value of a category of X is calculated as:   
*IV* = (*perc*.*Good* − *perc*.*Bad*)\**WOE*  
The IV of the categorical variables is the sum of information value of its individual categories.

options(scipen = 999, digits = 4)

IV(X=SimData$X.Cat, Y=SimData$Y.Binary)

#> 1] 0.162

#> attr(,“howgood”)

#> [1] “Highly Predictive”

“He who gives up code safety for code speed deserves neither.”

For more information and examples, visit [rstatistics.net](http://rstatistics.net)