Sistemas Inteligentes para la Gestión en la Empresa

Practice 1

Pre-processing of data and binary classification

Author

Abdullah Taher Sadoon AL-Musawi

Escuela Ténica Superior de Ingenierías Informáca y de Telecomunicació

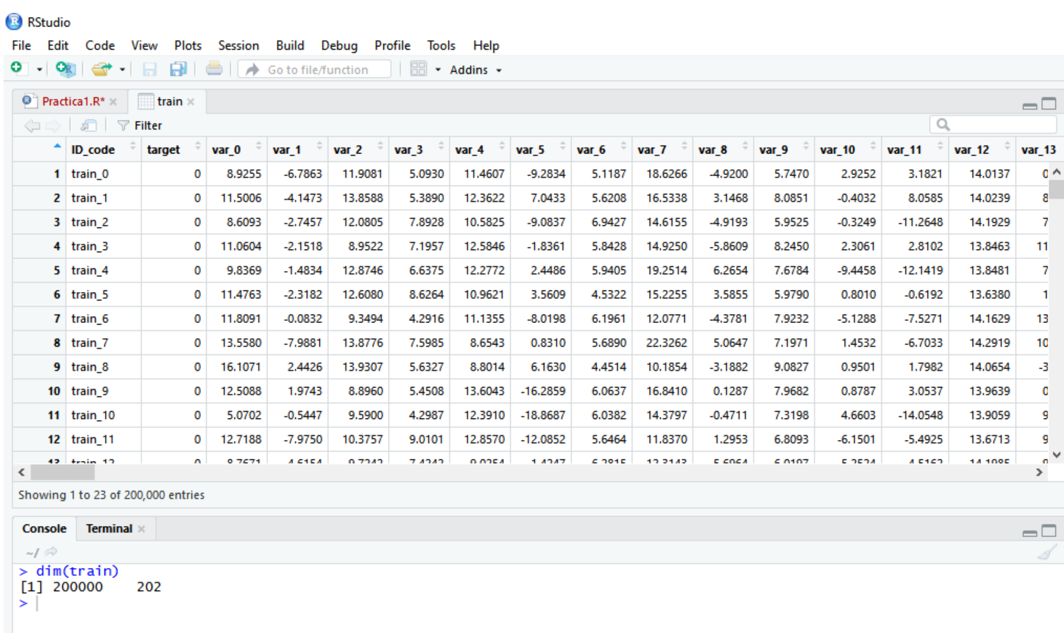
—

Granada, mayo de 2019

index of contents

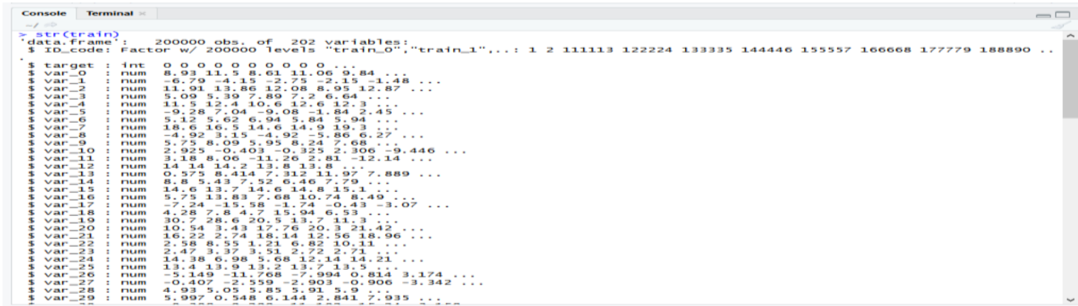
1. Data exploration

Initially Will be loading our dataset ´The Train_ok´ in R studio, to carry out the verification of the variables that contains and for checking the list provided in the excel file (Data Dictionary) provided for the realization of the practice.



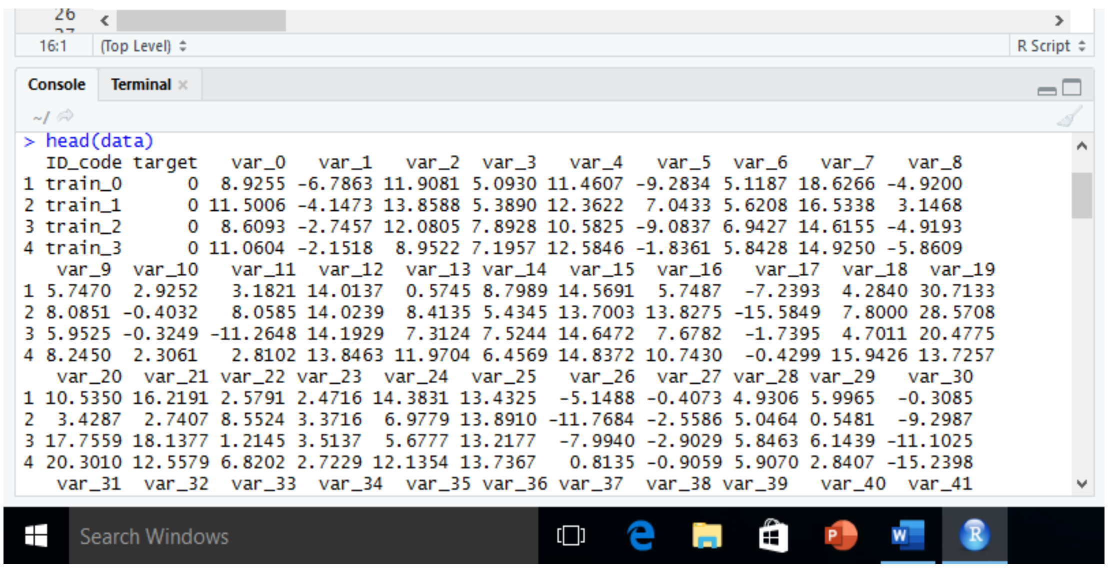
So, we have 202 variables(ID_code,target,var_0......var_199) and 200000 entries, so We will have a data set that you can handle with it.

Also, to get more information about the dataset, we can see the internal structure that has:

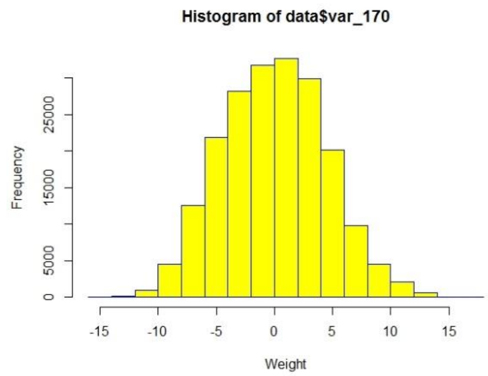


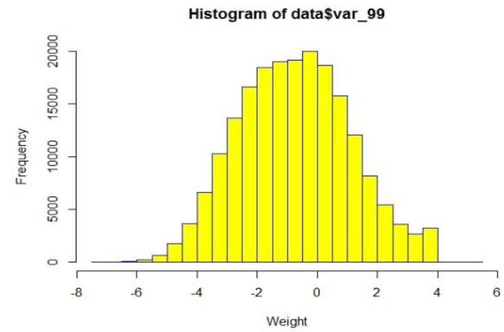
```
Console   Terminal
> str(train)
'data.frame':	200000 obs. of  202 variables:
 $ ID_code: Factor w/ 200000 levels "train_0","train_1",..: 1 2 111113 122224 133335 144446 155557 166668 177779 188890 ..
.
 $ target : int  0 0 0 0 0 0 0 0 0 0 ...
 $ var_0  : num  8.93 11.5 8.61 11.06 9.84 ...
 $ var_1  : num  -6.79 -4.15 -2.75 -2.15 -1.48 ...
 $ var_2  : num  11.91 13.86 12.08 8.95 12.87 ...
 $ var_3  : num  5.09 5.39 7.89 7.2 6.64 ...
 $ var_4  : num  11.5 12.4 10.6 12.6 12.3 ...
 $ var_5  : num  -9.28 7.04 -9.08 -1.84 2.45 ...
 $ var_6  : num  5.12 5.62 6.94 5.84 5.94 ...
 $ var_7  : num  18.6 16.5 14.6 14.9 19.3 ...
 $ var_8  : num  -4.92 3.15 -4.92 -5.86 6.27 ...
 $ var_9  : num  5.75 8.09 5.95 8.24 7.68 ...
 $ var_10 : num  2.925 -0.403 -0.325 2.306 -9.446 ...
 $ var_11 : num  3.18 8.06 -11.26 2.81 -12.14 ...
 $ var_12 : num  14 14 14.2 13.8 13.8 ...
 $ var_13 : num  0.575 8.414 7.312 11.97 7.889 ...
 $ var_14 : num  8.8 5.43 7.52 6.46 7.79 ...
 $ var_15 : num  14.6 13.7 14.6 14.8 15.1 ...
 $ var_16 : num  5.75 13.83 7.68 10.74 8.49 ...
 $ var_17 : num  -7.24 -15.58 -1.74 -0.43 -3.07 ...
 $ var_18 : num  4.28 7.8 4.7 15.94 6.53 ...
 $ var_19 : num  30.7 28.6 20.5 13.7 11.3 ...
 $ var_20 : num  10.54 3.43 17.76 20.3 21.42 ...
 $ var_21 : num  16.22 2.74 18.14 12.56 18.96 ...
 $ var_22 : num  2.58 8.55 1.21 6.82 10.11 ...
 $ var_23 : num  2.47 3.37 3.51 2.72 2.71 ...
 $ var_24 : num  14.38 6.98 5.68 12.14 14.21 ...
 $ var_25 : num  13.4 13.9 13.2 13.7 13.5 ...
 $ var_26 : num  -5.149 -11.768 -7.994 0.814 3.174 ...
 $ var_27 : num  -0.407 -2.559 -2.903 -0.906 -3.342 ...
 $ var_28 : num  4.93 5.05 5.85 5.91 5.9 ...
 $ var_29 : num  5.997 0.548 6.144 2.841 7.935 ...
```

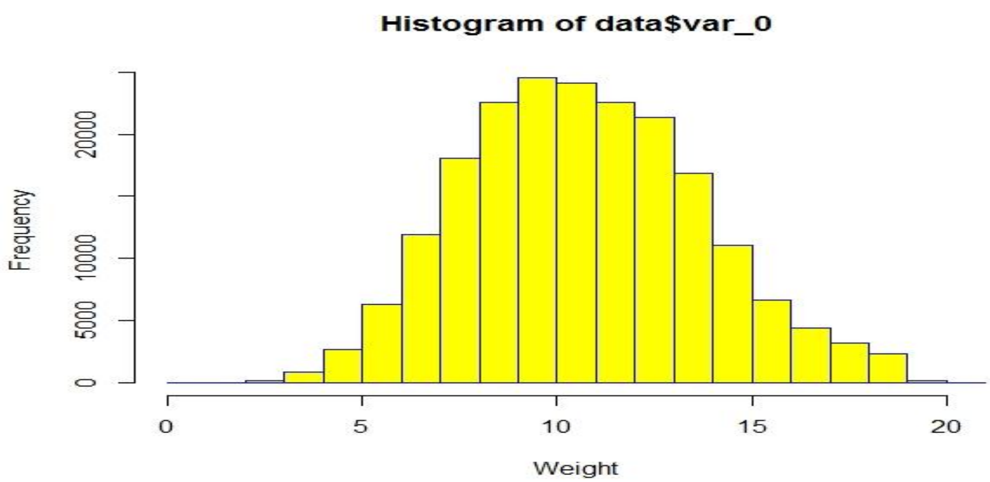
As we go exploring the dataset, we realize that we have variables very unbalanced cases for example with the variables var_0 ………var_199, so we will normalize these vars to make its value between 0-1.



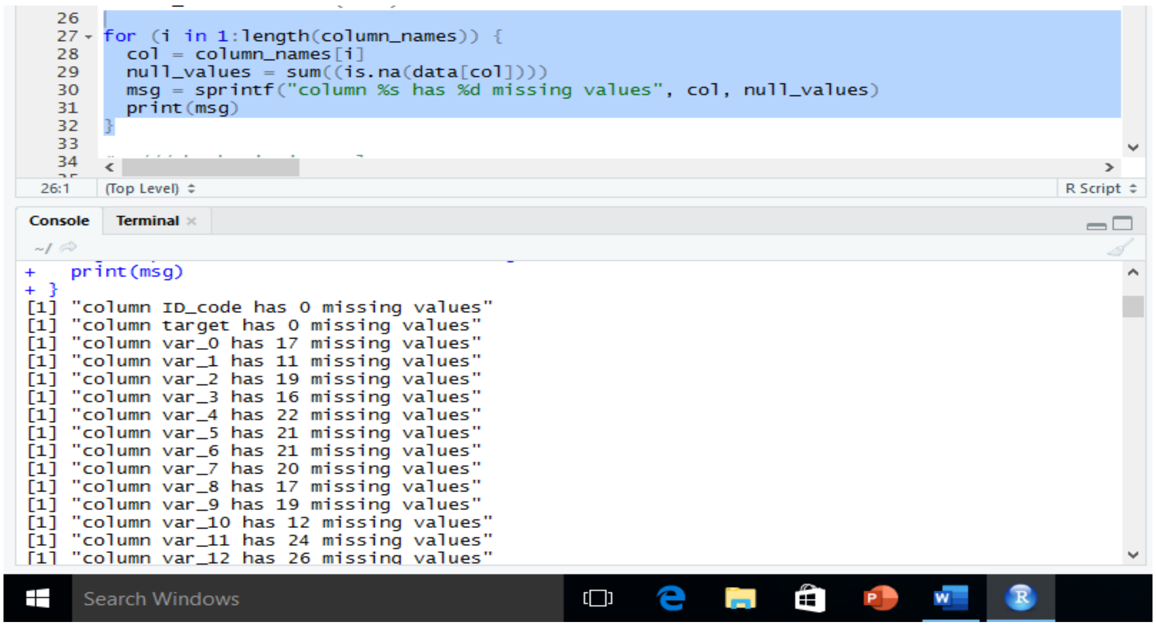
```
16:1   (Top Level) ÷                                              R Script ÷
Console   Terminal
~/
> head(data)
  ID_code target    var_0   var_1   var_2  var_3   var_4   var_5  var_6   var_7   var_8
1 train_0      0   8.9255 -6.7863 11.9081 5.0930 11.4607 -9.2834 5.1187 18.6266 -4.9200
2 train_1      0  11.5006 -4.1473 13.8588 5.3890 12.3622  7.0433 5.6208 16.5338  3.1468
3 train_2      0   8.6093 -2.7457 12.0805 7.8928 10.5825 -9.0837 6.9427 14.6155 -4.9193
4 train_3      0  11.0604 -2.1518  8.9522 7.1957 12.5846 -1.8361 5.8428 14.9250 -5.8609
   var_9  var_10   var_11  var_12  var_13 var_14  var_15  var_16   var_17  var_18  var_19
1 5.7470  2.9252   3.1821 14.0137  0.5745 8.7989 14.5691  5.7487  -7.2393  4.2840 30.7133
2 8.0851 -0.4032   8.0585 14.0239  8.4135 5.4345 13.7003 13.8275 -15.5849  7.8000 28.5708
3 5.9525 -0.3249 -11.2648 14.1929  7.3124 7.5244 14.6472  7.6782  -1.7395  4.7011 20.4775
4 8.2450  2.3061   2.8102 13.8463 11.9704 6.4569 14.8372 10.7430  -0.4299 15.9426 13.7257
   var_20  var_21 var_22 var_23   var_24  var_25   var_26  var_27 var_28 var_29   var_30
1 10.5350 16.2191 2.5791 2.4716 14.3831 13.4325  -5.1488 -0.4073 4.9306 5.9965  -0.3085
2  3.4287  2.7407 8.5524 3.3716  6.9779 13.8910 -11.7684 -2.5586 5.0464 0.5481  -9.2987
3 17.7559 18.1377 1.2145 3.5137  5.6777 13.2177  -7.9940 -2.9029 5.8463 6.1439 -11.1025
4 20.3010 12.5579 6.8202 2.7229 12.1354 13.7367   0.8135 -0.9059 5.9070 2.8407 -15.2398
   var_31  var_32  var_33  var_34  var_35 var_36 var_37  var_38 var_39  var_40  var_41
```


Histogram of data$var_170
Frequency
0
5000
15000
25000
-15
-10
-5
0
5
10
15
Weight


Histogram of data$var_99
Frequency
0
5000
10000
15000
20000
-8
-6
-4
-2
0
2
4
6
Weight


Histogram of data$var_0
Frequency
0
5000
10000
20000
0
5
10
15
20
Weight

that we have many missing values as NA's for example with the variables var_0…. var_199 for example var_0 has 17 missing values and var_4 has 22 missing values, so we have to complete it before doing classification processing.

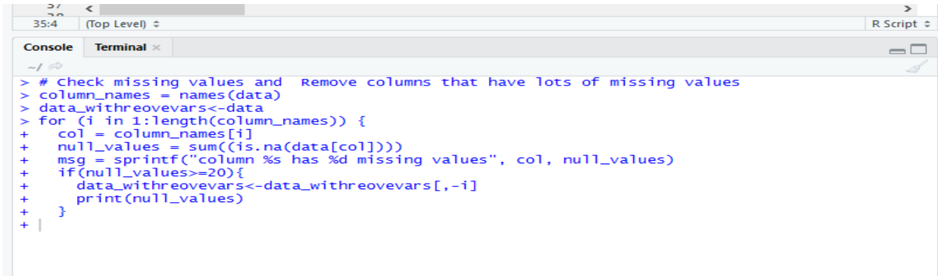


2. Preparing the data

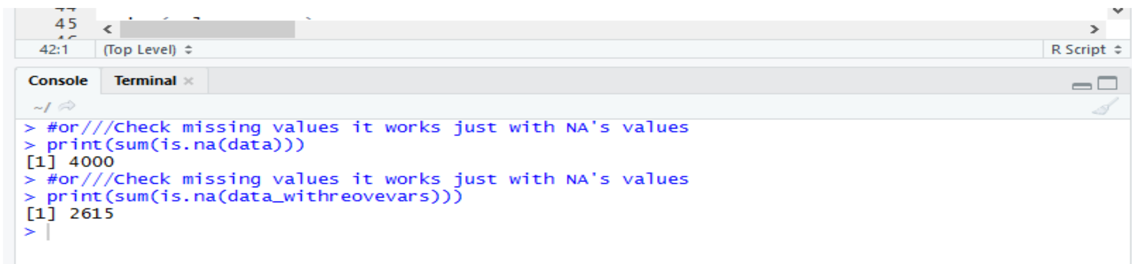
Now we're ready to start exploring missing data and rectifying it through imputation. There are a number of different ways we could go about doing this. Given the small size of the dataset, we probably should not opt for deleting either entire observations (rows) or variables (columns) containing missing values. We're left with the option of either replacing missing values with a sensible value given the distribution of the data, e.g., the mean, median or mode. Finally, we could go with prediction. We'll use both of the two latter methods and I'll rely on some data visualization to guide our decisions.

a- Elimination of variables with less information

➔ **Remove variables have 20 or more missing value**

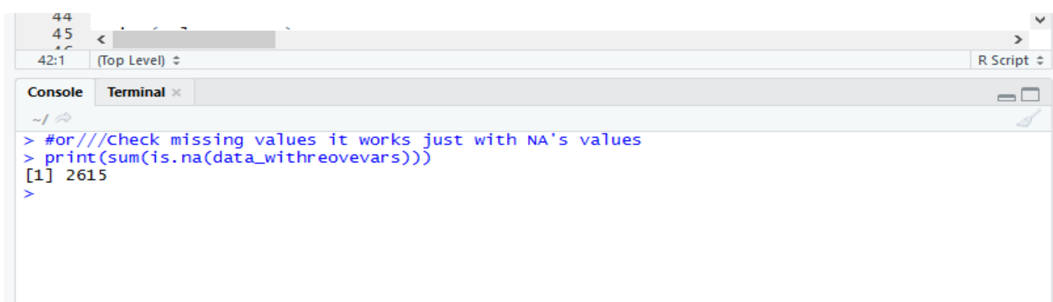


So we can see that we have variables with less missing values, the missing values have less to half of the origin data



b- missing values

One of the most common problems I have faced in Data Cleaning/Exploratory Analysis is handling the missing values. Firstly, understand that there is NO good way to deal with missing data. In our data set there are a lot of missing values as a NA's missing value there are 2615 missing values exactly, we can see them with that figure Below



so, we can fix it with library('mice') # imputation by using 'median' of variables

```r
#remove NA's
column_names<-names(data)
print(column_names)
for (i in 2:length(column_names)) {
  data[,i]<-ifelse(is.na(data[,i]),median(data[,i],na.rm = TRUE),data[,i])
}
```

After that we can Check missing values for our data set by ***md.pattern(data)***
Now we can see new data set without missing values, see the figure Below

.

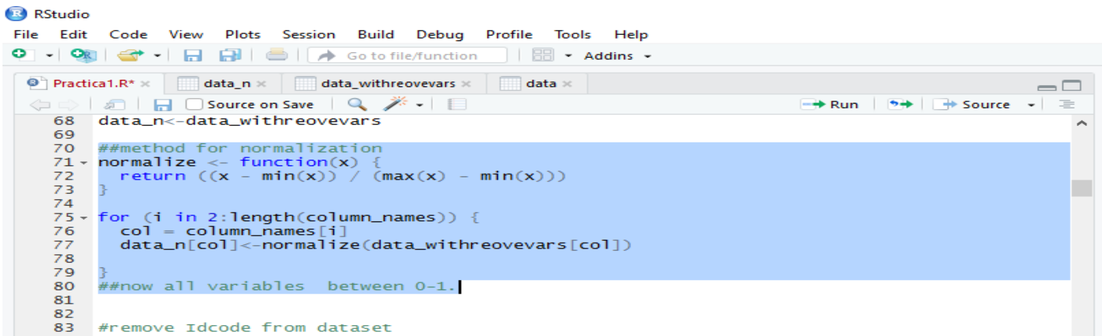
```
> print("The sum of NA's")
[1] "The sum of NA's"
> #or///Check missing values
> print(sum(is.na(data)))
[1] 0
>
> #or///Check missing values
> md.pattern(data)
 /\     /\
{  `---'  }
{  O   O  }
==>  V <==  No need for mice. This data set is completely observed.
 \  \|/  /
  `-----'

       ID_code target var_0 var_1 var_2 var_3 var_4 var_5 var_6 var_7 var_8 var_9 var_10
200000       1      1     1     1     1     1     1     1     1     1     1     1      1
```

c- Normalization

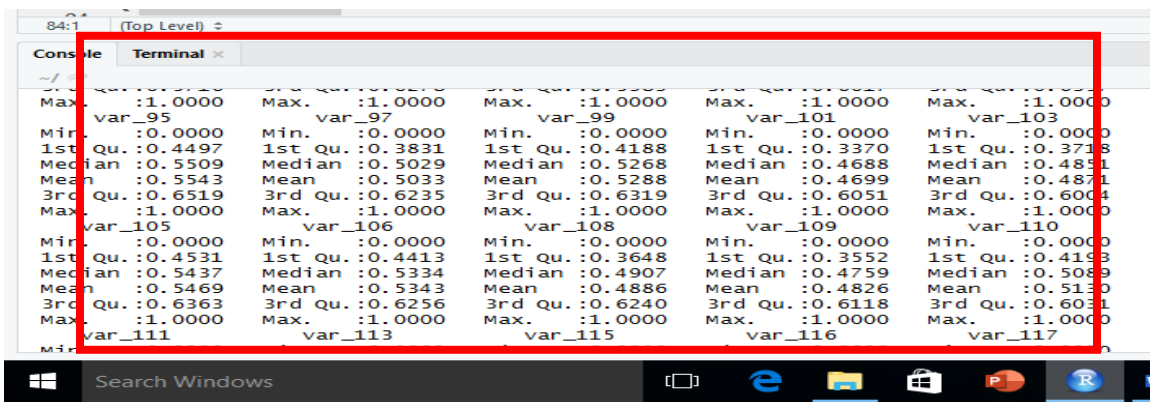
we realize that we have a lot of variables it's very unbalanced
for example, the values of var_0 be between (0.4084….10.6799) either that the values of var_1 be between (-15.043…. -1.628) and we have many cases same that, the Figure below show that

```
> #or///Check missing values it works just with NA's values
> print(sum(is.na(data_withreovevars)))
[1] 2615
> summary(data)
        ID_code              target             var_0             var_1
 train_0    :     1   Min.   :0.00000   Min.   : 0.4084   Min.   :-15.043
 train_1    :     1   1st Qu.:0.00000   1st Qu.: 8.4536   1st Qu.: -4.740
 train_10   :     1   Median :0.00000   Median :10.5248   Median : -1.608
 train_100  :     1   Mean   :0.09005   Mean   :10.6799   Mean   : -1.628
     var_2              var_3              var_4              var_5               var_6
 Min.   : 2.117   Min.   :-0.0402   Min.   : 5.075   Min.   :-32.5626   Min.   :2.347
 1st Qu.: 8.722   1st Qu.: 5.2542   1st Qu.: 9.883   1st Qu.:-11.1997   1st Qu.:4.768
 Median :10.580   Median : 6.8251   Median :11.108   Median : -4.8333   Median :5.385
 Mean   :10.715   Mean   : 6.7965   Mean   :11.078   Mean   : -5.0652   Mean   :5.409
     var_7             var_8              var_9              var_10              var_11
 Min.   : 5.35   Min.   :-10.5055   Min.   : 3.970   Min.   :-20.7313   Min.   :-26.095
 1st Qu.:13.94   1st Qu.: -2.3178   1st Qu.: 6.619   1st Qu.: -3.5949   1st Qu.: -7.510
```

My solve has been making all of values of variables between 0-1 by using (max-min values),



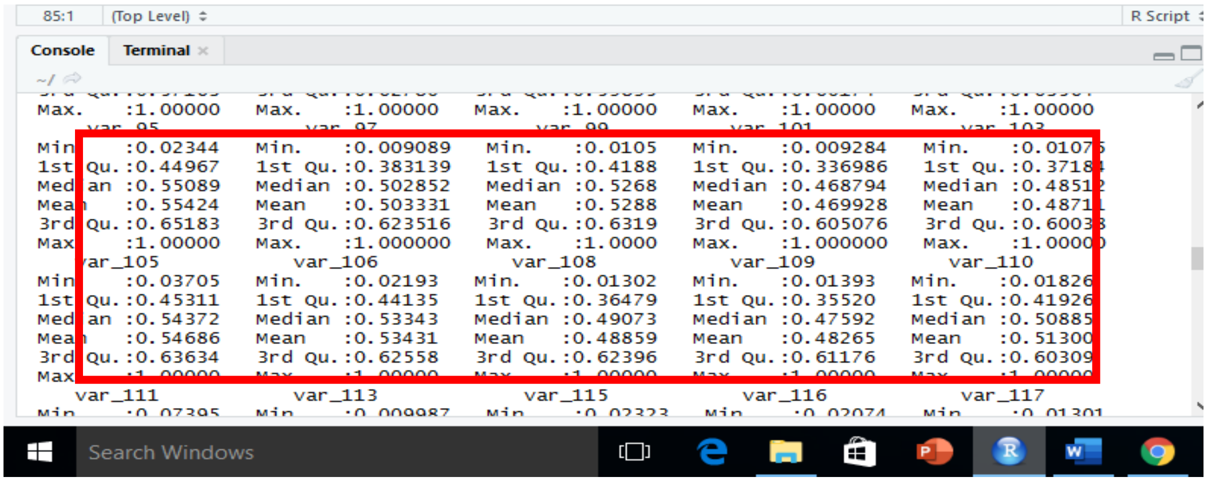
After that Solution will Appear values equal zero or very near of zero,

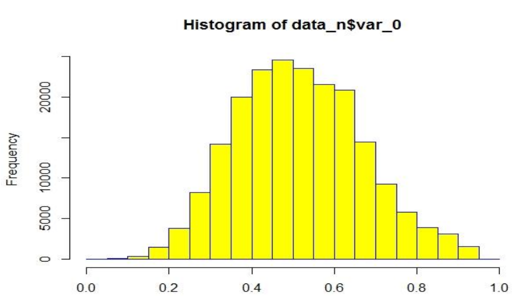


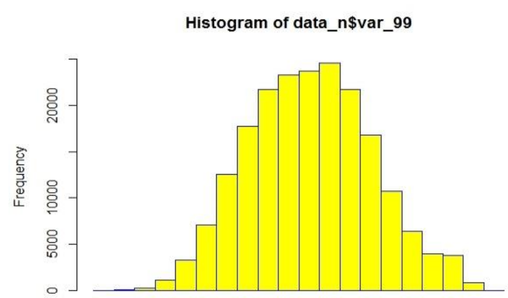
so will remove them with simple R code that be

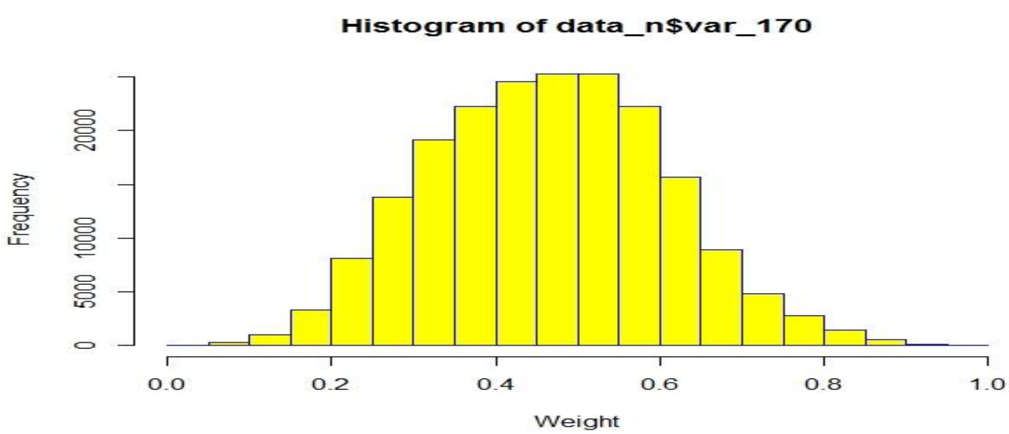
```
column_names=names(data_n)
#remove zero Values from dataset
for (i in 2:length(column_names)) {
  col = column_names[i]
  data_n=subset(data_n,data_n[,col]>0.009)
`
```

The summary of dataset with normalization will be (<1 and > 0.009) like these values in the figure below









Now our dataset is ready to split into two datasets ''train and test'', The data will be fragmented depending on target variable, the 70% of data will be in train dataset and 30% will be in test dataset the R code below show that.

```
## set the seed to make your partition reproducible
set.seed(123)
sample = sample.split(data_n$target, SplitRatio = .7)
train = subset(data_n, sample == TRUE)
test  = subset(data_n, sample == FALSE)
///
> dim(train)
[1] 139846    130
> dim(test)
[1] 59935   130
```

3- Classification techniques

- **OneR:**

  OneR, short for "One Rule", is a simple, yet accurate, classification algorithm that generates one rule for each predictor in the data, then selects the rule with the smallest total error as its "one rule". To create a rule for a predictor, we construct a frequency table for each predictor against the target. It has been shown that OneR produces rules only slightly less accurate than state-of-the-art classification algorithms while producing rules that are simple for humans to interpret.

- **OneR Algorithm:**

For each predictor, For each value of that predictor, make a rule as follows; Count how often each value of target (class) appears Find the most frequent class Make the rule assign that class to this value of the predictor Calculate the total error of the rules of each predictor Choose the predictor with the smallest total error.

Example:

Finding the best predictor with the smallest total error using OneR algorithm based on related frequency tables.

Which one is the best predictor ?

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

Frequency Tables

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3

The best predictor is:

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3

IF Outlook = Sunny THEN PlayGolf = Yes
IF Outlook = Overcast THEN PlayGolf = Yes
IF Outlook = Rainy THEN PlayGolf = No

Predictors Contribution

Simply, the total error calculated from the frequency tables is the measure of each predictor contribution. A low total error means a higher contribution to the predictability of the model.

Model Evaluation The following confusion matrix shows significant predictability power. OneR does not generate score or probability, which means evaluation charts (Gain, Lift, K-S and ROC) are not applicable.

Confusion Matrix		Play Golf			
		Yes	No		
OneR	Yes	7	2	Positive Predictive Value	0.78
	No	2	3	Negative Predictive Value	0.60
		Sensitivity	Specificity	Accuracy = 0.71	
		0.78	0.60		

- **The working with OneR**

  It's so easy to work with it in R Language we will need 'library(OneR)'
  Now we have train dataset to apply this algorithm and test for do testing and show the result, accuracy depending on the target that are 0,1 (Yes, No), the R code will be :

```
library(OneR)
train <- optbin(target ~ ., data = train)
model <- OneR(train, verbose = TRUE)
summary(model)
qplot(model)
pred <- predict(model, test)
eval_model(pred, test$target)
```

Now the result will be:

```
library(OneR)
train <- optbin(target ~ ., data = train)
model <- OneR(train, verbose = TRUE)
summary(model)
plot(model)
pred <- predict(model, test)
eval_model(pred, test$target)
```

```
> train <- optbin(target ~ ., data = train)
Warning message:
In optbin.data.frame(x = data, method = method, na.omit = na.omit) :
  target is numeric
> model <- OneR(train, verbose = TRUE)

    Attribute Accuracy
1 * var_0     90.99%
1   var_1     90.99%
1   var_2     90.99%
1   var_3     90.99%
1   var_5     90.99%
1   var_7     90.99%
1   var_9     90.99%
1   var_11    90.99%
1   var_12    90.99%
1   var_13    90.99%
```

```
> summary(model)

Call:
OneR.data.frame(x = train, verbose = TRUE)

Rules:
If var_0 = (0.0252,0.539] then target = 0
If var_0 = (0.539,1]      then target = 0

Accuracy:
127252 of 139846 instances classified correctly (90.99%)

Contingency table:
      var_0
target (0.0252,0.539] (0.539,1]    Sum
   0          * 73741   * 53511 127252
   1            6508      6086  12594
   Sum         80249     59597 139846
---
Maximum in each column: '*'

Pearson's Chi-squared test:
X-squared = 184.17, df = 1, p-value < 2.2e-16

> |
```

➢ KNN (K-Nearest Neighbour):

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique.

➢ KNN algorithm

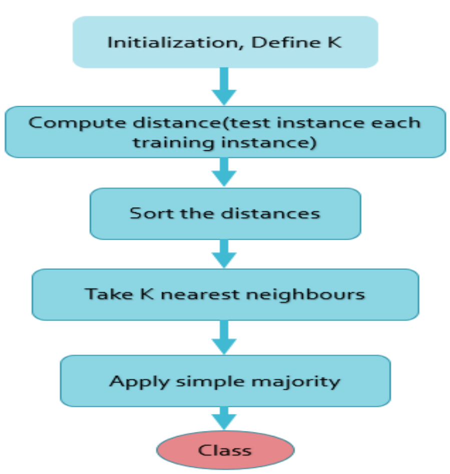
A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K = 1, then the case is simply assigned to the class of its nearest neighbor.

Distance functions

Euclidean $$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Manhattan $$\sum_{i=1}^{k}|x_i - y_i|$$

Minkowski $$\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$$



It should also be noted that all three distance measures are only valid for continuous variables. In the instance of categorical variables, the Hamming distance must be used. It also brings up the issue of standardization of the numerical variables between 0 and 1 when there is a mixture of numerical and categorical variables in the dataset.

Hamming Distance

$$D_H = \sum_{i=1}^{k} |x_i - y_i|$$

$$x = y \Rightarrow D = 0$$

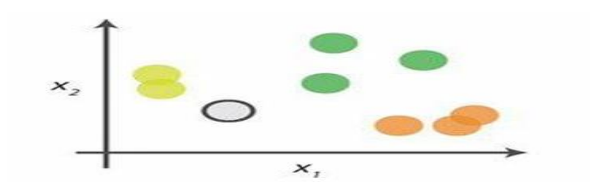
$$x \neq y \Rightarrow D = 1$$

X	Y	Distance
Male	Male	0
Male	Female	1

Choosing the optimal value for K is best done by first inspecting the data. In general, a large K value is more precise as it reduces the overall noise but there is no guarantee. Cross-validation is another way to retrospectively determine a good K value by using an independent dataset to validate the K value. Historically, the optimal K for most datasets has been between 3-10. That produces much better results than 1NN.

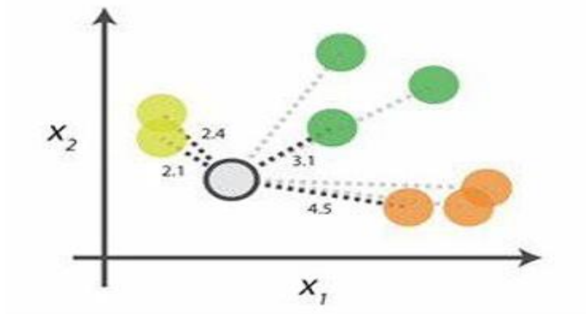
Example:

o.look at the data



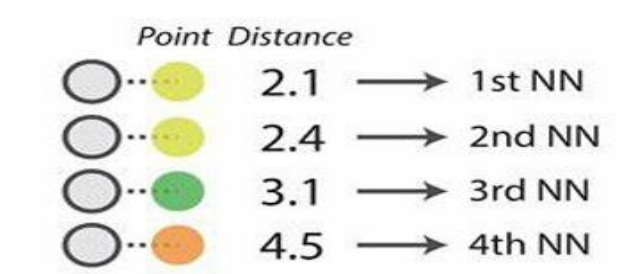
Say you want to classify the gray point into a class here, there are three potential classes, lime green, green and orange.

1.Calculate distances



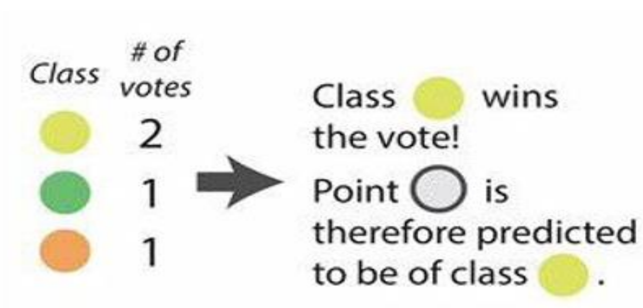
Start by calculating the distances between the grey point and all other points.

2.Find neighbours



Next , find the nearest neighbours by ranking points by increasing distance. The nearest neighbours (NNs) of the grey point are the ones closest in dataspace.

3.Vote on labels



Vote on the predicted class labels based on the classes of the k nearest neighbours . Here the labels were predicted based on the k=3 nearest neighbours

- **The working with OneR**

  It's so easy to work with it in R Language we will need 'library(class)'

Now we have train dataset to apply this algorithm and test for do testing and show the result, accuracy depending on the target that are 0,1 (Yes, No), We have built the model we also need to check the accuracy of the predicted values in test_s as to whether they match up with the known values in test_labels. To ensure this, we need to use the CrossTable() function available in the package 'gmodels', the R code will be :

```
train_lables <- data_withreovevars[sample,1]
test_lables <- data_withreovevars[140001:200000,1]
library(class)
library(gmodels)
knn <- knn(train = train_s,test = test_s ,cl = train_lables,k = 64)
CrossTable(x=test_lables,y=knn,prop.chisq = FALSE )
```

Now the result with K=64 will be:

```
Source
Console   Terminal
~/
> length(test_lables)
[1] 60000
> knn <- knn(train = train_s,test = test_s ,cl = train_lables,k = 64)
> CrossTable(x=test_lables,y=knn,prop.chisq = FALSE )


   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  60000


             | knn
 test_lables |         0 |         1 | Row Total |
-------------|-----------|-----------|-----------|
           0 |     49611 |      4932 |     54543 |
             |     0.910 |     0.090 |     0.909 |
             |     0.909 |     0.913 |           |
             |     0.827 |     0.082 |           |
-------------|-----------|-----------|-----------|
           1 |      4986 |       471 |      5457 |
             |     0.914 |     0.086 |     0.091 |
             |     0.091 |     0.087 |           |
             |     0.083 |     0.008 |           |
-------------|-----------|-----------|-----------|
Column Total |     54597 |      5403 |     60000 |
             |     0.910 |     0.090 |           |
-------------|-----------|-----------|-----------|
```

The test data consisted of 60000 observations. Out of which 49611 cases have been accurately predicted (TN->True Negatives) as zero (NO) in nature which constitutes 82.7%. Also, 471 out of 60000 observations were accurately predicted (TP-> True Positives) as One (Yes) in nature which constitutes 8%. Thus a total of 471 out of 60000 predictions where TP i.e, True Positive in nature.

The FN's if any poses a potential threat for the same reason and the main focus to increase the accuracy of the model is to reduce FN's.

There were 4932 cases of False Positives (FP) meaning 4932 cases were actually benign in nature but got predicted as One(yes).

The total accuracy of the model is 90.7 %((TN+TP)/60000) which shows that there may be chances to improve the model performance.

- **Improve the performance of the model**

  This can be taken into account by repeating the steps 3 and 4 and by changing the k-value. Generally, it is the square root of the observations and in this case, we took k=64.The k-value may be fluctuated in and around the value of 64 to check the increased accuracy of the model. We will try it out with values of your choice to increase the accuracy! Also, we need remembering, to keep the value of FN's as low as possible. But we should remember that Classification is sensitive to the correct selection of k
  small k? → less stable, influenced by noise
  larger k? → less precise, higher bias

4- Conclusions

the analysis of the dataset has been a laborious task from its size and the little intuitiveness of its values could not be seen at first view the most differentiating variables addition the great time that has taken the cleaning itself of the dataset which has taken most of the time since many tests have been done to finally stay with the most important variables,

In addition, the numerous problems obtained to obtain the models have been added, since the vast majority have taken several hours its execution which prevented that there was much room to test the different possibilities as for example in the case that has been attempted apply ***KNN*** but because it took more than 4 hours,

Finally, it should be noted that the model that has yielded the best results has been that of OneR. The problem of the dataset has been mainly the great imbalance between all values.

5- Bibliography

- http://www.saedsayad.com/oner.htm
- https://www.saedsayad.com/k_nearest_neighbors.htm
- https://www.analyticsvidhya.com/blog/2015/08/learning-concept-knn-algorithms-programming/
- https://cran.rproject.org/web/packages/OneR/vignettes/OneR.html
- https://www.tutorialspoint.com/r/r_histograms.htm
- https://www.rdocumentation.org/packages/psych/versions/1.8.12/topics/multi.hist