#Data contains 654 observations and 11 Variables

> **# Convert R1 into Factor variables Yes or No**

```
> data1$R1[data1$R1 == 0 ] <- 'No'
> data1$R1[data1$R1 == 1] <- 'Yes'
> data1$R1 <- factor(data1$R1)
>
```

> **#Create independent Samples for training, validation and test dataset**

```
> set.seed(1234)
> ind <- sample(3, nrow(data1), replace = T, prob = c(0.7, 0.15, 0.15))
> training <- data1[ind == 1,]
> test <- data1[ind == 2,]
> validation <- data1[ind == 3,]
```

```
> str(training)
'data.frame':	454 obs. of  11 variables:
$ A1 : int  1 0 0 1 1 1 0 1 1 1 ...
$ A2 : num  30.8 58.7 24.5 27.8 32.1 ...
$ A3 : num  0 4.46 0.5 1.54 4 ...
$ A8 : num  1.25 3.04 1.5 3.75 2.5 ...
$ A9 : int  1 1 1 1 1 1 1 1 1 0 ...
$ A10: int  0 0 1 0 1 1 1 1 1 1 ...
$ A11: int  1 6 0 5 0 0 0 0 0 0 ...
$ A12: int  1 1 1 0 0 0 1 1 0 0 ...
$ A14: int  202 43 280 100 360 164 80 180 52 128 ...
$ A15: int  0 560 824 3 0 31285 1349 314 1442 0 ...
$ R1 : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 2 2 …
```

```
> str(test)
'data.frame':	98 obs. of  11 variables:
```

```
$ A1 : int  1 1 1 1 1 0 0 1 1 0 ...
$ A2 : num  20.2 48.1 56.6 27.8 56.8 ...
$ A3 : num  5.625 6.04 18.5 0.585 12.25 ...
$ A8 : num  1.71 0.04 15 0.25 1.25 13.5 2 1 9.46 0.5 ...
$ A9 : int  1 0 1 1 1 1 1 1 1 1 ...
$ A10: int  1 1 0 0 0 1 1 1 1 1 ...
$ A11: int  0 0 17 2 4 0 0 0 0 0 ...
$ A12: int  1 1 0 1 0 0 1 0 0 0 ...
$ A14: int  120 0 0 260 200 980 368 240 200 171 ...
$ A15: int  0 2690 0 500 0 0 0 0 100 0 ...
$ R1 : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 1 1 1 1 ...

> str(validation)
'data.frame':	102 obs. of  11 variables:
$ A1 : int  1 0 1 1 1 1 1 1 1 1 ...
$ A2 : num  36.7 15.8 57.4 27.8 54.6 ...
$ A3 : num  4.415 0.585 8.5 1.5 9.415 ...
$ A8 : num  0.25 1.5 7 2 14.41 ...
$ A9 : int  1 1 1 1 1 0 1 1 1 1 ...
$ A10: int  0 0 0 0 0 1 1 1 0 0 ...
$ A11: int  10 2 3 11 11 0 0 0 11 1 ...
$ A12: int  0 1 1 0 0 1 0 0 1 0 ...
$ A14: int  320 100 0 434 30 100 400 320 80 520 ...
$ A15: int  0 0 0 35 300 0 5800 0 0 50000 ...
$ R1 : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 2 2 ...
```

```
> # Create train control prior to create KNN Model with number of iteration as 10
> #and cross - validation is 3 times, This will control all the computational overheads
> library(caret)
> trControl <- trainControl(method = "repeatedcv",
+                           number = 10,
+                           repeats = 3,
+                           classProbs = TRUE,
+                           summaryFunction = twoClassSummary)
>
>
> set.seed(222)
> fit <- train(R1 ~ .,
+              data = training,
+              method = 'knn',
+              tuneLength = 20,
+              trControl = trControl,
+              preProc = c("center", "scale"),
+              metric = "ROC",
+              tuneGrid = expand.grid(k = 1:60))
>
> #K Nearest neighbors
> fit
k-Nearest Neighbors

454 samples
10 predictor
  2 classes: 'No', 'Yes'
```

Pre-processing: centered (10), scaled (10)

Resampling: Cross-Validated (10 fold, repeated 3 times)

Summary of sample sizes: 408, 409, 409, 409, 408, 408, ...

Resampling results across tuning parameters:

k	ROC	Sens	Spec
1	0.7714048	0.8173333	0.7254762
2	0.8388429	0.8253333	0.7272222
3	0.8720175	0.8560000	0.7860317
4	0.8839270	0.8573333	0.7875397
5	0.8890444	0.8560000	0.8122222
6	0.8892222	0.8520000	0.8235714
7	0.8968206	0.8666667	0.8252381
8	0.8996365	0.8640000	0.8152381
9	0.9034190	0.8653333	0.8201587
10	0.9046603	0.8640000	0.8220635
11	0.9069063	0.8626667	0.8170635
12	0.9058460	0.8600000	0.8119048
13	0.9071556	0.8666667	0.7987302
14	0.9067556	0.8626667	0.8069841
15	0.9075159	0.8666667	0.8019841
16	0.9077508	0.8640000	0.7988095
17	0.9077032	0.8680000	0.7908730
18	0.9080365	0.8746667	0.7909524
19	0.9086540	0.8680000	0.7860317
20	0.9097667	0.8800000	0.7810317
21	0.9094127	0.8800000	0.7727778
22	0.9091905	0.8786667	0.7776190
23	0.9096190	0.8866667	0.7776190

24	0.9108222	0.8786667	0.7725397
25	0.9121667	0.8893333	0.7776190
26	0.9124095	0.8906667	0.7742857
27	0.9126984	0.8920000	0.7726984
28	0.9133492	0.8880000	0.7727778
29	0.9146524	0.8880000	0.7661905
30	0.9159222	0.8893333	0.7711111
31	0.9158143	0.8920000	0.7742857
32	0.9168651	0.8866667	0.7708730
33	0.9163206	0.8986667	0.7742857
34	0.9152032	0.9000000	0.7696032
35	0.9152714	0.9000000	0.7661111
36	0.9163317	0.8973333	0.7645238
37	0.9166698	0.8986667	0.7661905
38	0.9172492	0.8986667	0.7679365
39	0.9171651	0.9000000	0.7645238
40	0.9174254	0.8986667	0.7611905
41	0.9180365	0.9000000	0.7611905
42	0.9181111	0.9013333	0.7645238
43	0.9185587	0.8986667	0.7628571
44	0.9185905	0.9000000	0.7596825
45	0.9179032	0.9000000	0.7645238
46	0.9185905	0.9000000	0.7627778
47	0.9195524	0.9013333	0.7628571
48	0.9195794	0.9026667	0.7564286
49	0.9195540	0.9000000	0.7562698
50	0.9193556	0.9026667	0.7561905
51	0.9190063	0.9026667	0.7530952
52	0.9185000	0.9040000	0.7530159

53	0.9176048	0.9040000	0.7515079
54	0.9163698	0.9053333	0.7515079
55	0.9166603	0.9066667	0.7448413
56	0.9156905	0.9053333	0.7481746
57	0.9152698	0.9066667	0.7447619
58	0.9154746	0.9066667	0.7447619
59	0.9156730	0.9080000	0.7464286
60	0.9161635	0.9093333	0.7415079

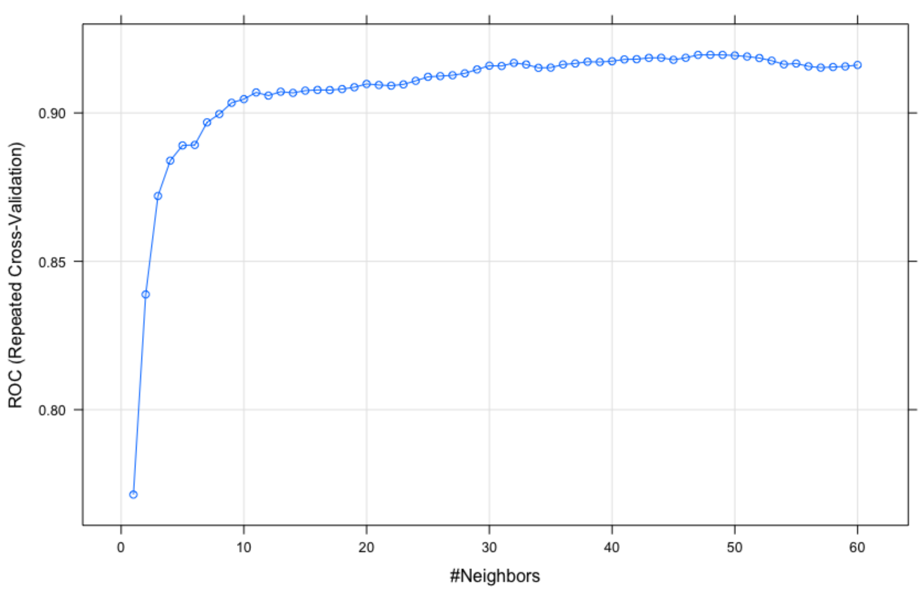
ROC was used to select the optimal model using the largest value.

The final value used for the model was k = 48.

>

> plot(fit)

>



> # ROC Curve values

> varImp(fit)

ROC curve variable importance

```
    Importance
A9     100.000
A11     72.020
A8      68.708
A10     60.547
A15     46.409
A3      31.391
A2      22.245
A14     18.801
A1       5.763
A12      0.000
>
> #Matrix and Stats
>
> pred <- predict(fit, newdata = test)
> confusionMatrix(pred,test$R1)
Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No  48   7
       Yes  5  38

               Accuracy : 0.8776
                 95% CI : (0.7959, 0.9351)
    No Information Rate : 0.5408
    P-Value [Acc > NIR] : 9.362e-13

                  Kappa : 0.7526
```

Mcnemar's Test P-Value : 0.7728

Sensitivity : 0.9057

Specificity : 0.8444

Pos Pred Value : 0.8727

Neg Pred Value : 0.8837

Prevalence : 0.5408

Detection Rate : 0.4898

Detection Prevalence : 0.5612

Balanced Accuracy : 0.8751

'Positive' Class : No

>

> **# Model accuracy is 87.76% with 84 correct classification out of 95**