Assignment_ML2

```
chooseCRANmirror(graphics = getOption("menu.graphics"), ind = 79,
                 local.only = FALSE)
install.packages("caret")
```

```
## Installing package into 'C:/Users/ibeme/Documents/R/win-library/4.1'
## (as 'lib' is unspecified)
```

```
## package 'caret' successfully unpacked and MD5 sums checked
```

```
## Warning: cannot remove prior installation of package 'caret'
```

```
## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying C:
## \Users\ibeme\Documents\R\win-library\4.1\00LOCK\caret\libs\x64\caret.dll to C:
## \Users\ibeme\Documents\R\win-library\4.1\caret\libs\x64\caret.dll: Permission
## denied
```

```
## Warning: restored 'caret'
```

```
##
## The downloaded binary packages are in
##  C:\Users\ibeme\AppData\Local\Temp\RtmpMfDDfd\downloaded_packages
```

```
install.packages("ISLR")
```

```
## Installing package into 'C:/Users/ibeme/Documents/R/win-library/4.1'
## (as 'lib' is unspecified)
```

```
## package 'ISLR' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\ibeme\AppData\Local\Temp\RtmpMfDDfd\downloaded_packages
```

```
install.packages("class")
```

```
## Installing package into 'C:/Users/ibeme/Documents/R/win-library/4.1'
## (as 'lib' is unspecified)
```

```
## package 'class' successfully unpacked and MD5 sums checked
```

```
## Warning: cannot remove prior installation of package 'class'
```

```
## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying C:
## \Users\ibeme\Documents\R\win-library\4.1\00LOCK\class\libs\x64\class.dll to C:
## \Users\ibeme\Documents\R\win-library\4.1\class\libs\x64\class.dll: Permission
## denied

## Warning: restored 'class'

##
## The downloaded binary packages are in
##  C:\Users\ibeme\AppData\Local\Temp\RtmpMfDDfd\downloaded_packages
```

```
library(class)
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

An Example

Data Exploration

```
# Importing the file
Universal_data<- read.csv("UniversalBank.csv")

#Eliminating variables [ID & Zipcode] from the dataset.

Universal_new_data <- Universal_data[c(-1,-5)]

#Converting Categorical variables into dummy Variables

dummy_family <- dummyVars(~Family, data = Universal_new_data)
head(predict(dummy_family,Universal_new_data))
```

```
##   Family
## 1      4
## 2      3
## 3      1
## 4      1
## 5      4
## 6      4
```

```
dummy_Education <- dummyVars(~Education,data = Universal_new_data)
head(predict(dummy_Education,Universal_new_data))
```

```
##   Education
## 1         1
## 2         1
## 3         1
## 4         2
## 5         2
## 6         2
```

Data Partitioning

```
# Splitting the data into training(60%) and validation(40%)

set.seed(123)
train_index <- createDataPartition(Universal_new_data$Personal.Loan,p=0.6,list = FALSE)
train_data <- Universal_new_data[train_index,] #3000 Observations
validation_data <- Universal_new_data[-train_index,] #2000 Observations
```

Generating Test Data

```
test_data<-data.frame(Age = 40, Experience = 10, Income = 84, Family = 2, CCAvg = 2, Education = 2, Mor
```

Data Normalization

```
# Copy the original data
train.norm.df <- train_data
valid.norm.df <- validation_data
test.norm.df <- test_data
traval.norm.df <- Universal_new_data #(Training + Validation data)

#Use preProcess() function to normalize numerical columns from the dataset

Values_z_normalised <- preProcess(train_data[,c(1:3,5,7)],method = c("center","scale"))

#Applying the normalised model on the training and validation data

train.norm.df[,c(1:3,5,7)] <-  predict(Values_z_normalised,train_data[,c(1:3,5,7)])
valid.norm.df[,c(1:3,5,7)] <-  predict(Values_z_normalised,validation_data[,c(1:3,5,7)])
#traval.norm.df[,c(1:3,5,7)] <-  predict(Values_z_normalised,Universal_new_data[,c(1:3,5,7)])
#test.norm.df <- predict(Values_z_normalised, test_data)

#summary(train.norm.df)
#var(train.norm.df[, c(1:3,5,7)])
#summary(valid.norm.df)
#var(valid.norm.df[, c(1:3,5,7)])
```

Modeling k-NN

```
set.seed(123)

Model.k.1<- knn(train=train.norm.df[,c(1:3,5,7)],test=valid.norm.df[,c(1:3,5,7)],cl= train.norm.df[,8],

actual=valid.norm.df[,8]
Prediction_prob =attr(Model.k.1,"prob")


table(Model.k.1, actual)
```

```
##          actual
## Model.k.1    0    1
##         0 1730   96
##         1   68  106
```

```
mean(Model.k.1 == actual)
```

```
## [1] 0.918
```

```
#row.names(train_data)[attr(Model.k.1, "Model.k.1.index")]
```

Classifying the customer using the k=1 [Performing KNN classification on test data]

```
# Renormalizing the (training+validation) data
set.seed(123)
Values_z_normalised2 <- preProcess(traval.norm.df[,c(1:3,5,7)], method = c("center","scale"))

traval.norm.df[,c(1:3,5,7)] <-  predict(Values_z_normalised2,Universal_new_data[,c(1:3,5,7)])
test.norm.df<- predict(Values_z_normalised2,test_data[,c(1:3,5,7)])


Prediction_test <- knn(train= traval.norm.df[,c(1:3,5,7)],test=test.norm.df,cl=traval.norm.df[,8],k=1,p

head(Prediction_test)
```

```
## [1] 0
## Levels: 0 1
```

```
Prediction_test_prob<-attr(Prediction_test,"prob")

Prediction_test_prob
```

```
## [1] 1
```

####Generating loop to find best k

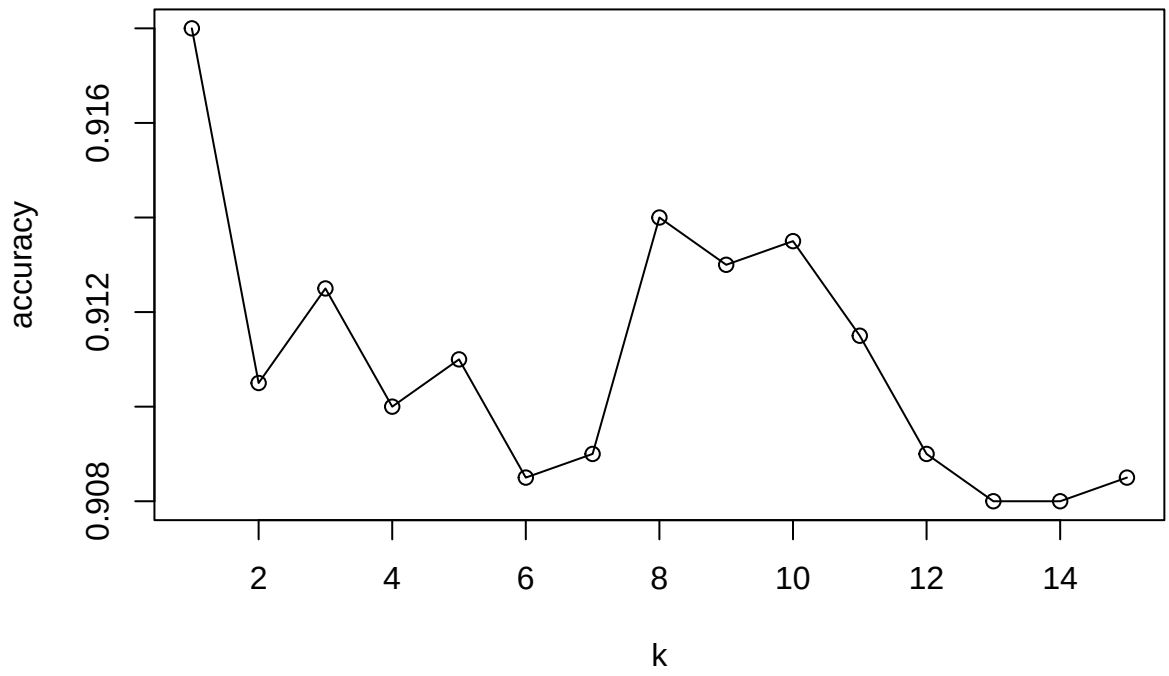
```r
#To determine k, we use the performance on the validation set.Here, we will vary the value of k from 1
# initialize a data frame with two columns: k, and accuracy.

library(caret)
set.seed(123)
accuracy.df <- data.frame(k = seq(1, 15, 1), accuracy = rep(0, 15))

# compute knn for different k on validation.
for(i in 1:15) {
  knn.pred <- knn(train.norm.df[,c(1:3,5,7)], valid.norm.df[,c(1:3,5,7)],
                  cl = train.norm.df[, 8], k = i,prob=TRUE)
  accuracy.df[i, 2] <- mean(knn.pred==actual)
}
accuracy.df
```

```
##     k accuracy
## 1   1   0.9180
## 2   2   0.9105
## 3   3   0.9125
## 4   4   0.9100
## 5   5   0.9110
## 6   6   0.9085
## 7   7   0.9090
## 8   8   0.9140
## 9   9   0.9130
## 10 10   0.9135
## 11 11   0.9115
## 12 12   0.9090
## 13 13   0.9080
## 14 14   0.9080
## 15 15   0.9085
```

```r
plot(accuracy.df,type='o')
```



Repartition

```
## Data Splitting (50% Training Data and 30% for validation data and 20% test data)
set.seed(123)
test_index1 <- createDataPartition(Universal_new_data$Personal.Loan,p=0.2,list = FALSE)
Test_Data2 <- Universal_new_data[test_index1,]# 1000 Rows (Test data)
train_vali_data <- Universal_new_data[-test_index1,]

train_index2 <- createDataPartition(train_vali_data$Personal.Loan,p=0.625,list = FALSE)
train_data2 <- train_vali_data[train_index2,] #2500 Rows (Training data)
validation_data2 <- train_vali_data[-train_index2,]#1500 Rows (validation data)

 NROW(Test_Data2)
```

```
## [1] 1000
```

```
 NROW(train_data2)
```

```
## [1] 2500
```

```
 NROW(validation_data2)
```

```
## [1] 1500
```

```r
 # Data Normalization

# Copy the original data
train.norm.df2 <- train_data2
valid.norm.df2 <- validation_data2
train_vali.norm.df <- train_vali_data
test.norm.df2 <-Test_Data2

#Use preProcess() function to normalize numerical columns from the Universal_new_data dataset

Values_z_normalised_repartition <- preProcess(train_data2[,c(1:3,5,7)],method = c("center","scale"))


train.norm.df2[,c(1:3,5,7)] <- predict(Values_z_normalised_repartition,train_data2[,c(1:3,5,7)])
valid.norm.df2[,c(1:3,5,7)] <- predict(Values_z_normalised_repartition,validation_data2[,c(1:3,5,7)])
train_vali.norm.df[,c(1:3,5,7)] <- predict(Values_z_normalised2,train_vali_data[,c(1:3,5,7)])
test.norm.df2[,c(1:3,5,7)] <-predict(Values_z_normalised_repartition,Test_Data2[,c(1:3,5,7)])


#summary(train.norm.df2)
var(train.norm.df2[, c(1:3,5,7)])
```

```
##                    Age   Experience       Income       CCAvg    Mortgage
## Age         1.00000000  0.994600424 -0.012432135 -0.03068362 -0.01454453
## Experience  0.99460042  1.000000000 -0.003516967 -0.02711063 -0.01399839
## Income     -0.01243213 -0.003516967  1.000000000  0.64709414  0.20669998
## CCAvg      -0.03068362 -0.027110631  0.647094136  1.00000000  0.11467646
## Mortgage   -0.01454453 -0.013998388  0.206699984  0.11467646  1.00000000
```

```r
summary(valid.norm.df2)
```

```
##       Age               Experience           Income             Family
##  Min.   :-1.922547   Min.   :-1.905010   Min.   :-1.39854   Min.   :1.000
##  1st Qu.:-0.791022   1st Qu.:-0.774038   1st Qu.:-0.71472   1st Qu.:1.000
##  Median :-0.007659   Median : 0.008943   Median :-0.18048   Median :2.000
##  Mean   : 0.038008   Mean   : 0.031447   Mean   : 0.04101   Mean   :2.394
##  3rd Qu.: 0.862744   3rd Qu.: 0.878922   3rd Qu.: 0.58883   3rd Qu.:3.000
##  Max.   : 1.907228   Max.   : 2.009895   Max.   : 3.21728   Max.   :4.000
##      CCAvg            Education        Mortgage         Personal.Loan
##  Min.   :-1.10529   Min.   :1.000   Min.   :-0.547467   Min.   :0.00
##  1st Qu.:-0.70265   1st Qu.:1.000   1st Qu.:-0.547467   1st Qu.:0.00
##  Median :-0.18496   Median :2.000   Median :-0.547467   Median :0.00
##  Mean   : 0.03919   Mean   :1.872   Mean   : 0.004023   Mean   :0.09
##  3rd Qu.: 0.39025   3rd Qu.:3.000   3rd Qu.: 0.451773   3rd Qu.:0.00
##  Max.   : 4.64677   Max.   :3.000   Max.   : 4.992006   Max.   :1.00
##  Securities.Account   CD.Account        Online         CreditCard
##  Min.   :0.0000     Min.   :0.000   Min.   :0.0000   Min.   :0.000
##  1st Qu.:0.0000     1st Qu.:0.000   1st Qu.:0.0000   1st Qu.:0.000
##  Median :0.0000     Median :0.000   Median :1.0000   Median :0.000
##  Mean   :0.1073     Mean   :0.068   Mean   :0.5993   Mean   :0.292
##  3rd Qu.:0.0000     3rd Qu.:0.000   3rd Qu.:1.0000   3rd Qu.:1.000
##  Max.   :1.0000     Max.   :1.000   Max.   :1.0000   Max.   :1.000
```

```
var(valid.norm.df2[, c(1:3,5,7)])
```

```
##                    Age  Experience     Income       CCAvg    Mortgage
## Age         0.98313414  0.97476588 -0.1105698 -0.06930295 -0.01803358
## Experience  0.97476588  0.97832584 -0.1041392 -0.07087030 -0.01547319
## Income     -0.11056981 -0.10413923  0.9733494  0.67338854  0.20948770
## CCAvg      -0.06930295 -0.07087030  0.6733885  1.09503588  0.10366299
## Mortgage   -0.01803358 -0.01547319  0.2094877  0.10366299  0.93739041
```

```
## Modeling k-NN

set.seed(123)
ModelNew.k.1<- knn(train.norm.df2[,c(1:3,5,7)],valid.norm.df2[,c(1:3,5,7)],cl= train.norm.df2[,8],k=1,p
print(ModelNew.k.1)
```

```
##    [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##   [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##   [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
##  [112] 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [149] 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0
##  [186] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
##  [223] 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [260] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [297] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
##  [334] 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0
##  [371] 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0
##  [408] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
##  [445] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [482] 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0
##  [519] 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0
##  [556] 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0
##  [593] 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
##  [630] 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
##  [667] 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0
##  [704] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
##  [741] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [778] 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [815] 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0
##  [852] 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0
##  [889] 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [926] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [963] 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1
## [1000] 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0
## [1037] 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1
## [1074] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0
## [1111] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
## [1148] 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
## [1185] 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
## [1222] 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
## [1259] 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0
## [1296] 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
## [1333] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1370] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0
```

```
## [1407] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1444] 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
## [1481] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
## attr(,"prob")
##    [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##   [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##   [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [297] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [334] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [371] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [408] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [445] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [482] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [519] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [556] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [593] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [630] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [667] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [704] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [741] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [778] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [815] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [852] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [889] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [926] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [963] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1000] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1037] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1074] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1111] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1148] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1185] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1222] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1259] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1296] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1333] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1370] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1407] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1444] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1481] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## Levels: 0 1
```

```
head(ModelNew.k.1)
```

```
## [1] 0 0 0 0 0 0
## Levels: 0 1
```

```
actual= valid.norm.df2[,8]
#row.names(train_data)[attr(Model.k.1, "Model.k.1.index")]
mean(ModelNew.k.1==actual)
```

```
## [1] 0.9166667
```

```
class_prob = attr(ModelNew.k.1,"prob")
class_prob
```

```
##    [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##   [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##   [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [297] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [334] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [371] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [408] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [445] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [482] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [519] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [556] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [593] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [630] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [667] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [704] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [741] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [778] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [815] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [852] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [889] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [926] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [963] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1000] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1037] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1074] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1111] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1148] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1185] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1222] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1259] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1296] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1333] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1370] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1407] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1444] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1481] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
# Prediction for the test data
Values_z_normalised3<- preProcess(train_vali_data[,c(1:3,5,7)], method = c("center","scale"))

train_vali.norm.df[,c(1:3,5,7)] <- predict(Values_z_normalised3,train_vali_data[,c(1:3,5,7)])
test.norm.df2[,c(1:3,5,7)]<- predict(Values_z_normalised3,Test_Data2[,c(1:3,5,7)])


Model_new3<- knn(train_vali.norm.df[,c(1:3,5,7)],test.norm.df2[,c(1:3,5,7)],cl=train_vali.norm.df[,8],k=
print(Model_new3)
```

```
##    [1] 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0
##   [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
##   [75] 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [112] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [149] 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
##  [186] 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
##  [223] 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
##  [260] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
##  [297] 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
##  [334] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
##  [371] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
##  [408] 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [445] 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0
##  [482] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0
##  [519] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
##  [556] 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [593] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [630] 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [667] 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [704] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
##  [741] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0
##  [778] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0
##  [815] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [852] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
##  [889] 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [926] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
##  [963] 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
## [1000] 0
## Levels: 0 1
```

```
head(Model_new3)
```

```
## [1] 0 1 0 0 0 0
## Levels: 0 1
```

```
actual= test.norm.df2[,8]
mean(Model_new3==actual)
```

```
## [1] 0.924
```

```
set.seed(123)
accuracy.df <- data.frame(k = seq(1, 15, 1), accuracy = rep(0, 15))

for(i in 1:15) {
  knn.pred <- knn(train_vali.norm.df[,c(1:3,5,7)], test.norm.df2[,c(1:3,5,7)],
                  cl = train_vali.norm.df[, 8], k = i,prob=TRUE)
  accuracy.df[i, 2] <- mean(knn.pred==actual)
}
accuracy.df
```

```
##     k accuracy
## 1   1    0.924
## 2   2    0.914
## 3   3    0.919
## 4   4    0.914
## 5   5    0.914
## 6   6    0.910
## 7   7    0.917
## 8   8    0.913
## 9   9    0.911
## 10 10    0.913
## 11 11    0.912
## 12 12    0.907
## 13 13    0.906
## 14 14    0.908
## 15 15    0.906
```

Including Confusion Matrix

You can also embed plots, for example:

```
confusionMatrix(ModelNew.k.1,as.factor(valid.norm.df2[,8]),positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1309   69
##          1   56   66
##
##                Accuracy : 0.9167
##                  95% CI : (0.9015, 0.9302)
##     No Information Rate : 0.91
##     P-Value [Acc > NIR] : 0.1966
##
##                   Kappa : 0.4682
##
##  Mcnemar's Test P-Value : 0.2831
##
##             Sensitivity : 0.48889
##             Specificity : 0.95897
##          Pos Pred Value : 0.54098
```

```
##          Neg Pred Value : 0.94993
##              Prevalence : 0.09000
##          Detection Rate : 0.04400
##    Detection Prevalence : 0.08133
##       Balanced Accuracy : 0.72393
##
##        'Positive' Class : 1
##
```

```
confusionMatrix(Model_new3,as.factor(test.norm.df2[,8]),positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 888  51
##          1  25  36
##
##                Accuracy : 0.924
##                  95% CI : (0.9058, 0.9397)
##     No Information Rate : 0.913
##     P-Value [Acc > NIR] : 0.118030
##
##                   Kappa : 0.4468
##
##  Mcnemar's Test P-Value : 0.004135
##
##             Sensitivity : 0.4138
##             Specificity : 0.9726
##          Pos Pred Value : 0.5902
##          Neg Pred Value : 0.9457
##              Prevalence : 0.0870
##          Detection Rate : 0.0360
##    Detection Prevalence : 0.0610
##       Balanced Accuracy : 0.6932
##
##        'Positive' Class : 1
##
```

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.