Stat844 project: Appendix

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 (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:

```
#for date preprocessing
library(lubridate)
```

```
## Warning: package 'lubridate' was built under R version 3.5.3
```

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
##
##     date
```

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.5.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:lubridate':
##
##     intersect, setdiff, union
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tictoc)

source('C:/Users/xpada/Downloads/scatter3d.R')
```

```
## Loading required package: nlme
```

```
##
## Attaching package: 'nlme'
```

```
## The following object is masked from 'package:dplyr':
##
##     collapse
```

```
## This is mgcv 1.8-26. For overview type 'help("mgcv-package")'.
```

```
set.seed(42)
dataset <- read.csv("googleplaystore.csv", header=TRUE, stringsAsFactors = FALSE)

#remove row 10473 as it left-shifted all the information
dataset = dataset[-10473,]

#do some data vizualization

#udpate date column to numeric
curr_date = as.Date("01/09/2018", "%d/%m/%Y")
data_size = dim(dataset)[1]
dates = dataset['Last.Updated']

for (i in 1:data_size){
  t1 = as.Date(dates[i,1], format = "%B %d, %Y")
  dataset['Last.Updated'][i,1] = as.numeric(difftime(curr_date,t1))
  #make size column numeric
  if(grepl('M',dataset['Size'][i,1], fixed=TRUE)){
    dataset['Size'][i,1] = as.numeric(gsub('M', '', dataset['Size'][i,1]))
  } else if(grepl('k',dataset['Size'][i,1], fixed=TRUE)){
    dataset['Size'][i,1] = as.numeric(gsub('k', '', dataset['Size'][i,1]))/1000
  }

  #make installs column numeric
  if(grepl('+',dataset['Installs'][i,1], fixed=TRUE)){
    dataset['Installs'][i,1] = gsub('[+]', '', dataset['Installs'][i,1])
  }
  if(grepl(',',dataset['Installs'][i,1], fixed=TRUE)){
    dataset['Installs'][i,1] = as.numeric(gsub('[,]', '', dataset['Installs'][i,1]))
  }

  #clean version columns
  if(grepl('^[0-9]{1,2}\\.[0-9]+',dataset['Current.Ver'][i,1])){
    dataset['Current.Ver'][i,1] = gsub('^[^0-9]*([0-9]+.[0-9]{1,3}).*','\\1',dataset['Current.Ve
r'][i,1])
  }else{
    dataset['Current.Ver'][i,1] = NA
  }

  if(grepl('^[0-9]{1,2}\\.[0-9]+',dataset['Android.Ver'][i,1])){
    dataset['Android.Ver'][i,1] = gsub('^[^0-9]*([0-9]+.[0-9]{1,3}).*','\\1',dataset['Android.Ve
r'][i,1])
  }else{
    dataset['Android.Ver'][i,1] = NA
  }

  if(grepl('$',dataset['Price'][i,1], fixed=TRUE)){
    dataset['Price'][i,1] = gsub('[$]','',dataset['Price'][i,1])
  }
}

#convert some columns from character to numeric
dataset$Last.Updated = sapply(dataset$Last.Updated, as.numeric)
```

```
dataset[dataset=='Varies with device'] = NA
dataset[dataset=='NaN'] = NA

#remove duplicated value
dataset = dataset[-which(duplicated(dataset$App)),]


#for now, remove all NA entries in response variables.
dataset = dataset[-which(is.na(dataset['Rating'])), ]

#only three column have missing values
#...(data vizualization)

#for now, impute with median
dataset$Size[which(is.na(dataset$Size))] = median(as.numeric(dataset$Size[which(!is.na(dataset$S
ize))]))
dataset$Current.Ver[which(is.na(dataset$Current.Ver))] = median(as.numeric(dataset$Current.Ver[w
hich(!is.na(dataset$Current.Ver))]))
dataset$Android.Ver[which(is.na(dataset$Android.Ver))] = median(as.numeric(dataset$Android.Ver[w
hich(!is.na(dataset$Android.Ver))]))

#convert some columns to numeric
dataset$Current.Ver = as.numeric(dataset$Current.Ver)
dataset$Android.Ver = as.numeric(dataset$Android.Ver)
dataset$Size = as.numeric(dataset$Size)
dataset$Price = as.numeric(dataset$Price)
dataset$Reviews = as.numeric(dataset$Reviews)
dataset$Installs = as.numeric(dataset$Installs)

#change app name to numeric app ID
dataset$App = as.numeric(1:nrow(dataset))

#Since Price covers the meaning of Type, then drop the Type column
# dataset['Type'] = list(NULL)

dataset=dataset %>% mutate_if(is.character, as.factor)


# dataset$Category = as.numeric(dataset$Category)

#add a new feature called Rating value indicator
t2= log(dataset$Installs+1)*log(dataset$Reviews+1)
dataset['RVI'] = t2
```

```
#data vizualization
library(ggplot2)
library(highcharter)
```

```
## Warning: package 'highcharter' was built under R version 3.5.3
```

```
## Highcharts (www.highcharts.com) is a Highsoft software product which is
```

```
## not free for commercial and Governmental use
```

```
library(binr)
library(Hmisc)
```

```
## Warning: package 'Hmisc' was built under R version 3.5.3
```

```
## Loading required package: lattice
```

```
## Loading required package: survival
```

```
## Loading required package: Formula
```

```
##
## Attaching package: 'Hmisc'
```

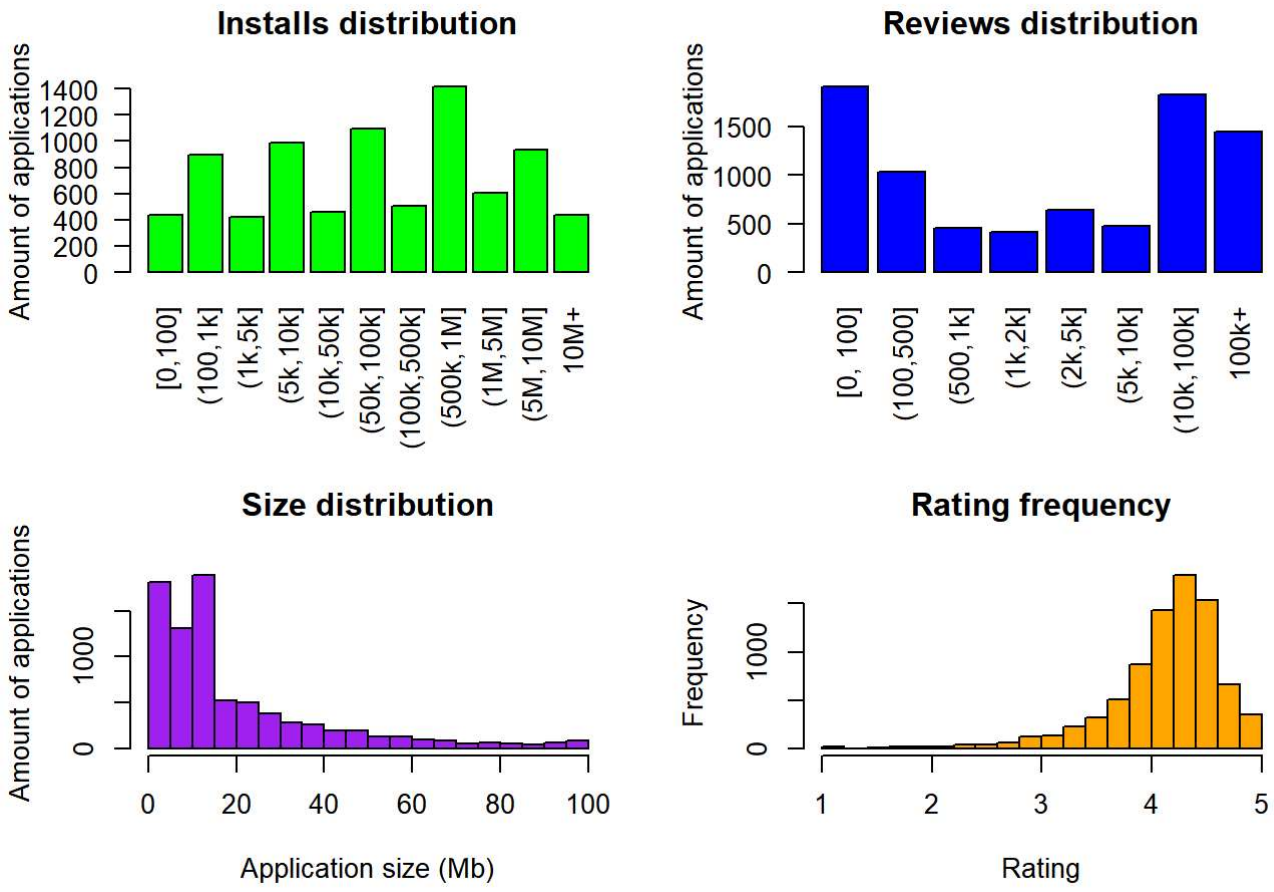
```
## The following objects are masked from 'package:dplyr':
##
##     src, summarize
```

```
## The following objects are masked from 'package:base':
##
##     format.pval, units
```

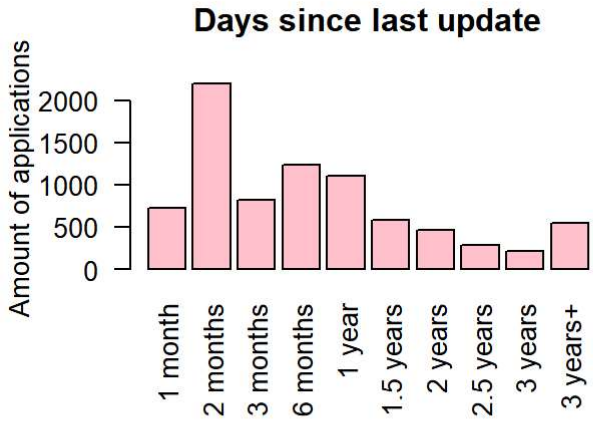
```
library(plotrix)
```

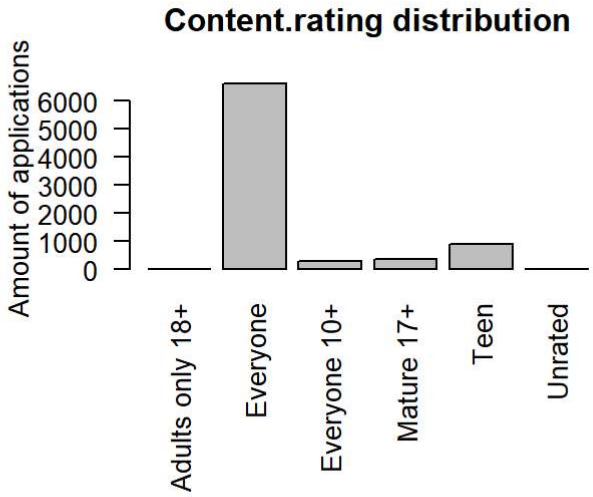
```
## Warning: package 'plotrix' was built under R version 3.5.3
```

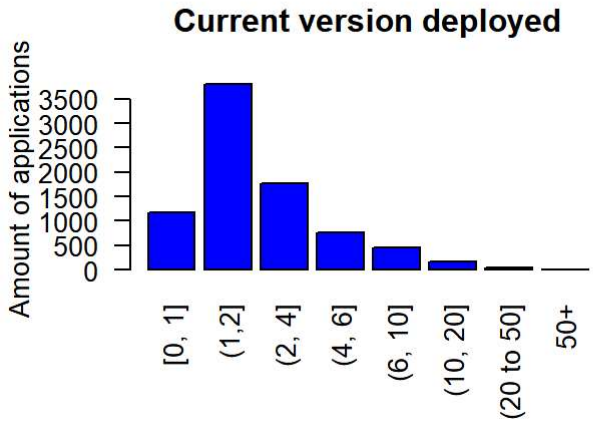
```
par(mfrow=c(2,2))

#installs
cut.Installs = cut(dataset$Installs, c(-Inf,100,1e+3,5e+3, 1e+4, 5e+4,1e+5,5e+5, 1e+6, 5e+6, 1e+
7,Inf))
levels(cut.Installs) = c('[0,100]', '(100,1k]', '(1k,5k]', '(5k,10k]', '(10k,50k]', '(50k,100k]'
, '(100k,500k]','(500k,1M]','(1M,5M]','(5M,10M]','10M+')
plot(cut.Installs, main='Installs distribution', ylab='Amount of applications', col='green', las
=2)

#Reviews
cut.Reviews = cut(dataset$Reviews, c(-Inf,100,500,1000,2000,5000,1e+4,1e+5,Inf))
levels(cut.Reviews) = c('[0, 100]', '(100,500]', '(500,1k]','(1k,2k]','(2k,5k]','(5k,10k]', '(10
k,100k]','100k+')
plot(cut.Reviews, main='Reviews distribution', col = 'blue', las=2, ylab='Amount of application
s')

#size
hist(dataset$Size, main='Size distribution', ylab = 'Amount of applications',xlab = 'Application
 size (Mb)', col='purple')

#Ratings
hist(dataset$Rating,xlab='Rating',ylab='Frequency',main='Rating frequency',col='orange')
```



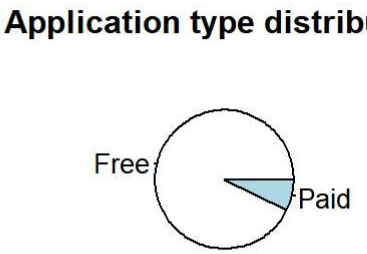
```
par(mfrow=c(2,2))
#Last updated
cut.days = cut(dataset$Last.Updated, c(-Inf,30,60,90,180, 365, 545, 730,910, 1095,Inf))
levels(cut.days) = c('1 month', '2 months', '3 months', '6 months', '1 year', '1.5 years', '2 ye
ars','2.5 years','3 years','3 years+')
plot(cut.days, main='Days since last update', ylab='Amount of applications', col='pink', las=2)

#content.rating
# labels=names(table(dataset$Content.Rating))
# pie3D(table(dataset$Content.Rating), labels=labels,main='Content rating distribution')
plot(dataset$Content.Rating, main='Content.rating distribution', ylab='Amount of applications',
 las=2)

#Current.ver
cut.ver = cut(dataset$Current.Ver, c(-Inf,1,2,4,6,10,20,50,Inf))
levels(cut.ver) = c('[0, 1]', '(1,2]', '(2, 4]','(4, 6]', '(6, 10]','(10, 20]','(20 to 50]','50
+')
plot(cut.ver, main='Current version deployed', ylab='Amount of applications', col='blue', las=2)
```

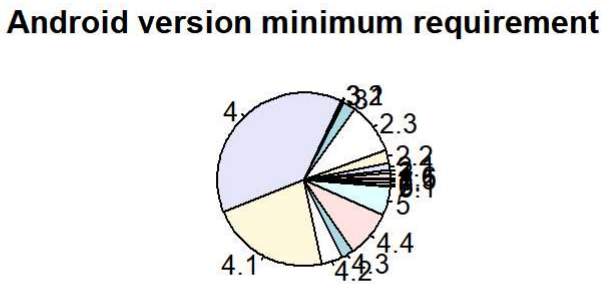




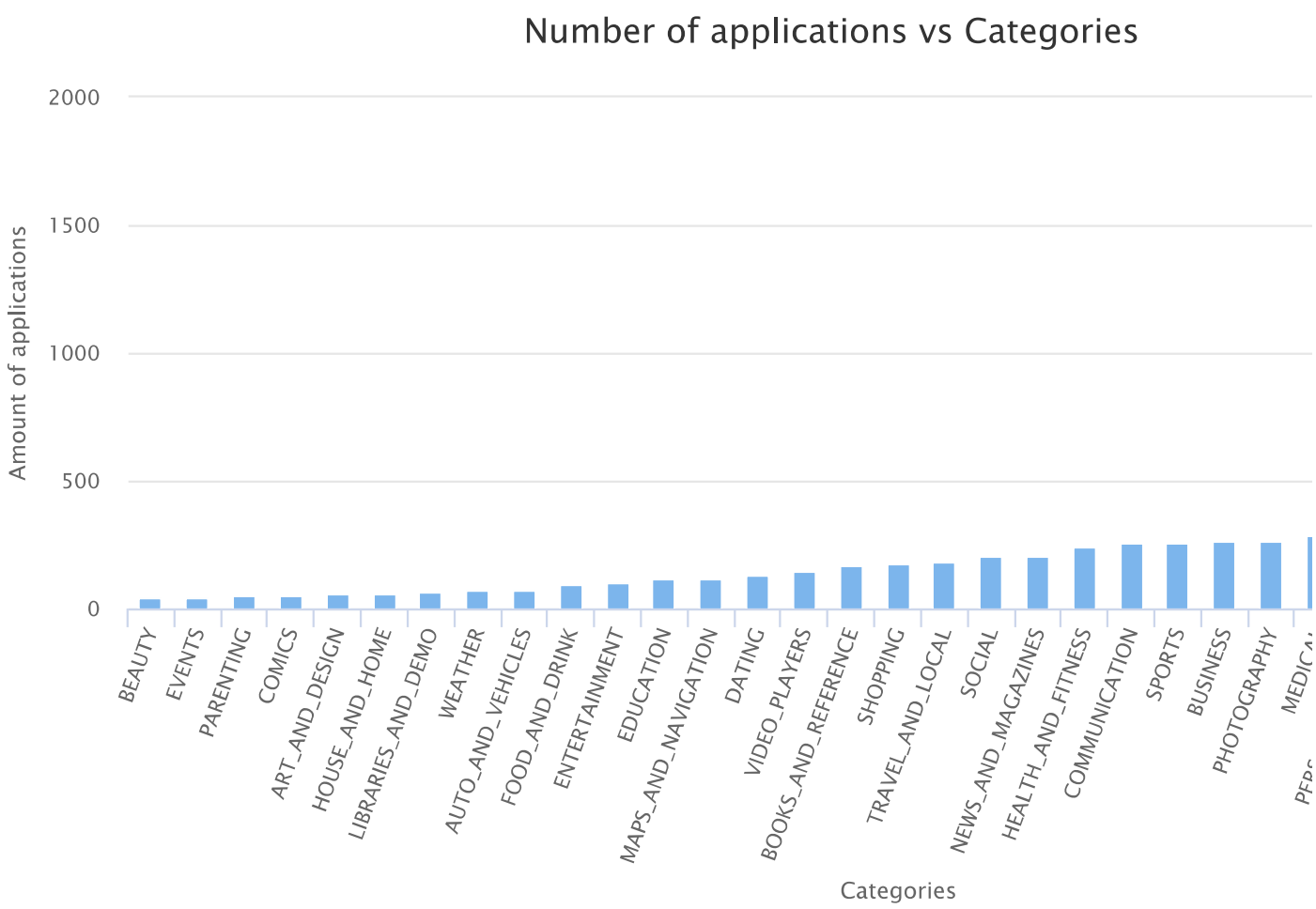


```
par(mfrow=c(2,2))
#type
labels=names(table(dataset$Type))
pie(table(dataset$Type), labels=labels, main='Application type distribution')
# plot(dataset$Type, main='Application type distribution', xlab = 'Type', ylab='Amount of applic
ations')

#Android.ver
labels = names(table(dataset$Android.Ver))
pie(table(dataset$Android.Ver), labels=labels, main='Android version minimum requirement', radiu
s=1,cex=1)
```

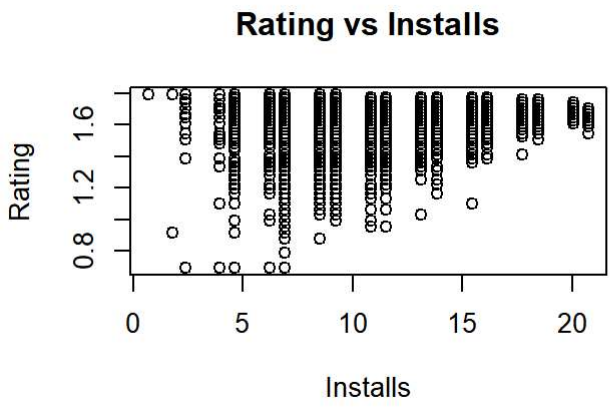


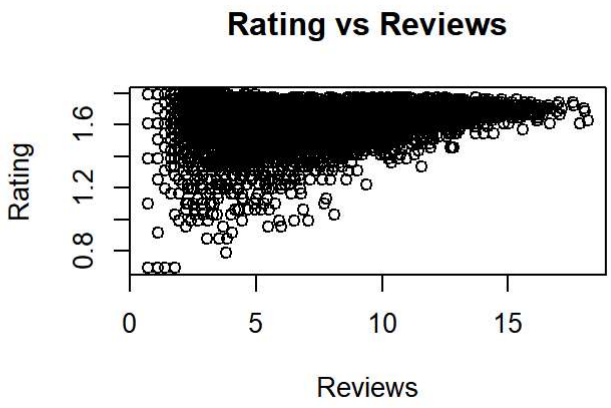



```
#Categories
dataset%>%
  count(Category)%>%
  arrange(n)%>%
  hchart(type = "column", hcaes(x = Category, y = n), main='Category Summary', xlab = 'Category'
, ylab='Amount') %>%
  hc_xAxis(title=list(text='Categories'),labels = list(rotation=-70,step=1)) %>%
  hc_yAxis(title=list(text='Amount of applications')) %>%
  hc_title(text='Number of applications vs Categories')
```




```
#Log transform some features
dataset['Rating'] = log(dataset['Rating']+1)
dataset['Installs'] = log(dataset['Installs']+1)
dataset['Reviews'] = log(dataset['Reviews'] + 1)
dataset['Size'] = log(dataset['Size'] + 1)

par(mfrow=c(2,2))
#installs vs rating
plot(dataset$Installs,dataset$Rating, main = 'Rating vs Installs', ylab='Rating', xlab = 'Instal
ls')

#reviews vs rating
plot(dataset$Reviews,dataset$Rating, main = 'Rating vs Reviews', ylab='Rating', xlab = 'Reviews'
)
```





smoothing method

```
library(FNN)
```

```
## Warning: package 'FNN' was built under R version 3.5.3
```

```
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
library(Metrics)
```

```
## Warning: package 'Metrics' was built under R version 3.5.3
```

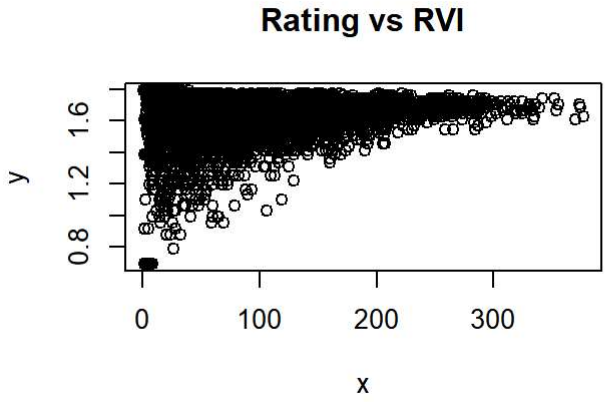
```
library(rgl)
```

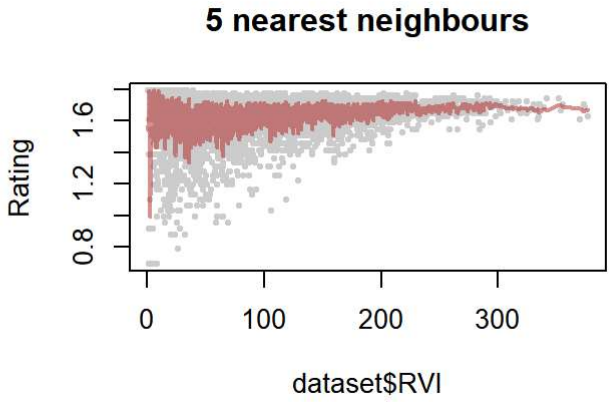
```
## Warning: package 'rgl' was built under R version 3.5.3
```

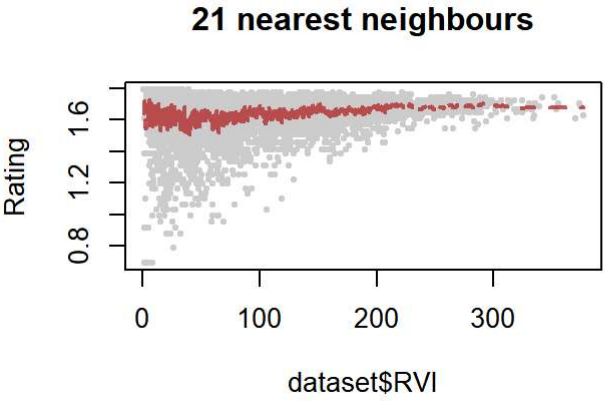
```
##
## Attaching package: 'rgl'
```

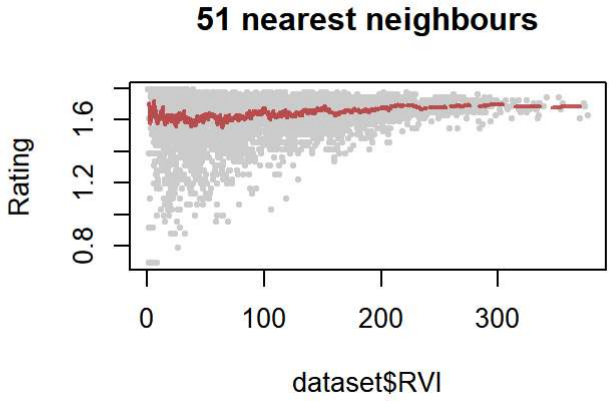
```
## The following object is masked from 'package:plotrix':
##
##     mtext3d
```

```
#this is the plot for the rating feature against the response variable
par(mfrow=c(2,2))
x=dataset$RVI
y=dataset$Rating
plot(x,y, main='Rating vs RVI')

# Let's try a few values for k
#
knn.fit5 <- knn.reg(x, y=y, k=5)
knn.fit21 <- knn.reg(x, y=y, k=21)
knn.fit51 <- knn.reg(x, y=y, k=51)
Xorder <- order(x)
plot(x,y,
     col="grey80", pch=19, cex=0.5,
     main = "5 nearest neighbours",
     xlab='dataset$RVI',
     ylab='Rating')
lines(x[Xorder], knn.fit5$pred[Xorder],
      col=adjustcolor("firebrick", 0.5),
      lwd=2)
plot(x,y,
     col="grey80", pch=19, cex=0.5,
     main = "21 nearest neighbours",
     xlab='dataset$RVI',
     ylab='Rating')
lines(x[Xorder], knn.fit21$pred[Xorder],
      col=adjustcolor("firebrick", 0.75),
      lwd=2, lty=2)
plot(x,y,
     col="grey80", pch=19, cex=0.5,
     main = "51 nearest neighbours",
     xlab='dataset$RVI',
     ylab='Rating')
lines(x[Xorder], knn.fit51$pred[Xorder],
      col=adjustcolor("firebrick", 0.75),
      lwd=2, lty=5)
```


Rating vs RVI
y
0.8
1.2
1.6
0
100
200
300
x

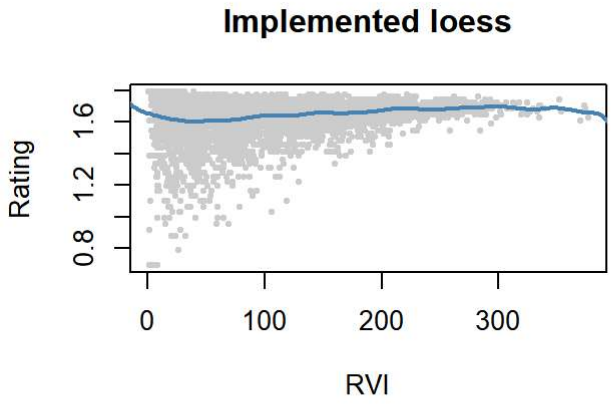

5 nearest neighbours
Rating
0.8
1.2
1.6
0
100
200
300
dataset$RVI

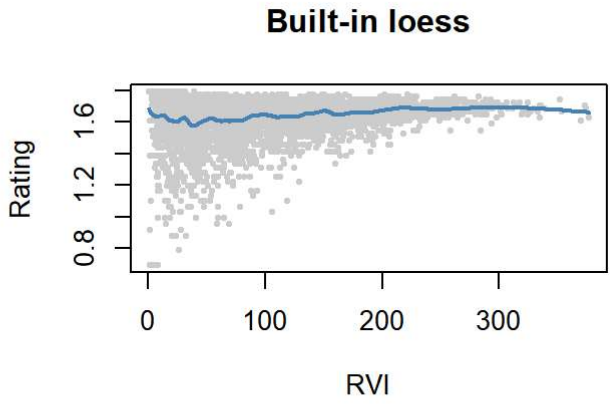

21 nearest neighbours
Rating
0.8
1.2
1.6
0
100
200
300
dataset$RVI


51 nearest neighbours
Rating
0.8
1.2
1.6
0
100
200
300
dataset$RVI

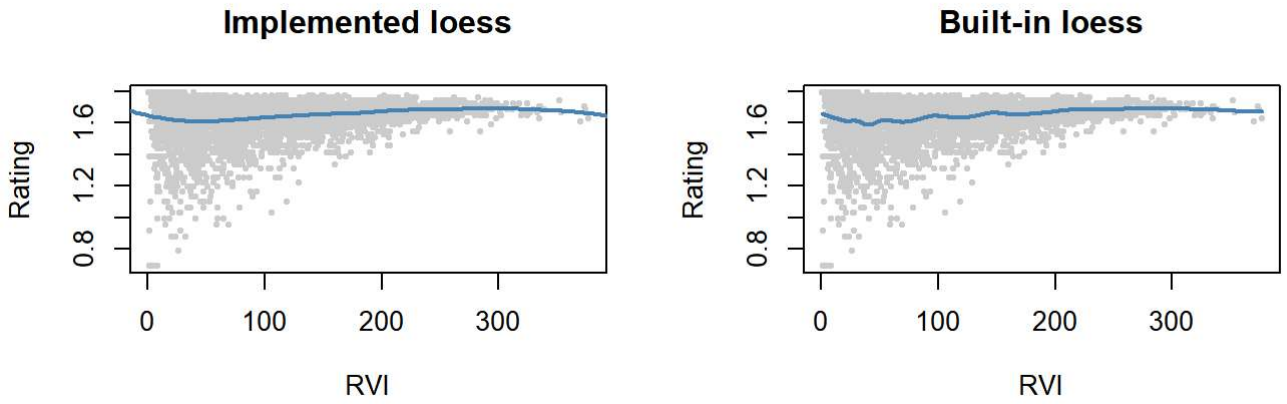
```
par(mfrow=c(2,2))
LoWeSS <- function(x, y,
                   xloc=NULL,
                   span=0.1,
                   weightFn=GaussWeight,
                   nlocs=200) {
  data <- data.frame(x=x, y=y)
  xrange <- extendrange(x)
  bandwidth <- (diff(xrange)*span)/4
  if (is.null(xloc)) {
    xloc <- seq(xrange[1], xrange[2],
                length.out=nlocs)
  }
  estimates <- vector("numeric", length=length(xloc))
  for (i in 1:length(xloc)) {
    weights <- weightFn(xloc[i], x, bandwidth)
    fit <- lm(y~x, data=data, weights=weights)
    pred <- predict(fit, newdata=data.frame(x=xloc[i]))
    estimates[i] <- pred
  }#return the estimates
  list(x = xloc, y = estimates)
}

GaussWeight <- function(xloc, x, h=1) {
  # Normal density
  dnorm(x, mean=xloc, sd=h)
}

N = nrow(dataset)
N_train = round(4*N/5)
N_test = N - N_train

id.train = sample(1:N, N_train, replace = FALSE)
id.test = setdiff(1:N,id.train)
test = dataset[id.test,]
train = dataset[id.train,]
actual = test$Rating

x = train$RVI
y = train$Rating

par(mfrow=c(2,2))
smooth <- LoWeSS(x,y) #default span=0.1
plot(x,y,
     col="grey80", pch=19, cex=0.5,
     main = "Implemented loess",
     xlab='RVI',
     ylab='Rating')
lines(smooth$x, smooth$y, col="steelblue", lwd=2)


#using the build in function loess in R that expresses more the concept of local weighted sums o
f squares
```

```
fit <- loess(y~x, data=train, span=0.1)
plot(x,y,
     col="grey80", pch=19, cex=0.5,
     main = "Built-in loess",
     xlab='RVI',
     ylab='Rating')
##Get the values predicted by the loess
pred <- predict(fit)
##Get the order of the x
Xorder <- order(x)
#
lines(x[Xorder],pred[Xorder], lwd=2, col="steelblue")
```





```
#tune the span paramter
par(mfrow=c(2,2))

#implemented loess
smooth <- LoWeSS(x,y, span=0.2)
plot(x,y,
     col="grey80", pch=19, cex=0.5,
     main = "Implemented loess",
     xlab='RVI',
     ylab='Rating')
lines(smooth$x, smooth$y, col="steelblue", lwd=2)

#build-in loess
fit <- loess(y~x, data=data.frame(train), span=0.2)
plot(x,y,
     col="grey80", pch=19, cex=0.5,
     main = "Built-in loess",
     xlab='RVI',
     ylab='Rating')

##Get the values predicted by the loess
pred <- predict(fit)
##
lines(x[Xorder],pred[Xorder], lwd=2, col="steelblue")
```



```
#5-fold cross validation
tic('loess computation time')
sum_mse = 0
sum_rmse=0
sum_rmsle=0
for (c in 0:4){
  id.test = (c*N_test):(c*N_test+N_test)
  id.train = setdiff(1:N, id.test)
  test = dataset[id.test,]
  train = dataset[id.train,]
  actual = test$Rating
  x = train$RVI
  y = train$Rating
  fit <- loess(y~x, data=train, span=0.1, control=loess.control(surface="direct"))
  pred = predict(fit,test$RVI)
  s.mse = mean((actual-pred)^2)
  s.rmse = rmse(actual,pred)
  s.rmsle = rmsle(actual,pred)
  sum_mse = sum_mse + s.mse
  sum_rmse = sum_rmse + s.rmse
  sum_rmsle = sum_rmsle + s.rmsle
}

loess.mse = sum_mse/5
loess.rmse = sum_rmse/5
loess.rmsle = sum_rmsle/5
toc()
```

```
## loess computation time: 2.3 sec elapsed
```

```
#3d scatter

#make sure to run the scatter3d.R before running blocks related to 3D scatter plots
id.train = sample(1:N, N_train, replace = FALSE)
id.test = setdiff(1:N,id.train)
test = dataset[id.test,]
train = dataset[id.train,]
actual = test$Rating


formula = dataset$Rating~dataset$Installs + dataset$Reviews
#more than just one variate
scatter3d(formula,data=train)
snapshot3d("smooth1.png")
```

```
#adding an interaction term, lets see the differences
scatter3d(formula,
          data=train,
          fit = c("linear", "interaction"))
snapshot3d("smooth2.png")
```

```
#quad with a non-additive quad that includes a cross product term xz
scatter3d(formula,
          data=train,
          fit=c("quadratic", "quadint")
)
snapshot3d("smooth3.png")
```

```
#compare loess and thin plate
scatter3d(formula,
          data=train,
          fit=c("loess","smooth"),
          df.loess=30, df.smooth=30)
snapshot3d("smooth4.png")
```

```
scatter3d(formula,
          data=train,
          fit=c("additive", 'smooth'), df.additive=15, df.smooth=30)
snapshot3d("smooth5.png")
```

```
library(mgcv)
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.5.3
```

```
##
## Attaching package: 'caret'
```

```
## The following objects are masked from 'package:Metrics':
##
##     precision, recall
```

```
## The following object is masked from 'package:survival':
##
##     cluster
```

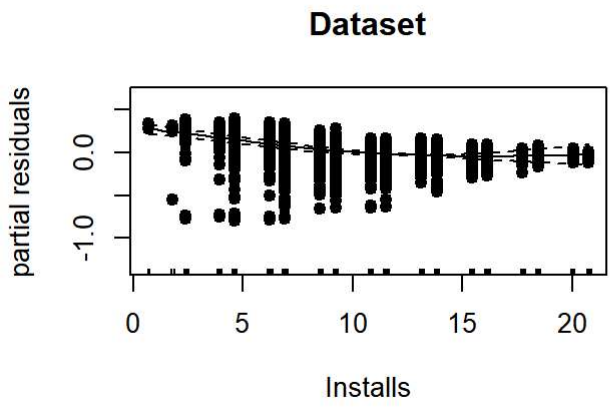
```
formula = Rating~
  Reviews + Size + Installs +
  Price + Last.Updated +
  Current.Ver + Android.Ver + RVI
tic('gam computation time')
gam.cv = train(formula,
            data = train,
            method = "gam",
            trControl = trainControl(method = "cv", number=5),
            tuneGrid = data.frame(method = "GCV.Cp", select = FALSE))
toc()
```

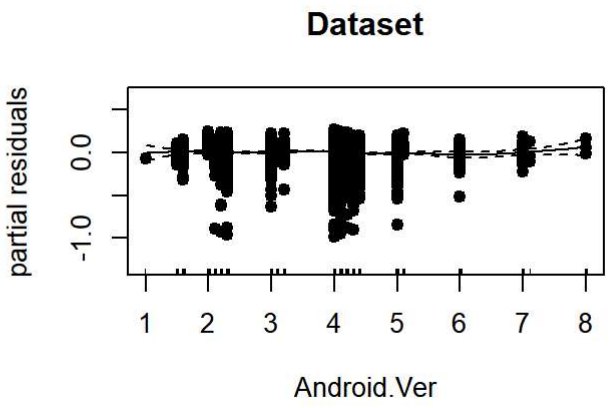
```
## gam computation time: 7.04 sec elapsed
```

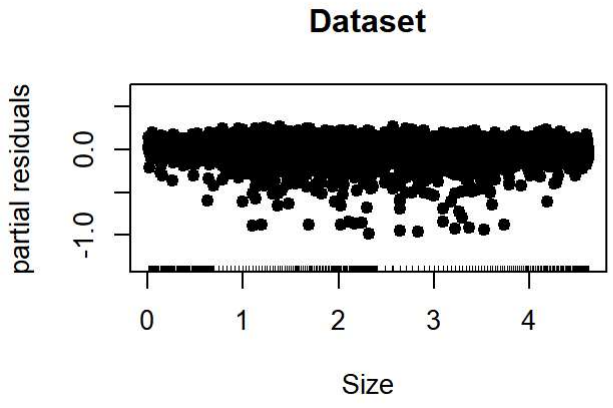
```
#final model using gam
gam.model = gam.cv$finalModel
summary(gam.model)
```

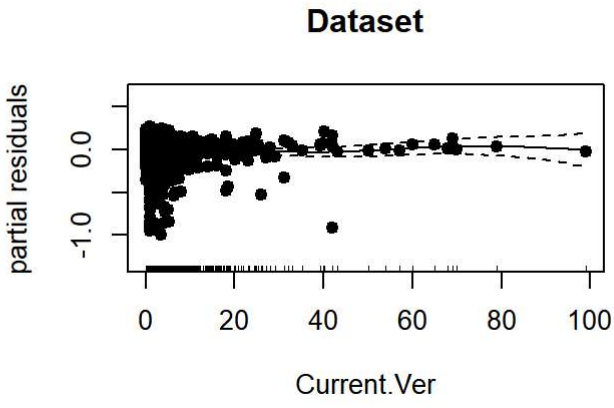
```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## .outcome ~ s(Installs) + s(Android.Ver) + s(Size) + s(Current.Ver) +
##     s(Last.Updated) + s(Reviews) + s(RVI)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.637446   0.001361    1203   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                   edf Ref.df      F  p-value
## s(Installs)     5.202  6.381 41.411  < 2e-16 ***
## s(Android.Ver)  7.205  8.021  3.065  0.00182 **
## s(Size)         3.140  3.930  4.003  0.00343 **
## s(Current.Ver)  5.377  6.445  2.739  0.01133 *
## s(Last.Updated) 2.880  3.618 18.216 3.36e-13 ***
## s(Reviews)      1.903  2.502 47.627  < 2e-16 ***
## s(RVI)          1.000  1.000  9.895  0.00166 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.125   Deviance explained = 12.9%
## GCV = 0.012196  Scale est. = 0.012145  n = 6557
```

```
par(mfrow=c(2,2))
plot(gam.model, main="Dataset",
     ylab="partial residuals", jit=TRUE,
     residuals =TRUE, pch=19)
```

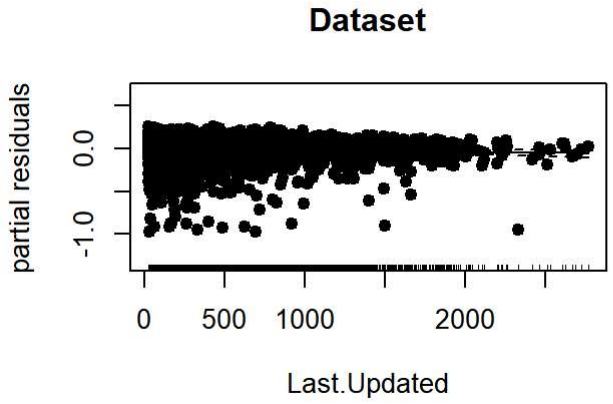


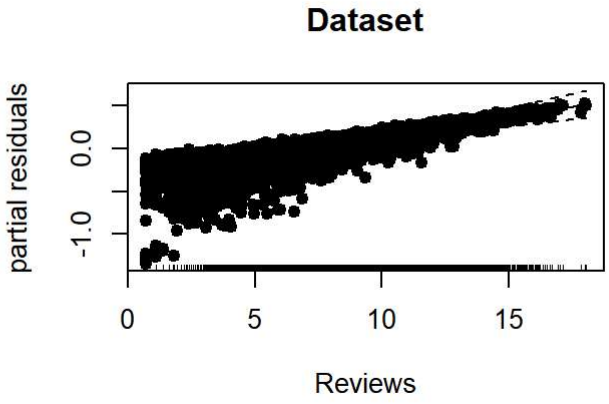


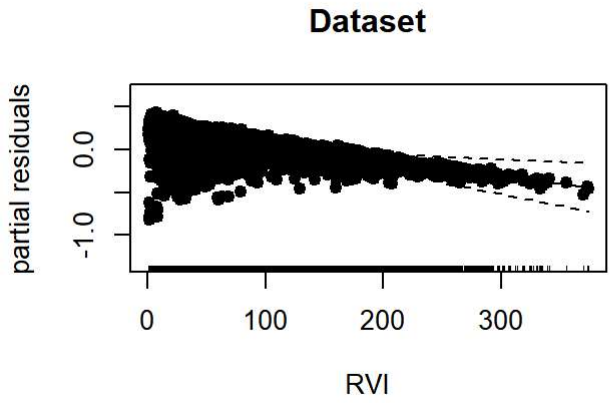





```
par(mfrow=c(1,1))
```








```
pred = predict(gam.model, test)
gam.mse = mean((actual-pred)^2)
gam.rmse = rmse(actual,pred)
gam.rmsle = rmsle(actual,pred)
gam.mse
```

```
## [1] 0.01343455
```

```
gam.rmse
```

```
## [1] 0.1159075
```

```
gam.rmsle
```

```
## [1] 0.04747901
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.5.3
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

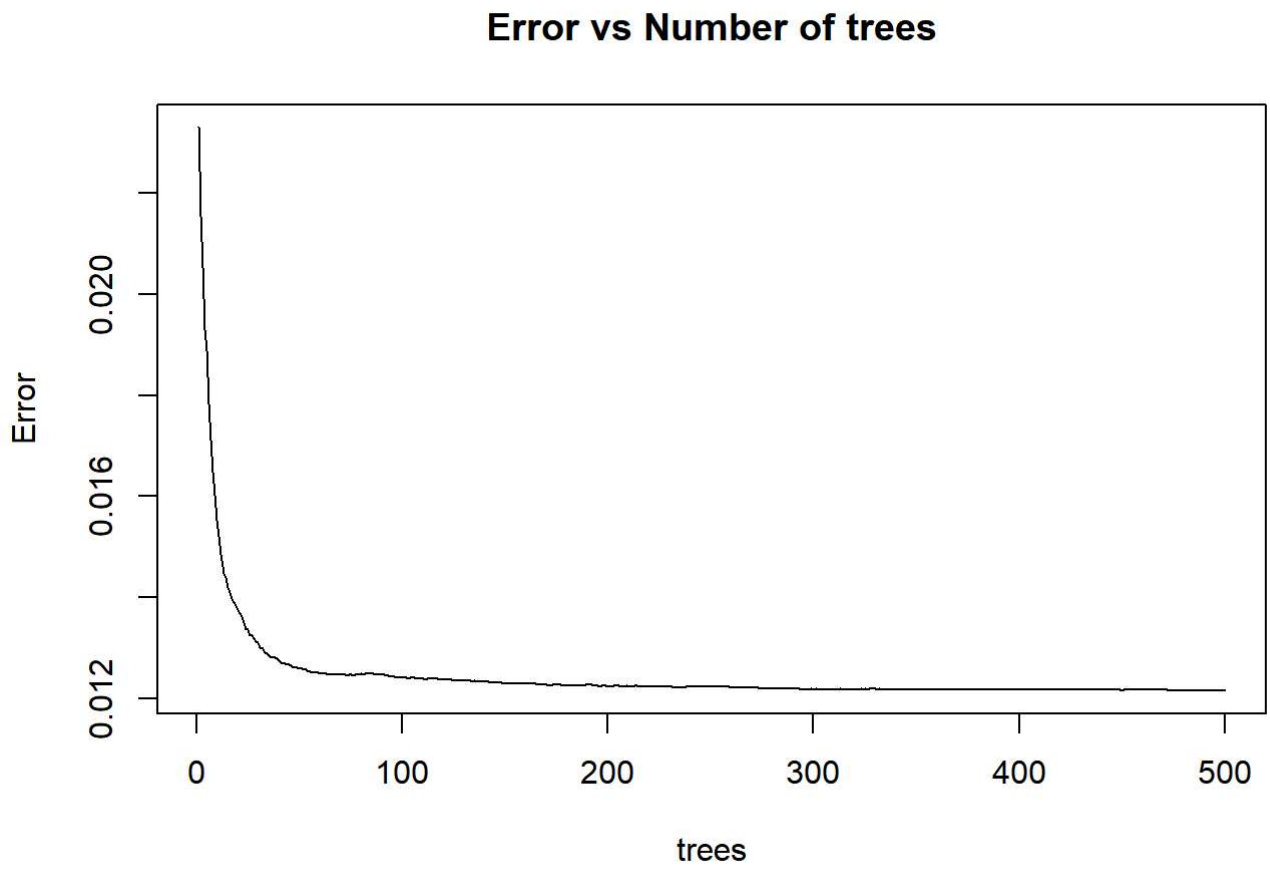
```
library(Metrics)
library(caTools)
library(caret)
library(ranger)
```

```
## Warning: package 'ranger' was built under R version 3.5.3
```

```
##
## Attaching package: 'ranger'
```

```
## The following object is masked from 'package:randomForest':
##
##     importance
```

```
set.seed(42)

p = dim(dataset)[2]-3

formula = Rating~
  Category + Reviews + Size + Installs + Type +
  Price + Content.Rating + Last.Updated +
  Current.Ver + Android.Ver + RVI

#get train and test set
N = nrow(dataset)
N_train = round(4*N/5)
N_test = N - N_train

id.train = sample(1:N, N_train, replace = FALSE)
id.test = setdiff(1:N,id.train)
train = dataset[id.train,]
test = dataset[id.test,]
actual = test$Rating

#Let us see the optimal number of trees to use
rf = randomForest(formula, data = dataset, mtry = p)
plot(rf, main='Error vs Number of trees')
```



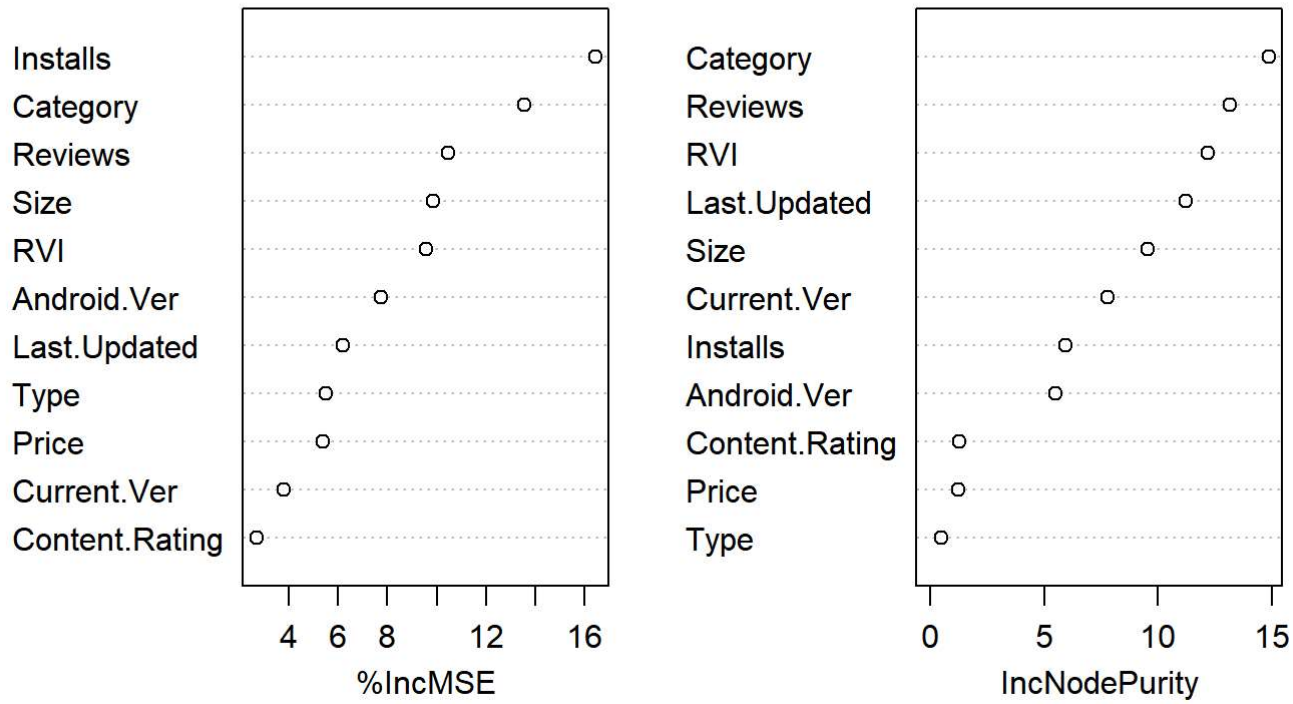
```
rf = randomForest(formula, data = dataset, subset=id.train, ntree = 100, mtry = 4, importance =
T)
pred = predict(rf, test)
randomForest::importance(rf,type=2)
```

```
##                IncNodePurity
## Category          14.8372886
## Reviews           13.1337939
## Size               9.5399410
## Installs           5.9376540
## Type               0.4965294
## Price              1.2375910
## Content.Rating     1.2840395
## Last.Updated      11.1915845
## Current.Ver        7.7693320
## Android.Ver        5.5051394
## RVI               12.1732754
```

```
randomForest::importance(rf,type=1)
```

```
##                  %IncMSE
## Category       13.569302
## Reviews        10.476561
## Size            9.858821
## Installs       16.418454
## Type            5.519791
## Price           5.392508
## Content.Rating  2.687412
## Last.Updated    6.216858
## Current.Ver     3.804108
## Android.Ver     7.736484
## RVI             9.555204
```

```
varImpPlot(rf, main='Feature importance')
```




```
#5-fold CV using caret

fit_control <- trainControl(## 5-fold CV
                            method = "cv",
                            number = 5)

tic('ranger cv computation time')
rf_fit <- train(formula,
                data = train,
                method = "ranger",
                num.trees=100,
                importance = 'permutation',
                trControl = fit_control)
toc()
```

```
## ranger cv computation time: 26.5 sec elapsed
```

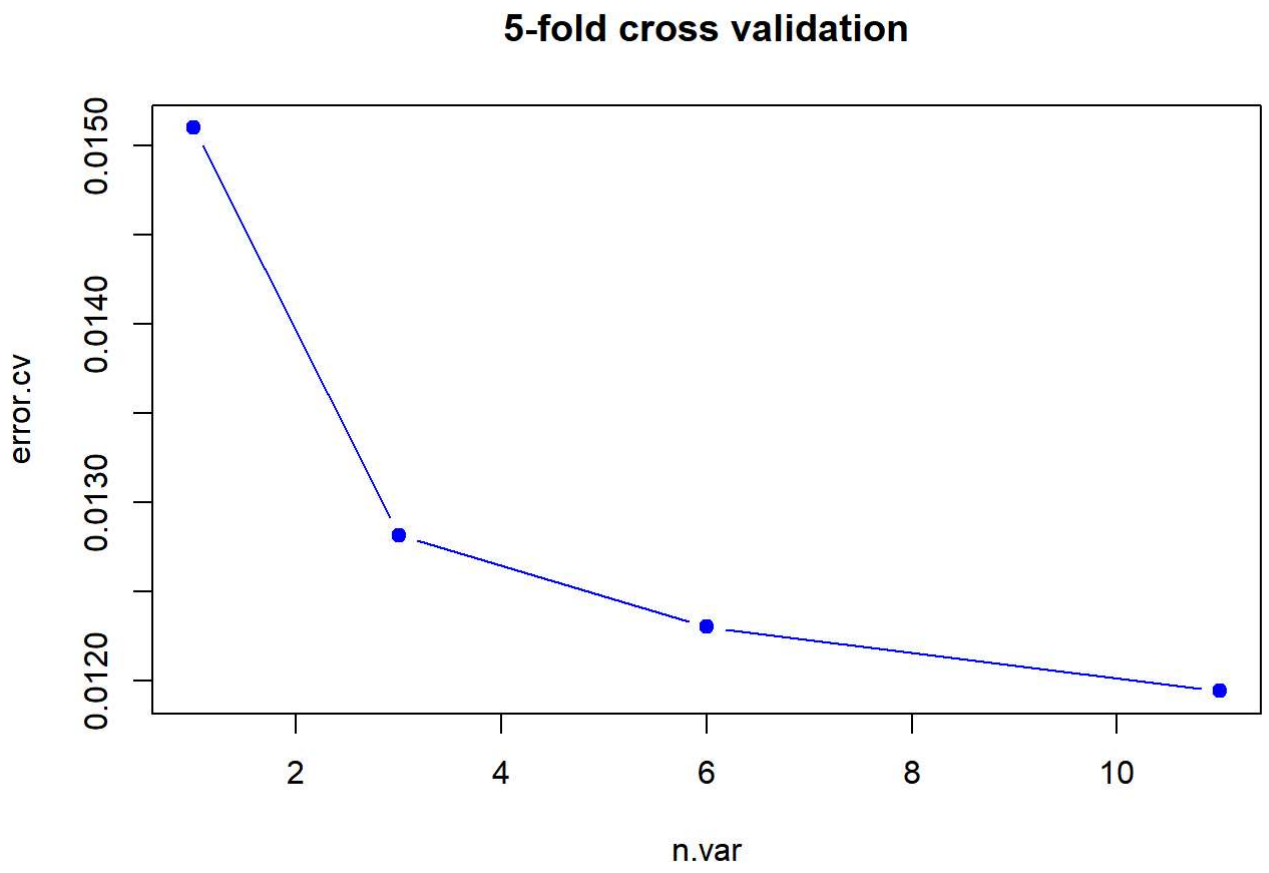
```
rf_fit
```

```
## Random Forest
##
## 6557 samples
##   11 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 5246, 5246, 5245, 5246, 5245
## Resampling results across tuning parameters:
##
##   mtry  splitrule   RMSE       Rsquared    MAE
##    2    variance    0.1126210  0.11128243  0.07350890
##    2    extratrees  0.1143194  0.09333174  0.07491469
##   24    variance    0.1109714  0.12681155  0.07101770
##   24    extratrees  0.1103584  0.13334252  0.07031007
##   46    variance    0.1113863  0.12519439  0.07122624
##   46    extratrees  0.1113328  0.12644265  0.07087978
##
## Tuning parameter 'min.node.size' was held constant at a value of 5
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were mtry = 24, splitrule =
##  extratrees and min.node.size = 5.
```

```
varImp(rf_fit)
```

```
## ranger variable importance
##
##   only 20 most important variables shown (out of 46)
##
##                             Overall
## Reviews                     100.000
## Installs                     83.592
## RVI                          49.134
## Size                         14.078
## Last.Updated                 12.340
## CategoryTOOLS                 8.193
## CategoryGAME                  7.686
## Android.Ver                   6.637
## TypePaid                      6.227
## CategoryHEALTH_AND_FITNESS    5.200
## Current.Ver                   4.905
## Content.RatingEveryone        3.402
## Content.RatingMature 17+      3.046
## CategoryFINANCE               2.835
## CategoryDATING                2.731
## Content.RatingTeen            2.553
## CategoryPHOTOGRAPHY           2.361
## CategoryMAPS_AND_NAVIGATION   2.326
## CategoryPERSONALIZATION       2.218
## CategorySOCIAL                2.129
```

```
trainy = train$Rating
trainx = train[,c('RVI','Category','Reviews','Size','Installs','Type','Price','Content.Rating',
'Last.Updated','Current.Ver','Android.Ver')]

#5-fold validation
dataset.rfcv = rfcv(trainx=trainx,trainy=trainy,cv.fold=5)
with(dataset.rfcv, plot(n.var, error.cv, pch = 19, type="b", col="blue", main='5-fold cross vali
dation'))
```



```
#5-fold cross validation

sum_mse.full = 0
sum_rmse.full=0
sum_rmsle.full=0
tic('rf.full computation time')
for (c in 0:4){
  id.test = (c*N_test):(c*N_test+N_test)
  id.train = setdiff(1:N, id.test)
  test = dataset[id.test,]
  train = dataset[id.train,]
  actual = test$Rating
  rf.full = randomForest(formula, data = dataset, subset=id.train, ntree = 100, mtry = p, import
ance = T)
  pred.full = predict(rf.full, test)

  s.mse.full = mean((actual-pred.full)^2)
  s.rmse.full = rmse(actual,pred.full)
  s.rmsle.full = rmsle(actual,pred.full)
  sum_mse.full = sum_mse.full + s.mse.full
  sum_rmse.full = sum_rmse.full + s.rmse.full
  sum_rmsle.full = sum_rmsle.full + s.rmsle.full
}

rf.full.mse = sum_mse/5
rf.full.rmse = sum_rmse/5
rf.full.rmsle = sum_rmsle/5
toc()
```

```
## rf.full computation time: 101.73 sec elapsed
```

```
tic('rf computation time')
sum_mse = 0
sum_rmse=0
sum_rmsle=0
for (c in 0:4){
  id.test = (c*N_test):(c*N_test+N_test)
  id.train = setdiff(1:N, id.test)
  test = dataset[id.test,]
  train = dataset[id.train,]
  actual = test$Rating
  rf = randomForest(formula, data = dataset, subset=id.train, ntree = 100, mtry = 4, importance
 = T)
  pred = predict(rf, test)
  s.mse = mean((actual-pred)^2)
  s.rmse = rmse(actual,pred)
  s.rmsle = rmsle(actual,pred)
  sum_mse = sum_mse + s.mse
  sum_rmse = sum_rmse + s.rmse
  sum_rmsle = sum_rmsle + s.rmsle
}

rf.mse = sum_mse/5
rf.rmse = sum_rmse/5
rf.rmsle = sum_rmsle/5
toc()
```

```
## rf computation time: 45.13 sec elapsed
```

```
rf.mse
```

```
## [1] 0.01284883
```

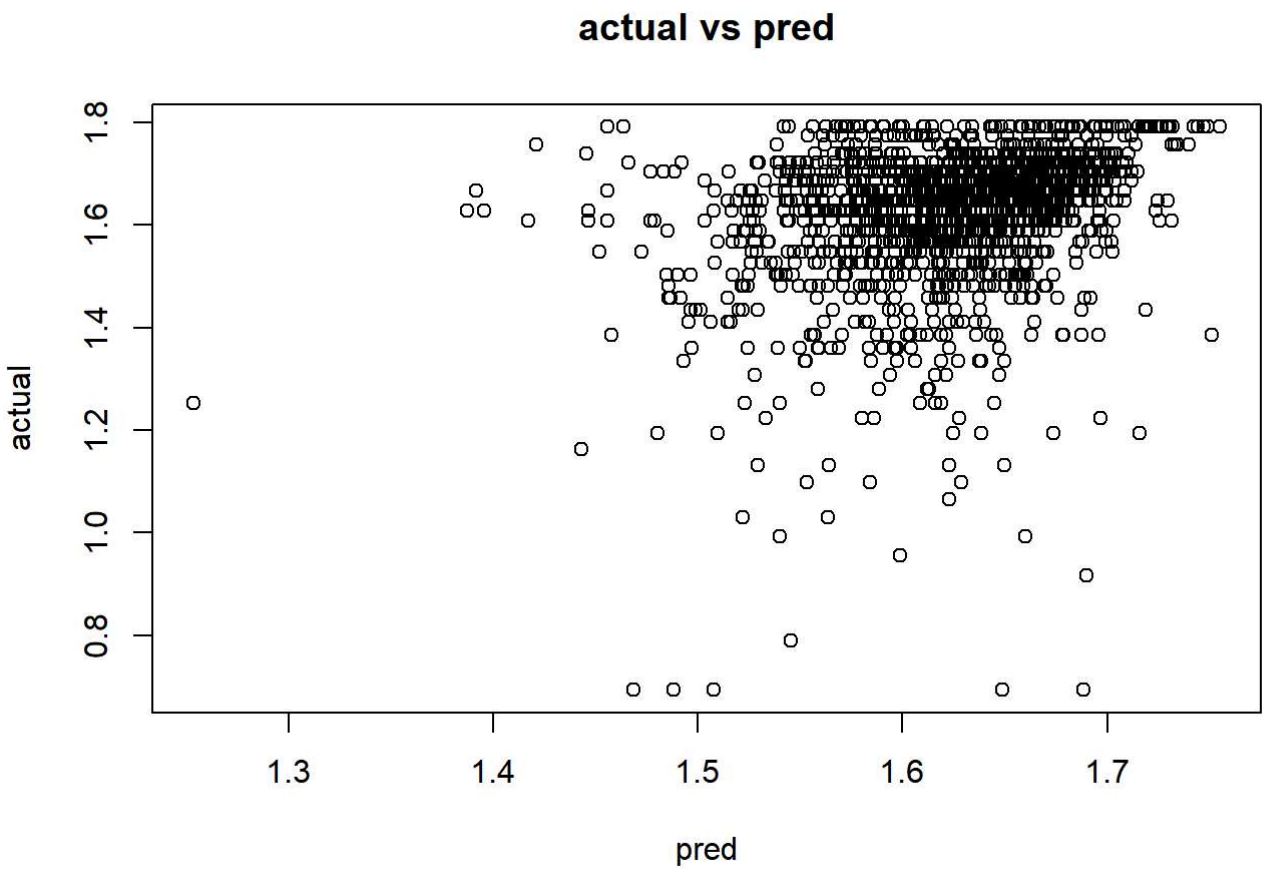
```
rf.rmse
```

```
## [1] 0.1105794
```

```
rf.rmsle
```

```
## [1] 0.0450894
```

```
plot(pred,actual, main='actual vs pred')
```




```
#Boosting method
library(rpart)
```

```
##
## Attaching package: 'rpart'
```

```
## The following object is masked from 'package:survival':
##
##     solder
```

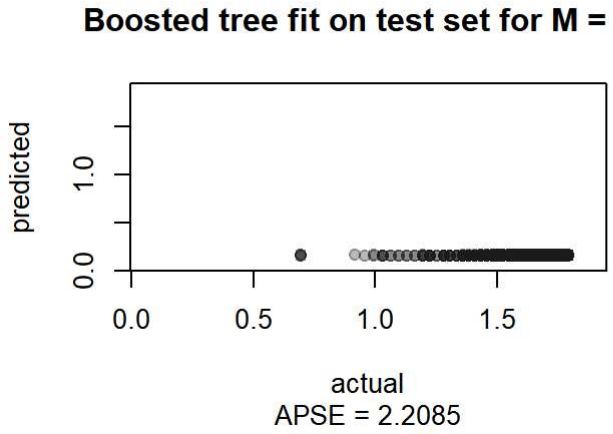
```
library(gbm)
```

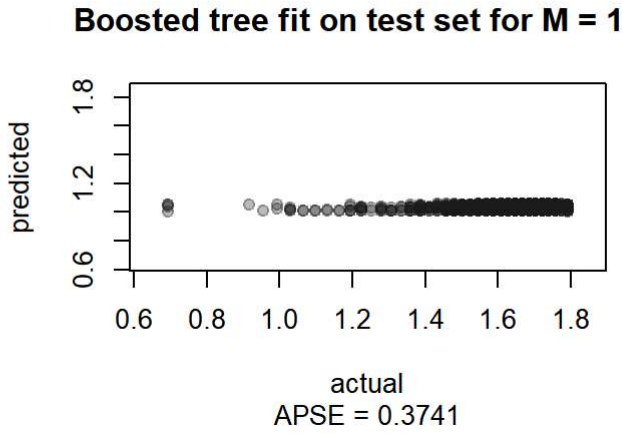
```
## Warning: package 'gbm' was built under R version 3.5.3
```

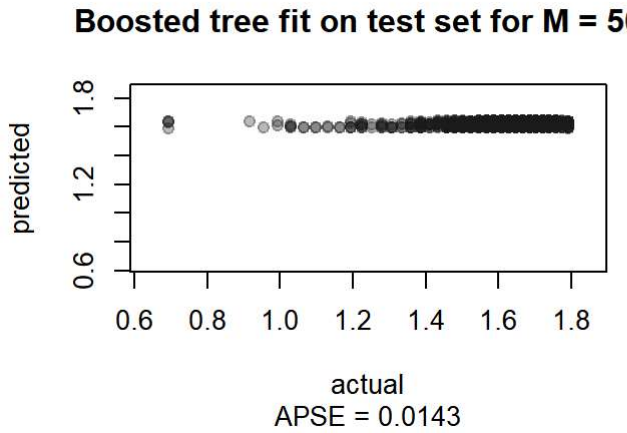
```
## Loaded gbm 2.1.5
```

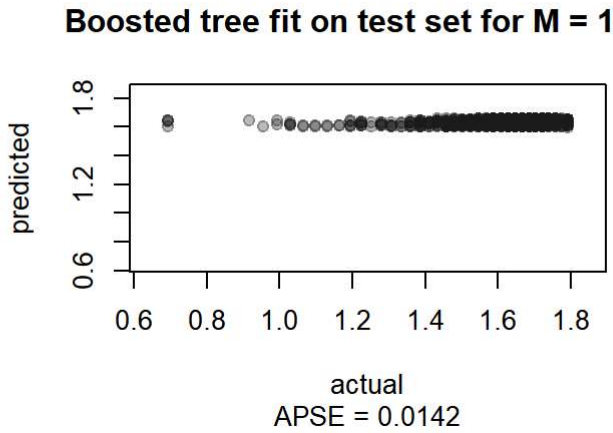
```
library(mgcv)
library(caret)
set.seed(42)

formula = 'Rating~
  Category + Reviews + Size + Installs + Type +
  Price + Content.Rating + Last.Updated +
  Current.Ver + Android.Ver + RVI'


get.response <- function(fittedTree, test.data){
  f <- formula(fittedTree)
  terms <- terms(f)
  response.id <- attr(terms, "response")
  response <- as.list(attr(terms, "variables"))[[response.id + 1]]
  with(test.data, eval(response))
}

get.newdata <- function(fittedTree, test.data){
  f <- formula(fittedTree)
  terms <- terms(f)
  as.list(test.data[,attr(terms, "term.labels")])
}

boostTree <- function(formula, data,
                      lam=0.01, M = 10,
                      control=rpart.control(), ...) {
  # Break the formula into pieces
  formula.sides <- strsplit(formula, "~")[[1]]
  response.string <- formula.sides[1]
  rhs.formula <- formula.sides[2]
  # Construct the boost formula
  bformula <- paste("resid", rhs.formula, sep=" ~ ")
  # Initialize the resid and explanatory variates
  resid <- get.response(formula, data)
  xvars <- get.newdata(formula, data)
  # Calculate the boostings
  Trees <- Map(
    function(i) {
      # update data frame with current resid
      rdata <- data.frame(resid=resid, xvars)
      # Fit the tree
      tree <- rpart(bformula, data = rdata, control=control, ...)
      # Update the residuals
      # (Note the <<- assignment to escape this closure)
      resid <<- resid - lam * predict(tree)
      # Return the tree
      tree }
    , 1:M)
  # Return the boosted function
  function(newdata){
    if (missing(newdata)) {
      predictions <- Map(function(tree) {
```

```
        # Boost piece
        lam * predict(tree)
      }, Trees)
    } else {
      predictions <- Map(function(tree){
        # New data needs to be a list
        if (is.data.frame(newdata)) {
          newdata.tree <- get.newdata(tree, newdata)
        } else {
          newdata.tree <- newdata
        }
        # Boost piece
        lam * predict(tree, newdata=newdata.tree)
      }, Trees)
    }#
    #Gather the results together
    Reduce(`+`, predictions)
  }
}

N = nrow(dataset)
N_train = round(0.8*N)
N_test = N-N_train
id.train <- sample(1:N, N_train, replace=FALSE)
id.test <- setdiff(1:N, id.train)
# Split the data into training and test sets.
dataset.train <- dataset[id.train,]
dataset.test <- dataset[id.test,]
actual.test = dataset.test$Rating

par(mfrow=c(2,2))
for (M in c(10, 100, 500, 1000)) {
  # Get the boosted tree
  predict.boostedTree <- boostTree(formula, data=dataset.train, M=M, lam = 0.01)
  # Use it to predict
  pred <- predict.boostedTree(dataset.test)
  # Now see how well it does
  lims <- extendrange(c(actual.test, pred))
  plot(get.response(formula, dataset.test), pred, pch=19,
       col = adjustcolor("grey10", 0.3),
       xlim=lims, ylim=lims, xlab="actual", ylab="predicted",
       main = paste("Boosted tree fit on test set for M =", M),
       sub = paste("APSE =", round(mean((actual.test - pred)^2), 4) )
  )
}
```



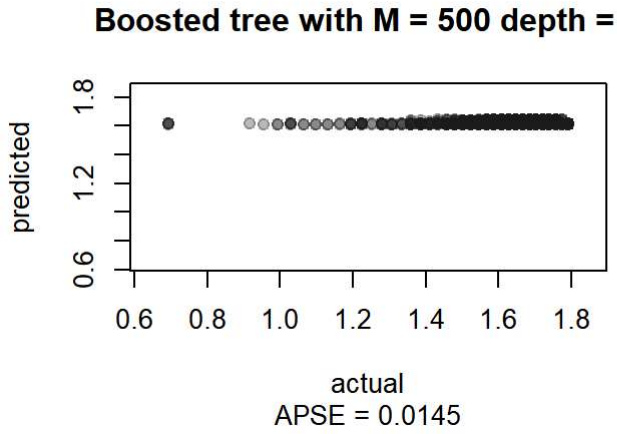


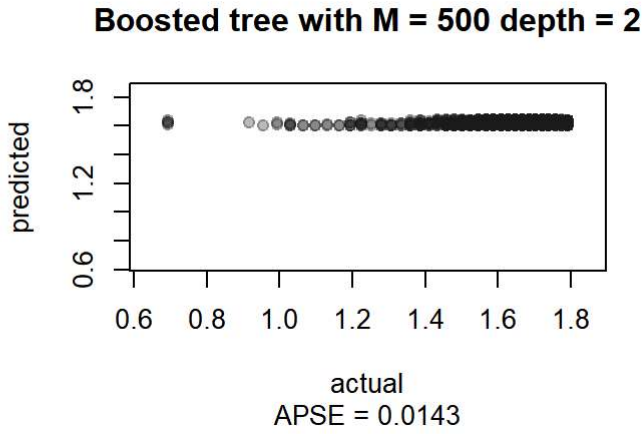





```
#we see that M = 500 and M = 1000 have similar APSE, hence stay with M = 500
```

```
par(mfrow=c(2,2))
# Now do it for varying depth
for (max.depth in c(1, 2)) {
  # Get the boosted tree
  predict.boostedTree <- boostTree(formula, data = dataset.train,
                                   M = 500, lam = 0.01,
                                   control = rpart.control(maxdepth = max.depth))
  # Use it to predict
  pred <- predict.boostedTree(dataset.test)
  # Now see how well it does
  lims <- extendrange(c(actual.test, pred))
  plot(get.response(formula, dataset.test), pred, pch=19,
       col = adjustcolor("grey10", 0.3),
       xlim=lims, ylim=lims, xlab="actual", ylab="predicted",
       main = paste("Boosted tree with M = 500",
                    "depth =", max.depth),
       sub = paste("APSE =", round(mean((actual.test - pred)^2), 4) )
  )
}
```






```
#try gradient boosting
par(mfrow=c(3,1), mar=c(5,8,4,1)+.1)
dataset.boost <- gbm(as.formula(formula), data=dataset.train, shrinkage = 1, n.trees = 500)
```

```
## Distribution not specified, assuming gaussian ...
```

```
#relative influence
rel.inf <- relative.influence(dataset.boost)
```

```
## n.trees not given. Using 500 trees.
```

```
rel.inf
```

```
##       Category        Reviews           Size       Installs           Type
##     56.4123475     34.2458380      1.8303947      4.0380132      0.0000000
##          Price Content.Rating   Last.Updated    Current.Ver    Android.Ver
##      0.3189781      0.0000000      2.6883774      0.7357100      0.8288414
##            RVI
##     43.5375172
```

```
#Scaled relative influence
rel.inf <- relative.influence(dataset.boost, scale. = TRUE, sort. = TRUE)
```

```
## n.trees not given. Using 500 trees.
```

```
rel.inf
```

```
##       Category            RVI        Reviews       Installs   Last.Updated
##    1.000000000    0.771772832    0.607062807    0.071580307    0.047655832
##           Size    Android.Ver    Current.Ver          Price Content.Rating
##    0.032446703    0.014692553    0.013041649    0.005654402    0.000000000
##           Type
##    0.000000000
```

```
rel.inf <- rel.inf/sum(rel.inf)
round(100 * rel.inf, 2)
```

```
##       Category            RVI        Reviews       Installs   Last.Updated
##          39.00          30.10          23.68           2.79           1.86
##           Size    Android.Ver    Current.Ver          Price Content.Rating
##           1.27           0.57           0.51           0.22           0.00
##           Type
##           0.00
```

```
summary(dataset.boost, main = "gbm #1, shrinkage = 1", las=1)
```

	var <fctr>	**rel.inf** <dbl>
Category	Category	39.0029735
RVI	RVI	30.1014353
Reviews	Reviews	23.6772545
Installs	Installs	2.7918448
Last.Updated	Last.Updated	1.8587191
Size	Size	1.2655179
Android.Ver	Android.Ver	0.5730533
Current.Ver	Current.Ver	0.5086631
Price	Price	0.2205385
Type	Type	0.0000000

1-10 of 11 rows Previous **1** 2 Next

```
dataset.boost <- gbm(as.formula(formula), data=dataset.train, shrinkage = 1, n.trees = 500)
```

```
## Distribution not specified, assuming gaussian ...
```

```
summary(dataset.boost, main = "gbm #2, shrinkage = 1", las=1)
```

	var <fctr>	rel.inf <dbl>
Category	Category	43.1544233
Reviews	Reviews	26.9836193
RVI	RVI	21.8043477
Installs	Installs	3.1345975
Current.Ver	Current.Ver	1.6294753
Last.Updated	Last.Updated	1.3303601
Size	Size	0.8155511
Android.Ver	Android.Ver	0.6110890
Price	Price	0.5365366
Type	Type	0.0000000

1-10 of 11 rows Previous **1** 2 Next

```
#when no randomness, then review is the most important, so very different results!
dataset.boost <- gbm(as.formula(formula), data=dataset.train, shrinkage = 1, n.trees = 500, bag.
fraction=1)
```

```
## Distribution not specified, assuming gaussian ...
```

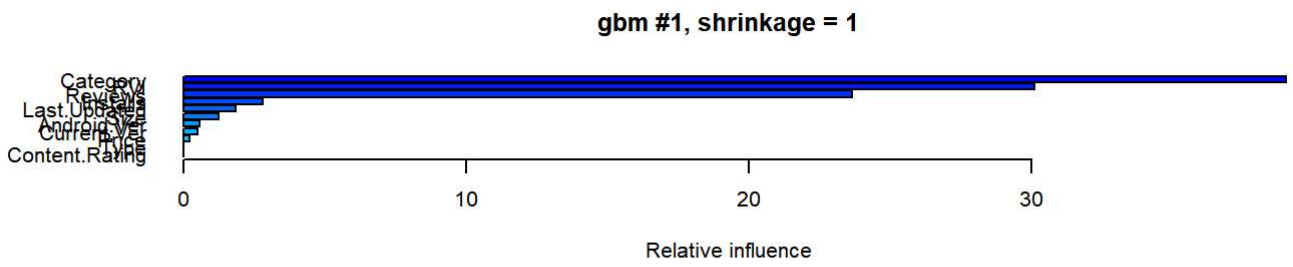
```
rel.inf <- relative.influence(dataset.boost, scale. = TRUE, sort. = TRUE)
```

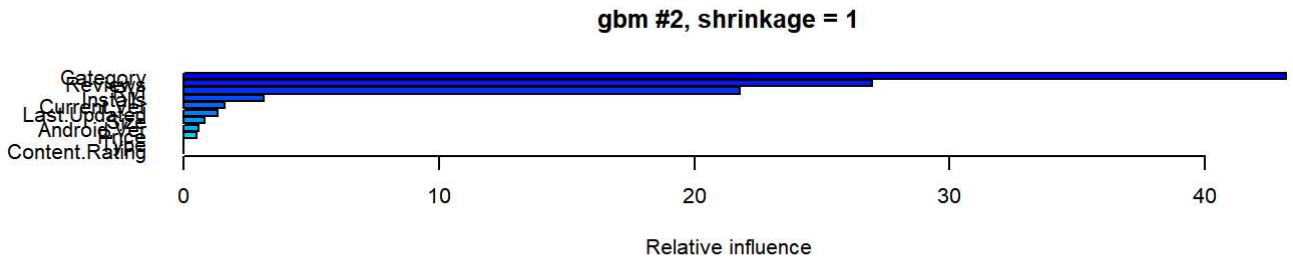
```
## n.trees not given. Using 500 trees.
```

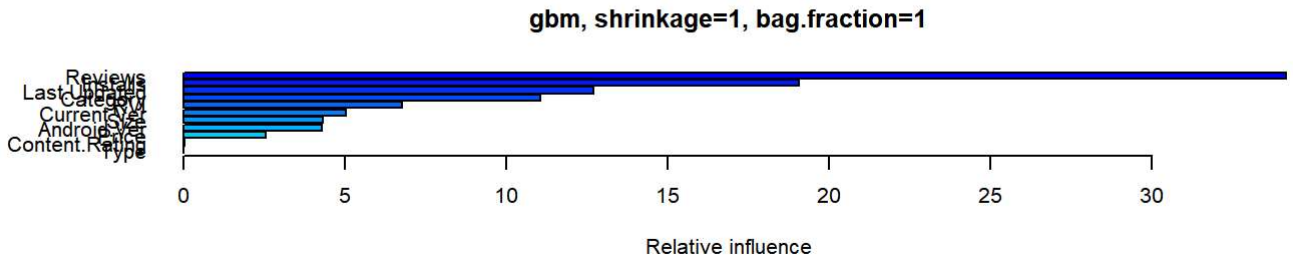
```
rel.inf <- rel.inf/sum(rel.inf)
round(100 * rel.inf, 2)
```

```
##        Reviews       Installs   Last.Updated       Category            RVI
##          34.14          19.06          12.70          11.07           6.76
##    Current.Ver           Size    Android.Ver          Price Content.Rating
##           5.05           4.33           4.30           2.54           0.05
##           Type
##           0.00
```

```
summary(dataset.boost, main = "gbm, shrinkage=1, bag.fraction=1", las=1)
```







	var <fctr>	rel.inf <dbl>
Reviews	Reviews	34.14173793
Installs	Installs	19.06244804
Last.Updated	Last.Updated	12.70076644
Category	Category	11.07144628
RVI	RVI	6.75851567
Current.Ver	Current.Ver	5.04672489
Size	Size	4.33390083
Android.Ver	Android.Ver	4.29702039
Price	Price	2.53817435
Content.Rating	Content.Rating	0.04926517

1-10 of 11 rows Previous **1** 2 Next

```
par(mfrow=c(1,1), mar=c(5.1,4.1,6.1,1.1))
```

```
#cross validation

formula = Rating~
  Category + Reviews + Size + Installs + Type +
Price + Content.Rating + Last.Updated +
Current.Ver + Android.Ver + RVI

fitControl <- trainControl(## 5-fold CV
  method = "cv",
  number = 5)

gbmGrid <-  expand.grid(interaction.depth = c(1, 3, 5),
                        n.trees = 500,
                        shrinkage = c(0.1,0.5,1),
                        n.minobsinnode = 10)


tic('gradient boosting computational time')
gbmFit <- train(formula, data = dataset.train,
                 method = "gbm",
                 trControl = fitControl,
                 verbose = FALSE,
                 tuneGrid = gbmGrid)
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution =
## "bernoulli", : variable 42: Content.RatingUnrated has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution =
## "bernoulli", : variable 42: Content.RatingUnrated has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution =
## "bernoulli", : variable 42: Content.RatingUnrated has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution =
## "bernoulli", : variable 42: Content.RatingUnrated has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution =
## "bernoulli", : variable 42: Content.RatingUnrated has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution =
## "bernoulli", : variable 42: Content.RatingUnrated has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution =
## "bernoulli", : variable 42: Content.RatingUnrated has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution =
## "bernoulli", : variable 42: Content.RatingUnrated has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution =
## "bernoulli", : variable 42: Content.RatingUnrated has no variation.
```

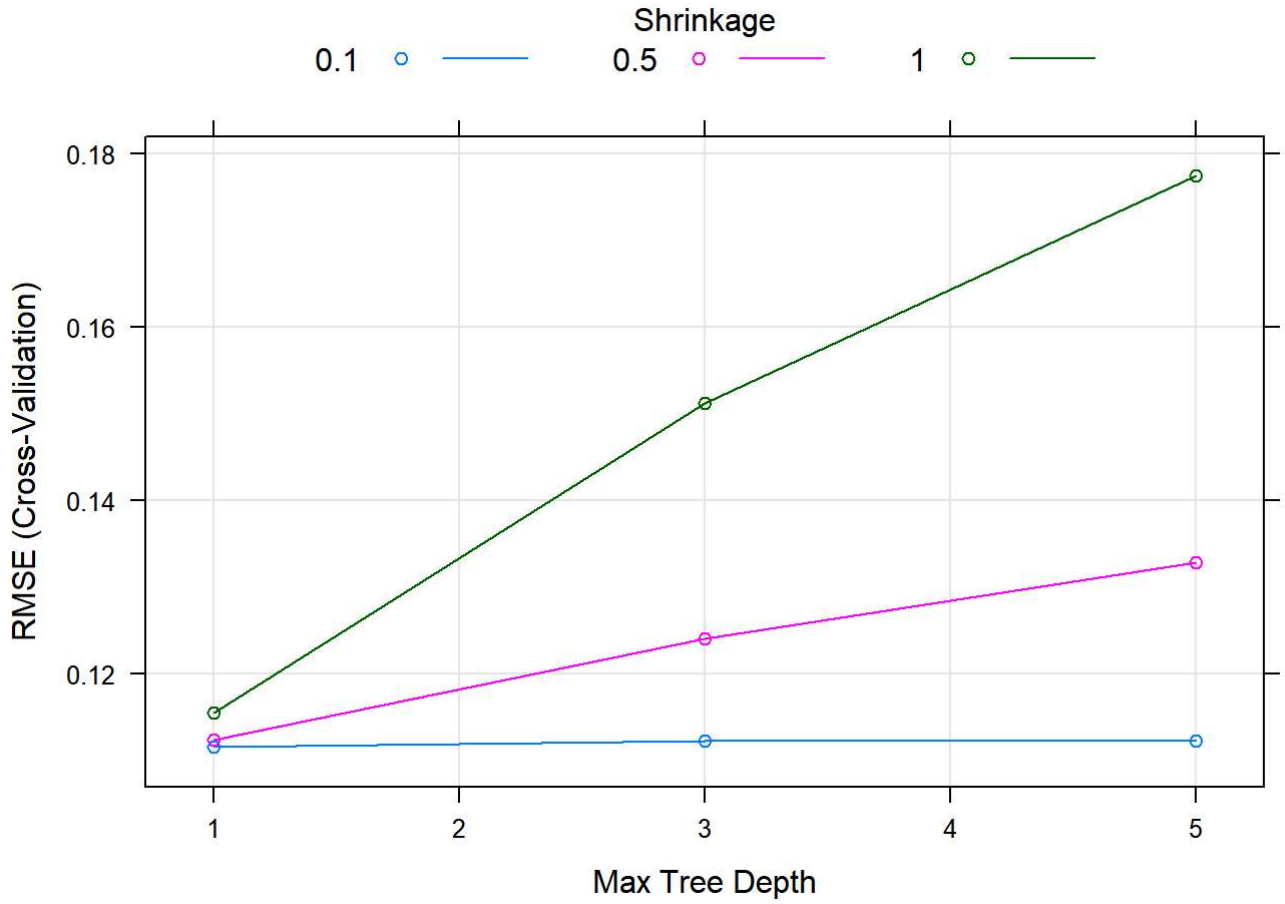
```
toc()
```

```
## gradient boosting computational time: 146.59 sec elapsed
```

```
gbmFit
```

```
## Stochastic Gradient Boosting
##
## 6557 samples
##   11 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 5246, 5246, 5246, 5244, 5246
## Resampling results across tuning parameters:
##
##   shrinkage  interaction.depth  RMSE       Rsquared    MAE
##   0.1        1                  0.1115315  0.10908371  0.07290355
##   0.1        3                  0.1122810  0.11298829  0.07178664
##   0.1        5                  0.1123090  0.12220750  0.07150356
##   0.5        1                  0.1124145  0.10450016  0.07277014
##   0.5        3                  0.1240746  0.07837581  0.07927762
##   0.5        5                  0.1328866  0.06449921  0.08604768
##   1.0        1                  0.1154923  0.08382180  0.07512727
##   1.0        3                  0.1512122  0.03783238  0.09685337
##   1.0        5                  0.1774205  0.03357915  0.11537771
##
## Tuning parameter 'n.trees' was held constant at a value of 500
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were n.trees = 500,
##  interaction.depth = 1, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
plot(gbmFit)
```




```
gbm_pred = predict(gbmFit, test)
gbm.mse = mean((actual-gbm_pred)^2)
gbm.rmse = rmse(actual,gbm_pred)
gbm.rmsle = rmsle(actual,gbm_pred)
gbm.mse
```

```
## [1] 0.01536917
```

```
gbm.rmse
```

```
## [1] 0.1239724
```

```
gbm.rmsle
```

```
## [1] 0.05125983
```

```
fitControl <- trainControl(## 5-fold CV
  method = "cv",
  number = 5)
tic('xgb computation time')
xgb_fit <- train(formula, data = train, method = "xgbTree",
                 trControl=fitControl)
toc()
```

```
## xgb computation time: 78.78 sec elapsed
```

```
xgb_pred = predict(xgb_fit, test)
xgb.mse = mean((actual-xgb_pred)^2)
xgb.rmse = rmse(actual,xgb_pred)
xgb.rmsle = rmsle(actual,xgb_pred)
xgb.mse
```

```
## [1] 0.01641214
```

```
xgb.rmse
```

```
## [1] 0.1281099
```

```
xgb.rmsle
```

```
## [1] 0.05295567
```