Simulation

Christian Wagner and Masaoud Haidar

3/10/2022

Setup

```r
#install.packages('DMwR')
#install.packages(c('xts', 'quantmod', 'abind', 'ROCR'))
#install.packages( "DMwR_0.4.1.tar.gz", repos=NULL, type="source" )
library(DMwR)
```

Generate simulation data

```r
# set parameters
set.seed(98)
reps = 200
n = 1000

beta = c(log(0.05 / 0.95) - 4.15, 1, 1) # Was -2, but changed to -4.15 to get 0.05
X_mean = 1
X1_var = X_mean ^ 2
X2_cond_var = 1
X2_var = X2_cond_var + X1_var

# list of dataframes
sims <- vector("list", reps)

for(i in 1:reps){
  # generate covariates. X_1 is an exponential with rate 1 and
  # X_2 is a normal with mean X_1 and SD = 1
  X1 <- rexp(n, X_mean)
  X2 <- rnorm(n, X1, X2_cond_var)

  # generate response variable. Y_i is distributed bernoulli
  # s.t. logit(P(Y_i = 1)) = beta_0 + beta_1 * X_1 + beta_2 * X_2
  p = exp(beta[1] + beta[2] * X1 + beta[3] * X2) /
    (1 + exp(beta[1] + beta[2] * X1 + beta[3] * X2))
  Y = rbinom(n, 1, p)
  #Y = rep(NA, n)
  #for(j in 1:n){
  #  Y[j] = rbinom(1, 1, p[j])
  #}
  # can we do rbinom(n, 1, p) instead of the loop?

  df = data.frame("X1" = X1, "X2" = X2, "Y" = Y)
  sims[[i]] = df
}
```

Helper functions:

```r
corr = function(a, b){
  # Computes the correlation between two vectors a and b
  num = mean(a * b) - mean(a) * mean(b)
  denom = sd(a) * sd(b)
  return(num / denom)
}

get_empty_df = function(sz = reps){
  # creates and returns an empty dataframe to
  # be populated by simulation results.
  df = data.frame(
    mean_X1 = rep(NA, sz),
    mean_X2 = rep(NA, sz),
    var_X1 = rep(NA, sz),
    var_X2 = rep(NA, sz),
    corr_X1_X2 = rep(NA, sz),
    beta0_star = rep(NA, sz),
    beta1_star = rep(NA, sz),
    beta2_star = rep(NA, sz),
    F1 = rep(NA, sz)
  )
  return(df)
}

get_perc_over = function(Y, desired_ratio){
  # Given a response variable Y and the desired proportion of 1s in Y,
  # this function calculates the required percentage of oversampling that
  # is required in order to achieve the desired proportion of 1s in Y.
  # This is designed so the output can be directly passed to the SMOTE function.
  num1 = sum(Y == 1)
  num0 = sum(Y == 0)
  # solved mathematically
  perc = 100 * desired_ratio * num0 / (num1 *(1 - desired_ratio))
  return(perc)
}
```

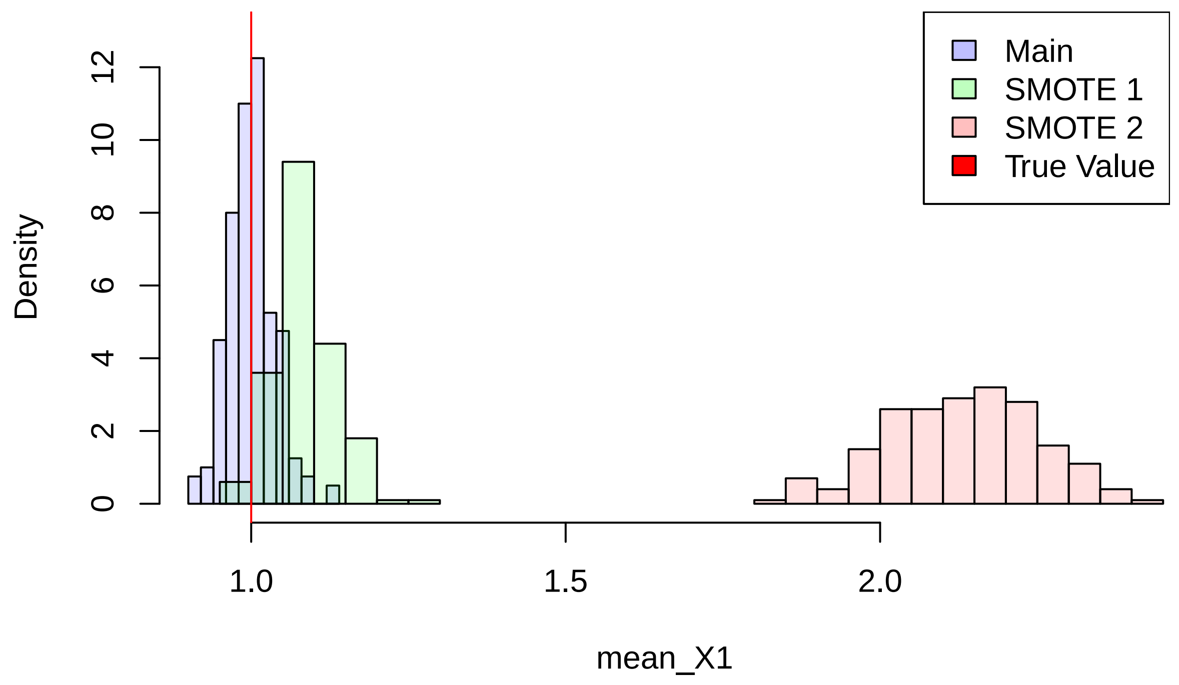
```r
compute_metrics = function(df, df_test){
  # Given a data frame (Either the original generated dataset or the
  # augmented dataset) and a test data frame, this function calculates all
  # the metrics we are interested in and returns them.
  # This is designed to return a row that can be added to a
  # dataframe of results.

  # distribution metrics:
  mean_X1 = mean(df$X1)
  mean_X2 = mean(df$X2)
  var_X1 = var(df$X1)
  var_X2 = var(df$X2)

  # Correlation metric:
  corr_X1_X2 = corr(df$X1, df$X2)
  # Coefficient metrics:
  model = glm(Y ~ X1 + X2, data = df, family = "binomial")
  beta_star = model$coefficients

  # F1 score on test data
  Y_star = (predict(model, newdata = df_test, type = "response") > 0.5) * 1
  TP = sum(Y_star == df_test$Y & Y_star == 1)
  FP = sum(Y_star != df_test$Y & Y_star == 1)
  FN = sum(Y_star != df_test$Y & Y_star == 0)
  precision = TP / (TP + FP)
  recall = TP / (TP + FN)
  F1 = 2 * (precision * recall) / (precision + recall)

  return(list(mean_X1,
              mean_X2,
              var_X1,
              var_X2,
              corr_X1_X2,
              beta_star[1],
              beta_star[2],
              beta_star[3],
              F1))
}
```

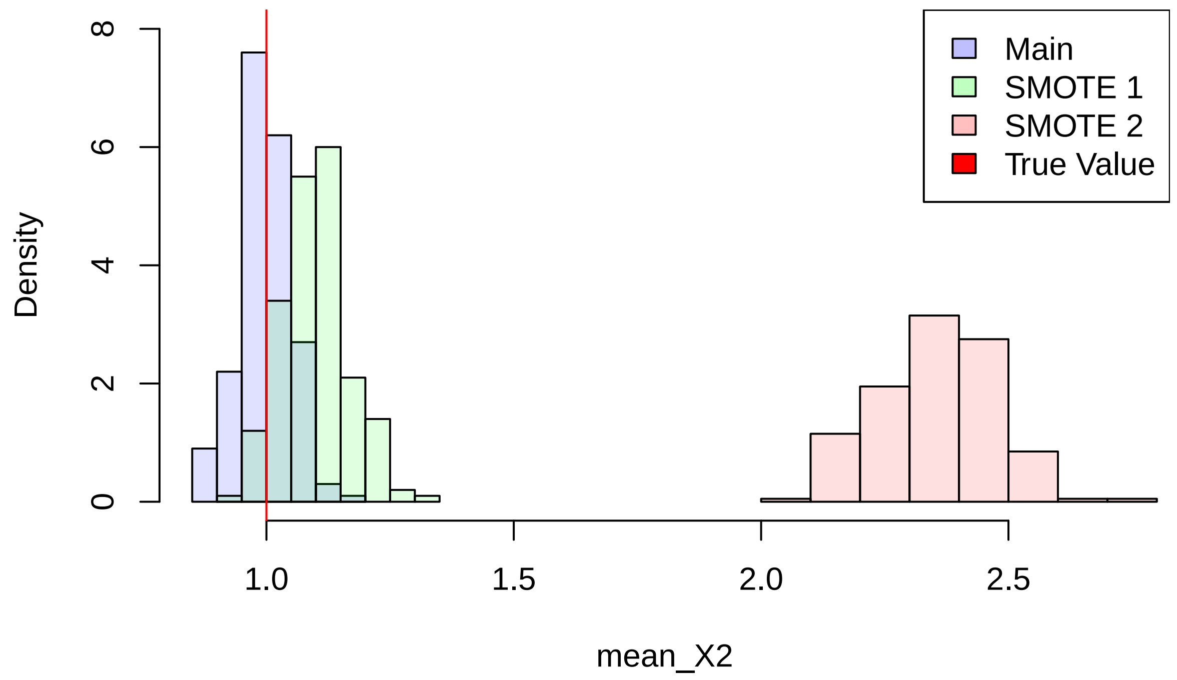
Simulation

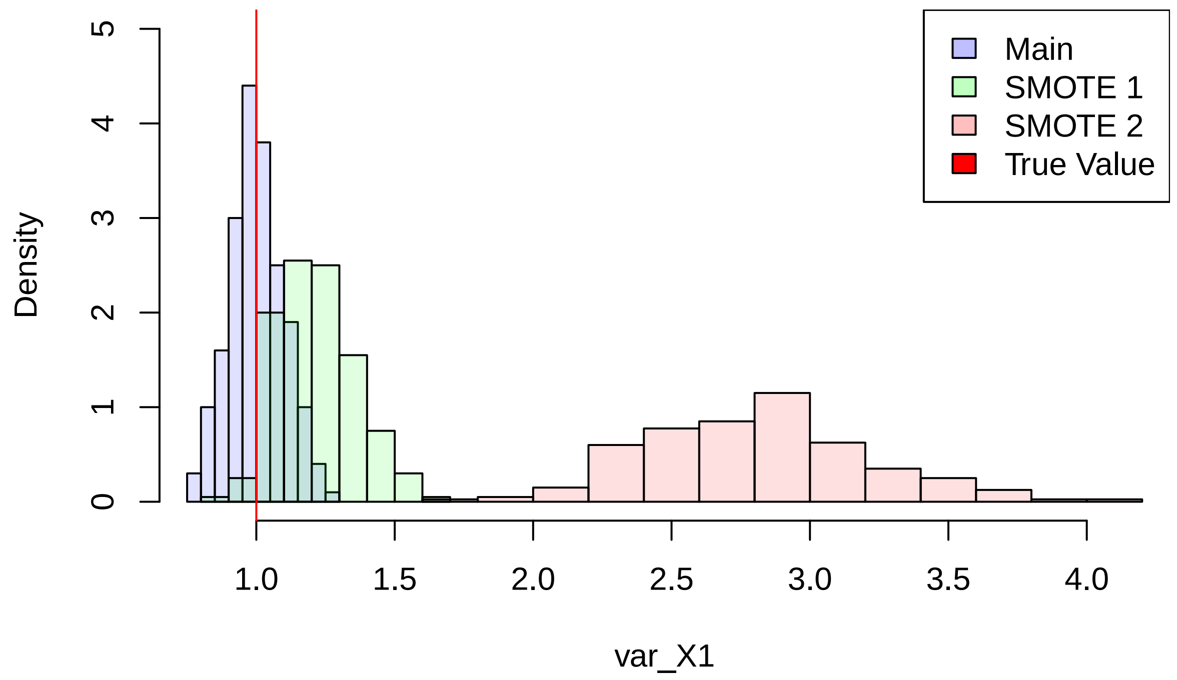
```r
# The three dataframes of results
res_base = get_empty_df()
res_smote1 = get_empty_df()
res_smote2 = get_empty_df()

#means = rep(NA, reps)
for (i in 1:reps){
  # Get the i-th data frame and split it into train and test
  df = sims[[i]]
  df_train = df[-1:-(0.2 * n), ]
  df_test = df[1:(0.2 * n), ]

  # Calculate the metrics for the original dataset
  res_base[i, ] = compute_metrics(df_train, df_test)

  # Convert Y into factor as required for the SMOTE function
  df_train$Y = factor(df_train$Y)

  # Augment the data according to two oversampling rates
  df_smote1 = SMOTE(Y ~ .,
                    df_train,
                    perc.over = get_perc_over(df_train$Y, 0.1),
                    perc.under = 1800,
                    k = 5)
  df_smote2 = SMOTE(Y ~ .,
                    df_train,
                    perc.over = get_perc_over(df_train$Y, 0.5),
                    perc.under = 106,
                    k = 5)

  # Convert from factor back to numbers
  df_smote1$Y = (df_smote1$Y == 1)*1
  df_smote2$Y = (df_smote2$Y == 1)*1

  # Calculate the metrics for the two SMOTE datasets
  res_smote1[i, ] = compute_metrics(df_smote1, df_test)
  res_smote2[i, ] = compute_metrics(df_smote2, df_test)
}
```

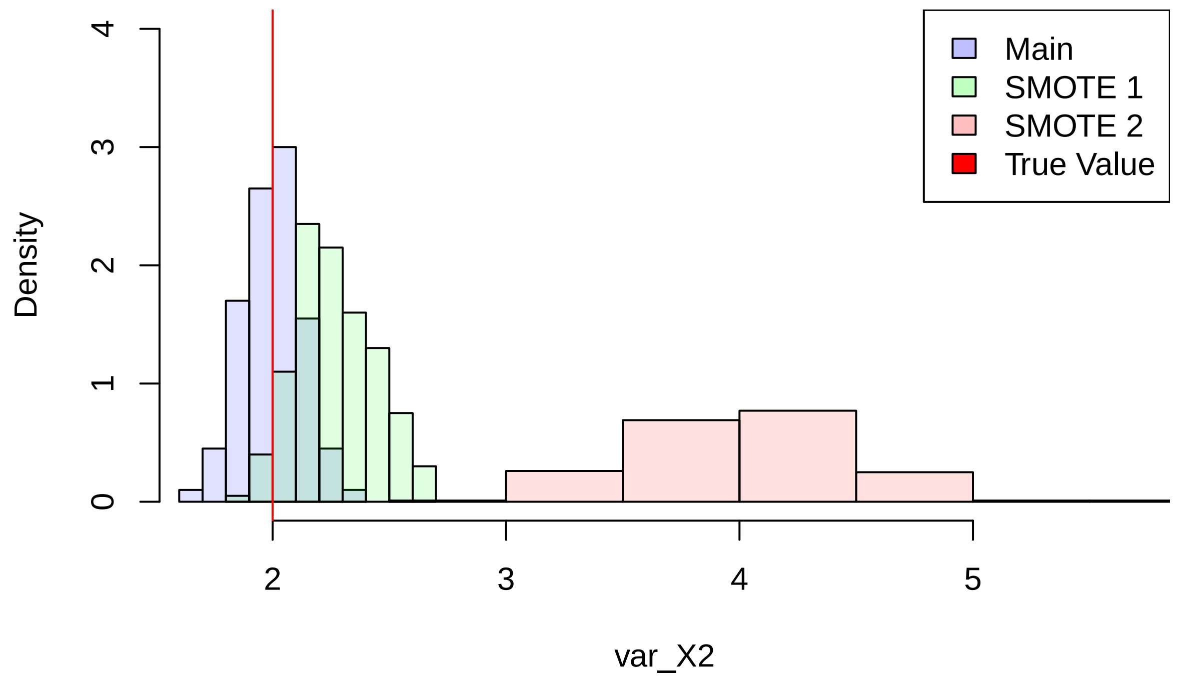
Results: Histograms

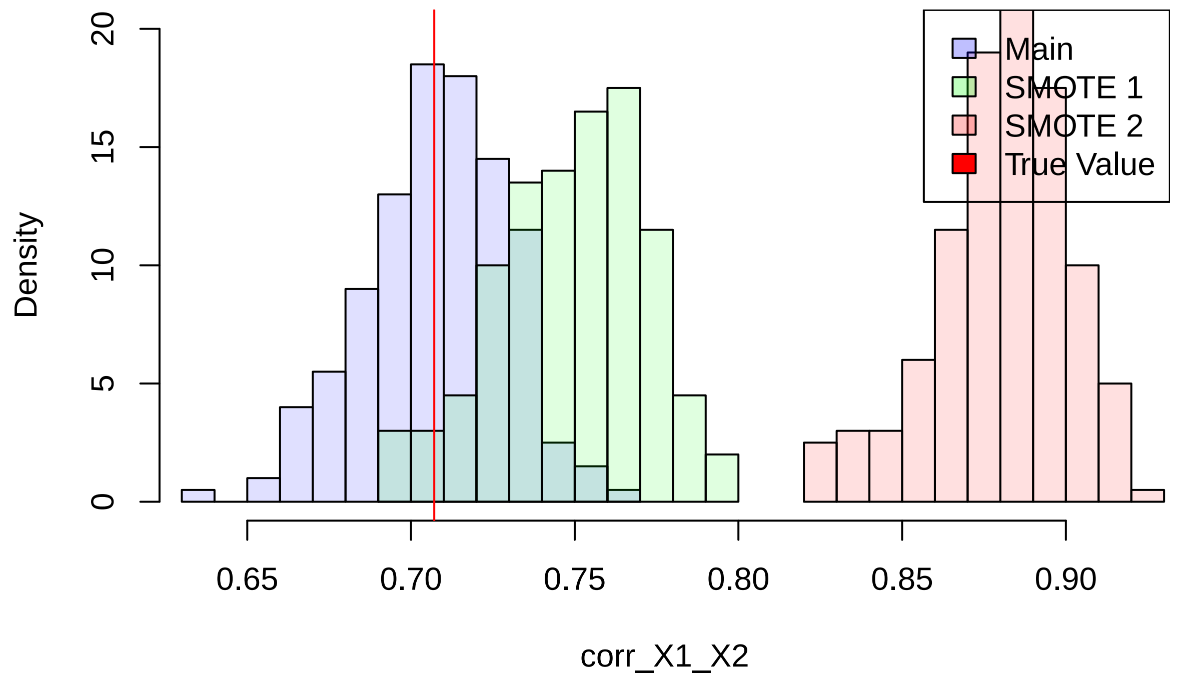
```r
# Plot histograms of all variables according with their real values (if
# those exist.)
theoritical_values = c(1, 1, 1, 2, 1 / sqrt(2), beta[1], beta[2], beta[3], NA)
ylims = c(13, 8, 5, 4, 20, 0.6, 2, 1.4, 5)

for (i in 1:9){
  col_name = names(res_base)[i]
  # uncomment this next line to save the figure as png
  #png(file=paste(c("hist_", col_name, ".png"), collapse = ""),width=600, height=350)

  mn1 = min(res_base[, col_name], na.rm = T)
  mn2 = min(res_smote1[, col_name], na.rm = T)
  mn3 = min(res_smote2[, col_name], na.rm = T)
  mn = min(mn1, mn2, mn3)

  mx1 = max(res_base[, col_name], na.rm = T)
  mx2 = max(res_smote1[, col_name], na.rm = T)
  mx3 = max(res_smote2[, col_name], na.rm = T)
  mx = max(mx1, mx2, mx3)

  hist(res_base[, col_name],
       main = " ",
       xlab = col_name,
       col = rgb(0,0,1,1/8),
       xlim = c(mn, mx),
       ylim = c(0, ylims[i]),
       freq = F)

  hist(res_smote1[, col_name],
       xlab = col_name,
       add = T,
       col = rgb(0,1,0,1/8),
       freq = F)

  hist(res_smote2[, col_name],
       xlab = col_name,
       add = T,
       col = rgb(1,0,0,1/8),
       freq = F)



  if (!is.na(theoritical_values[i])){
    abline(v = theoritical_values[i], col = "red")
    legend("topright",
         legend=c("Main", "SMOTE 1", "SMOTE 2", "True Value"),
         fill = c(rgb(0, 0, 1, 1/4),
                  rgb(0, 1, 0, 1/4),
                  rgb(1, 0, 0, 1/4),
                  "red"))
  }
  else{
    legend("topright",
         legend=c("Main", "SMOTE 1", "SMOTE 2"),
         fill = c(rgb(0, 0, 1, 1/4),
                  rgb(0, 1, 0, 1/4),
                  rgb(1, 0, 0, 1/4)))
  }
  # uncomment this next line to save the figure as png
  # dev.off()
}
```



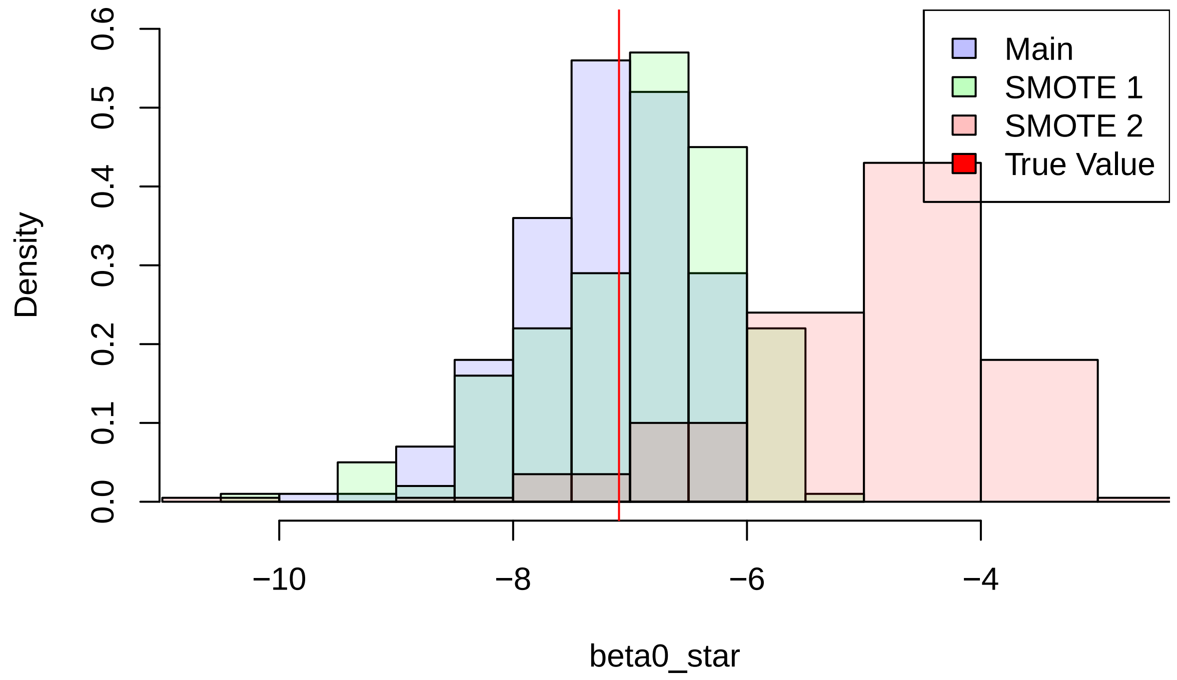

Density
mean_X2
Main
SMOTE 1
SMOTE 2
True Value
0
2
4
6
8
1.0
1.5
2.0
2.5


Main
SMOTE 1
SMOTE 2
True Value
Density
var_X1
0
1
2
3
4
5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

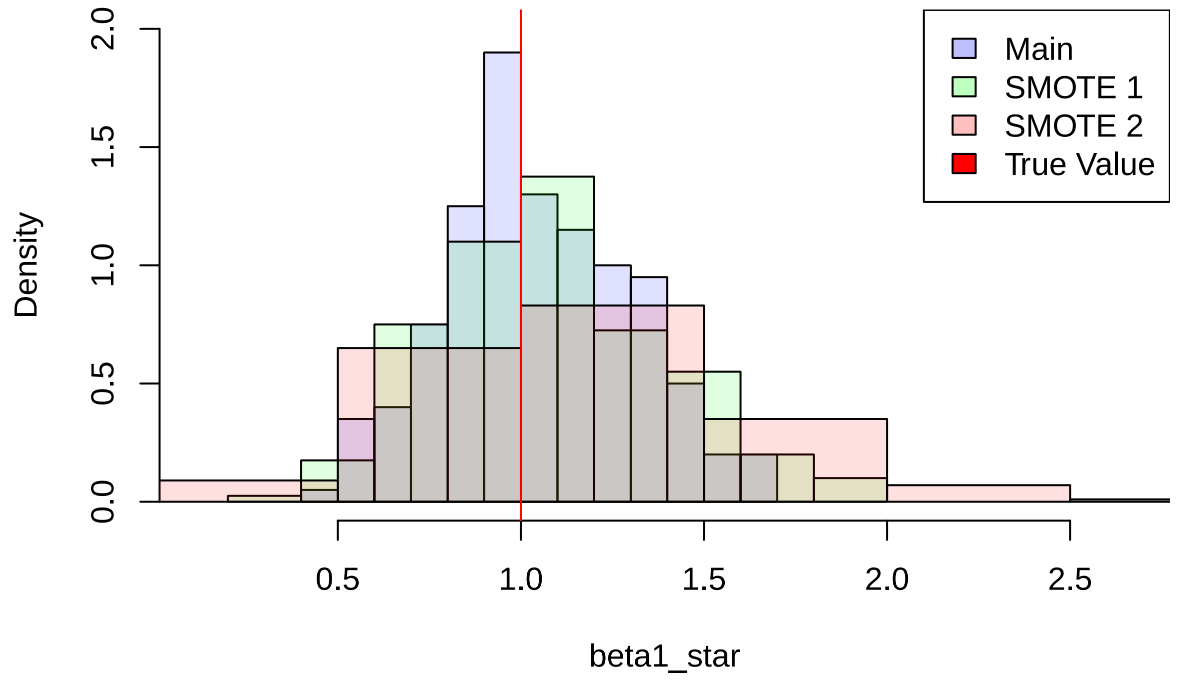

Density
var_X2
Main
SMOTE 1
SMOTE 2
True Value
0
1
2
3
4
2
3
4
5

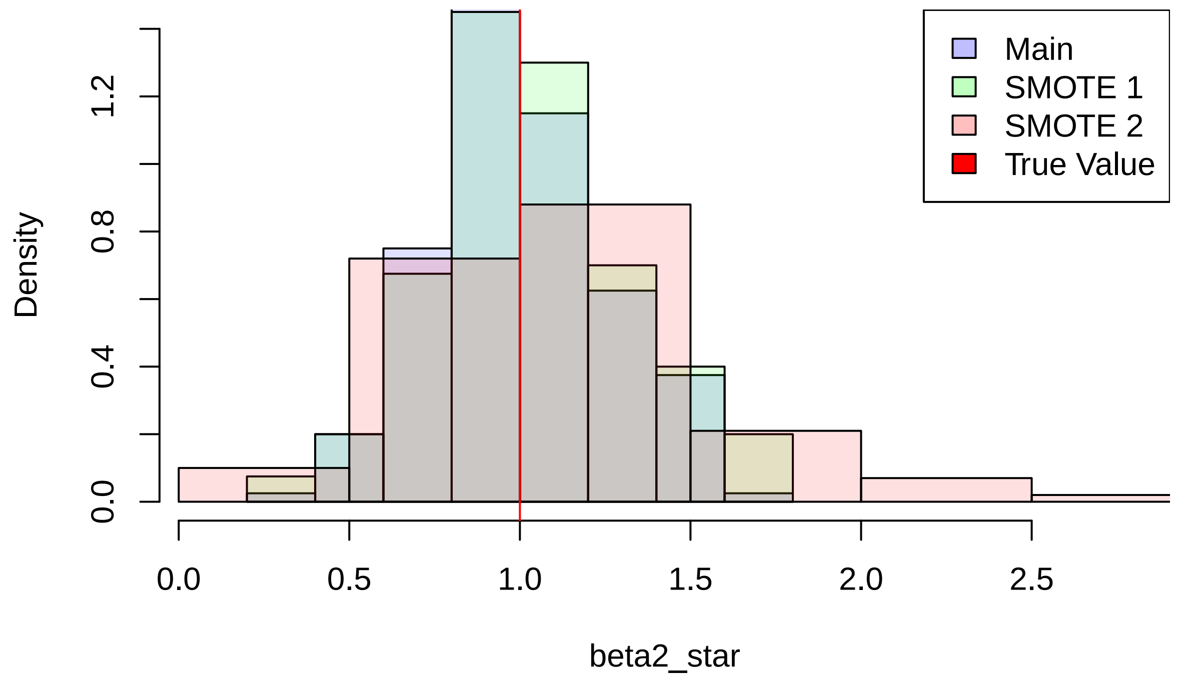


Main
SMOTE 1
SMOTE 2
True Value
Density
corr_X1_X2
0
5
10
15
20
0.65
0.70
0.75
0.80
0.85
0.90

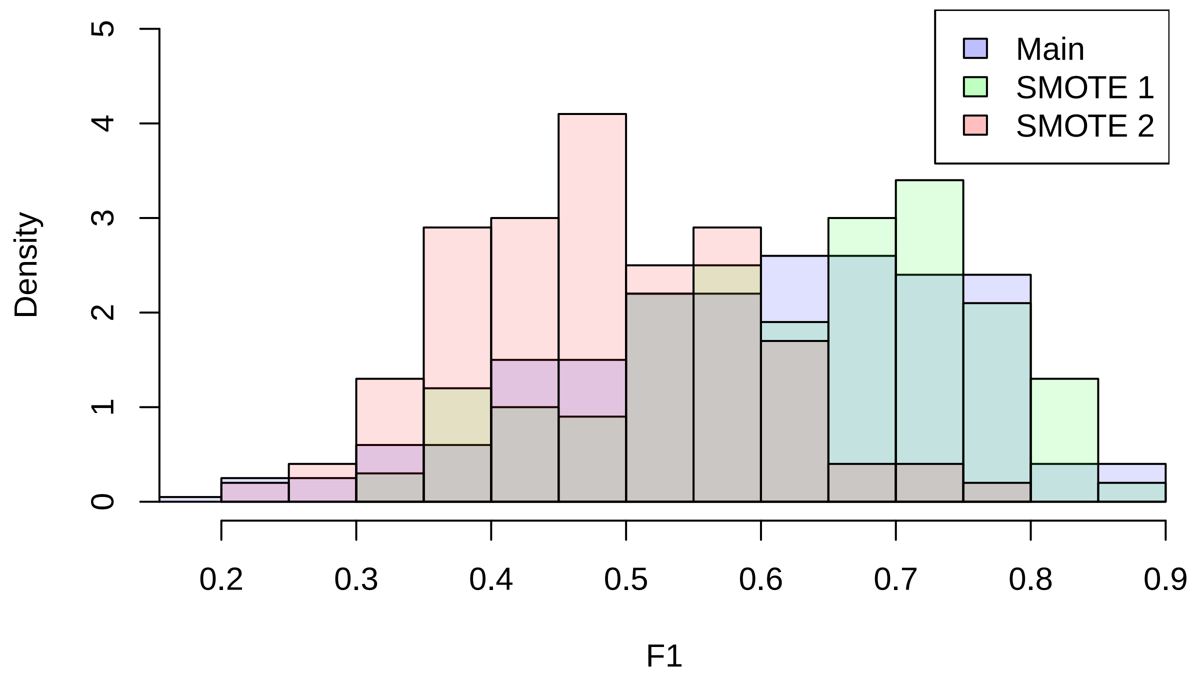


Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
−10
−8
−6
−4
beta0_star
Main
SMOTE 1
SMOTE 2
True Value


Main
SMOTE 1
SMOTE 2
True Value
Density
beta1_star
0.0
0.5
1.0
1.5
2.0
2.5



Main
SMOTE 1
SMOTE 2
True Value
Density
0.0
0.4
0.8
1.2
0.0
0.5
1.0
1.5
2.0
2.5
beta2_star


Main
SMOTE 1
SMOTE 2
Density
F1
0
1
2
3
4
5
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

Results: Table

```
# Print a table of the results
get_mean = function(df){
  return(list(
    mean(df$mean_X1),
    mean(df$mean_X2),
    mean(df$var_X1),
    mean(df$var_X2),
    mean(df$corr_X1_X2),
    mean(df$beta0_star),
    mean(df$beta1_star),
    mean(df$beta2_star),
    mean(df$F1)))
}

res_comparision = get_empty_df(3)
res_comparision[1, ] = get_mean(res_base)
res_comparision[2, ] = get_mean(res_smote1)
res_comparision[3, ] = get_mean(res_smote2)
res_comparision
```

```
##    mean_X1   mean_X2   var_X1   var_X2 corr_X1_X2 beta0_star beta1_star
## 1 1.000229 0.9958496 1.001767 1.999820  0.7077176  -7.238039   1.049425
## 2 1.085639 1.0986097 1.218497 2.268150  0.7492140  -6.914132   1.091067
## 3 2.129992 2.3523018 2.806397 4.010298  0.8800853  -4.904668   1.173547
##   beta2_star        F1
## 1  0.9966389 0.6042718
## 2  1.0407546 0.6334307
## 3  1.1293139 0.4849455
```

Results: Hypothesis testing for difference in Beta Estimates and F1 scores

```
# data frame with parameter estimates categorized by smote level
beta1 = c(res_base$beta1_star, res_smote1$beta1_star, res_smote2$beta1_star)
beta2 = c(res_base$beta2_star, res_smote1$beta2_star, res_smote2$beta2_star)
mean1 = c(res_base$mean_X1, res_smote1$mean_X1, res_smote2$mean_X1)
mean2 = c(res_base$mean_X2, res_smote1$mean_X2, res_smote2$mean_X2)
var1 = c(res_base$var_X1, res_smote1$var_X1, res_smote2$var_X1)
var2 = c(res_base$var_X2, res_smote1$var_X2, res_smote2$var_X2)
corr = c(res_base$corr_X1_X2, res_smote1$corr_X1_X2, res_smote2$corr_X1_X2)
F1 = c(res_base$F1, res_smote1$F1, res_smote2$F1)
smote_level = factor(c(rep(0, 200), rep(0.1, 200), rep(0.5, 200)))

df_params <- data.frame("beta1" = beta1, "beta2" = beta2, "mean1" = mean1,
                        "mean2" = mean2, "var1" = var1, "var2" = var2,
                        "corr" = corr, "F1" = F1, "smote_level" = smote_level)
```

Now, we can conduct two ANOVA tests, to see if the difference in smote level as seen in the histograms is significant at $\alpha = 0.05$.

```
# conduct an anova test on the beta estimates
anova1 <- aov(beta1 ~ smote_level, data = df_params)
summary(anova1)
```

```
##              Df Sum Sq Mean Sq F value  Pr(>F)
## smote_level   2   1.60  0.7981    6.85 0.00114 **
## Residuals   597  69.56  0.1165
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Testing for β_1, we get a test statistic of $F_{2,597} = 6.85$ which corresponds to a p-value of 0.001. Thus, altering the oversampling rate does have a significant effect on β_1 estimates.

```
anova2 <- aov(beta2 ~ smote_level, data = df_params)
summary(anova2)
```

```
##              Df Sum Sq Mean Sq F value   Pr(>F)
## smote_level   2   1.83  0.9131   8.401 0.000252 ***
## Residuals   597  64.89  0.1087
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Testing for β_2, we get a test statistic of $F_{2,597} = 8.401$ which corresponds to a p-value of 0.00025. Thus, altering the oversampling rate does have a significant effect on β_2 estimates.

Focusing in on both smote options explored, with final ratios of 0.1 and 0.5, we can conduct a t-test on the F1 scores of each resulting model using Fisher's strict null hypothesis.

```
t.test(df_params$F1[df_params$smote_level == 0.1], df_params$F1[df_params$smote_level == 0.5])
```

```
##
##  Welch Two Sample t-test
##
## data:  df_params$F1[df_params$smote_level == 0.1] and df_params$F1[df_params$smote_level == 0.5]
## t = 12.588, df = 380.72, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.1252915 0.1716788
## sample estimates:
## mean of x mean of y
## 0.6334307 0.4849455
```

Here, we get a test statistic of $t_{380} = 12.588$ which corresponds to a p-value of less than 2.2e-16, a highly significant result. Therefore, the F1 score of the less aggressive smote model with a 0.1 positive rate is significantly higher than that of the 0.5 positive oversampling smote.

We can also perform this test to compare between the original data and the smote model with a level of 0.1.

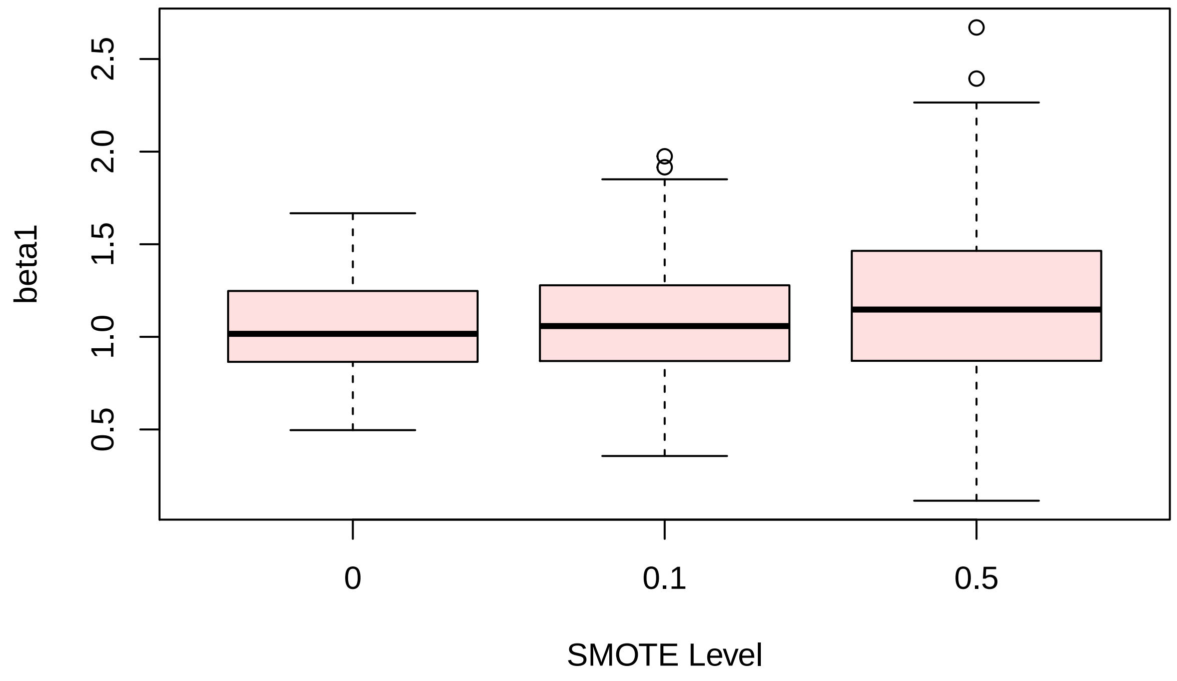
```
t.test(df_params$F1[df_params$smote_level == 0.1], df_params$F1[df_params$smote_level == 0])
```

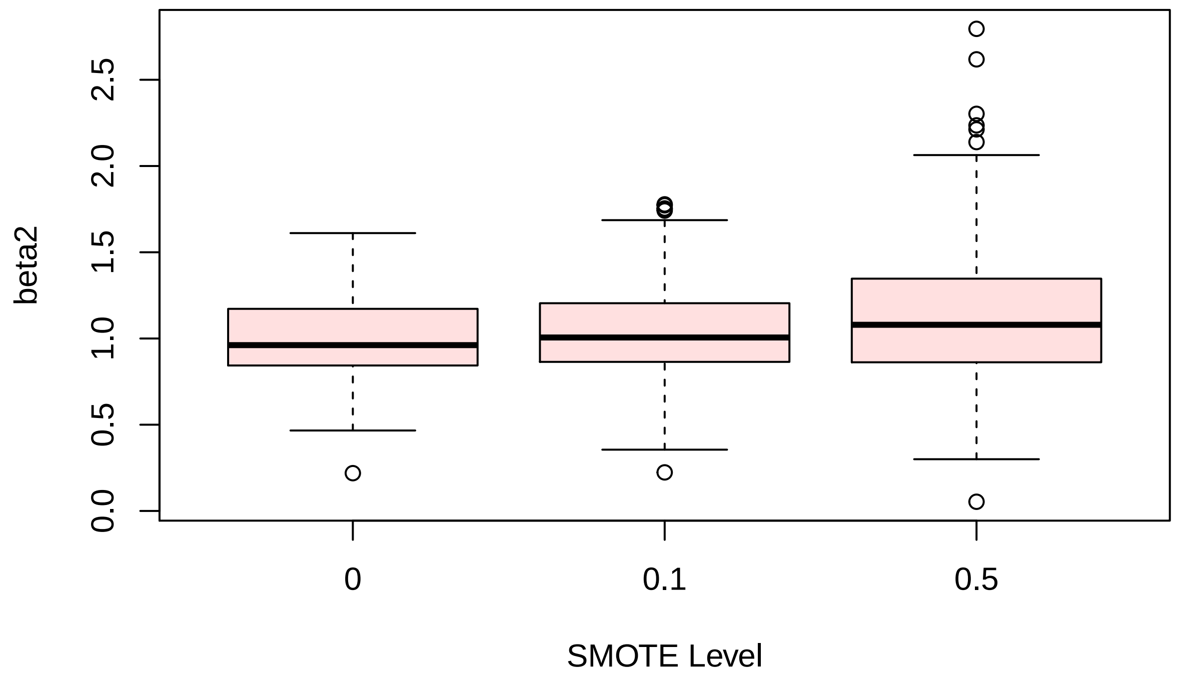
```
##
##  Welch Two Sample t-test
##
## data:  df_params$F1[df_params$smote_level == 0.1] and df_params$F1[df_params$smote_level == 0]
## t = 2.1585, df = 395.77, p-value = 0.03149
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.002600593 0.055717275
## sample estimates:
## mean of x mean of y
## 0.6334307 0.6042718
```

While the F1 score on these two datasets is close, we can see that using smote with level 0.1 did significantly improve the F1 score.

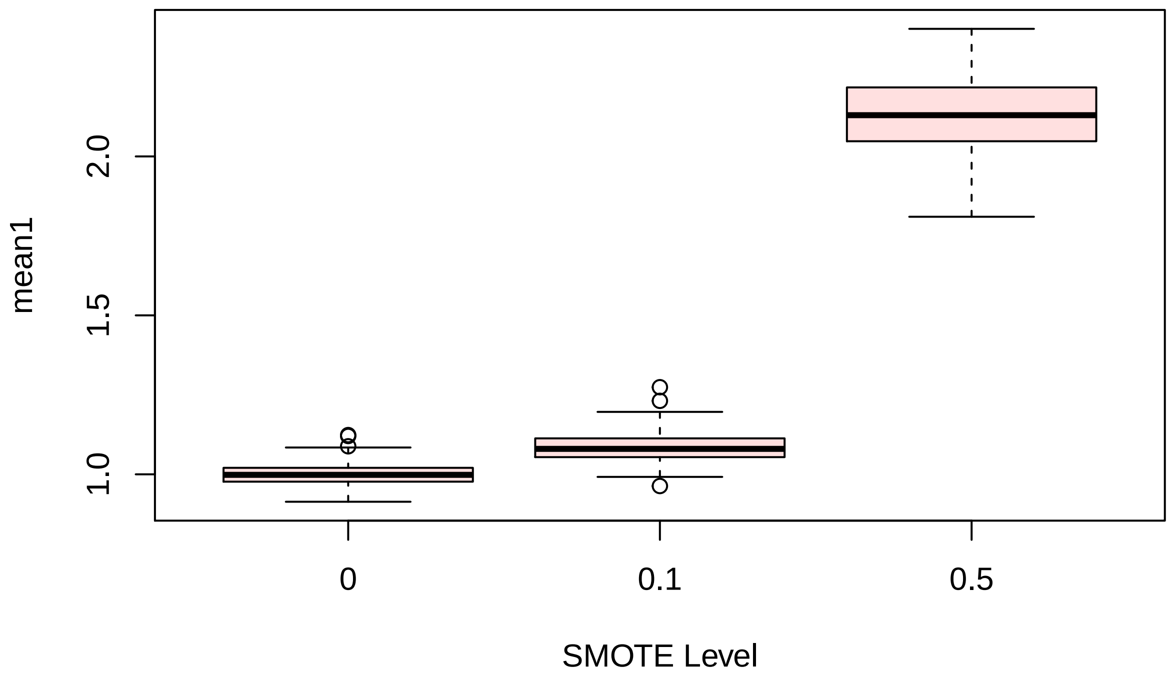
Results: Boxplots

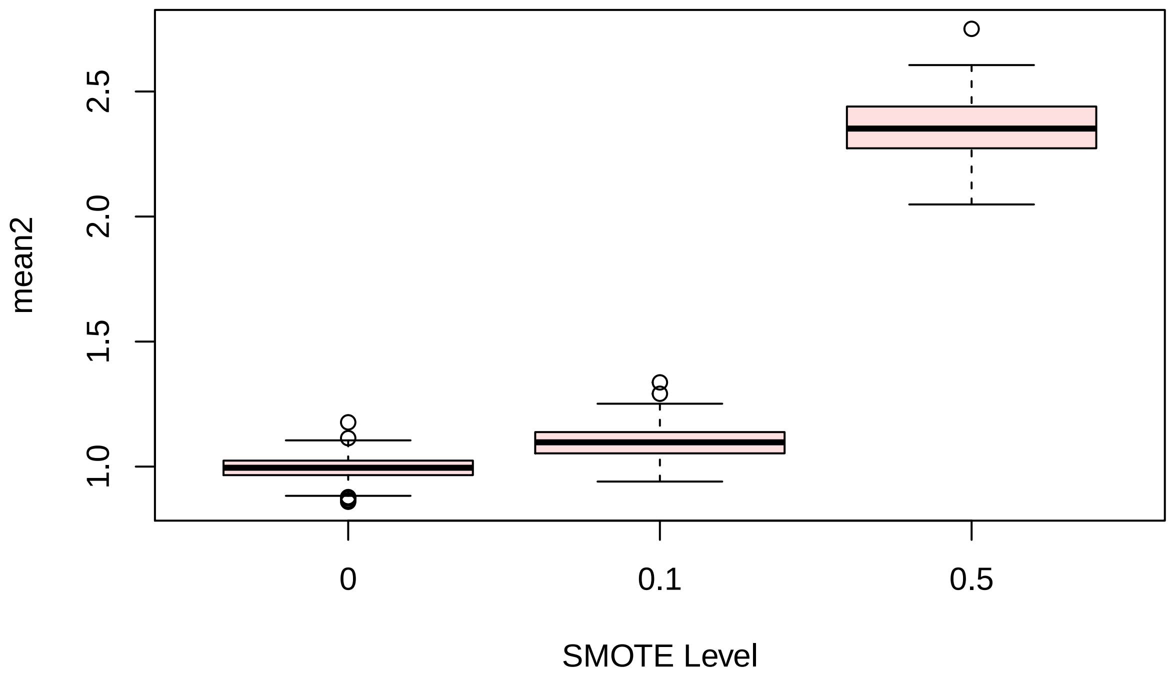
```r
# Plot boxplots for all variables
for (i in 1:8){
  col_name = names(df_params)[i]
  # uncomment this next line to save the figure as png
  #png(file=paste(c("boxplot_", col_name, ".png"), collapse = ""),width=600, height=350)

  boxplot(df_params[, col_name] ~ df_params$smote_level,
          main = " ",
          xlab = "SMOTE Level",
          ylab = col_name,
          col = rgb(1,0,0,1/8))

  # uncomment this next line to save the figure as png
  #dev.off()
}
```

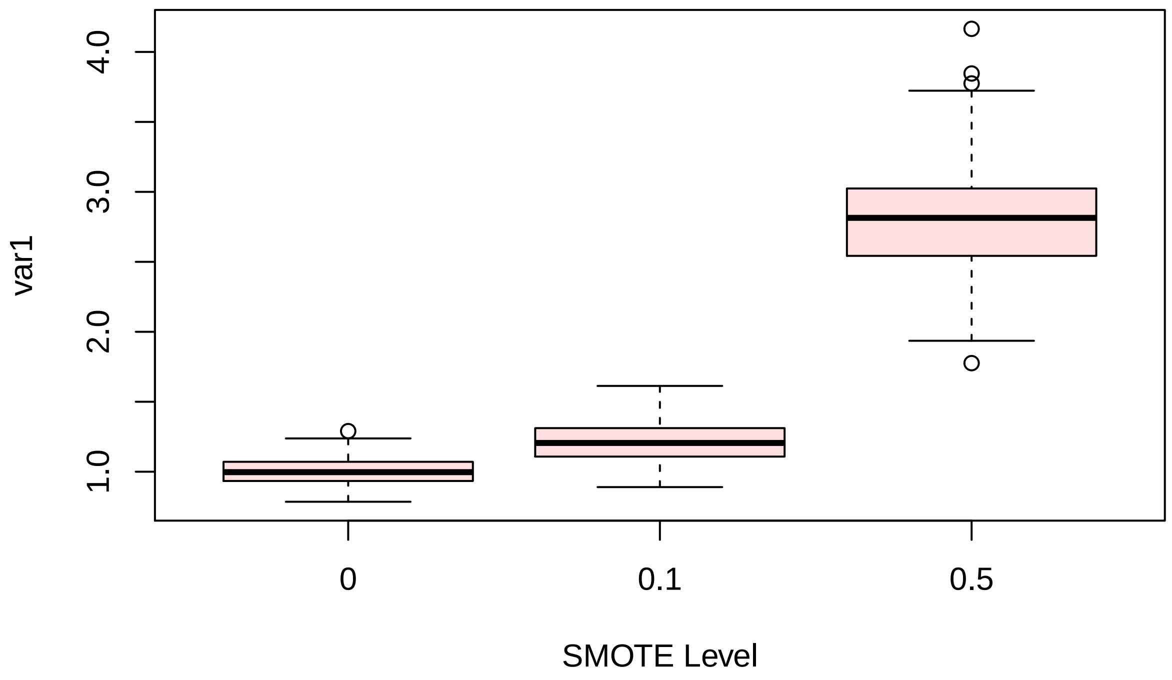


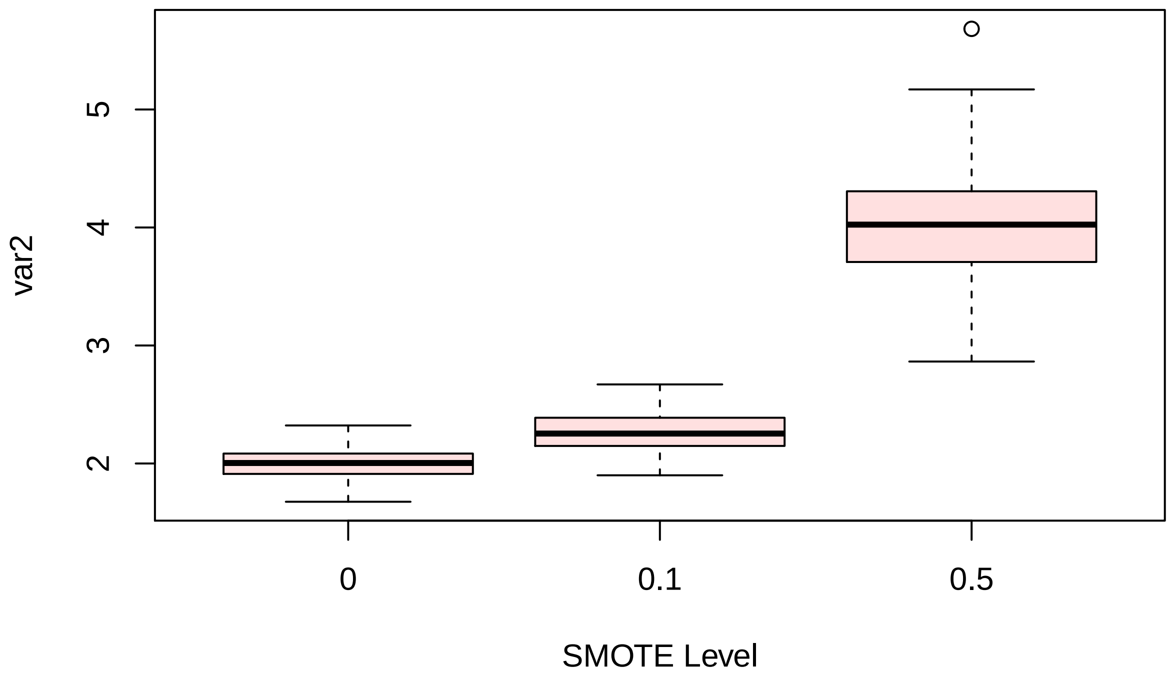


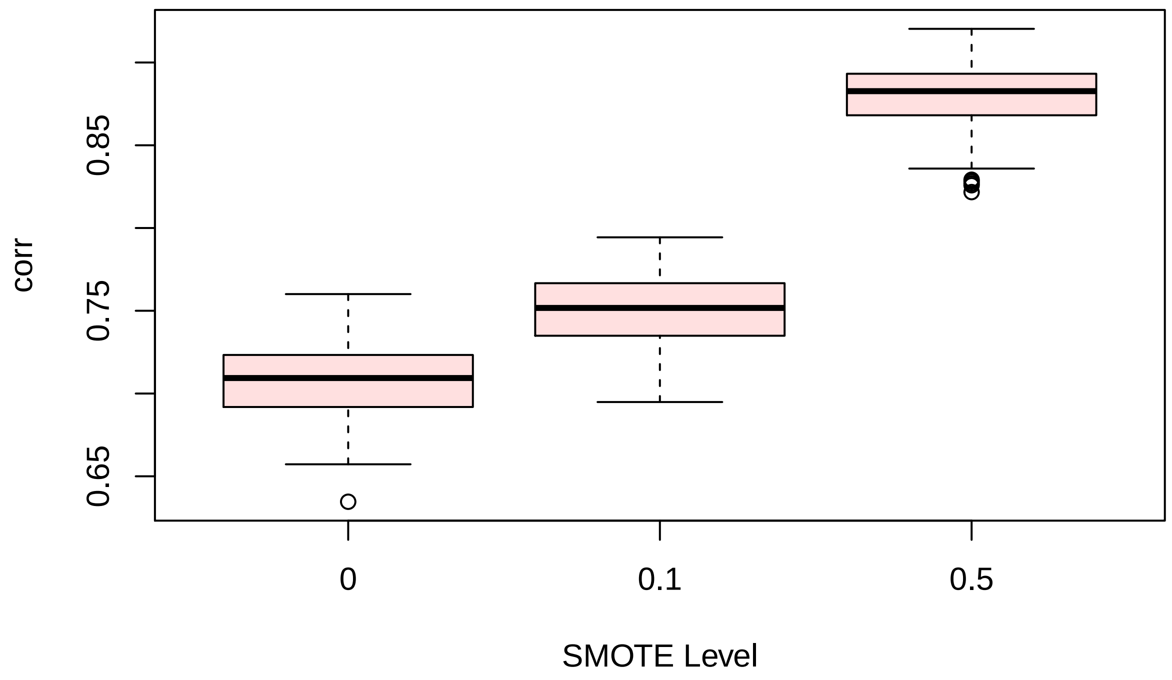
beta2
0.0
0.5
1.0
1.5
2.0
2.5
0
0.1
0.5
SMOTE Level


mean1
2.0
1.5
1.0
0
0.1
0.5
SMOTE Level

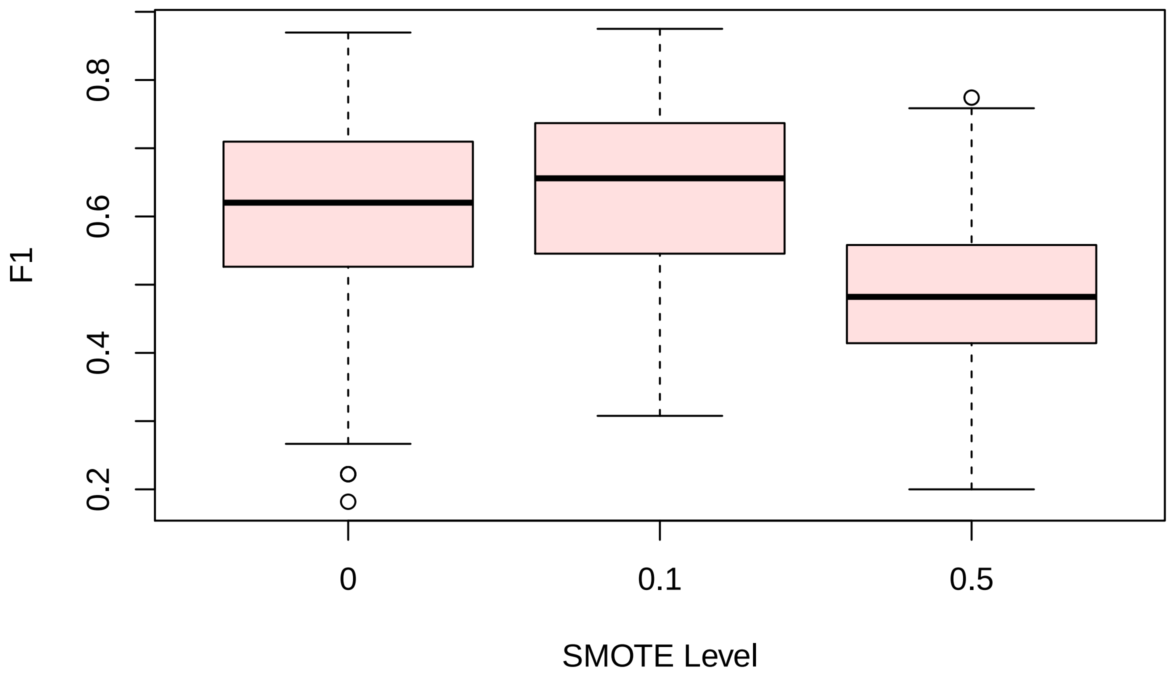

mean2
2.5
2.0
1.5
1.0
0
0.1
0.5
SMOTE Level


4.0
3.0
2.0
1.0
var1
0
0.1
0.5
SMOTE Level


var2
5
4
3
2
0
0.1
0.5
SMOTE Level



corr
0.65
0.75
0.85
0
0.1
0.5
SMOTE Level


F1
0.2
0.4
0.6
0.8
0
0.1
0.5
SMOTE Level