```
<-------- PD--------------------------->

# Import Data

install.packages("xlsx")
library(readxl)
getwd()
oneyearPD <- read_excel("oneyearpd.xlsx")

# Data structure

install.packages("dplyr")

library(dplyr)

dplyr::glimpse(oneyearPD)

# Rounding arrears count field

oneyearPD$max_arrears_12m<- round(oneyearPD$max_arrears_12m,4)
oneyearPD$arrears_months<- round(oneyearPD$arrears_months,4)

# Default flag definition


oneyearPD<- dplyr::mutate(oneyearPD,
                  default_event = if_else(oneyearPD$arrears_event == 1 |
                                          oneyearPD$term_expiry_event == 1 |
                                          oneyearPD$bankrupt_event == 1, 1,0))

# Recode default event variables for more convenient use
# 0-default, 1-non-default

oneyearPD$default_flag<-
  dplyr::if_else(oneyearPD$default_event == 1,0,1)

# Perform a stratified sampling: 70% train and 30% test

install.packages("caret")
library(caret)
set.seed(2122)
train.index <- caret::createDataPartition(oneyearPD$default_event,
                                          p = .7, list = FALSE)
train <- oneyearPD[ train.index,]
test <- oneyearPD[-train.index,]

# Information value assessment

install.packages("smbinning")
library(smbinning)
iv_analysis<- smbinning.sumiv(df=train,y="default_flag")

# Plot IV summary table

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

```
smbinning.sumiv.plot(iv_analysis,cex=1)

# Correlation check

woe_vars<- train %>%
  dplyr::select(starts_with("woe"))
woe_corr<- cor(as.matrix(woe_vars), method = 'spearman')

# Graphical inspection
library(corrplot)
corrplot(woe_corr, method = 'number')

#Discarding a highly correlated variable
woe_vars_clean<- woe_vars %>%
  dplyr::select( -woe_max_arrears_bal_6m)

#Loading required packages and attaching the database
library(MASS)
attach(train)

# Fitting a full logistic regression model
logit_full<- glm(default_event~ woe_bureau_score+
                   woe_annual_income+woe_emp_length+woe_max_arrears_12m
                 +woe_months_since_recent_cc_delinq+woe_num_ccj+woe_cc_util,
                 family = binomial(link = 'logit'), data = train)
logit_stepwise<- stepAIC(logit_full, k=qchisq(0.05, 1,
                                      lower.tail=F), direction = 'both')

# Detaching the dataset
detach(train)

# Define the scaling function
scaled_score <- function(logit, odds, offset = 500, pdo = 20) {
  b <- pdo / log(2)
  a <- offset - b * log(odds)
  round(a + b * log((1 - logit) / logit))
}

# Load the necessary library
library(dplyr)

# Use fitted model to score both test and train datasets
predict_logit_test <- predict(logit_stepwise, newdata = test, type = "response")
predict_logit_train <- predict(logit_stepwise, newdata = train, type = "response")

# Apply the scaling function to obtain scores
test_scores <- sapply(predict_logit_test, function(x) scaled_score(x, odds = 1))
train_scores <- sapply(predict_logit_train, function(x) scaled_score(x, odds = 1))

# Add scores to the test and train datasets
test <- test %>% mutate(score = test_scores)
train <- train %>% mutate(score = train_scores)

# Display the first few rows of the scored datasets
head(test)
```

```
head(train)

# Add predicted probabilities to test and train datasets
test$predict_logit <- predict(logit_stepwise, newdata = test, type = "response")
train$predict_logit <- predict(logit_stepwise, newdata = train, type = "response")

# Add a sample indicator column to differentiate between train and test data
train$sample <- 'train'
test$sample <- 'test'

# Combine train and test datasets into a single dataset
data_whole <- rbind(train, test)

# Select specific columns to create the final scored dataset
data_score <- data_whole %>%
  dplyr::select(id, default_event, default_flag, woe_bureau_score,
                woe_annual_income, woe_max_arrears_12m,
                woe_months_since_recent_cc_delinq,
                woe_cc_util, sample, predict_logit)

# Define scoring parameters and calculate the final score
# 72 = odds, 660 = offset, 40 = points to double the odds (pdo)
data_score$score <- sapply(data_score$predict_logit, function(x) scaled_score(x, odds = 72, offset = 660,
pdo = 40))

# Display the first few rows of the scored data
head(data_score)


# Load the pROC library
library(pROC)

# Plot the ROC curve for the training data
plot(roc(train$default_event, train$predict_logit, direction = "<"),
     col = "blue", lwd = 3, main = "ROC Curve")

1. Gini index
library(optiRum)
gini_train<- optiRum::giniCoef(train$predict_logit,
train$default_event)
print(gini_train)


<---------------- Lifetime PD---------------------->

LTPD <- read_excel("Lifetime_PD.xlsx")

# Fit random forest

train_def <- LTPD %>% dplyr::filter(sample=="train")
test_def <- LTPD %>% dplyr::filter(sample=="test")
train_def$def_char=as.factor(ifelse(train_def$default_flag==0,
                                    'No','Yes'))

test_def$def_char=as.factor(ifelse(test_def$default_flag==0,
```

```
                              'No','Yes'))

library(randomForest)
set.seed(123)
rf_def <- randomForest(def_char ~ tob+ltv_utd+
                         gdp+uer+cpi+hpi+ir+gdp_lag, data=train_def, mtry=3,
                       ntree=50, importance=TRUE, na.action=na.omit)
importance(rf_def)

<------- LGD----------------->


# 1. Updload data
lgd <- read_excel("data_lgd_year.xlsx")
# 2. Explorative analysis
# Box-plot months to maturity vs. flag_sold
boxplot(data_lgd$months_to_maturity~data_lgd$flag_sold,
horizontal=T, frame=F, col='light blue',
main='Months to Maturity (Sold=1 vs. Non-Sold=0)')
# Create train and test samples
data_lgd$flag_sold_1<- dplyr::if_else(data_lgd$flag_sold == 1,0,1)
library(caret)
set.seed(2122)
train_index <- caret::createDataPartition(data_lgd$flag_sold_1, p = .7,
list = FALSE)
train <- data_lgd[ train_index,]
test <- data_lgd[-train_index,]
# Binning analysis
#IV analysis
library(smbinning)
iv_analysis<- smbinning.sumiv(df=train,y='flag_sold_1')

library(MASS)
attach(train)
logit_full<- glm(flag_sold~ woe_months_to_maturity+woe_ltv_utd+
woe_tob, family = binomial(link = 'logit'), data = train)
logit_stepwise<- stepAIC(logit_full,
k= qchisq(0.05, 1, lower.tail=F), direction = 'both')
detach(train)
#Inspect results:
summary(logit_stepwise)
#Predict and compute Gini index
#Predict
predict_logit_train<- predict(logit_stepwise,
newdata = train, type = 'response')
train$predict_logit <- predict(logit_stepwise,
newdata = train, type = 'response')
#Gini index calculation
library(optiRum)
gini_train<- optiRum::giniCoef(train$predict_logit,
train$flag_sold)


# TOBIT LGD MODELING
```

```
library(AER)

# 1. Upload data
data_lgd<- read_excel("data_lgd_year.xlsx")

# Overview of the database structure
library(dplyr)
dplyr::glimpse(data_lgd)

# Histogram of the variable: ltv_utd
data_lgd$lossrate<-data_lgd$shortfall/
  data_lgd$balance_at_default
hist(data_lgd$lossrate)

# Create train and test samples
data_lgd$flag_sold_1<-
  dplyr::if_else(data_lgd$flag_sold == 1,0,1)

library(caret)
set.seed(2122)
train_index <- caret::createDataPartition(data_lgd$flag_sold_1,
                                          p = .7, list = FALSE)
trainlgd <- data_lgd[ train_index,]
testlgd <- data_lgd[-train_index,]

# Fit tobit regression
fit_tobit <- tobit(lossrate ~ ltv_utd, data = trainlgd)
summary(fit_tobit)

<------ LGD BETA REGRESSION ----->

library(betareg)
train <- trainlgd %>%
  dplyr::mutate(lossrate_new= ifelse(lossrate==1,0.9999,
                                     no=ifelse(lossrate==0,0.0001,lossrate)))
fit_beta <- betareg( lossrate_new ~ ltv_utd+tob+region,
                     data = trainlgd)
summary(fit_beta)

# <---------------------EAD Modeling--------------------->

#Upload data
ead <- read_excel("ead.xlsx")
library(dplyr)

#Train and test samples
set.seed(1234) #set seed in order to reproduce sampling results

train_sample <- caret::createDataPartition(ead$year,
p=0.7, list=FALSE)
train_ead <- ead[train_sample, ]
test_ead <- ead[-train_sample, ]
# Run beta regression
library(betareg)
beta_ead <- betareg(uti_def ~ uti_ini+gdp, data=train_ead)
```

CCF Modeling

```
library(VGAM)
tobit_ead <- vglm(ccf_ratio ~ uti_ini+gdp,
tobit(Lower=0, Upper=25, type.f = 'cens'), data=train_ead)
```