Predictive Analytics: Application in the Credit Risk Domain Case Study Teaching (CST)-Vignette in cheat sheet style ("group project cover sheet")

Bastigkeit Moritz, Ennser Valentin, Grabherr Elias, Nafees Muhammad Talha

Nov. 27, 2025 (Scorecard_Intro_2511.Rmd)

Contents

Table 1: Exemplary View of the Data Table

creditability	status.of.existing.checking.account	duration.in.month
good	... < 0 DM	6
bad	0 <= ... < 200 DM	48
good	no checking account	12
good	... < 0 DM	42
bad	... < 0 DM	24
good	no checking account	36

1 Data Loading and Preparation

1.1 Loading libraries

```
library(scorecard)
library(tidyverse)
library(knitr)
```

1.2 Loading external data

```
data("germancredit")
```

1.3 Data Selection

To train our models to predict the creditworthiness, the following variables were selected as they capture key aspects of an applicant's financial situation.

The five predictor variables are:

(status.of.existing.checking.account) (categorical) This is a categorical variable that describes the applicant's checking account condition using qualitative labels such as "no checking account", "<0 DM", etc.

(duration.in.month) (numeric) This is a numeric variable indicating the length of the loan contract in months, reflecting how long the applicant will take to repay off the credit.

(credit.history) (categorical) This is another categorical variable that describes the applicant's past repayment behaviour, ranging from values of "no credits taken" and "all credits in this bank paid back duly" to "critical account".

(savings.account.and.bonds) (categorical) This is another variable that categorises the applicant's savings level, both in their savings account and bonds. There are several categories such as "unknown/no savings account" to " < 100 DM" to separate those into groups.

(purpose) (categorical) Purpose reflects a categorical variable that gives insight into what the applicant intends to do with the credit. Values range from "used car" and "new car" to "education", etc.

The following chunk contains the code for generating Table **??**.

Table 2: Information Value of Predictor Variables

variable	info_value
status.of.existing.checking.account	0.67
duration.in.month	0.33
credit.history	0.29
savings.account.and.bonds	0.20
purpose	0.17

1.4 Information Values

In a first step, the information value of the predictor variables is regarded to get a general overview of the data quality. This information value (IV) measures how well each predictor variable can separate "good borrowers" from "bad borrowers" (in relation to creditability). In practical terms, a higher IV relates to stronger predictive power of the underlying variable, with general thresholds of being good predictors of values larger than 0.3.

1.5 Filtering and Splitting Data

In the following step, the data is filtered to exclude rows with missing values and split into train and validation set. The split chosen is at a ratio of .75 to .25. This is being done to have two independent samples for training and validating the model at a later point.

```
## v Variable filtering on 1000 rows and 5 columns in 00:00:00
## v 0 variables are removed in total
```

1.6 Specifying dummy variable for credit defaults: default.list

For being able to statistically analyze and test the results from the scorecard the default.list is established that contains the default values of the response variable

```r
default.list <- data_f.list %>%
  lapply(function(x) x$creditability)
```

Hint: The default.list is needed for calculating the population stability indes (PSI) with the function perf_psi() and the gains table with the function gains_table().

Exemplarily showing the content of the default.list

```r
default.list %>% str()
```

```
## List of 2
##  $ train   : int [1:727] 0 0 1 0 0 0 1 1 0 1 ...
##  $ validate: int [1:273] 1 0 0 1 0 1 1 0 0 1 ...
```

2 Weight-Of-Evidence (WoE)-based transformation of predictor variables

2.1 WoE-based binning of train and validate samples: bins.list

WoE-based classing, i.e. binning and grouping of predictor variables

```
bins.list <- data_f.list$train %>%
  woebin("creditability")
```

```
## v Binning on 727 rows and 6 columns in 00:00:10
```

Hint: The default binning method is method="width". Other methods are

- "frequ" that support numerical variables as well as
- "tree" and "chimerge" supporting both, i.e. numerical and categorical variables which are used in the optimal binning approach.

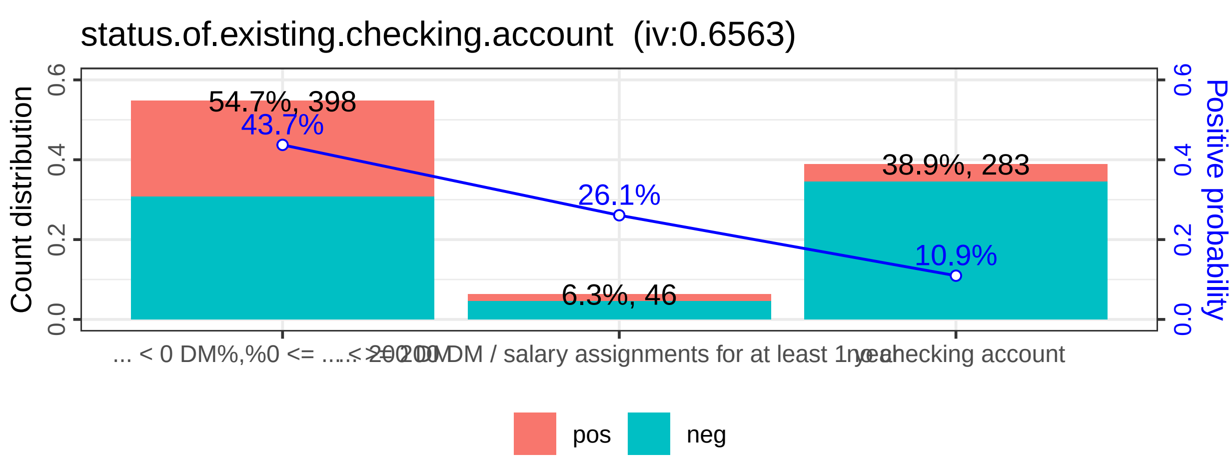
```
bins.list %>% names()
```

```
## [1] "status.of.existing.checking.account" "duration.in.month"
## [3] "credit.history"                      "savings.account.and.bonds"
## [5] "purpose"
```

Plotting the bins (including bin statistics)

```
bins.list$status.of.existing.checking.account %>%
  woebin_plot()
```

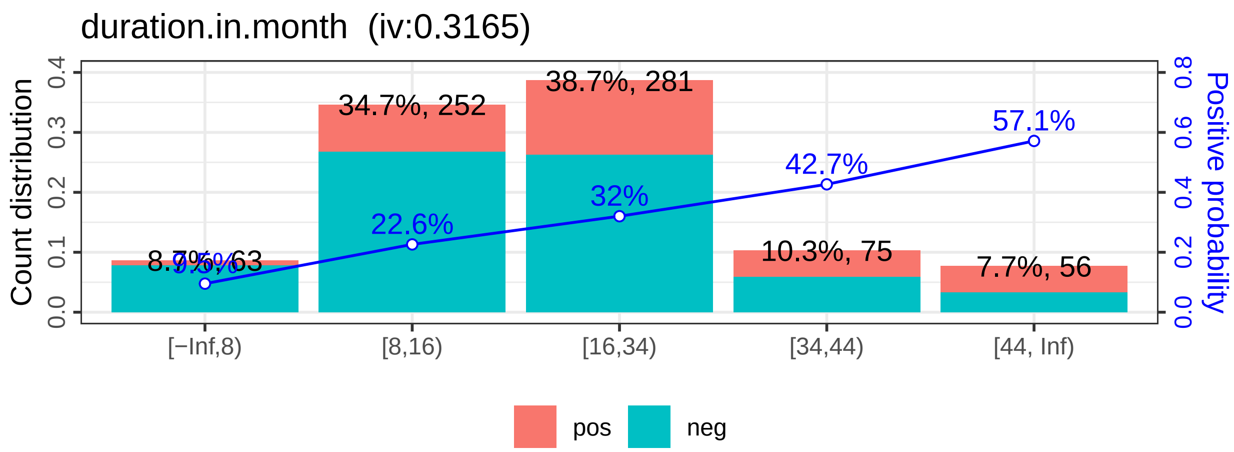
```
## $status.of.existing.checking.account
```



Hint: credit.amount does not have an acceptable structure of the default rates (positive probability) over the bins like e.g. a linear or u-curve structure; hence it should not be included in the scorecard model!

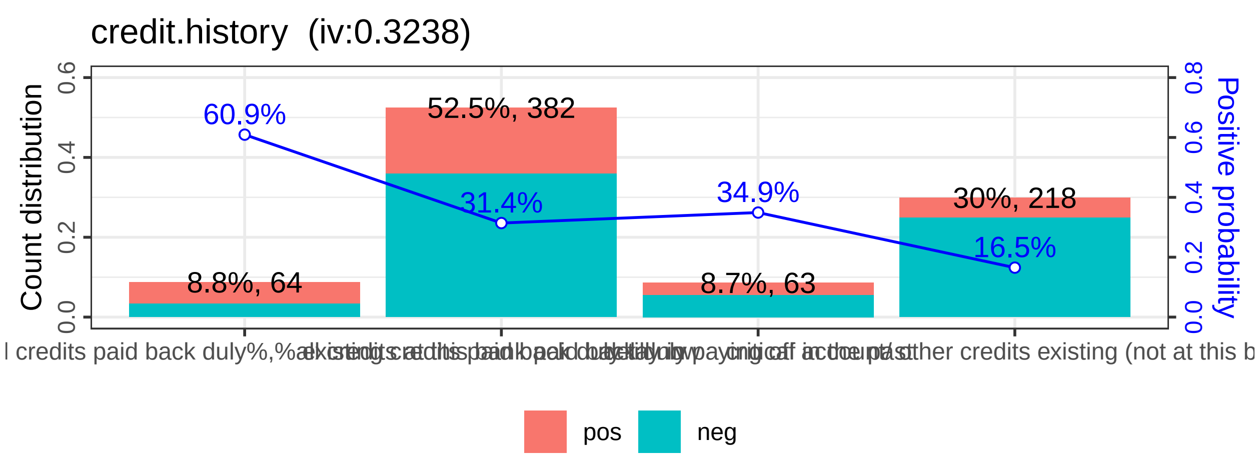
```r
bins.list$duration.in.month %>%
  woebin_plot()
```

```
## $duration.in.month
```



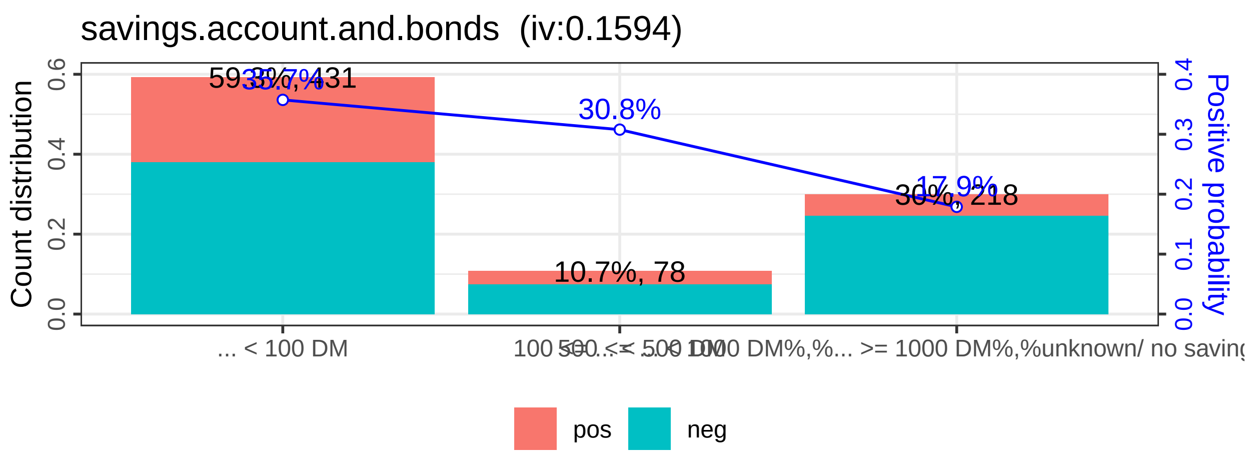

```r
bins.list$credit.history %>%
  woebin_plot()
```

```
## $credit.history
```



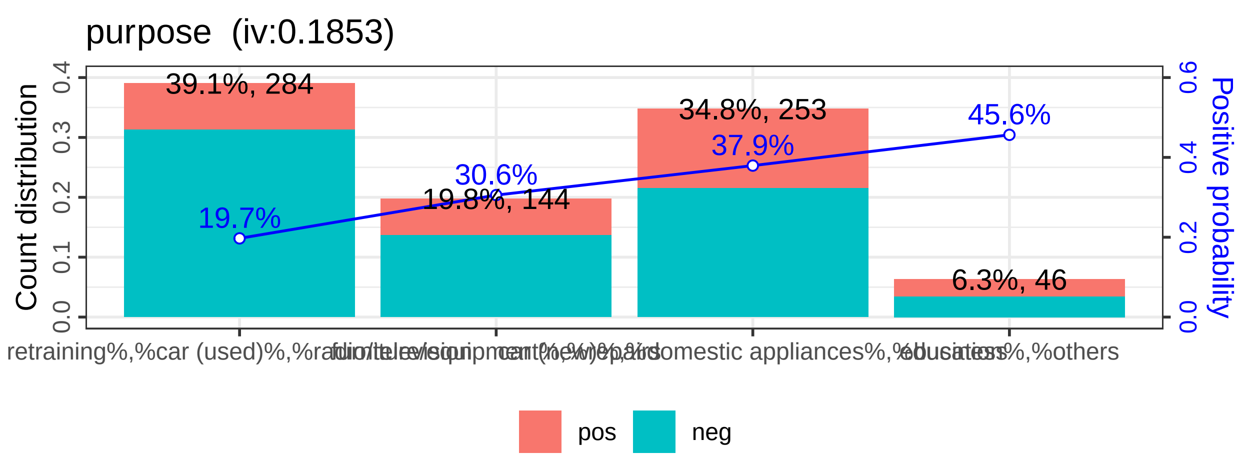

```r
bins.list$savings.account.and.bonds %>%
  woebin_plot()
```

```
## $savings.account.and.bonds
```




```r
bins.list$purpose %>%
  woebin_plot()
```

```
## $purpose
```



Hint: duration.in.month has lineare structure of the default rates; hence it should be included in the scorecard model!

Excursion: Manual bin-adjustments

Bins can be altered manually by

1. Saving the bin list generated in the woebin() function via e.g. save_as="breaks2410.list"
2. Loading the saved R-file "breaks2410.list.R", editing the breaks as needed and storing the file
3. Sourcing the edited and stored "breaks2410.list.R" file with the "source(…)$value" function
4. Binning the data again with the "woebin()" function with the additional argument "break_list"

ad 1)

```r
bins.list <- data_f.list$train %>%
  woebin("creditability",
         save_as = "breaks2410.list")
```

ad 3)

```r
breaksList <- source("breaks2410.list.R")$value
```

ad 4)

```r
bins.list <- data_f.list$train %>%
  woebin("creditability",
         breaks_list = "breaksList")
```

Hint: The above code chunks are not yet evaluated, as they are performed only when the original binning does not deliver beneficial results.

2.2 WoE-based transforming of predictor variables: data_woe.list

2.2.1 WoE-based transforming of train and validate data: data_woe.list

Transforming splitted sample: Needed for train/validate analysis

```r
data_woe.list <- data_f.list %>%
  lapply(function(x) woebin_ply(x, bins.list))
```

```
## v Woe transformating on 727 rows and 5 columns in 00:00:10
```

```
## v Woe transformating on 273 rows and 5 columns in 00:00:10
```

```r
data_woe.list %>% lapply(class)
```

```
## $train
## [1] "data.table" "data.frame"
##
## $validate
## [1] "data.table" "data.frame"
```

```r
data_woe.list %>% lapply(dim)
```

```
## $train
## [1] 727   6
##
## $validate
## [1] 273   6
```

```r
#data_woe.list$train %>%
#  select(creditability, credit.amount_woe, duration.in.month_woe) %>%
#  head()
```

3 Generalized linear model (glm): Regressing predictors against responses

3.1 Logistic regression of WoE-transformed predictors: glm(.,data_woe.list$train)

The WoE-based logistic regression is the preferred regression approach as it delivers the most compact regression models.

3.1.1 Constructing and calibrating the WoE-based logistic regression model

```
data_woe.glm <- glm(creditability ~ .,
                    family = binomial(),
                    data = data_woe.list$train)
```

3.1.2 Investigating the fitted regression model

```
data_woe.glm$aic
```

```
## [1] 711.2915
```

Summary of regression: summary()

```
data_woe.glm %>% summary()
```

```
##
## Call:
## glm(formula = creditability ~ ., family = binomial(), data = data_woe.list$train)
##
## Coefficients:
##                                        Estimate Std. Error z value Pr(>|z|)
## (Intercept)                            -0.84422    0.09532  -8.857  < 2e-16
## status.of.existing.checking.account_woe 0.83286    0.12266   6.790 1.12e-11
## duration.in.month_woe                   0.96608    0.17657   5.471 4.46e-08
## credit.history_woe                      0.78853    0.16945   4.653 3.26e-06
## savings.account.and.bonds_woe           0.86020    0.24723   3.479 0.000503
## purpose_woe                             0.94490    0.22095   4.277 1.90e-05
##
## (Intercept)                            ***
## status.of.existing.checking.account_woe ***
## duration.in.month_woe                   ***
## credit.history_woe                      ***
## savings.account.and.bonds_woe           ***
## purpose_woe                             ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 886.32  on 726  degrees of freedom
## Residual deviance: 699.29  on 721  degrees of freedom
## AIC: 711.29
##
## Number of Fisher Scoring iterations: 5
```

3.2 Logistic regression of original predictors: glm(.,data_f.list$train)

```
#data_f.glm <- glm(creditability ~ .,
 #                 family = binomial(),
  #                data = data_f.list$train %>%
   #                  select(creditability,
     #                       credit.amount,
    #                        duration.in.month,
   #                         credit.history))
```

Hint: For simplicity only three original predictors are included in the logistic regression model

```
#data_f.glm$aic
```

```
#data_f.glm$xlevels
```

For getting a compact summary the function summary() is customized and formatted

```
formatSummary <- function(model_summary) {
  aux_coeff <- model_summary$coefficients[, 1]
  aux_prob <- model_summary$coefficients[, 4]
  aux_stars <- symnum(aux_prob,
                      corr = FALSE,
                      na = FALSE,
                      cutpoints = c(0, 0.001, 0.01, 0.05, 0.1, 1),
                      symbols = c("***", "**", "*", ".", " "))
  names(aux_coeff) <- str_trunc(names(aux_coeff),
                               width = 40)
  aux_result <- data.frame(Estimate = aux_coeff,
                          Prob_z = aux_prob,
                          "Stars" = aux_stars)
  return(aux_result)
}
```

```
#summary(data_f.glm) %>% formatSummary()
```

4 Building scorecard-models (scm) and calculating scorepoints

Scorepoints are calculated by combing scorecard-model, which combines bin and glm information, with individual data

4.1 Building scm-models: Combining bins.list & data_woe.glm in scorecard()

Building the scorecard via bin and glm information resulting from train sample

```
scorecard.scm <- bins.list %>% scorecard(data_woe.glm)
```

```
scorecard.scm %>% names()
```

```
## [1] "basepoints"                          "status.of.existing.checking.account"
## [3] "duration.in.month"                   "credit.history"
## [5] "savings.account.and.bonds"           "purpose"
```

Investigating the content of the “woe-based” scorecard model

```
scorecard.scm$basepoints
```

```
##      variable    bin    woe points
##        <char> <lgcl> <lgcl>  <num>
## 1: basepoints     NA     NA    449
```

```
scorecard.scm$duration.in.month[,1:8]
```

```
##             variable       bin count count_distr   neg   pos    posprob
##               <char>    <char> <int>       <num> <int> <int>      <num>
## 1: duration.in.month  [-Inf,8)    63  0.08665750    57     6 0.0952381
## 2: duration.in.month    [8,16)   252  0.34662999   195    57 0.2261905
## 3: duration.in.month   [16,34)   281  0.38651994   191    90 0.3202847
## 4: duration.in.month   [34,44)    75  0.10316369    43    32 0.4266667
## 5: duration.in.month [44, Inf)    56  0.07702889    24    32 0.5714286
##           woe
##         <num>
## 1: -1.3967784
## 2: -0.3754349
## 3:  0.1020496
## 4:  0.5590492
## 5:  1.1421954
```

```
scorecard.scm$duration.in.month[,c(1,9:13)]
```

```
##             variable      bin_iv  total_iv breaks is_special_values points
##               <char>       <num>     <num> <char>            <lgcl>  <num>
## 1: duration.in.month 0.117489928 0.3165171      8             FALSE     97
## 2: duration.in.month 0.044932100 0.3165171     16             FALSE     26
## 3: duration.in.month 0.004106144 0.3165171     34             FALSE     -7
## 4: duration.in.month 0.035304912 0.3165171     44             FALSE    -39
## 5: duration.in.month 0.114683977 0.3165171    Inf             FALSE    -80
```

4.2 Calculating scorepoints: Combinig individual data.df & scm-model in scorecard_ply()

Generating a score list

```
score.list <- data_f.list %>%
  lapply(function(x) scorecard_ply(x, scorecard.scm))
```

Hint: The only_total_score=TRUE (= default argument) has to be used for providing two compatible lists for further processing. If scores to the different predictors are of interest, the two separate, i.e. train and validate samples have to analyzed individually with the argument only_total_score=FALSE.

```
score.list %>% names()
```

```
## [1] "train"    "validate"
```

```
score.list$train %>%
  head()
```

```
##    score
##    <num>
## 1:   633
## 2:   351
## 3:   351
## 4:   477
## 5:   553
## 6:   625
```

```
score.list$validate %>%
  head()
```

```
##    score
##    <num>
## 1:   349
## 2:   531
## 3:   390
## 4:   287
## 5:   455
## 6:   408
```

5 WoE-based predicting (forecasting) of probabilities and scorepoints

5.1 Predicting probabilities: Combining data_woe.list & data_woe.glm in predict()

```
predProb.list <- data_woe.list %>%
  lapply(function(x) predict(data_woe.glm,
                             type = 'response',
                             x))
```

Hint: Due to the fact that the data_woe.glm was calibrated for the train sample two different types of prediction can be destinguished, i.e. the in-sample (IS) prediction by using the train sample in the predict()-function, and the out-of-sample (OoS) prediction by using the test sample in the predict()-function.

```
predProb.list %>% names()
```

```
## [1] "train"    "validate"
```

```
predProb.list$train %>% head() # In-Sample prediction
```

```
##          1          2          3          4          5          6
## 0.03257030 0.62644835 0.62511261 0.22923725 0.09405324 0.03635829
```

```
predProb.list$validate %>% head() # Out-of-Sample prediction
```

```
##         1         2         3         4         5         6
## 0.6293632 0.1222070 0.4915309 0.8008039 0.2815743 0.4315283
```

5.2 Predicting scorepoints: Retrieving predictions from score.list generated in scorecard_ply()

The prediction of the scorepoints is alread incorported in the built scorecard.

```
score.list$train %>%
  head()
```

```
##    score
##    <num>
## 1:   633
## 2:   351
## 3:   351
## 4:   477
## 5:   553
## 6:   625
```

```r
score.list$validate %>%
  head()
```

```
##    score
##    <num>
## 1:   349
## 2:   531
## 3:   390
## 4:   287
## 5:   455
## 6:   408
```

6 Scorecard Validation: Statistical testing of forecasting accuracy

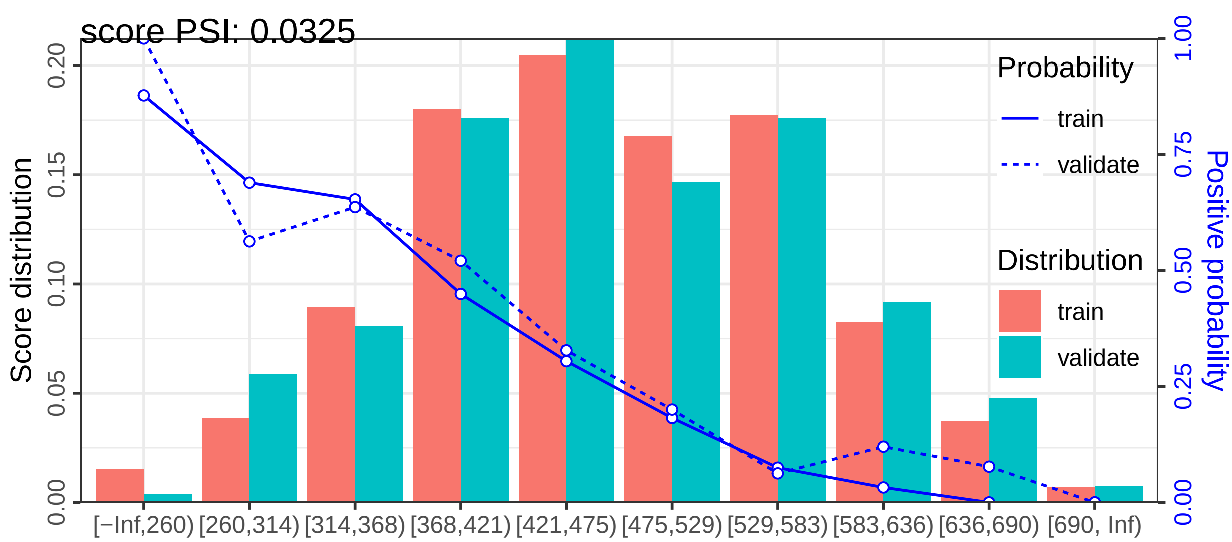
6.1 Checking stability of score and probility distributions: Population Stability Index (PSI)

```
psi.list <- perf_psi(score = score.list,
                     label = default.list,
                     return_distr_dat = TRUE)
```

Hint: More details of per_psi() function are given @ https://www.rdocumentation.org/packages/scorecard/versions/0.1.9/topics/perf_psi

```
psi.list$pic
```

```
## $score
```




```
psi.list %>% names()
```

```
## [1] "pic" "psi" "dat"
```

```
psi.list$dat %>% names()
```

```
## [1] "score"
```

```
psi.list$dat$score[,1:9] %>% head()
```

```
## Key: <datset>
##     datset        bin count cum_count   neg   pos cum_neg cum_pos count_distr
##     <fctr>     <fctr> <int>     <int> <int> <int>   <int>   <int>       <num>
## 1:   train [-Inf,260)    11        11     1    10       1      10      0.0151
```

```
## 2:  train  [260,314)   28        39     8    20       9      30       0.0385
## 3:  train  [314,368)   65       104    21    44      30      74       0.0894
## 4:  train  [368,421)  131       235    70    61     100     135       0.1802
## 5:  train  [421,475)  149       384   102    47     202     182       0.2050
## 6:  train  [475,529)  122       506    99    23     301     205       0.1678
```

```
psi.list$psi
```

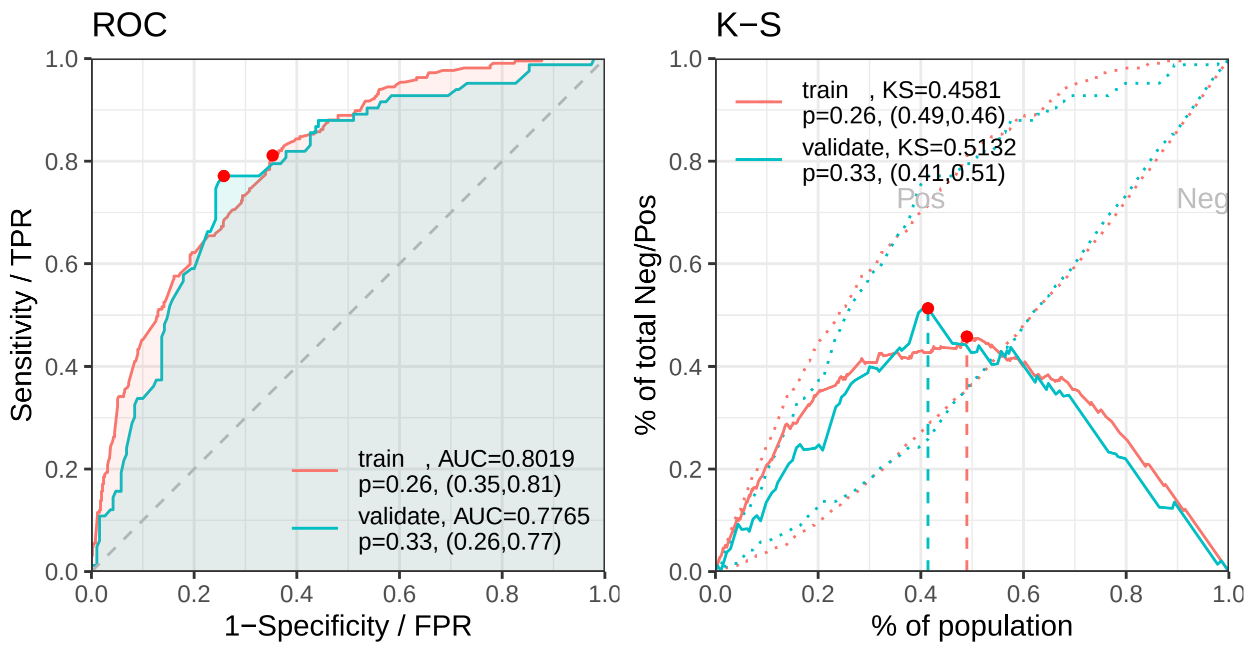
```
##    variable        dataset        psi
##      <char>         <char>      <num>
## 1:    score train_validate 0.03247355
```

```
perf_psi(score, label = NULL, title = NULL, x_limits = NULL,
  x_tick_break = 50, show_plot = TRUE, seed = 186,
  return_distr_dat = FALSE)
# e.g. # x_limits = c(250, 700),
#      # x_tick_break = 50,
```

6.2 IS & OoS testing probability prediction accuracy: perf_eva(.,predProb.list)

probability prediction accuracy

```
ProbPredAccuracy <- perf_eva(pred = predProb.list,
                             label = default.list,
                             binomial_metric = c("rmse","auc","gini"),
                             show_plot=c("roc","ks"),
                             confusion_matrix = TRUE)
```




```r
names(ProbPredAccuracy)
```

```
## [1] "binomial_metric"  "confusion_matrix" "pic"
```

```r
ProbPredAccuracy$binomial_metric
```

```
## $train
##          RMSE       AUC      Gini
##         <num>     <num>     <num>
## 1: 0.3998542 0.8018569 0.6037137
##
## $validate
##          RMSE      AUC     Gini
##         <num>    <num>    <num>
## 1: 0.4150501 0.776506 0.553012
```

```r
ProbPredAccuracy$confusion_matrix
```

```
## $train
##     label pred_0 pred_1     error
##    <char>  <num>  <num>     <num>
## 1:      0    330    180 0.3529412
## 2:      1     41    176 0.1889401
## 3:  total    371    356 0.3039890
```

```
##
## $validate
##     label pred_0 pred_1     error
##    <char>  <num>  <num>     <num>
## 1:      0    122     68 0.3578947
## 2:      1     17     66 0.2048193
## 3:  total    139    134 0.3113553
```

Excursion

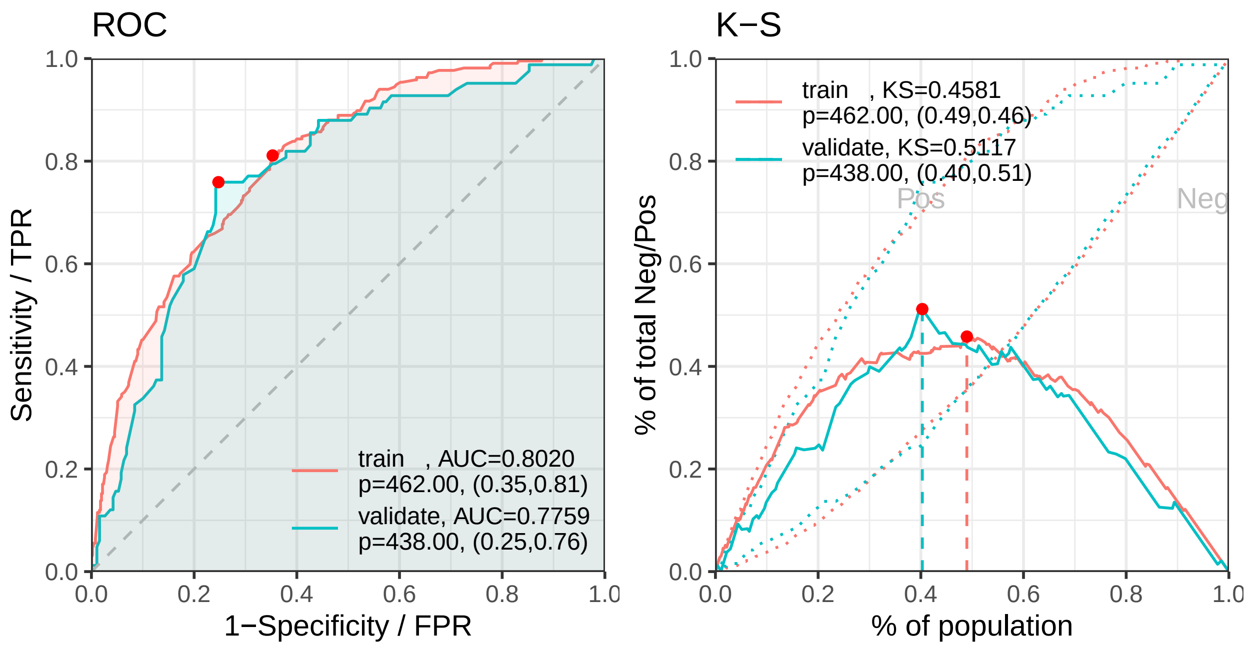
```r
perf_eva(pred = predProb.list,
         label = default.list,
         binomial_metric = c("rmse","auc","gini"),
         show_plot= FALSE,
         confusion_matrix = FALSE)
```

```
## $binomial_metric
## $binomial_metric$train
##          RMSE       AUC      Gini
##         <num>     <num>     <num>
## 1: 0.3998542 0.8018569 0.6037137
##
## $binomial_metric$validate
##          RMSE      AUC     Gini
##         <num>    <num>    <num>
## 1: 0.4150501 0.776506 0.553012
```

6.3 IS & OoS testing scorepoint prediction accuracy: perf_eva(.,score.list)

scorepoint prediction accuracy

```r
ScorePredAccuracy <- perf_eva(pred = score.list,
                              label = default.list,
                              binomial_metric = c("rmse","auc","gini"),
                              show_plot=c("roc","ks"),
                              confusion_matrix = TRUE)
```




```r
names(ScorePredAccuracy)
```

```
## [1] "binomial_metric"  "confusion_matrix" "pic"
```

```r
ScorePredAccuracy$binomial_metric
```

```
## $train
##          AUC      Gini
##        <num>     <num>
## 1: 0.8019563 0.6039125
##
## $validate
##          AUC      Gini
##        <num>     <num>
## 1: 0.7759036 0.5518072
```

```r
ScorePredAccuracy$confusion_matrix
```

```
## $train
##     label pred_0 pred_1     error
##    <char>  <num>  <num>     <num>
## 1:      0    330    180 0.3529412
## 2:      1     41    176 0.1889401
## 3:  total    371    356 0.3039890
```

```
##
## $validate
##     label pred_0 pred_1     error
##    <char>  <num>  <num>     <num>
## 1:      0    122     68 0.3578947
## 2:      1     17     66 0.2048193
## 3:  total    139    134 0.3113553
```

7 Appendix

7.1 Appendix: Essay style with formulas in LaTeX language

Group project assignment: Write a scholarly essay with full sentences, correct citations and LaTeX formulas.

Example essay style: From a statistical perspective the transition from the MPS to the VaR framework is related to switching the perspective from considering moments (parameters) of random variables, i.e. μ and σ, to considering the quantiles and corresponding probabilities of these variables. Specifically, the VaR measure specifies the risk of a random variable ($\tilde{P}$) via the threshold quantile (VaR) that is exceeded into the negative direction (i.e. $P \leq VaR$) with the loss probability (α) or respectively, is exceeded into the positive direction (i.e. $P > VaR$) with the complementary probability, i.e. the confidence level $(1 - \alpha)$.

7.2 Appendix: Generating tables, figures, cross references and citations

```
data.df[1:100,2:3] %>% plot()
```

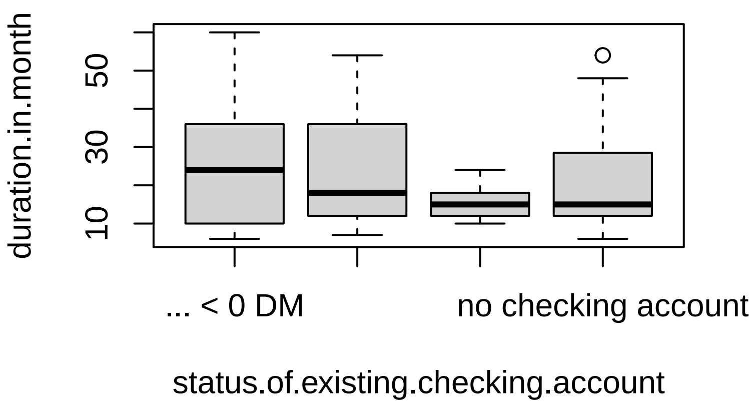


Figure 1: Amount vs. Duration

Figure **??** is a sample figure where the credit.amount is scatter plotted against the duration.in.month.

Formulas without numbering

$$\Pr\{\tilde{P} \leq VaR\} = \alpha$$

Formulas with numbering (and labeling which is needed for referencing)

$$\Pr\{\tilde{P} \leq VaR\} = \alpha \tag{1}$$

Formula (**??**) is a sample formula defining the Value at Risk.

Always cite original literature to avoid plagiarism: e.g. **?** or (**?**). Don't forget to cite page numbers as well for literal citations, e.g. (**?**, p. 100).