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

1 Abstract

Credit risk assessment is a central task in retail banking, ensuring that financial institutions can effectively discriminate between creditworthy and non-creditworthy applicants. Modern credit scoring increasingly relies on data-driven methods that combine statistical modelling, machine learning techniques and domain-specific transformations to produce interpretable and stable credit-risk predictions.
This project applies a **Weight-of-Evidence (WoE)** and **Information Value (IV)**–driven scorecard modelling framework using the *German Credit* dataset. The workflow includes data preparation, IV-based feature assessment, supervised binning, WoE transformation, logistic regression modelling, scorecard generation and out-of-sample validation. Performance is evaluated through stability measures such as the **Population Stability Index (PSI)** and accuracy metrics including AUC, Gini and RMSE. The resulting scorecard provides a transparent, regulator-compliant and empirically robust tool for credit-risk prediction.

2 Introduction

Credit risk modelling aims to quantify the likelihood that a borrower will fail to meet contractual repayment obligations. As banks, insurers and financial intermediaries increasingly rely on data-driven decision frameworks, scorecards have become the industry standard for credit underwriting due to their transparency, interpretability and regulatory acceptance (Hand and Henley, 1997; Thomas et al., 2002). In contrast to purely algorithmic black-box models, scorecards preserve a clear link between economic reasoning, data transformations and model parameters—an aspect that is essential for auditability and explainability in regulated environments.

The aim of this project is to build and validate a **predictive scorecard model** based on the *German Credit* dataset. To achieve this, the project adopts a structured analytical workflow aligned with established credit-risk modelling guidelines (Anderson, 2007; Siddiqi, 2017). The key methodological components are:

1. **Data Filtering and Variable Selection** – Choosing a subset of variables that capture behavioural and financial characteristics of borrowers.
2. **Information Value Analysis (IV)** – Quantifying the predictive strength of candidate variables.
3. **Supervised Binning and Weight-of-Evidence Transformation (WoE)** – Creating monotonic predictor–default relationships and ensuring stable logistic-regression estimation.
4. **Logistic Regression Modelling** – Fitting a parsimonious and interpretable model linking WoE-transformed predictors to the probability of default.
5. **Scorecard Generation** – Translating the regression coefficients and bin structures into a practical scorecard with additive scorepoints.
6. **Model Validation** – Assessing predictive accuracy (AUC, Gini, RMSE), calibration, and population stability (PSI) for both in-sample (IS) and out-of-sample (OoS) samples.

Using WoE transformations is beneficial because it enforces monotonicity, reduces the influence of outliers and yields logistic models with minimal multicollinearity (Siddiqi, 2017). This ensures that the final scorecard is both empirically robust.

Overall, this project demonstrates how predictive analytics can be applied to credit-risk modelling to produce a validated scorecard. The methodological steps should lead to a replicable blueprint for building credit-risk models.

3 Data Loading and Preparation

3.1 Loading libraries

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

3.2 Loading external data

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

3.3 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.

3.4 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.

Table 1: 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

3.5 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 .65 to .35. 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
```

3.6 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

```
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

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

```
## List of 2
##  $ train   : int [1:635] 0 0 1 0 0 0 1 1 0 0 ...
##  $ validate: int [1:365] 1 0 0 1 0 1 1 1 0 0 ...
```

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

4.1 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 635 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.

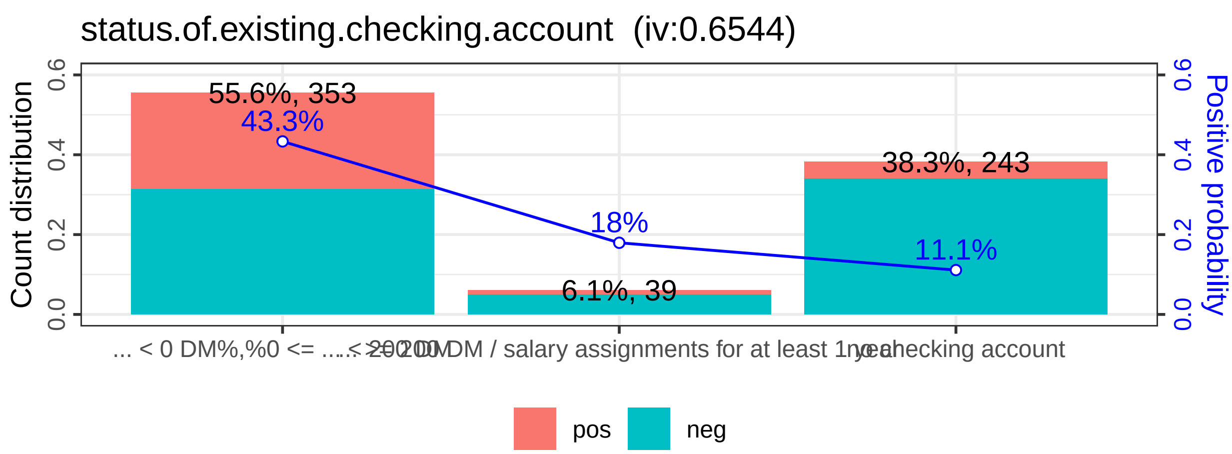
```
bins.list %>% names()
```

```
## [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()
```

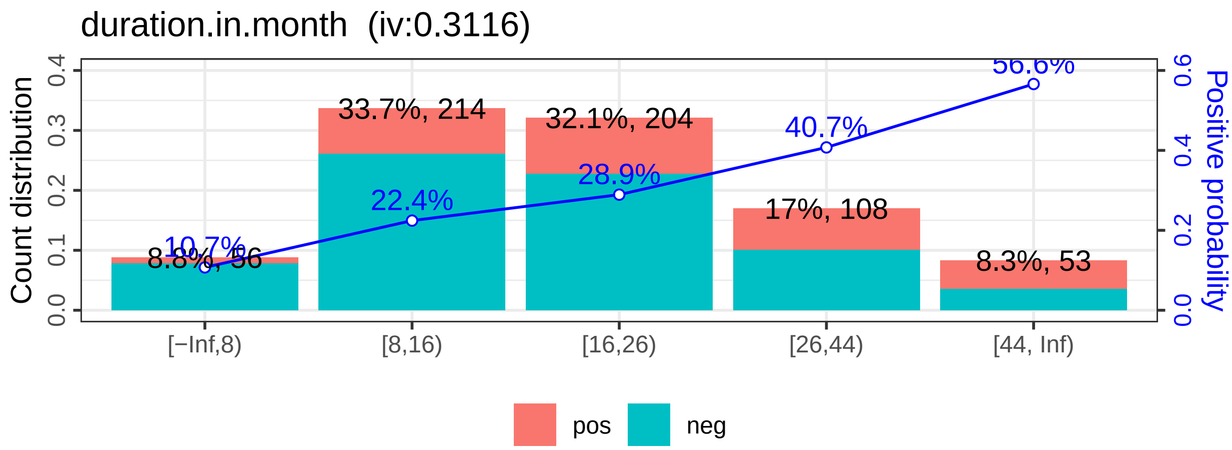
```
## $status.of.existing.checking.account
```



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!

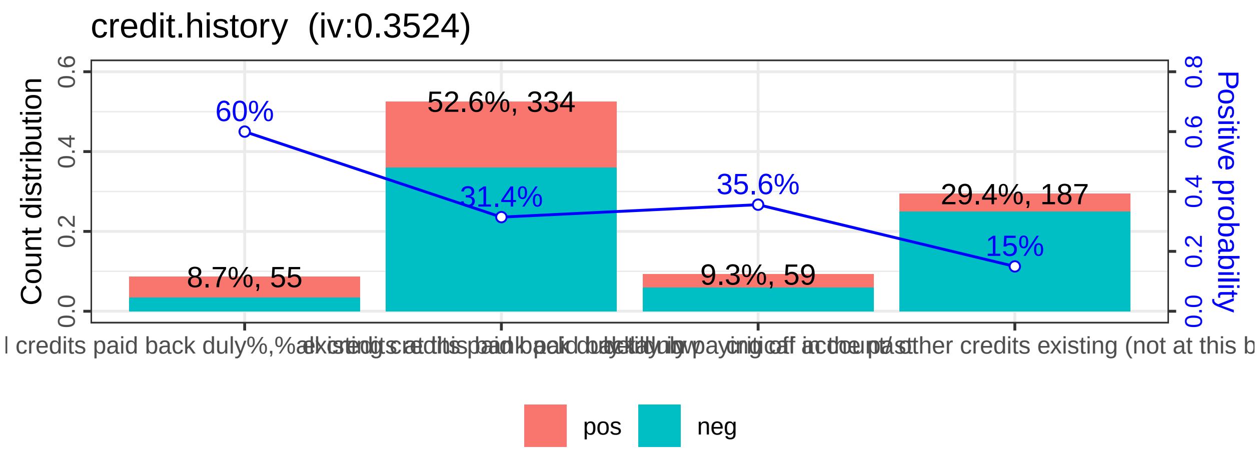
```
bins.list$duration.in.month %>%
  woebin_plot()
```

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



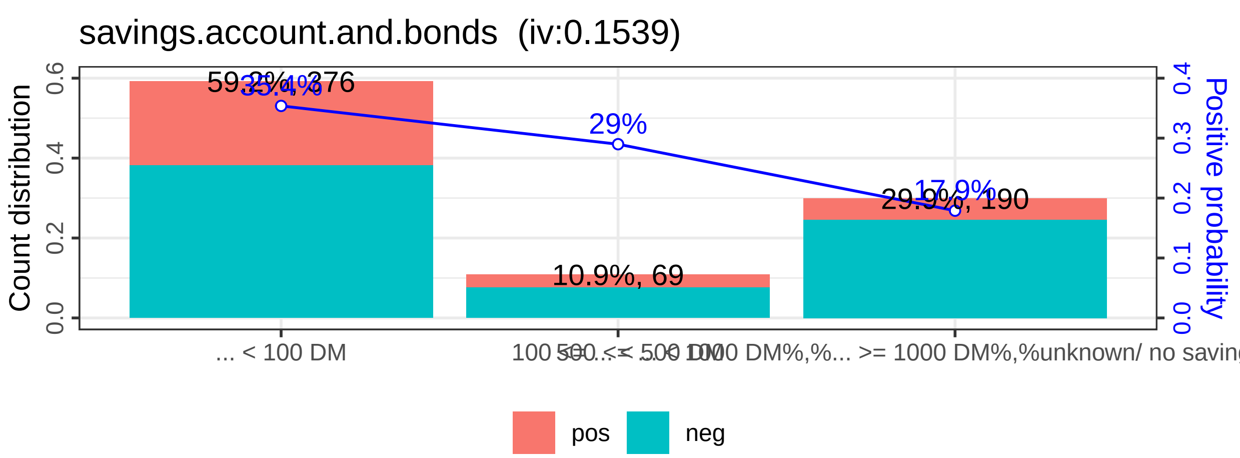

```
bins.list$credit.history %>%
  woebin_plot()
```

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



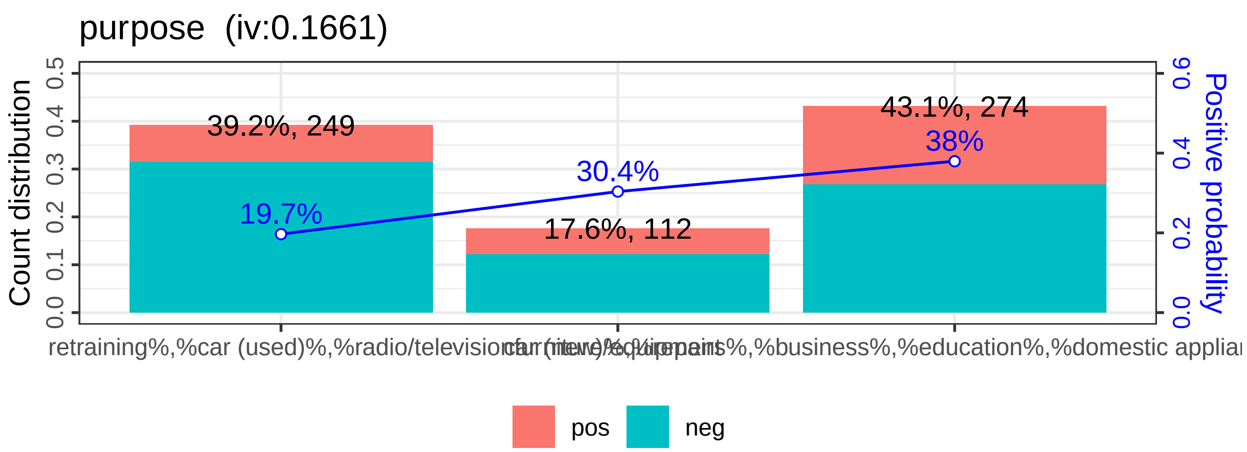

```
bins.list$savings.account.and.bonds %>%
  woebin_plot()
```

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




```
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)

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

ad 3)

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

ad 4)

```
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.

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

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

Transforming splitted sample: Needed for train/validate analysis

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

```
## v Woe transforming on 635 rows and 5 columns in 00:00:10
## v Woe transforming on 365 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] 635   6
##
## $validate
## [1] 365   6
```

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

```
## v Woe transformating on 635 rows and 5 columns in 00:00:00
## v Woe transformating on 365 rows and 5 columns in 00:00:00
```

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

5.1 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.

5.1.1 Constructing and calibrating the WoE-based logistic regression model

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

5.1.2 Investigating the fitted regression model

```r
data_woe.glm$aic
```

```
## [1] 619.9392
```

Summary of regression: summary()

```r
summary(data_woe.glm)
```

```
##
## Call:
## glm(formula = creditability ~ ., family = binomial(), data = data_woe.list$train)
##
## Coefficients:
##                                        Estimate Std. Error z value Pr(>|z|)
## (Intercept)                             -0.8628     0.1023  -8.435  < 2e-16
## status.of.existing.checking.account_woe   0.8195     0.1310   6.254 3.99e-10
## duration.in.month_woe                     0.9772     0.1897   5.152 2.58e-07
## credit.history_woe                        0.7643     0.1754   4.357 1.32e-05
## savings.account.and.bonds_woe             0.8944     0.2697   3.316 0.000912
## purpose_woe                               0.9840     0.2506   3.927 8.59e-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: 769.77  on 634  degrees of freedom
## Residual deviance: 607.94  on 629  degrees of freedom
## AIC: 619.94
##
## Number of Fisher Scoring iterations: 5
```

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

```
data_f.glm <- glm(
  creditability ~ status.of.existing.checking.account + duration.in.month + credit.history,
  family = binomial(),
  data = data_f.list$train
)
```

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

```
data_f.glm$aic
```

```
## [1] 656.1663
```

```
data_f.glm$xlevels
```

```
## $status.of.existing.checking.account
## [1] "... < 0 DM"
## [2] "0 <= ... < 200 DM"
## [3] "... >= 200 DM / salary assignments for at least 1 year"
## [4] "no checking account"
##
## $credit.history
## [1] "no credits taken/ all credits paid back duly"
## [2] "all credits at this bank paid back duly"
## [3] "existing credits paid back duly till now"
## [4] "delay in paying off in the past"
## [5] "critical account/ other credits existing (not at this bank)"
```

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()
```

```
##                                              Estimate       Prob_z Stars
## (Intercept)                                0.36752107 4.879382e-01
## status.of.existing.checking.account0 ... -0.36279765 1.142842e-01
## status.of.existing.checking.account..... -1.25349211 6.084361e-03    **
## status.of.existing.checking.accountno... -1.85705728 3.799756e-12   ***
## duration.in.month                          0.03353515 9.890970e-06   ***
## credit.historyall credits at this ban... -0.54363263 3.753992e-01
## credit.historyexisting credits paid b... -1.18265150 1.571509e-02     *
```

```
## credit.historydelay in paying off in ... -1.00684149 7.091344e-02     .
## credit.historycritical account/ other... -1.91787010 2.306172e-04   ***
```

6 Building scorecard-models (scm) and calculating scorepoints

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

6.1 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    450
```

```
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)    56  0.08818898    50     6 0.1071429
## 2: duration.in.month    [8,16)   214  0.33700787   166    48 0.2242991
## 3: duration.in.month   [16,26)   204  0.32125984   145    59 0.2892157
## 4: duration.in.month   [26,44)   108  0.17007874    64    44 0.4074074
## 5: duration.in.month [44, Inf)    53  0.08346457    23    30 0.5660377
##            woe
##          <num>
## 1: -1.24657892
## 2: -0.36710216
## 3: -0.02551168
## 4:  0.49899117
## 5:  1.13938778
```

```
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 0.0991299271 0.3115523      8             FALSE     88
## 2: duration.in.month 0.0417950298 0.3115523     16             FALSE     26
## 3: duration.in.month 0.0002079889 0.3115523     26             FALSE      2
## 4: duration.in.month 0.0461252338 0.3115523     44             FALSE    -35
## 5: duration.in.month 0.1242941288 0.3115523    Inf             FALSE    -80
```

6.2 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:   630
## 2:   354
## 3:   357
## 4:   496
## 5:   557
## 6:   622
```

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

```
##    score
##    <num>
## 1:   350
## 2:   551
## 3:   395
## 4:   285
## 5:   456
## 6:   420
```

7 WoE-based predicting (forecasting) of probabilities and score-points

7.1 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
## 0.03390496 0.61729222 0.60866854 0.18293009 0.08764694 0.03756245
```

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

```
##         1         2         3         4         5         6
## 0.6311053 0.0952773 0.4777985 0.8080544 0.2818604 0.3935574
```

7.2 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:   630
## 2:   354
## 3:   357
## 4:   496
## 5:   557
## 6:   622
```

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

```
##    score
##    <num>
## 1:   350
## 2:   551
## 3:   395
## 4:   285
## 5:   456
## 6:   420
```

8 Scorecard Validation: Statistical testing of forecasting accuracy

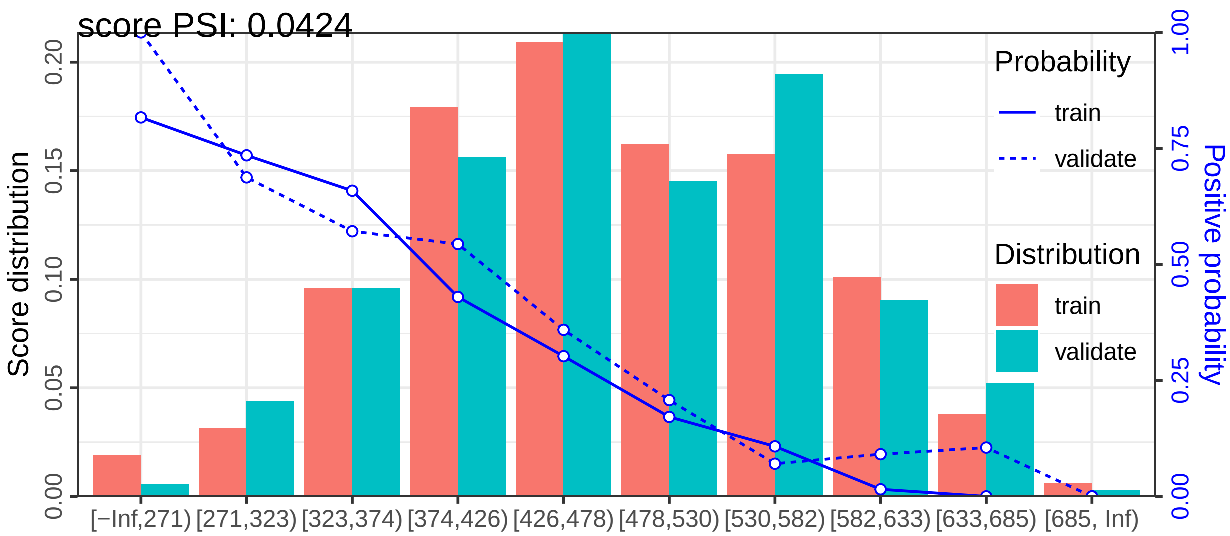
8.1 Checking stability of score and probility distributions: Population Stability Index (PSI)

```
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,271)    12        12     2    10       2      10      0.0189
## 2:  train  [271,323)    20        32     5    15       7      25      0.0315
## 3:  train  [323,374)    61        93    20    41      27      66      0.0961
## 4:  train  [374,426)   114       207    64    50      91     116      0.1795
## 5:  train  [426,478)   133       340    92    41     183     157      0.2094
## 6:  train  [478,530)   103       443    85    18     268     175      0.1622
```

```
psi.list$psi
```

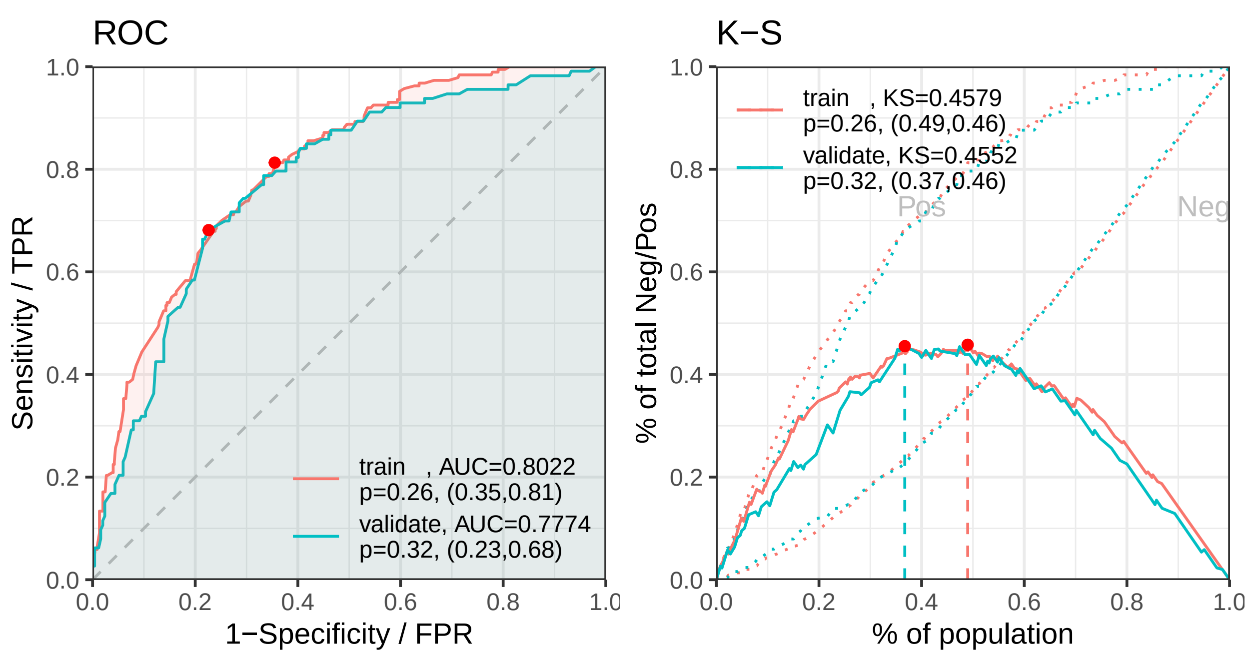
```
##    variable        dataset        psi
##      <char>         <char>      <num>
## 1:    score train_validate 0.04239616
```

```r
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,
```

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

probability prediction accuracy

```r
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: 0.3987765 0.8022465 0.6044929
##
## $validate
##         RMSE       AUC     Gini
##        <num>     <num>    <num>
## 1: 0.4153243 0.7773915 0.554783
```

```r
ProbPredAccuracy$confusion_matrix
```

```
## $train
```

```
##     label pred_0 pred_1     error
##    <char>  <num>  <num>     <num>
## 1:      0    289    159 0.3549107
## 2:      1     35    152 0.1871658
## 3:  total    324    311 0.3055118
##
## $validate
##     label pred_0 pred_1     error
##    <char>  <num>  <num>     <num>
## 1:      0    168     84 0.3333333
## 2:      1     25     88 0.2212389
## 3:  total    193    172 0.2986301
```

Excursion

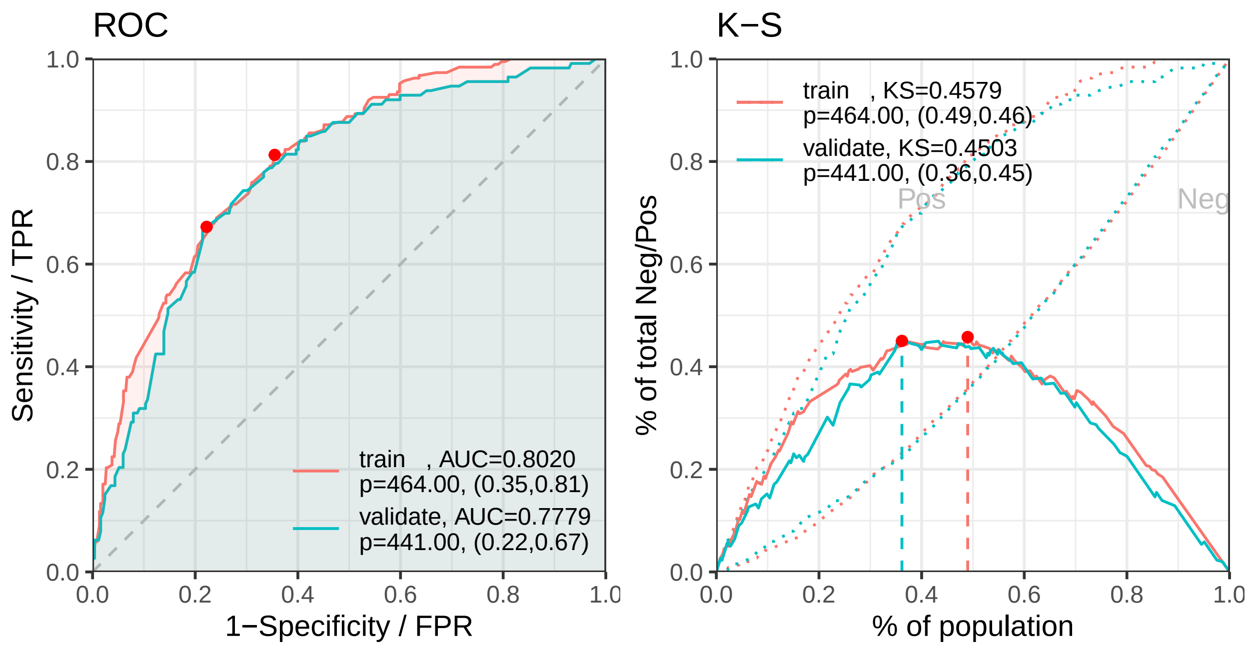
```
perf_eva(pred = predProb.list,
         label = default.list,
         binomial_metric = c("rmse","auc","gini"),
         show_plot= FALSE,
         confusion_matrix = FALSE)
```

```
## $binomial_metric
## $binomial_metric$train
##         RMSE       AUC      Gini
##        <num>     <num>     <num>
## 1: 0.3987765 0.8022465 0.6044929
##
## $binomial_metric$validate
##         RMSE       AUC     Gini
##        <num>     <num>    <num>
## 1: 0.4153243 0.7773915 0.554783
```

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

scorepoint prediction accuracy

```
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: 0.801966 0.6039319
##
## $validate
##           AUC      Gini
##         <num>     <num>
## 1: 0.7779007 0.5558014
```

```r
ScorePredAccuracy$confusion_matrix
```

```
## $train
##     label pred_0 pred_1     error
##    <char>  <num>  <num>     <num>
## 1:      0    289    159 0.3549107
## 2:      1     35    152 0.1871658
## 3:  total    324    311 0.3055118
##
## $validate
##     label pred_0 pred_1     error
##    <char>  <num>  <num>     <num>
## 1:      0    168     84 0.3333333
## 2:      1     25     88 0.2212389
## 3:  total    193    172 0.2986301
```

9 Appendix

9.1 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 $(1 - \alpha)$.

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

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

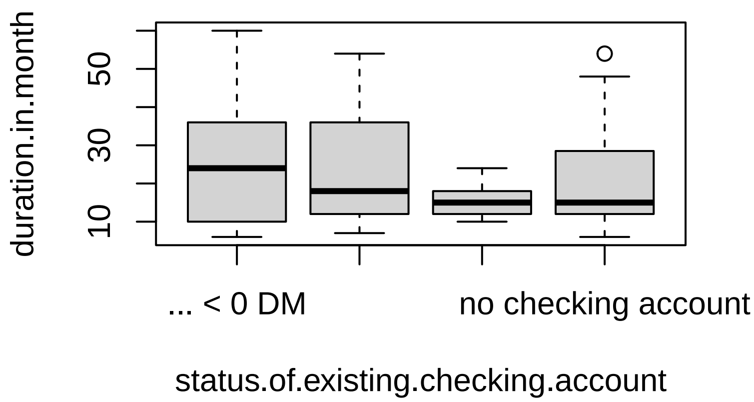


Figure 1: Amount vs. Duration

#Figure 1 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 (1) is a sample formula defining the Value at Risk.

Always cite original literature to avoid plagiarism: e.g. Schwaiger (2016) or (Schwaiger, 2016). Don't forget to cite page numbers as well for literal citations, e.g. (Schwaiger, 2016, p. 100).

References

Anderson, R. (2007). *The Credit Scoring Toolkit: Theory and Practice for Retail Credit Risk Management and Decision Automation*. Oxford University Press.

Hand, D. J. and Henley, W. E. (1997). Statistical classification methods in consumer credit scoring: A review. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 160(3):523–541.

Schwaiger, W. S. (2016). *Investition und Finanzierung*. TU-MV Media Verlag, ISBN-13: 978-3903024311.

Siddiqi, N. (2017). *Intelligent Credit Scoring: Building and Implementing Predictive Models*. Wiley, 2 edition.

Thomas, L. C., Edelman, D. B., and Crook, J. N. (2002). *Credit Scoring and Its Applications*. SIAM.