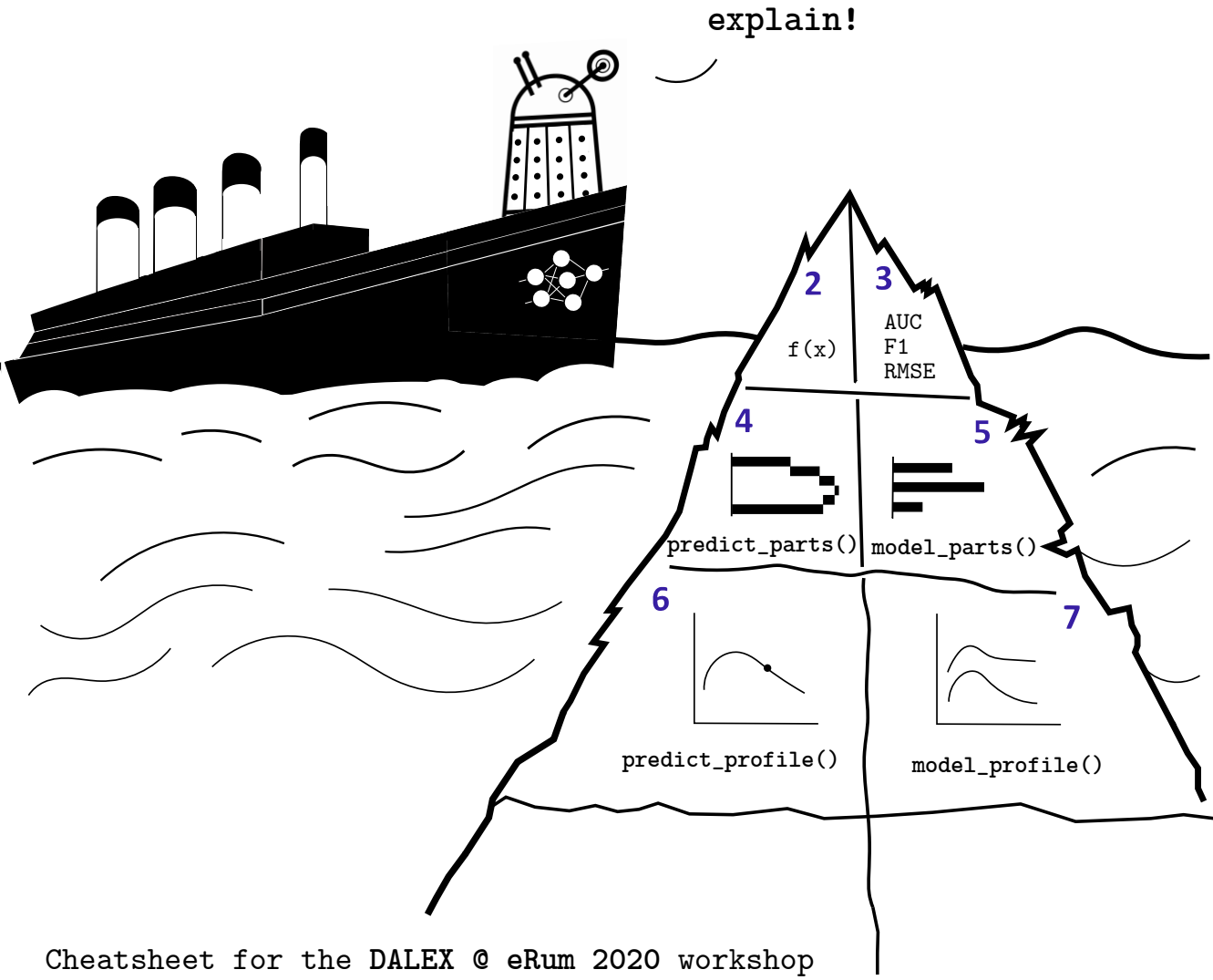
The hitchhiker guide to Explanatory Model Analysis



Cheatsheet for the **DALEX @ eRum 2020** workshop
Content: **Przemysław Biecek**
Cover: **Anna Kozak**

[1] EMA is a subfield of eXplainable Artificial Intelligence. It covers the tools and processes for analysing predictive models in order to better understand their behaviour.
[2] Instance level = local explanations - methods to explore the model from the perspective of a single prediction.
[3] Dataset level = global explanations - analysis of the overall model behaviour, the perspective of a population.

Why XAI[1]?

D Instance level explanations[2] help **to debug a model** / understand why the model was wrong.

A Dataset level explanations[3] help **to audit a model**, check if it complies with the requirements.

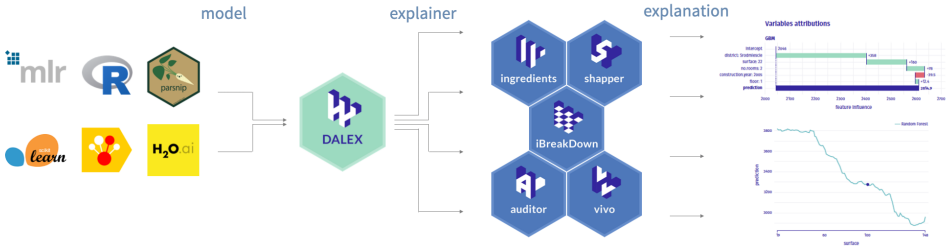
L Understand **what to do to leverage** a prediction for a model.

E Allow to confront the model with **expert knowledge**. If consistent we are more likely to **trust the prediction**.

X **Examine** models e.g. in the **champion-challenger analysis**.

DALEX architecture

```
ranger(y ~ ., data = df)  %>%  explain()  %>%  model_parts()  %>%  plot()
```



[4] Explainer is an object / adapter that wraps the model and creates an uniform structure and interface for operations.

explainer
model
data: data.frame
y: numeric
y_hat: numeric
predict_function: function (model, data)
residuals: numeric
residual_function: function(model, data, y)
weights: numeric
model_info: list(package, ver, type)
class: character
label: character

First explainer[4]

Examples are based on the `titanic_imputed` data set,

```
library("DALEX")
head(titanic_imputed, 1)
#   gender age class    embarked  fare sibsp parch survived
# 1   male  42   3rd Southampton  7.11     0     0        0
```

random forest classification model for `survived`

```
library("ranger")
model_ranger <- ranger(survived ~ ., data = titanic_imputed,
         classification = TRUE, probability = TRUE)
exp_ranger <- explain(model_ranger,
        data = titanic_imputed, y = titanic_imputed$survived)
predict(exp_ranger, titanic_imputed[1,])
## [1] 0.1032819
```

and a logistical regression models with splines.

```
library("rms")
model_rms <- lrm(survived ~ rcs(age)*gender + rcs(fare) + class,
        data = titanic_imputed)
exp_rms <- explain(model_rms,
        data = titanic_imputed,
        y = titanic_imputed$survived,
        predict_function = function(m, x) predict(m, x, type = "fitted"),
        label = "Logistic_with_splines")
exp_rms$model_info
## Package: rms  version: 5.1.4
## Task type: classification
```

How good is the model?

The evaluation of the model performance for the classification is based on different measures than for the regression.
For regression, commonly used measures are Mean squared error MSE[5] and Rooted mean squared error RMSE[6].
For classification, commonly used measures are Accuracy[7], Precision[8] and Recall[9] and F1 score[10].

[5] $MSE(f) = \frac{1}{n}\sum_i^n (f(x_i) - y_i)^2$
[6] $RMSE(f) = \sqrt{MSE(f, X, y)}$
[7] $ACC(f) = (TP + TN)/n$
[8] $Prec(f) = TP/(TP + FP)$
[9] $Recall(f) = TP/(TP + FN)$
[10] $F1(f) = 2\frac{Prec(f) * Recall(f)}{Prec(f) + Recall(f)}$

Model performance

Model exploration starts with an assessment of how good is the model. The `DALEX::model_performance` function calculates a set of the most common measures for the specified model.

```
mp_ranger <- model_performance(exp_ranger)
mp_ranger
## Measures for:  classification
## recall     : 0.5977496
## precision  : 0.9081197
## f1         : 0.72095
## accuracy   : 0.8509289
## auc        : 0.752529
```

```
mp_rms <- model_performance(exp_rms)
mp_rms
## Measures for:  classification
## recall     : 0.5977496
## precision  : 0.767148
## f1         : 0.6719368
## accuracy   : 0.8119619
## auc        : 0.8174259
```

Note: The model is evaluated on the data given in the explainer. Use `DALEX::update_data()` to specify another dataset.
Note: Explainer knows whether the model is for classification or regression, so it automatically selects the right measures. It can be overridden if needed.

Performance charts

The S3 generic `plot` function draws a graphical summary of the model performance. With the `geom` argument, one can determine the type of chart.

```
# Boxplots
plot(mp_ranger, mp_rms, geom = "boxplot")
# ROC curves
plot(mp_ranger, mp_rms, geom = "roc")
# LIFT curves
plot(mp_ranger, mp_rms, geom = "lift")
```

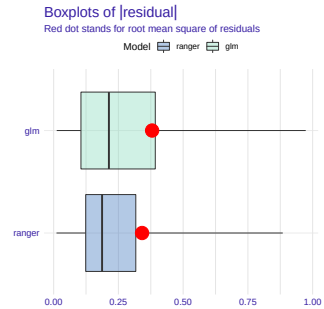


Figure 1: Boxplots for residuals

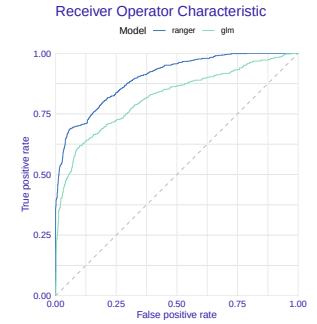


Figure 2: ROC curves

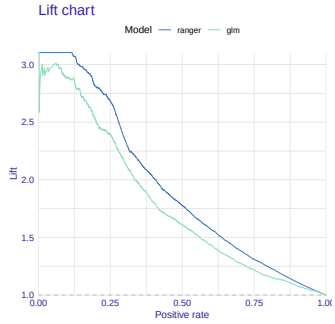


Figure 3: LIFT curves

Which variables have an impact on a model prediction?

Once we calculate the model prediction, the question often arises which variables had the greatest impact on it.

```
henry <- titanic_imputed[1,]
predict(exp_ranger, henry)
## [1] 0.1032819
```

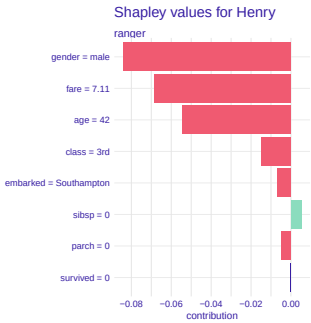


Figure 4: Shapley values assess the impact of each variable on a model prediction. The attributions add up to model prediction.

For linear models it is easy to assess the impact of individual variables because there is one coefficient for each variable.
It turns out that such attributions can be calculated for any predictive model. The most popular model agnostic method are Shapley values. They may be calculated with a `predict_parts()` function.

```
sh_ranger <- predict_parts(exp_ranger, henry, type = "shap")
plot(sh_ranger, show_boxplots = FALSE)
```

The Shapley values are additive. For models with interactions, it is often too much of a simplification. The Break Down method allows for the identification of interactions.

```
bd_ranger <- predict_parts(exp_ranger, henry, type = "break_down_interactions")
bd_ranger
##                                 contribution
## ranger: intercept                      0.322
## ranger: gender = male                 -0.102
## ranger: fare:class = 7.11:3rd         -0.046
## ranger: age = 42                      -0.059
## ranger: sibsp = 0                      0.002
## ranger: embarked = Southampton        -0.008
## ranger: parch = 0                     -0.005
## ranger: survived = 0                   0.000
## ranger: prediction                     0.103

plot(bd_ranger)
```

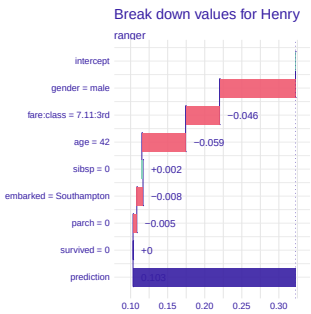


Figure 5: Break down values assess the impact of variable or their interactions on model predictions.

The `show_boxplots` argument allows you to highlight the stability bars of the estimated attributions.
Other possible values of the `type` argument are `oscillations`, `shap`, `break_down`, `break_down_interactions`.
With `order` one can force a certain sequence of variables.

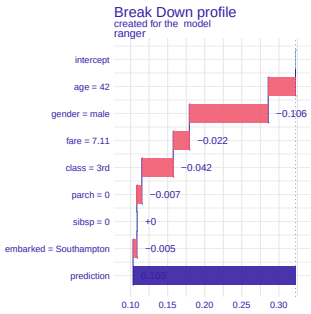


Figure 6: Break down values for selected order of variables.

```
bd_ranger <- predict_parts(exp_ranger, henry,
        order = c("age", "gender", "fare", "class",
                  "parch", "sibsp", "embarked"))
plot(bd_ranger)
```

By default, functions such as `model_parts`, `predict_parts`, `model_profiles` do not calculate statistics on the entire data set, but on `n_samples` of random cases, and the entire procedure is repeated `B` times to estimate the error bars.

Which variables have an impact on the model?

Many models have built-in ways of assessing the importance of variables. The procedure described below is universal and does not depend on the model structure.

Note that if a variable is important in a model, then after its permutation the model predictions should be less accurate. The permutation importance of a variable `i` is the difference between the model performance for the original data and the model performance measured on data with the permutated variable `i`[11].

The `DALEX::model_parts()` function calculates the importance of variables. The `type` argument determines whether to subtract (`type="difference"`), divide (`type="ratio"`) or present original loss functions (`type="raw"`).

```
mp_ranger <- model_parts(exp_ranger, type = "difference")
mp_ranger
##       variable mean_dropout_loss  label
## 1 _full_model_        0.00000000 ranger
## 2        parch        0.01751121 ranger
## 3        sibsp        0.02057990 ranger
## 4     embarked        0.02507604 ranger
## 5          age        0.07177812 ranger
## 6         fare        0.09148897 ranger
## 7        class        0.09509643 ranger
## 8       gender        0.20906310 ranger
## 9   _baseline_        0.39712142 ranger

plot(mp_ranger, show_boxplots = FALSE)
```

The importance of variables is a convenient tool to compare models. It is enough to put several importance objects to the generic S3 `plot()` function.

```
mp_ranger <- model_parts(exp_ranger)
mp_rms <- model_parts(exp_rms)
plot(mp_ranger, mp_rms, show_boxplots = FALSE)
```

By default, RMSE is used for regression and 1-AUC for classification problems. But you can change the loss function by specifying the `loss_function` argument.

[11] $V(f,i) = L^i_{perm}(f) - L_{org}(f)$, where $L_{org}(f)$ is the value of loss function for original data, while $L^i_{perm}(f)$ is the value of loss function after permuted i-th variable

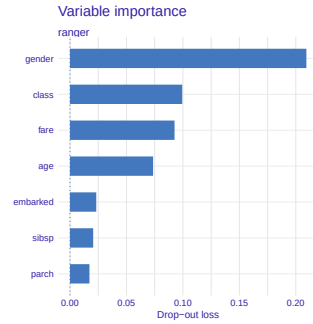


Figure 7: Importance scores $V(f,i)$

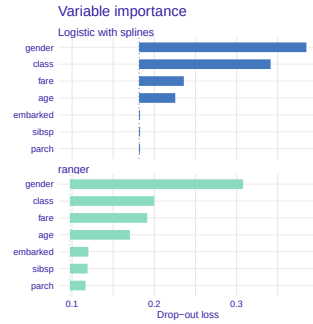


Figure 8: Variable importance plots for two models. Each bar starts in $L^i_{perm}(f)$ and ends in $L_{org}(f)$.

What if?

Ceteris-paribus profiles[12] show how the model response would change for a selected observation if one of the coordinates of that observation were changed while leaving the other coordinates unchanged.

The `predict_profiles()` function calculated CP profiles for a selected observation, model and vector of variables (all continuous variables by default).

```
cp_ranger <- predict_profile(exp_ranger, henry)
cp_ranger
## Top profiles    :
##       gender        age class    embarked fare sibsp parch     _yhat_
## 1     female 42.0000000   3rd Southampton 7.11     0     0 0.42139892
## 1.1     male 42.0000000   3rd Southampton 7.11     0     0 0.09811404
## 11      male  0.1666667   3rd Southampton 7.11     0     0 0.51780598
##       _vname_ _ids_ _label_
## 1      gender     1  ranger
## 1.1    gender     1  ranger
## 11        age     1  ranger
```

CP profiles can be visualized with the generic `plot()` function.

```
plot(cp_ranger, variables = c("age", "fare"))
```

For technical reasons, quantitative and qualitative variables cannot be shown in a single chart. So if you want to show the importance for quality variables you need to plot them separately.

```
plot(cp_ranger, variables = "class", categorical_type = "bars")
```

The `plot` function can combine different models, making it easier to see similarities and differences. The `color` argument allows you to highlight models in the figure.

```
cp_rms <- predict_profile(exp_rms, henry)
plot(cp_ranger, cp_rms, variables = "age", color = "_label_")
```

Oscillations

Local importance of variables can be measured as oscillations of CP plots. The greater the variability of the CP profile, the more important is the variable. Set `type = "oscillations"` in the `predict_parts` function.

```
 predict_parts(exp_rms, henry, type = "oscillations")
##    _vname_ _ids_ oscillations
## 1   gender     1   0.28444819
## 3    class     1   0.13305595
## 2      age     1   0.08664857
## 4     fare     1   0.02131783
```

[12] Ceteris Paribus is a Latin phrase for "other things being equal.

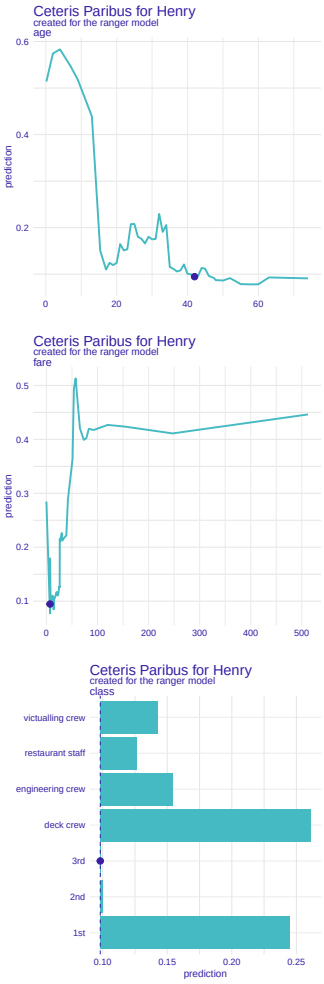


Figure 9: The dot shows the observation under analysis. CP profile shows how the model prediction will change for changes in the selected variable.

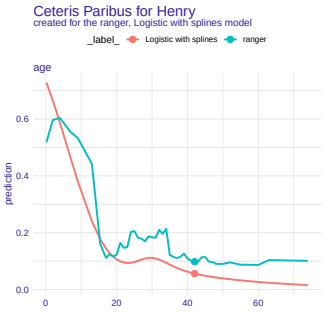


Figure 10: CP profiles for two models.

Partial dependence profiles

Partial dependence profiles are averages from CP profiles for all (or a large enough number) observations.
The `model_profiles()` function calculates PD profiles for a specified model and variables (all by default).

```
mp_ranger <- model_profile(exp_ranger)
```

Profiles can be then drawn with the `plot()` function. See an example in Figure 11.

```
plot(mp_ranger, variables = "age")
```

Grouped partial dependence profiles

By default, the average is calculated for all observations. But with the argument `groups=` one can specify a factor variable in which CP profiles will be averaged. See an example in Figure 12.

```
mp_ranger <- model_profile(exp_ranger, groups = "gender")
plot(mp_ranger, variables = "age")
```

Clustered partial dependence profiles

If the model is additive, all CP profiles are parallel. But if the model has interactions, CP profiles may have different shapes for different observations. Defining the k argument allows to find and calculate the average in k segments of CP profiles.

```
mp_ranger <- model_profile(exp_ranger, k = 3, center = TRUE)
```

PDP profiles do not take into account the correlation structure between the variables. For correlated variables, the Ceteris paribus assumption may not make sense. The `model_profile` function can also calculate other types of aggregates, such as marginal profiles and accumulated local profiles. To do this, specify the argument `type=` for `"conditional"` or `"accumulated"`.

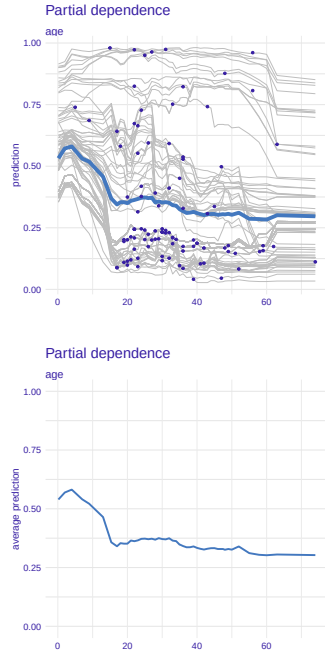


Figure 11: Partial dependence profile for age variable.

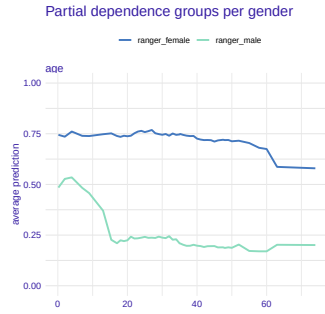


Figure 12: Partial dependence for age in groups defined by gender.

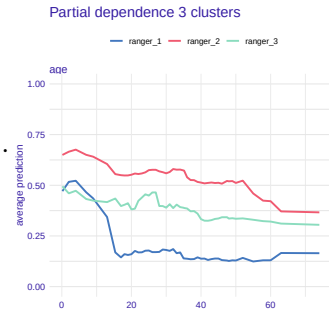


Figure 13: Partial dependence for three segments of CP profiles.

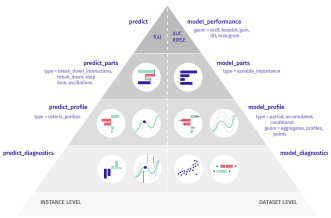


Figure 14: XAI pyramid

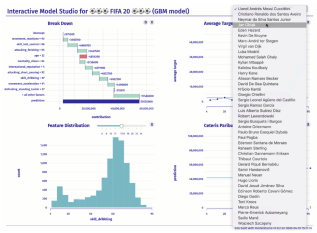


Figure 15: modelStudio dashboard

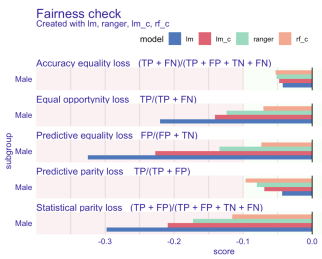


Figure 16: Fairness check

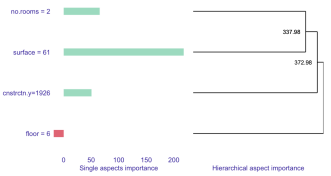


Figure 17: Hierarchical analysis of variable's importance with triplot

What's next?

This brochure presents the selected solutions from the Explanatory Model Analysis area. The solutions developed by our team include more packages. All of them are available at GitHub under the organization `https://github.com/ModelOriented/`. Here are some highlights. Find more at `http://DrWhy.AI`.

- `DALEXtra` is a package with additional functionalities facilitating work with models created in python (scikitlearn, keras), H2O, mlr, mlr3 or other popular frameworks. All functions described above will also work for models created in other libraries or programming languages, provided that these models are properly wrapped.
- `modelStudio` and `ArenaR` are packages based on the evolving grammar of interactive model exploration. They generate a serverless HTML dashboard (one line of code) that consists of various XAI views that can be freely combined.
- `fairmodels` is a package that implements various measures of model bias and fairness. Measures such as Equal opportunity loss, Predictive equality loss, Predictive parity loss or Statistical parity loss. It is easy for one or more explainers to investigate whether they generate unfair decisions.
- `triplot` is a package analyzing the importance of variables taking into account the structure of correlation between the variables. Apart from analysing how important a single variable is, we also have an analysis of how important groups of correlated variables are. Groups can be defined by ourselves or with a hierarchical cluster analysis.

Acknowledgments

Many thanks to the workshop assistants Anna Kozak and Szymon Maksymiuk and the organizers of the great eRum 2020 conference.

Work on these packages was financially supported by the Polish National Science Centre under Opus Grant number 2016/21/B/ST6/02176 and 2017/27/B/ST6/0130.