EARL conference

Explainable Machine Learning.

WORKSHOP 3.

Objectives

At the end of this workshop you will be able to:

- Describe explainable AI methods
- Explain the importance of building explainable models to other stakeholders
- Implement a range of XAI techniques with DALEX

You won’t learn much about:

- Other responsible ML tools like fairness
- Different packages pros and cons

Requirements

You will need

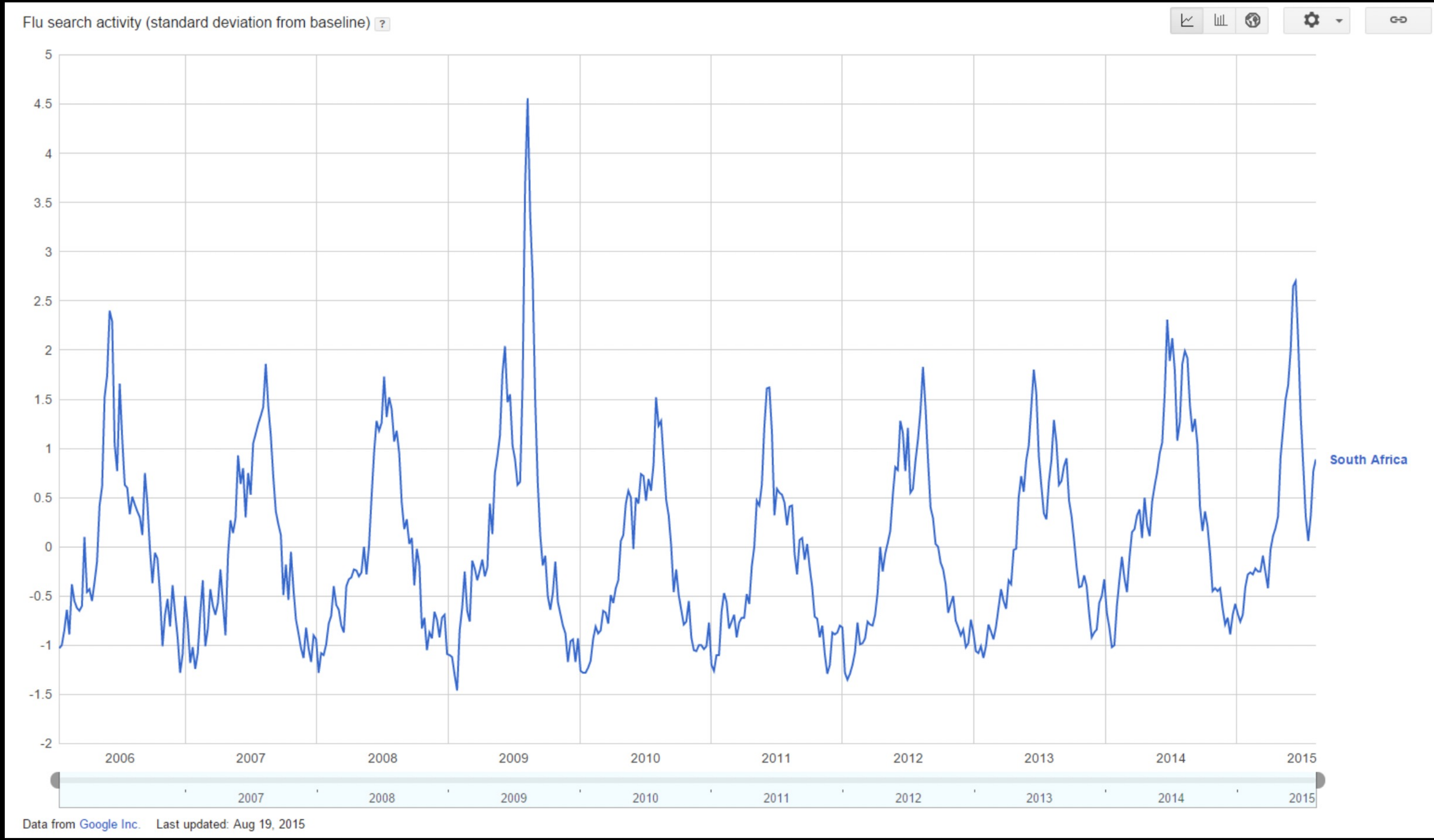
- Some understanding of modelling concepts.
- R and RStudio installed.

You won't need

- Prior knowledge of explainable AI methods.

Once upon a time…

There was a great algorithm

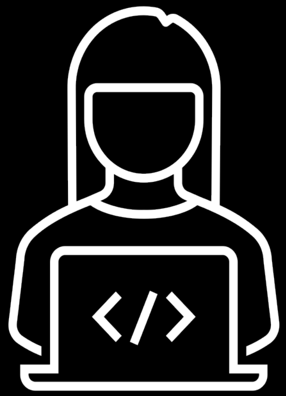


And it was retired, not so happily, ever after

Because it stopped working

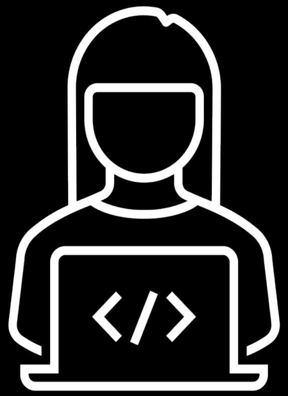
- Missed a summer outbreak
- Overestimated winter outbreaks

Why should you care?

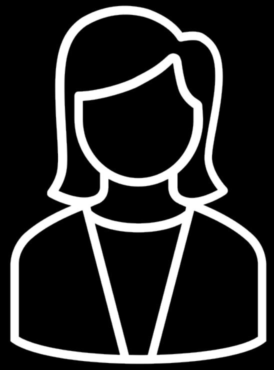


Data Scientist

Why should you care?

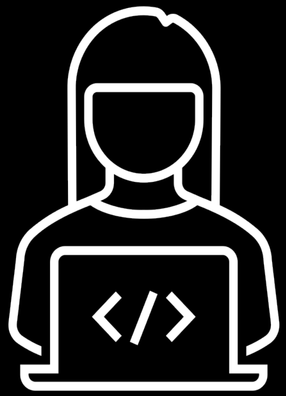


Data Scientist

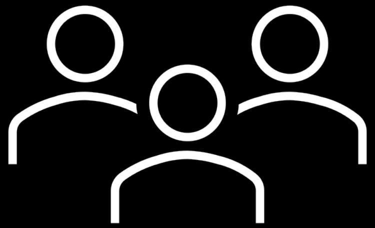


Business stakeholder

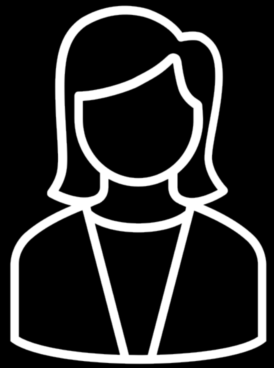
Why should you care?



Data Scientist

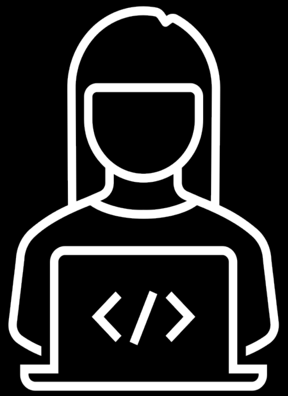


Consumer

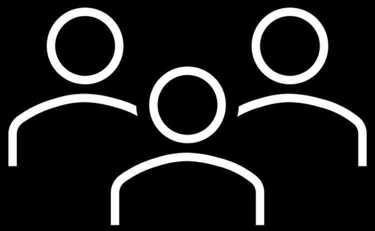


Business
stakeholder

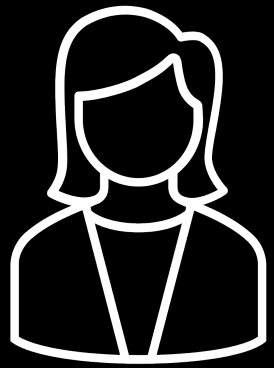
Why should you care?



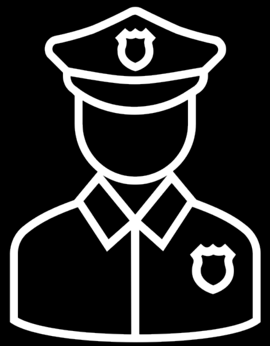
Data Scientist



Consumer



Business stakeholder



Regulator

How to explain your AI

Publish algorithms

Use interpretable models

Analysis > Modelling

Proxy models

Model analysis

How to explain your AI

Publish algorithms

Use interpretable models

Analysis > Modelling

Proxy models

Model analysis

Types of model analysis

Global
Variable importance for model Feature Importance
Variable variability affect on prediction Partial Dependence Plots (PDP) Accumulated Local Effects (ALE)
Model diagnostics Residual plots, Variable vs. prediction plots

Types of model analysis

Global	Local
Variable importance for model Feature Importance	**Variable importance for single prediction** Break Down (BD) SHAP LIME
Variable variability affect on prediction Partial Dependence Plots (PDP) Accumulated Local Effects (ALE)	
Model diagnostics Residual plots, Variable vs. prediction plots	

Types of model analysis

Global	Local
Variable importance for model Feature Importance	**Variable importance for single prediction** Break Down (BD) SHAP LIME
Variable variability affect on prediction Partial Dependence Plots (PDP) Accumulated Local Effects (ALE)	**Sensitivity analysis** Ceteris Paribus (CP) Individual Conditional Expectations (ICE)
Model diagnostics Residual plots, Variable vs. prediction plots	**Predict diagnostics** Local residual density plot

Chapter 1
Pre-requisites

This workshop proceeds under the assumption that the reader is familiar with the following concepts:

- Modelling in R
- Statistical methods of model evaluation

Chapter 2

Introduction

Why is XAI important?

Modelling packages, such as `tidymodels` or `caret` offer a common syntax to conveniently test a selection of Machine Learning models and select the best performing one based on a pre-defined metric. These tools have been invaluable in enabling fast iteration and efficient innovation to productionise ML/AI powered products.

A potentially undesirable side effect of this workflow is that it leads to considering very different models as equivalent black-boxes that can be swapped in and out of the pipeline.

This, in turn, makes it difficult to detect certain problems early enough. Insufficiently tested models quickly lose their effectiveness, lead to unfair decisions, discriminate, are deferred by users, and do not provide the option to appeal (Maksymiuk et al, 2021[1]).

How can you use it?

XAI methods are often helpful, and sometimes, necessary. For example, when:

- A model makes incorrect decisions
- You work with inquisitive stakeholders who need to understand the underlying dynamics of the model to trust it
- You want to validate an assumption or increase domain knowledge
- GDPR requires you to be able to justify automated decisions to the people who are impacted by your models
- You want to make a responsible model recommendation, knowing not just if the model makes accurate decisions but *how* it gets there

What is DALEX / DALEXtra?

The DALEX and DALEXtra packages are implementations of a range of XAI methods. According to the package authors themselves[2]:

> The DALEX package xrays any model and helps to explore and explain its behaviour, helps to understand how complex models are working. The main function explain() creates a wrapper around a predictive model. Wrapped models may then be explored and compared with a collection of local and global explainers.

[1] https://arxiv.org/pdf/2009.13248.pdf
[2] https://github.com/ModelOriented/DALEX

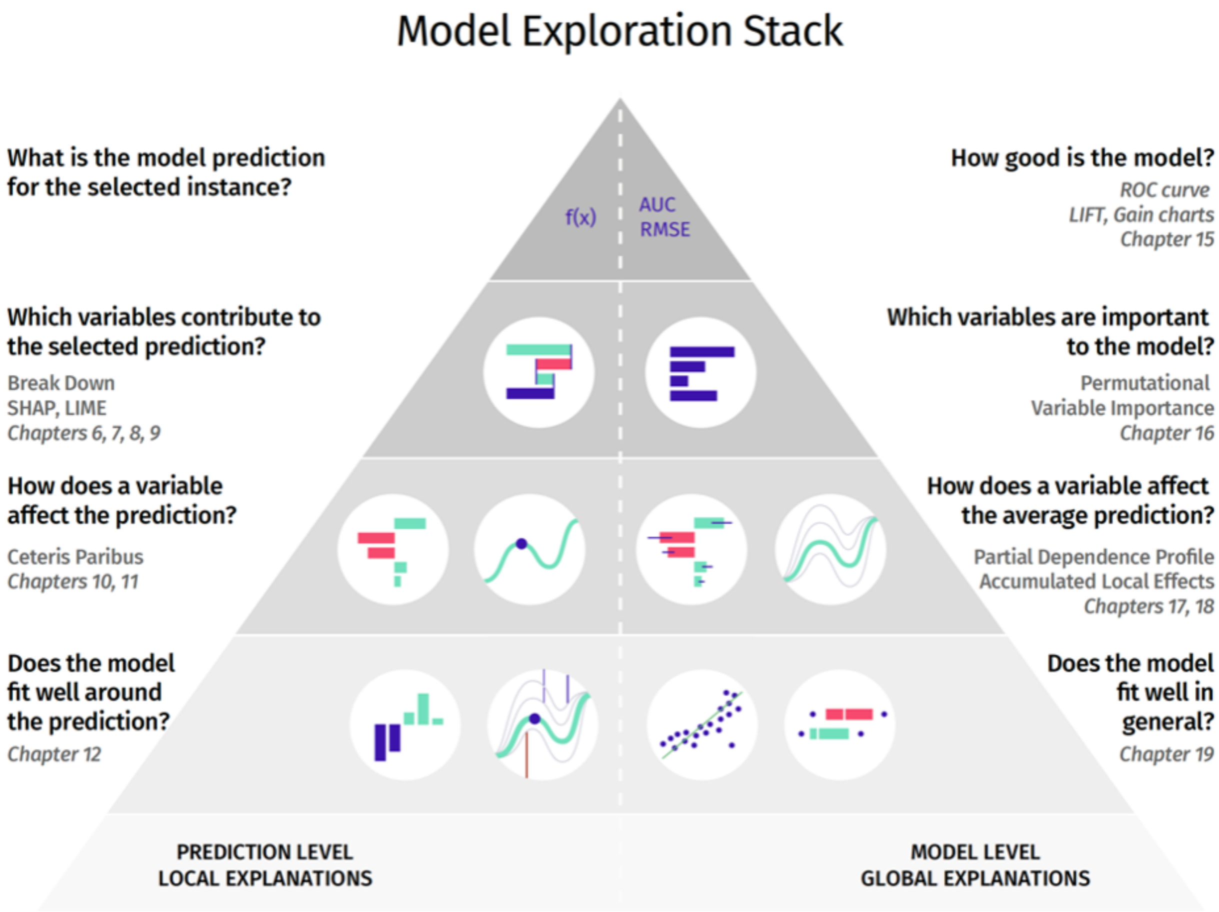
Why did we chose them?

There is a plethora of great packages in R to deep dive into your models predictions. Among the most downloaded in CRAN you can find: `lime`, `smbinning`, `iml`, `pdp` and many others. We picked DALEX for this workshop because:

- DALEX works with a lot of modelling frameworks
- It gives a standardised grammar to call lots the different model agnostic methods
- It is one of the most popular R package on CRAN and Github

What are we covering in this workshop?

There are many taxonomies of explainable AI methods. For the purpose of this workshop, we will follow the one made by Przemyslaw Biecek and Tomasz Burzykowski in their book Explanatory Model Analysis[3].



Model exploration techniques, as presented in Explanatory Model Analysis: Explore, Explain, and Examine Predictive Models. With examples in R and Python. Chapters listed are in reference to the book.

[3] https://ema.drwhy.ai/introduction.html

We will first look at the model performance and diagnostics, then look at global explainers, such as feature importance, partial dependency plots and accumulated local effects, before exploring local explainers, including break down plots, Shapley values and LIME.

Chapter 3

Environment set up

First things first, make sure you have the right packages installed and load the relevant libraries as well as the data used for this tutorial.

```
# packages_to_install <- c('DALEX', 'randomForest', 'tidyverse')
# install.packages(packages_to_install)

# Load packages
library(DALEX)
library(randomForest)
library(tidyverse)

# load data
data("apartments")
data("apartments_test")
```

Let's add a random variable in both the training and the test set to see how this impacts the feature importance and other explainable AI methods. We will also set number of rooms and floor to be factor variables.

```
data_transform <- function(data){
  data %>%
    mutate(random_var = runif(dim(data)[1]),
           no.rooms = as.factor(no.rooms),
           floor = as.factor(floor))
}

apartments <- apartments %>% data_transform()

apartments_test <- apartments_test %>% data_transform()
```

To get acquainted with the dataset, have a quick look at what we're working with.

```r
summary(apartments)
#>     m2.price     construction.year    surface           floor
#>  Min.    :1607   Min.    :1920       Min.    : 20.00   2      :116
#>  1st Qu.:2857   1st Qu.:1943       1st Qu.: 53.00   9      :108
#>  Median :3386   Median :1965       Median : 85.50   10     :108
#>  Mean   :3487   Mean   :1965       Mean   : 85.59   6      :104
#>  3rd Qu.:4018   3rd Qu.:1988       3rd Qu.:118.00   7      :103
#>  Max.   :6595   Max.   :2010       Max.   :150.00   8      :103
#>                                                     (Other):358
#>  no.rooms         district      random_var
#>  1: 99     Mokotow     :107   Min.    :0.0000839
#>  2:202     Wola        :106   1st Qu.:0.2437661
#>  3:231     Ursus       :105   Median :0.5002432
#>  4:223     Ursynow     :103   Mean   :0.5003114
#>  5:198     Srodmiescie:100   3rd Qu.:0.7589129
#>  6: 47     Bemowo      : 98   Max.   :0.9990245
#>            (Other)     :381
```

Chapter 4

Make two models

We build two models, one linear regression model and one tree based model. This will allow us to compare explanation methods for different types of models.

Linear models are inherently explainable while random forests are not. Illustrating explanation methods with both should bring some clarity on the underlying methodology of the explainers and showcase the value add of XAI methods for models that are not inherently explainable.

```
set.seed(220808)

# train RF model
apartments_rf_model <-
  randomForest::randomForest(m2.price ~ .,
                             data = apartments)

# train LM model
apartments_lm_model <-  lm(m2.price ~ .,
                           data = apartments)

# predict on test set using RF
predicted_rf <- predict(apartments_rf_model,
                        apartments_test)

# predict on test set using LM
predicted_lm <- predict(apartments_lm_model,
                        apartments_test)
```

5

Build the explainers

How does DALEX work?

DALEX is a wrapper around a model. The first step is therefore to wrap the model with the `explain` function.

Let's have a look at the output from this code. It is useful to refer to the documentation[4] to understand what the function expects.

```
explainer_lm <- DALEX::explain(model = apartments_lm_model,
                               data = apartments_test[,2:7], # test
        data, excluding outcome
                               y = apartments_test$m2.price
                               ) # test data outcome column
#> Preparation of a new explainer is initiated
#>   -> model label       :  lm  (  default  )
#>   -> data              :  9000  rows  6  cols
#>   -> target variable   :  9000  values
#>   -> predict function  :  yhat.lm  will be used (  default  )
#>   -> predicted values  :  No value for predict function target
        column. (  default  )
#>   -> model_info        :  package stats , ver. 4.2.0 , task
        regression (  default  )
#>   -> predicted values  :  numerical, min =  1812.558 , mean =
        3505.844 , max =  6266.988
#>   -> residual function :  difference between y and yhat (
        default  )
#>   -> residuals         :  numerical, min =  -301.1235 , mean =
        5.679968 , max =  563.5238
#>   A new explainer has been created!

explainer_lm
#> Model label:  lm
#> Model class:  lm
#> Data head  :
#>      construction.year surface floor no.rooms    district
        random_var
#> 1001              1976     131     3        5 Srodmiescie
        0.1404973
#> 1002              1978     112     9        4     Mokotow
        0.3263997
```

[4] https://www.rdocumentation.org/packages/DALEX/versions/2.4.2/topics/explain.default

```
explainer_rf <- DALEX::explain(model = apartments_rf_model,
                               data = apartments_test[,2:7],
                               y = apartments_test$m2.price)
#> Preparation of a new explainer is initiated
#>   -> model label       :  randomForest  (  default  )
#>   -> data              :  9000  rows  6  cols
#>   -> target variable   :  9000  values
#>   -> predict function  :  yhat.randomForest  will be used (
        default  )
#>   -> predicted values  :  No value for predict function target
        column. (  default  )
#>   -> model_info        :  package randomForest , ver. 4.7.1.1 ,
        task regression (  default  )
#>   -> predicted values  :  numerical, min =  1867.907 , mean =
        3502.116 , max =  6161.301
#>   -> residual function :  difference between y and yhat (
        default  )
#>   -> residuals         :  numerical, min =  -615.3261 , mean =
        9.40788 , max =  947.041
#>   A new explainer has been created!

# show object
explainer_rf
#> Model label:  randomForest
#> Model class:  randomForest.formula,randomForest
#> Data head  :
#>      construction.year surface floor no.rooms    district
        random_var
#> 1001              1976     131     3        5 Srodmiescie
        0.1404973
#> 1002              1978     112     9        4     Mokotow
        0.3263997
```

The console prints the model information and the data head. Now, explore the explainer object: what is available to you through this object?

```
Preparation of a new explainer is initiated
  -> model label       :  your model label (e.g. Random Forest)
  -> data              :  dimensions of your data
  -> target variable   :  dimensions of your target variable
  -> predict function  :  the predict function for your model
  -> predicted values  :  Column in the model prediction object of
the positive class (Not relevant here)
  -> model_info        :  model package, version and type (here:
regression)
  -> predicted values  :  description of the output of your predict
function
  -> residual function :  difference between y and yhat (  default
)
  -> residuals         :  description of the residuals, after
running the predict function
```

Chapter 6

Model performance

The usual first step of a Machine Learning workflow is to check the model performance. Our example is a regression problem so the root mean squared error is a relevant metric.

DALEX has a `model_performance` function that creates an object containing the residuals, which you can explore yourself or use the plot method of the DALEX object to plot common performance metrics.

6.1 Make the model performance object

```
mp_lm <- model_performance(explainer_lm)
mp_lm
#> Measures for:  regression
#> mse        : 81450.82
#> rmse       : 285.3959
#> r2         : 0.8995432
#> mad        : 222.8607
#>
#> Residuals:
#>        0%       10%       20%       30%       40%       50%
        60%
#> -301.1235 -236.7025 -218.2758 -200.4750 -185.7940 -163.8277
        -134.1419
#>       70%       80%       90%      100%
#>  350.7932  392.8023  427.8070  563.5238
```

```
mp_rf <- model_performance(explainer_rf)
mp_rf
#> Measures for:  regression
#> mse        : 41056.92
#> rmse       : 202.6251
#> r2         : 0.9493627
#> mad        : 116.7112
#>
#> Residuals:
#>          0%         10%         20%         30%         40%
#> -615.326133 -203.760343 -141.216828 -100.328513  -67.101563
#>         50%         60%         70%         80%         90%
#>  -30.871354    7.079974   69.388912  159.319895  274.509470
#>        100%
#>  947.041032
```

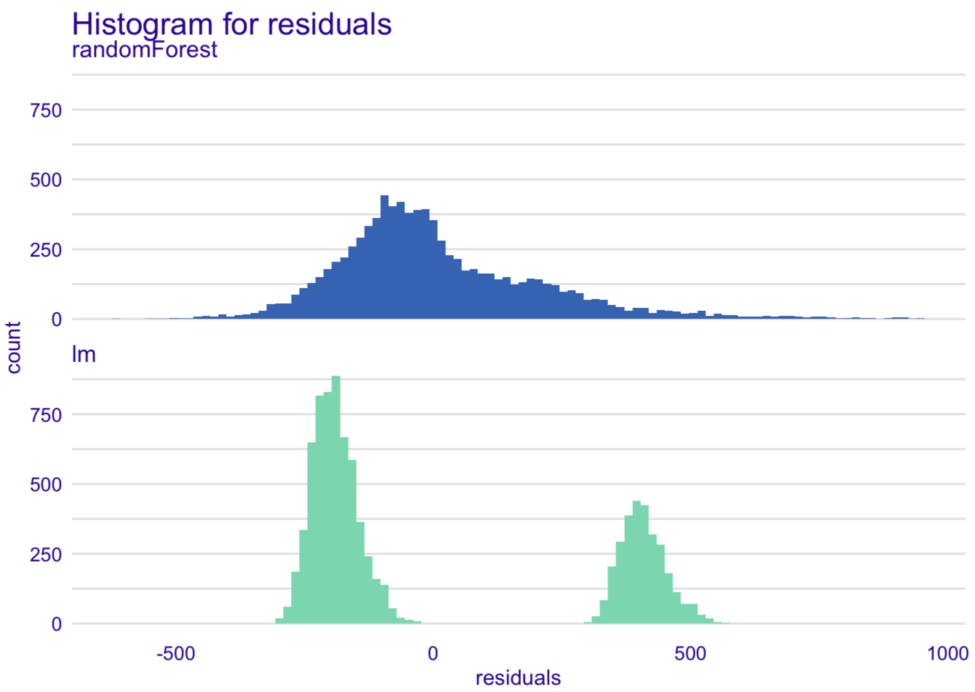
6.1.1 Exercise

Explore the structure of the model performance objects created.

Hint: Use the `str()` function.

6.2 Plot residuals

```
plot(mp_lm, mp_rf, geom = "histogram")
```



6.2.1 Exercise

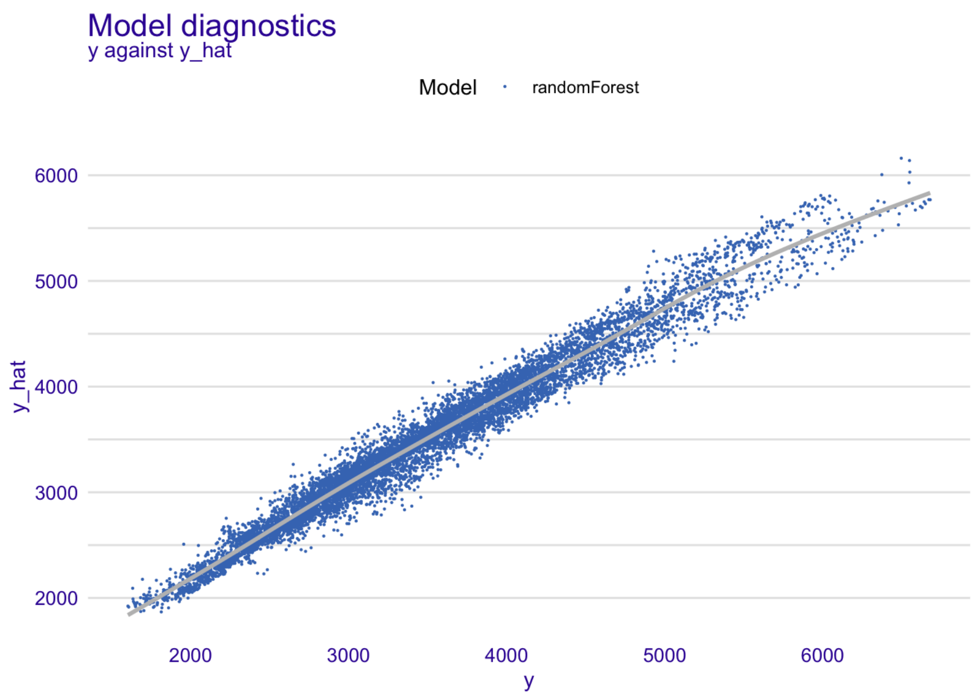
Plot the model performance as a boxplot.

Chapter 7
Model Diagnostics

Model diagnostics allow you to explore the relationship between your model outcome and the different features. The first step is to plot predicted values against observed ones.

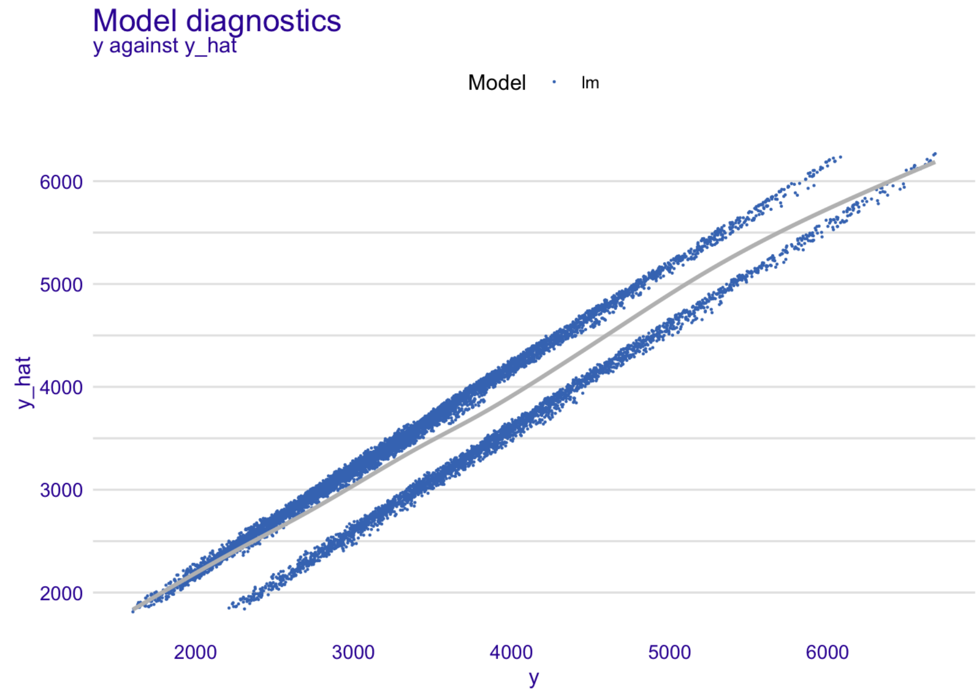
```
md_rf <- model_diagnostics(explainer_rf)
md_lm <- model_diagnostics(explainer_lm)

plot(md_rf, variable = "y", yvariable = "y_hat")
```



As expected, the random forest is a really good model.

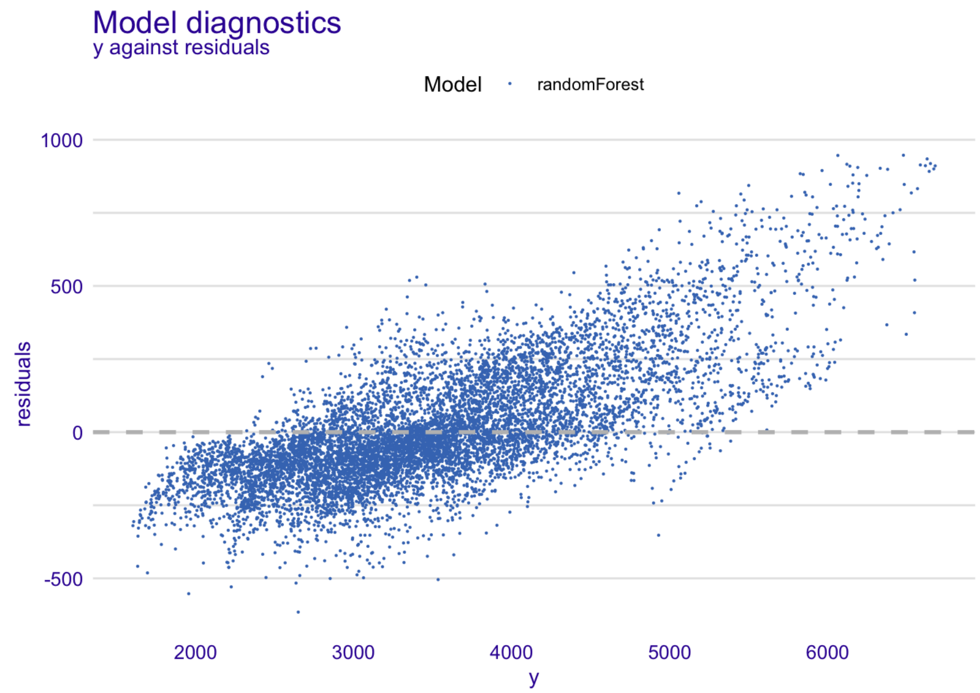
```
plot(md_lm, variable = "y", yvariable = "y_hat")
```



The two groups of residuals are represented on the y vs. y_hat plot.

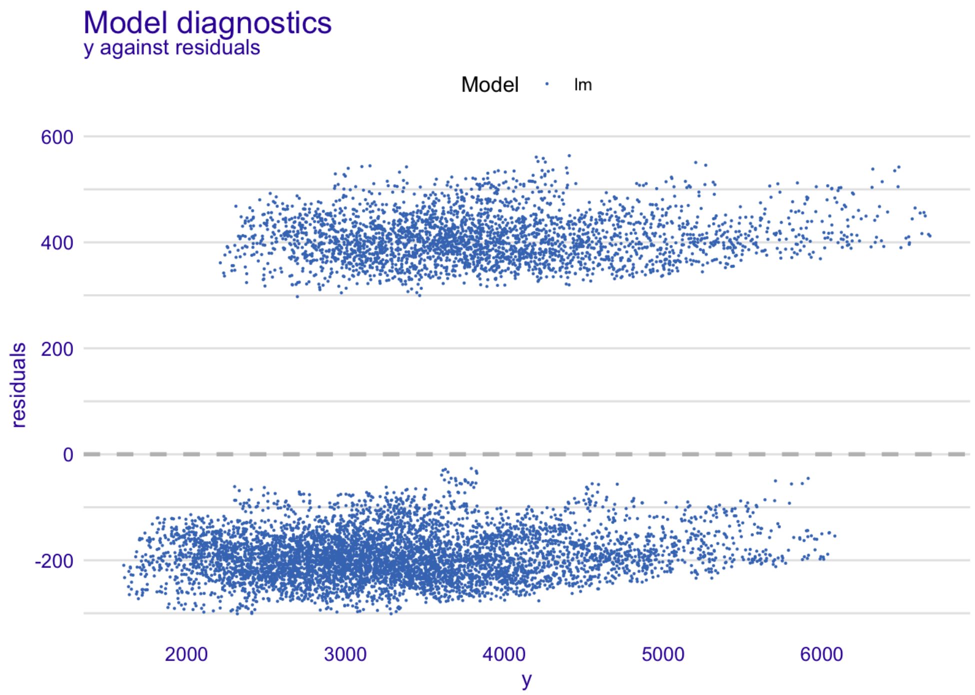
Let’s take a look at how residuals correlate with observed.

```
plot(md_rf, variable = "y", yvariable = "residuals", smooth = FALSE)
```



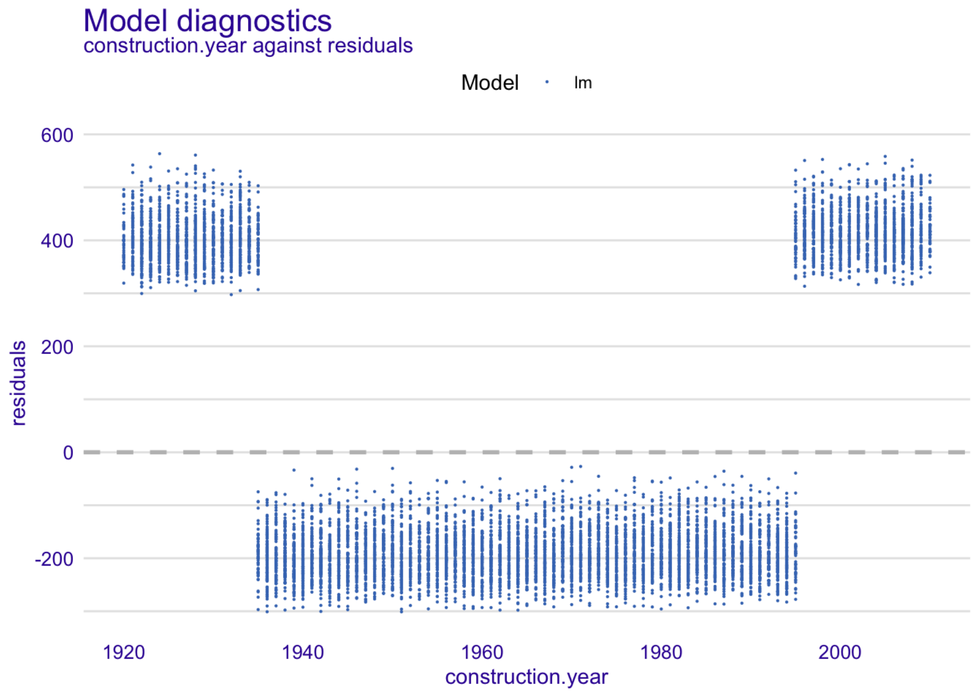
From the plots above we can conclude that Random Forest underestimates for high value flats and overestimates low value flats.

```
plot(md_lm, variable = "y", yvariable = "residuals", smooth = FALSE)
```



Linear model shows two groups of residuals! The residuals are not independent and identically distributed. Let's explore which variables have a non-linear effect on the estimate.

```
plot(md_lm, variable = "construction.year", yvariable = "residuals",
         smooth = FALSE)
```



Construction year is non-linearly correlated to the residuals. This is the pattern we observe in the model residuals for the linear regression.

7.0.1 Exercise

Create some additional diagnostic plots for the random and linear model.

Consider:

1. Exploring different variables.
2. Comparing both models for one particular variable.
3. Plotting the observed values against the absolute residuals.
4. Plotting absolute residuals (hint: use `yvariable="abs_residuals"`).

Chapter 8
Global explainer

8.1 Variable Importance

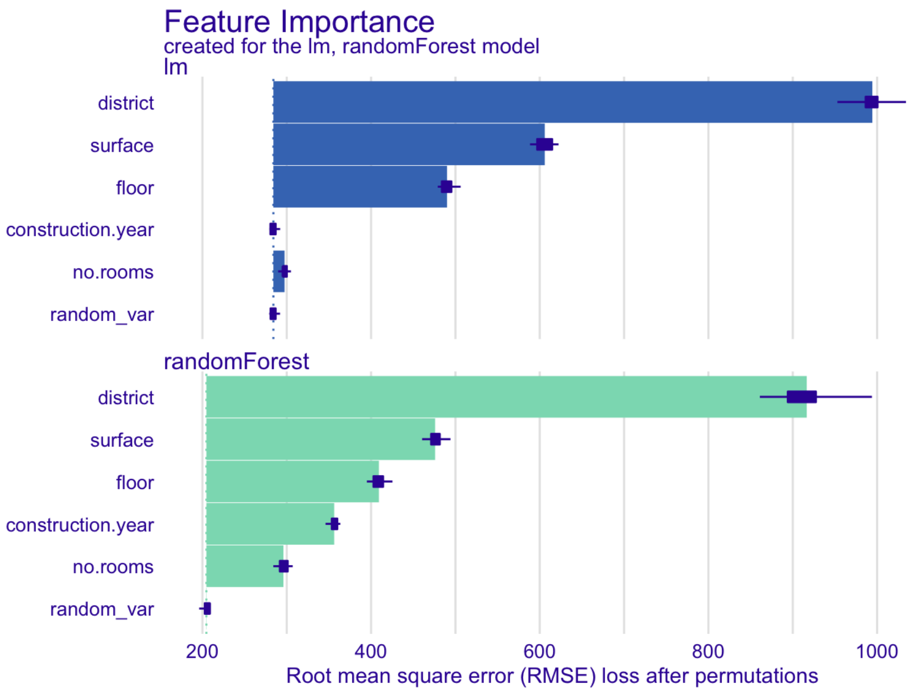
To understand the importance of an explanatory variable, we can use a method provided by *DALEX*, described by Fisher, Rudin, and Dominici (http://jmlr.org/papers/v20/18-760.html).

The `model_parts` function begins by calculating the loss for the full model (the default loss function for regression models is RMSE). Then, the values of one variable are permuted and the loss is recalculated. This is repeated for each variable.

As the perturbations are random, we repeat the process a number of times and average the results. The greater the change in loss is, the more important we deem the variable to be.

```
set.seed(220817)
fi_rf <- DALEX::model_parts(explainer_rf, B = 10)
fi_lm <- DALEX::model_parts(explainer_lm, B = 10)

plot(fi_rf, fi_lm)
```



The baseline of the bars above represent the loss of the full model. The bars represent the average loss when when the variable is permuted. The boxplots show the range of losses for the (B=10) permutations calculated. Note the difference in importance of the `construction.year` variable.

8.2 Partial Dependency Plots (PDP)

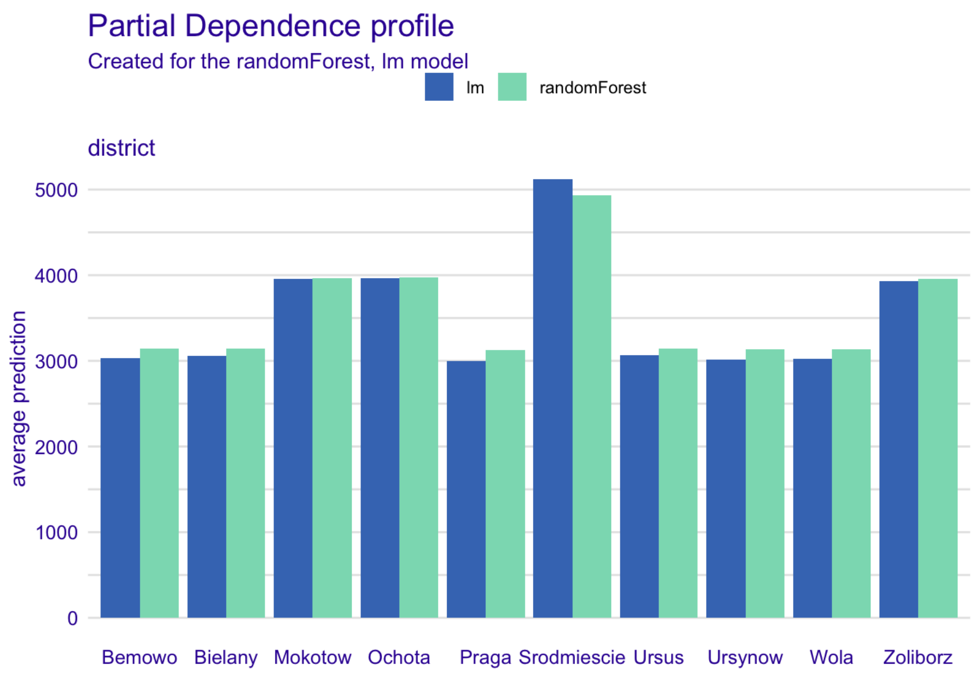
Partial dependence plots (PDPs) describe the marginal effect of one or two features on the target variable.

8.2.1 Categorical variables

In the case of categorical variables, this is easy to calculate. We just set the value of the category we are interested in to be the same for all observations. In the case of the apartments dataset, we can set the district of each apartment to be identical and compute the average prediction to understand the marginal effect of that district on apartment value.

If you are familiar with Warsaw, you might know Srodmiescie is the city center, Ochota, Mokotow and Zoliborz are well connected to the center and the rest of the districts are further out from the city center. The effect of being closer or further from the city center is evident, when we calculate the PDP.

```
pdp_cat_rf <- model_profile(explainer_rf,
                            variables = "district",
                            type = "partial")
pdp_cat_lm <- model_profile(explainer_lm,
                            variables = "district",
                            type = "partial")
plot(pdp_cat_rf, pdp_cat_lm)
```



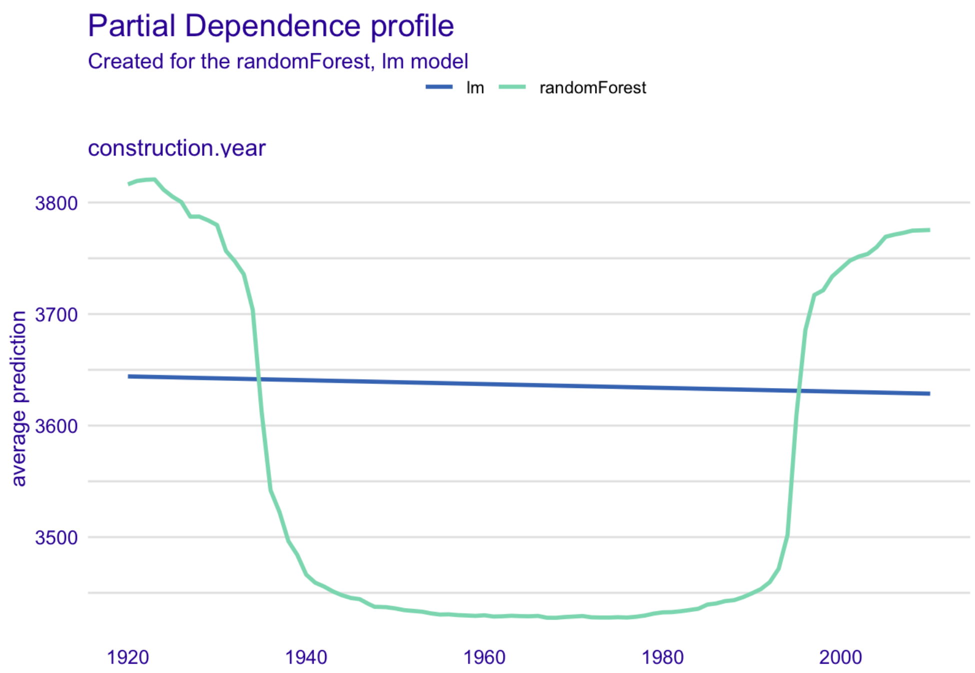
8.2.2 Continuous variables

In the case of continuous variables, a PDP can reveal whether the relationship between a feature and target is linear, monotonic, or something else. The computation for the partial dependence of apartment value on construction year, is as follows:

1. Pick a year
2. Set the year of construction of every observation to that year
3. Use this modified dataset to calculate the average apartment price
4. Repeat steps 1-3 for every year and draw a curve

For other continuous variables, these steps are followed for each unique feature value.

```
pdp_rf <- model_profile(explainer_rf, variables =
         "construction.year", type = "partial")
pdp_lm <- model_profile(explainer_lm, variables =
         "construction.year", type = "partial")
plot(pdp_rf, pdp_lm)
```



The PDP above shows that the linear model has a linear partial dependence relationship with construction year, whereas the relationship in the random forest model is non-linear and non-monotonous.

The benefits of using PDPs are that they are easy to implement and the computation behind it is fairly intuitive. However, while the results of this method can be easy to interpret, it assumes that the features are independent.

As described by Christopher Molner in his book 'Interpretable Machine Learning',

> "If the feature for which you computed the PDP is not correlated with the other features, then the PDPs perfectly represent how the feature influences the prediction on average. In the uncorrelated case, the interpretation is clear: The partial dependence plot shows how the average prediction in your dataset changes when the j-th feature is changed."

If the assumption of independence is violated, your PDP assumes the existence of very unlikely data points. For example, we know that the square footage of a flat, correlates with the number of rooms. The PDP assumes that there exists a flat that is 20 squared meters with 6 bedrooms. The next section introduces a tool that does not need to make the assumption of feature independence.

8.2.3 Exercise

Create PDPs for both models with `variables = NULL`. What relationships do you observe?

8.3 Accumulated Local Effect (ALE)

PDPs suffer from two great limitations when features are correlated:

1. they assume the existence of unlikely data points
2. they cannot disentangle the effect of each feature. In the above example, you would be estimating the effect of the square footage and the number of room together

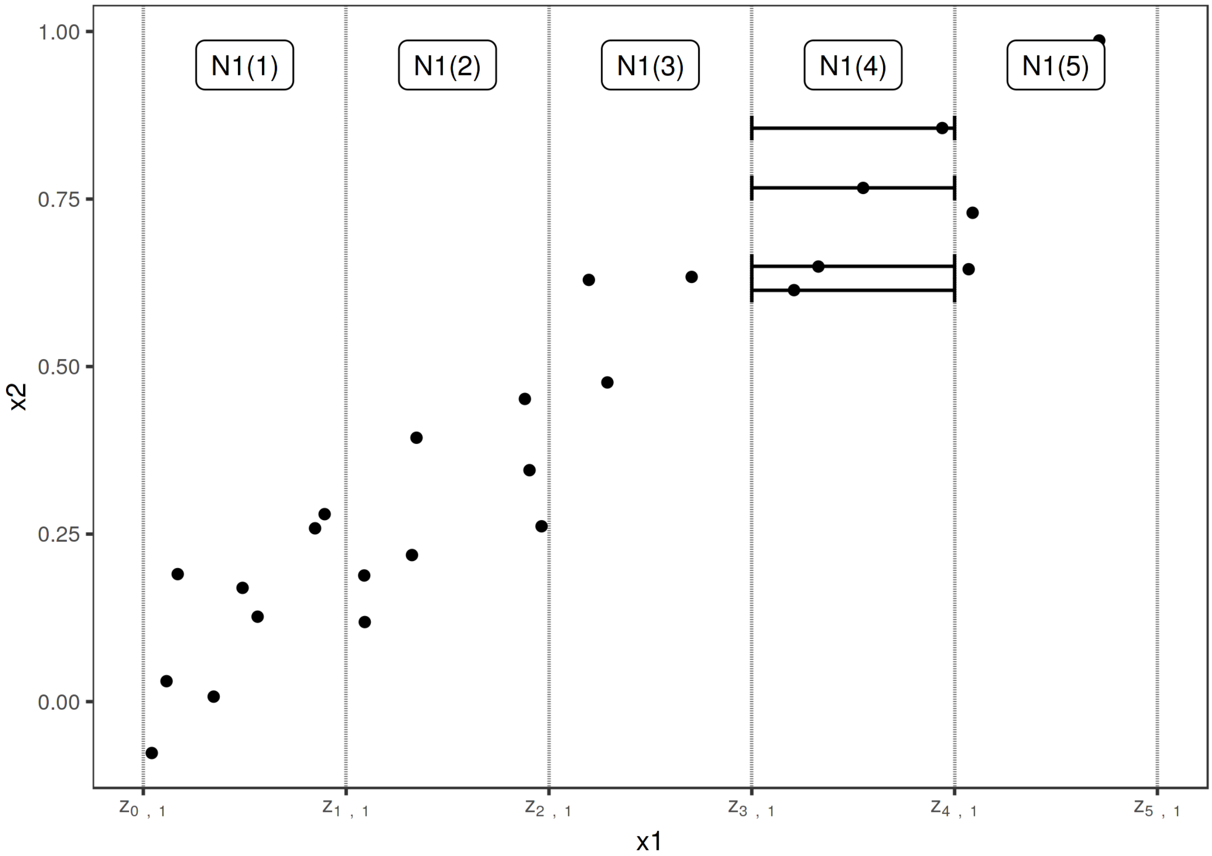
Accumulated Local Effect (ALE) curves solve this problem by using a difference in prediction rather than an average.

The process is as follows:

1. select an interval around a value of interest
2. for each points in that interval, find the difference between the observation if the feature was set to the upper bound vs the lower bound
3. Average these differences
4. repeat steps 1-3 and draw a curve through plotted points

The function of Accumulated local effects (ALE) is similar to PDPs, in that it also works to reveal how a feature affects predictions on average. However, it addresses a major drawback of PDPs and does not rely on variables being independent.

To better understand the ALE method, consider the apartments with surface areas in the range (20,22). For each observation in that interval, calculate the difference between its predicted value if we set the surface area to be 22m^2 vs if we set the surface area to be 20m^2. Sum these differences and divide by the number of observations to find the average effect of an apartment having a surface area of 21m^2. The figure below illustrates this process.



ALE calculation for variable x1, correlated with x2, as shown in Interpretable Machine Learning: A Guide for Making Black Box Models Explainable[5]

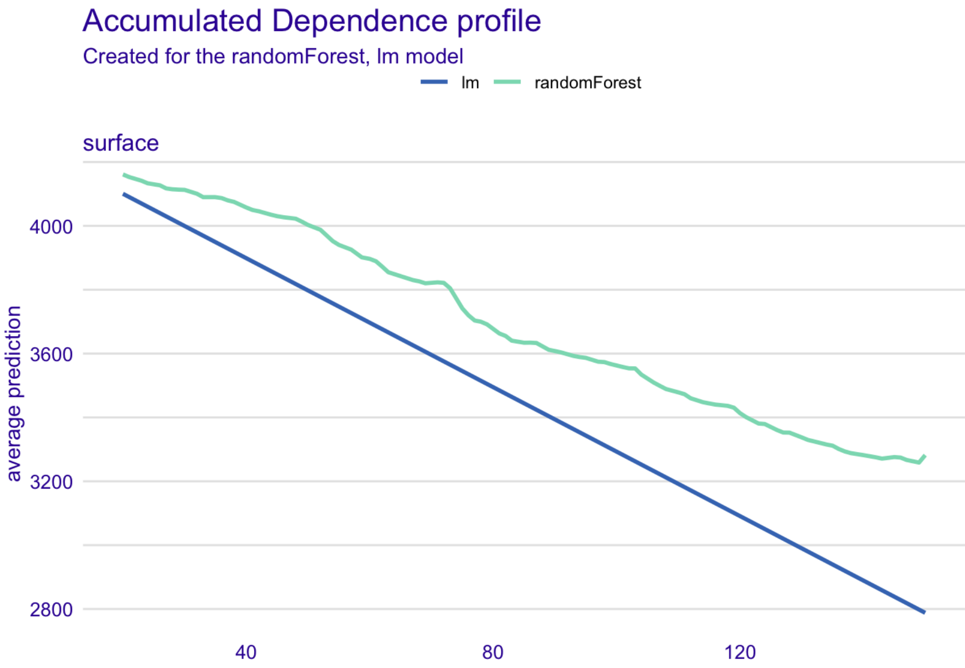
In addition to being an unbiased alternative to PDPs, ALE plots are still easy to interpret and faster to compute. On the down side, as effects are calculated per interval, interpretation of effects across intervals is not possible when features are strongly correlated.

```
ale_rf <- model_profile(explainer_rf,
                        variables = "surface",
                        type = "accumulated")

ale_lm <- model_profile(explainer_lm,
                        variables = "surface",
                        type = "accumulated"
                        )

plot(ale_rf, ale_lm)
```

[5]https://christophm.github.io/interpretable-ml-book/ale.html


Accumulated Dependence profile
Created for the randomForest, lm model
lm
randomForest
surface
average prediction
4000
3600
3200
2800
40
80
120

Chapter 9
Local explainer

Local explainer methods explore individual predictions and can help answer a question such as: which variable is having the greatest impact on *my* apartment's value (as opposed to the average value of apartments in Warsaw).

9.1 Break down

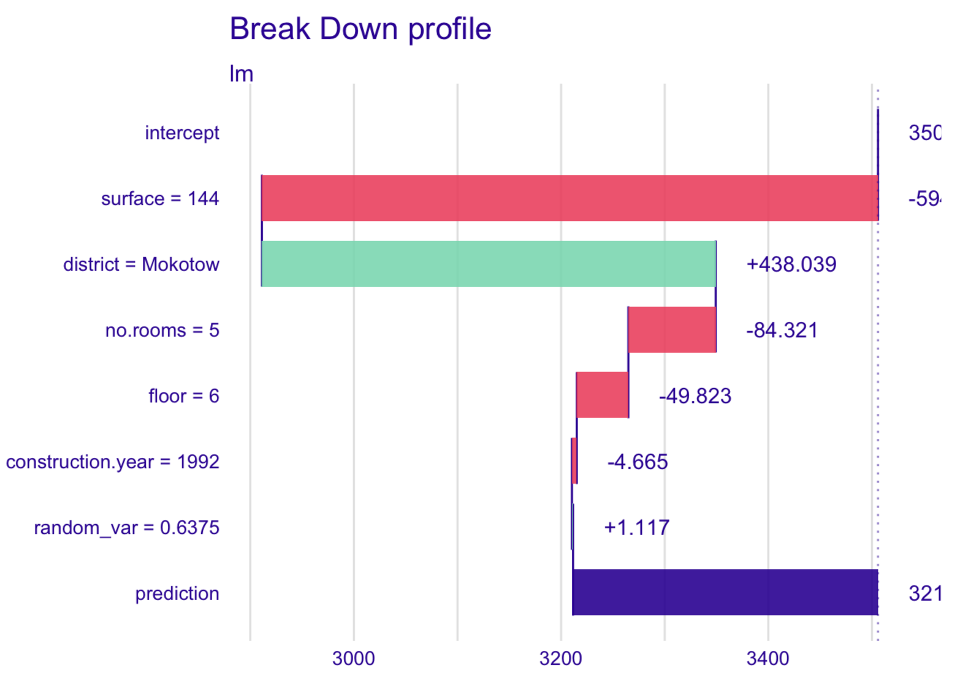
A break-down plot is a local explainer aiming to answer this question.

To understand the effect of variables on the value of apartment 5 in this dataset, we begin with the average price of all apartments, then (in the case of the LM), consider we set the surface area of all apartments in the dataset to be 144m^2, now we have a new average price. Next we set the district of all observations to be Mokotow, then we set the number of rooms to be 5 etc.. In the end, we end up with the prediction for apartment 5, as we have set all variables to be identical to that of apartment 5.

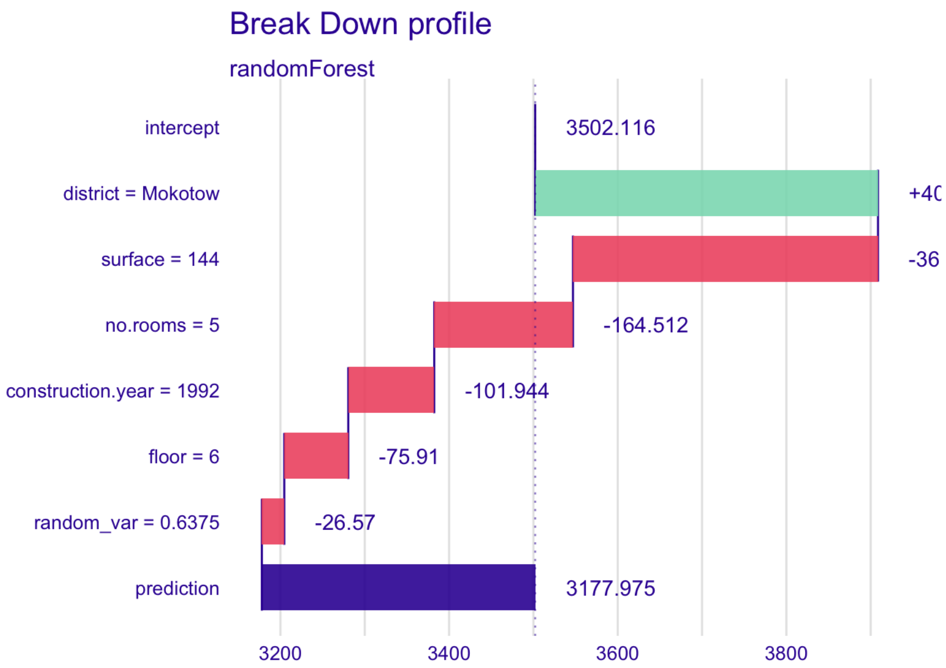
Note that the increase or decrease in estimated price depends on the order we set the variables.

```
bd_rf <- predict_parts(explainer_rf, new_observation =
         apartments[5,])
bd_lm <- predict_parts(explainer_lm, new_observation =
         apartments[5,])

plot(bd_lm)
```



```r
plot(bd_rf)
```



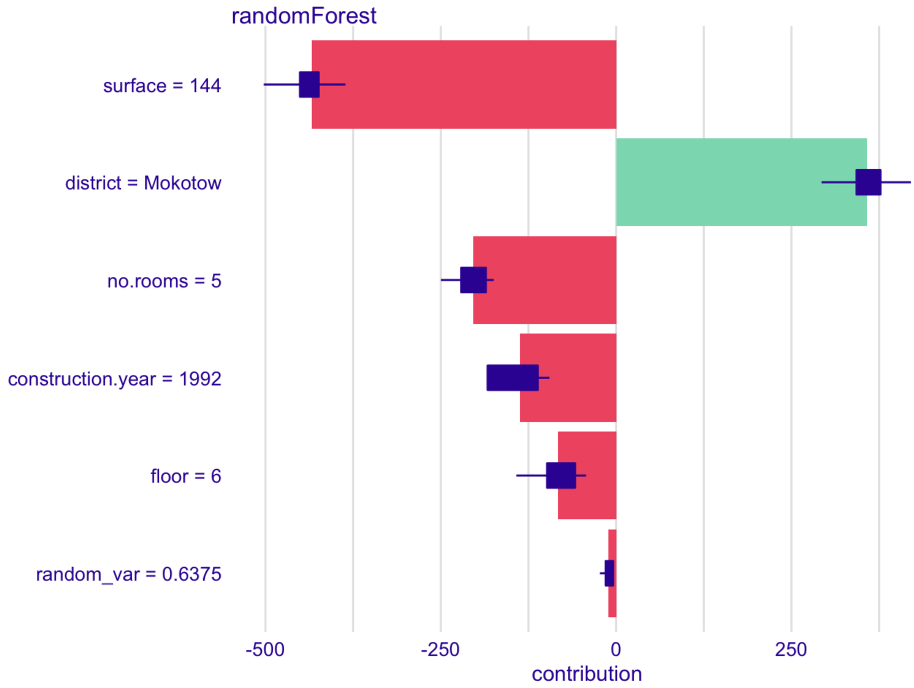
Break-down plots are model-agnostic, compact and easy to understand. The main drawback is that the order of variables becomes important if features are correlated or the if the model contains interaction terms.

9.2 Shapley Values

Exploring Shapley values is one way to solve the break down plot problem, as it takes the effects of variable order into consideration. In fact, we can think of Shapley values as the average effect of each variable for a number of break-down plots with random variable orderings.

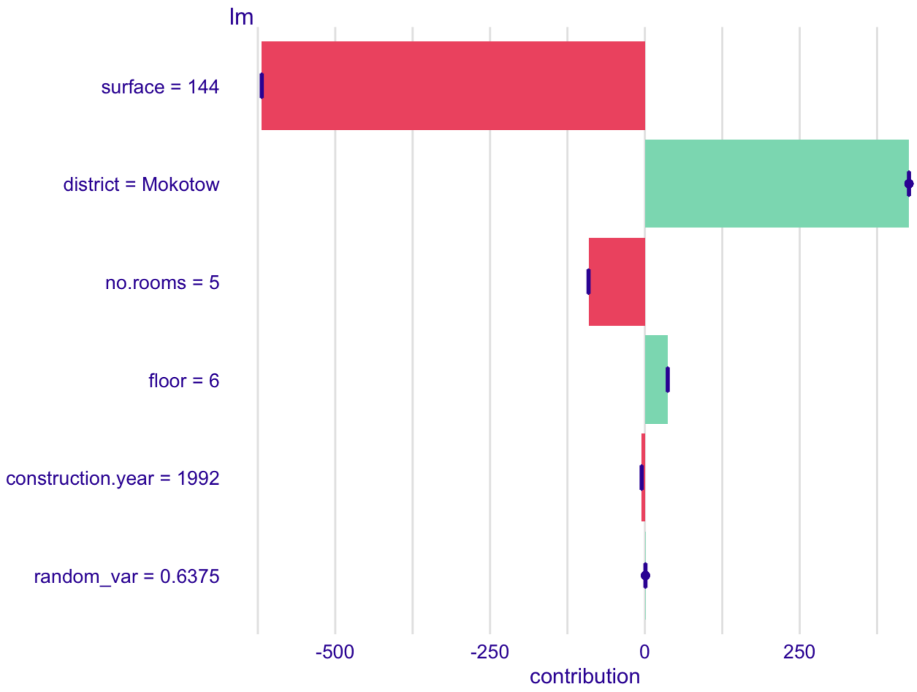
Shapley values originate in game theory and aim to solve the problem that given varying contributions from different players in a cooperative situation, how might the surplus gains be most fairly distributed? Similarly, we can look at a model and ask, how can we best score variables in terms on significance of contribution to predictions?

```
shap_rf <- predict_parts(explainer_rf, new_observation =
        apartments[5,],
                     type = "shap", N =50,
                     B =50)
plot(shap_rf)
```




```
shap_lm <- predict_parts(explainer_lm, new_observation =
        apartments[5,],
                        type = "shap", N =50,
                        B =50)


plot(shap_lm)
```



SHAP provides an intuitive solution to the break down plot variable ordering problem, however, for large models, the calculations required may be overly time-consuming.

9.3 Local Interpretable Model-agnostic Explanations (LIME)

In models with thousands or even millions of features, SHAP and BD are not appropriate: calculating the Shapley Values for this many features implies a prohibitively large number of iterations and breaking down a prediction in millions of tiny parts might end up being meaningless.

Large feature sets are very in genomics and when working with text or image data. In such cases, sparse explanations with a small number of variables offer a useful alternative. One of these sparse explainer is the Local Interpretable Model-agnostic Explanations (LIME) method.

The intuition behind LIME is best illustrated with the image below. In this scenario, we want to explain how the complex model behaves for the data point represented by the cross. The complex model's predictions are represented by the green and orange areas.

The LIME method consists in creating an artificial dataset around the data point we need to explain. Then, we can fit an explainable model on this artificial dataset to locally approximate the predictions of the black-box model. In the image below, the dotted line represents a linear model fitted on the artificial dataset.

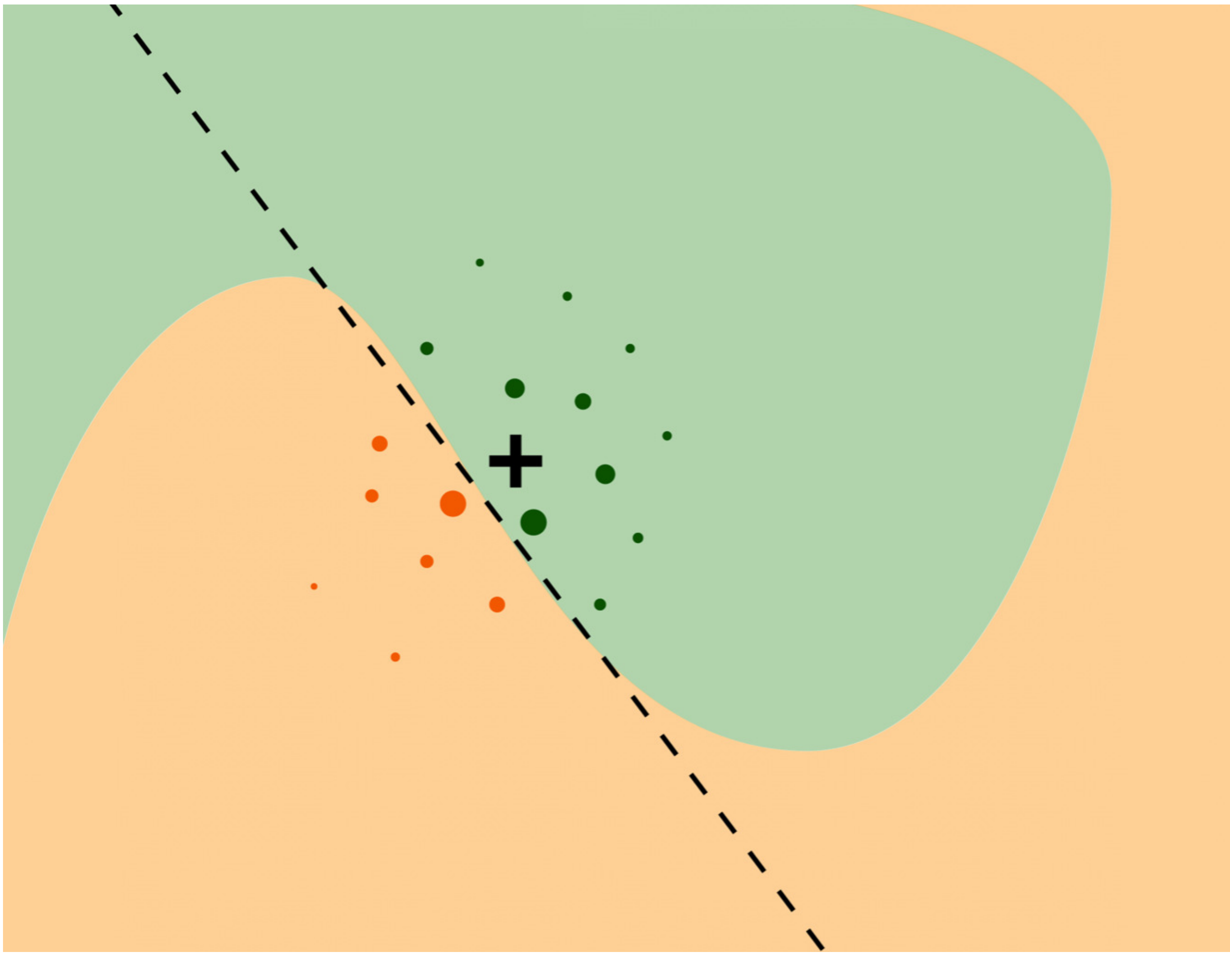


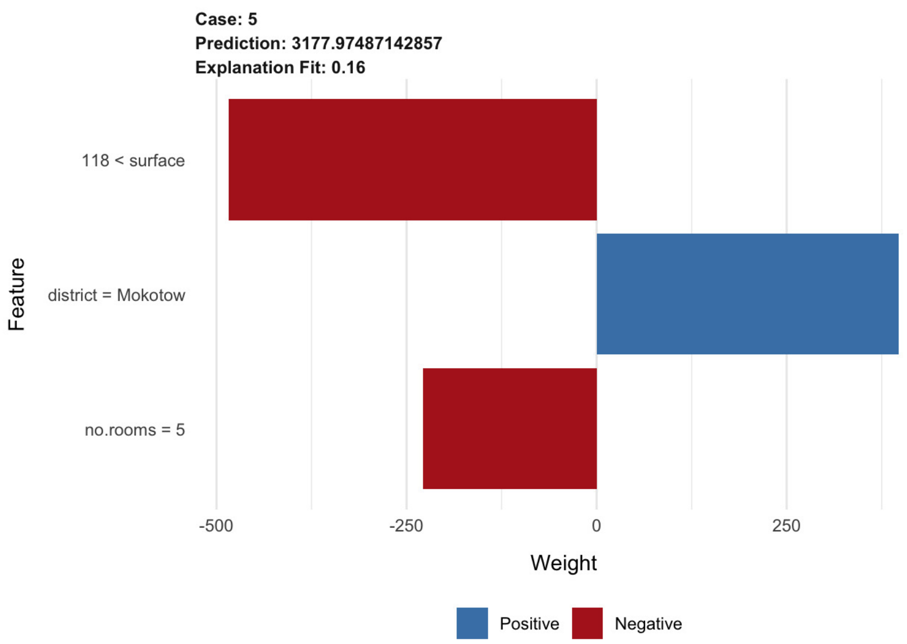
Illustration of the intuition behind LIME. Explanatory Model Analysis, Przemyslaw Biecek and Tomasz Burzykowski[6]

[6] https://ema.drwhy.ai/LIME.html

```
set.seed(220808)
library(DALEXtra)
library(lime)

model_type.dalex_explainer <- DALEXtra::model_type.dalex_explainer
predict_model.dalex_explainer <-
        DALEXtra::predict_model.dalex_explainer

library(localModel)
lime_apt5 <- predict_surrogate(explainer = explainer_rf,
                               predict_model.dalex_explainer,
                               model_type.dalex_explainer,
                                 new_observation = apartments[5, ],
                                 n_features = 3,
                                 n_permutations = 1000,
                                 type = "lime")
#> Warning in explain.data.frame(x = new_observation, explainer =
#> lime_model, : "labels" and "n_labels" arguments are ignored when
#> explaining regression models
plot(lime_apt5)
```



Chapter 10

Exercise solutions

6.1.1 Exercise

str(mp_rf)

6.2.1 Exercise

plot(mp_lm, mp_rf, geom = “boxplot”)

7.0.1 Exercise

plot(md_rf, md_lm)

plot(md_rf, md_lm, variable = “construction.year”)

plot(md_rf, variable = “y”, yvariable = “abs_residuals”)

plot(md_rf, variable = “ids”)

8.2.3 Exercise

plot(model_profile(explainer_lm, variables = NULL, type = “partial”))

plot(model_profile(explainer_rf, variables = NULL, type = “partial”))

11 Resources

11 Resources

This workshop was largely inspired by the work of other people who contributed to open source resources.

- Szymon Maksymiuk, Alicja Gosiewska, Przemysław Biecek (2021), **Landscape of R packages for eXplainable Artificial Intelligence**[7]
- Christoph Molnar (2022), **Interpretable Machine Learning: A Guide for Making Black Box Models Explainable**[8]
- Przemyslaw Biecek, Tomasz Burzykowski (2020), **Explanatory Model Analysis: Explore, Explain, and Examine Predictive Models. With examples in R and Python**[9]
- Seungjun (Josh) Kim (2021), **Explainable AI (XAI) Methods Part 1 — Partial Dependence Plot (PDP)**[10]
- Przemyslaw Biecek (2020),**DALEX v 1.0 and the Explanatory Model Analysis**[11]
- David Dalpiaz(2022), **Applied Statistics with R**[12]

[7] https://arxiv.org/pdf/2009.13248.pdf
[8] https://christophm.github.io/interpretable-ml-book/pdp.html
[9] https://ema.drwhy.ai/breakDown.html
[10] https://towardsdatascience.com/explainable-ai-xai-methods-part-1-partial-dependence-plot-pdp-349441901a3d
[11] https://www.r-bloggers.com/2020/02/dalex-v-1-0-and-the-explanatory-model-analysis/
[12] https://book.stat420.org/model-diagnostics.html

```
###########################
# Explainable AI workshop #
###########################

# Environment set up -----------

# Install packages
packages_to_install <- c('DALEX', 'randomForest', 'dplyr', 'localModel',
                         'DALEXtra')
install.packages(packages_to_install)

# Load packages
library(DALEX)
library(DALEXtra)
library(randomForest)
library(tidyverse)
library(DALEXtra)
library(lime)
library(localModel)

# load data
data("apartments")
data("apartments_test")

# transform data
data_transform <- function(data){
  data %>%
    mutate(random_var = runif(dim(data)[1]),
           no.rooms = as.factor(no.rooms),
           floor = as.factor(floor))
}

apartments <- apartments %>% data_transform()

apartments_test <- apartments_test %>% data_transform()

head(apartments)
```

```
# Make two models --------------
set.seed(220808)

# train RF model
apartments_rf_model <-
  randomForest::randomForest(m2.price ~ .,
                             data = apartments)

# train LM model
apartments_lm_model <-  lm(m2.price ~ .,
                           data = apartments)

# predict on test set using RF
predicted_rf <- predict(apartments_rf_model,
                        apartments_test)

# predict on test set using LM
predicted_lm <- predict(apartments_lm_model,
                        apartments_test)

# Build the explainers --------------
## explainer for LM


## explainer for RF model



# Model performance -------------------
## Make the model performance object with model_performance

## use the str() function to explore the object

## Plot residuals

### plot one graph

### plot both on one graph
```

try with geom = "histogram"

try with geom = "boxplot"

Model Diagnostics ------------

model_diagnostics object with model_diagnostics()

plot diagnostics: y against y_hat

plot y against residuals (residuals = observed - predicted if residuals > 0 model underestimates)

plot construction.year against residuals

try different variables

try both model for one variable

try absolute residuals

Global explainer -------------
Feature Importance

Partial Dependency Plots (PDP)
use the model_profile() function
categorical vars

Continuous variables
pdp one continuous var

pdp all variables

Accumulated Local Effects (ALE)

```
### create ALE curve with model_profile()

## plot both together

# Local explainer -------------
## Break down of predictions with predict_parts()

## Shapley Values with predict_parts()


# LIME: Local Interpretable Model-agnostic Explanations --------
set.seed(220808)
library(DALEXtra)
library(lime)
library(localModel)

# create a model_type

# create surrogate model

# explain prediction with predict_surrogate()

# Try for another flat

# Try another LIME implementation
```

Thank you!