POLITECNICO DI TORINO

Mathematical Engineering
Master's Degree

Master's Thesis

Variable importance in modern regression with application to supplier productivity analysis

Supervisor:
Prof. Mauro Gasparini

Author:
Simona Fiorito

A.A. 2017-2018

A me stessa

Abstract

In the last decades the amount of available data increased extremely and the data analytics field evolved and expanded accordingly. Starting from linear regression many other algorithms have been developed, leading to the spread of machine learning methods. Such methods are able to implement more complex regression models, achieving higher accuracy, but often at the cost of interpretability. In this master's thesis a study about the interpretation of some regression models has been carried out. Methods to evaluate variable importance and to display efficiently feature effects on the variable of interest are explained; the respective R functions that implement such methods are illustrated through the presentation of a case study developed by the author during an internship in Tetra Pak.

Acknowledgements

Si ringraziano i miei amici per essere rimasti tali, nonostante la mia latitanza dovuta agli impegni universitari.

La mia famiglia per il loro sempre presente supporto, affetto e pazienza.

I colleghi di Tetra Pak per il calore mostratomi e per la loro simpatia e stravaganza, che mi hanno accompagnato durante il percorso di stage rendendolo un'esperienza unica, da portare nel cuore, ma soprattutto per le immense e innumerevoli risate.

Grazie ai miei compagni universitari che mi hanno accompagnato in questo percorso dall'inizio alla fine, senza i quali non avrei raggiunto questo traguardo così velocemente.

Addio e grazie per i pesci.

Disclaimer

Due to confidentiality reasons only a simplified version of the real problem is reported in this thesis and reported results are illustrative examples, which do not reflect the real outcome of the work.

Contents

List of Figures

Chapter 1

Introduction

Technological progress is leading the world, and companies in particular, to a true digital revolution. We have entered the Big Data Era, so called because of the huge amount of available data, that is further increasing. With the rise in volume of data to analyse, it was necessary to develop new techniques and methods to handle and interpret them, bringing increasingly attention to machine learning algorithms.

Recently, part of the attention moved to the interpretation of such algorithms behaviour. Indeed, many of them are able to deliver pretty accurate results, however, without providing an explanation of the logic behind them. Such algorithms are called black box methods.

Being able to understand the variable dynamics carrying to the results, would allow gaining insights on complex systems impossible to describe without resorting to black box methods. However, not managing to unravel such complexity does not implicate just giving up on further knowledge. Indeed, there are many reported cases of black box models applied to prediction and classification problem that resulted to not work properly due to biases hidden in the learning process [12]. Thus, it should not be placed blind trust in such algorithms, without carrying out some preliminary analysis to assess which logic driving them.

In Chapter 2 is provided a description of some of the most applied regression models with relative measures useful to asses variable importance. Thereafter, with the purpose of deliver insights as effectively as possible, Chapter 3 focuses on visualisation tools for interpretability. Chapter 4 presents a case study along with applications in R of the previous methods. Finally, in Chapter 5 a summary is provided.

Chapter 2

Variable importance in regression models

Regression analysis is a substantial branch of statistical modelling. Its purpose is the study of relationships between a variable of interest y, also called target variable or response variable, and a set of p explanatory variables, $\{x_k\}_{k=1}^{p}$, also called features or independent variables. In particular, thanks to regression analysis is both possible to predict the target variable value, given the values of the features, and to understand the influence of each independent variable on the one of interest. Different regression models accomplish these two tasks to different extents.

Each regression method describes the relationships between the variables by mean of some function $y = f(x)$, characterised by M parameters $\Lambda = \{\lambda_m\}_{m=1}^{M}$. Given a dataset $(Y, \mathbf{X})$ an approximation $\hat{y} = \hat{f}(x)$ of such function can be computed, assigning to the parameters the values $\hat{\Lambda} = \{\hat{\lambda}_m\}_{m=1}^{M}$, which minimise a loss function $L(y, \hat{f}(x))$ that quantifies the prediction error of $\hat{f}(x)$:

$$\hat{\Lambda} = \arg\min_{\Lambda} \sum_{i=1}^{N} L(Y_i, f(\mathbf{X}_i; \Lambda)). \tag{2.1}$$

The methods presented below are briefly described, more detail can be found in [4, 8].

2.1 Linear regression

Linear regression has been the very first form of regression analysis. This model approximates the relationship - that is not necessarily linear - between the target and the independent variables by means of the following linear (in the β's) equation:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \epsilon, \tag{2.2}$$

where Y and X_k, with $k = 1, ..., p$, are the vectors collecting the observed values of the target and the independents variables, respectively; β_k, with $k = 1, ..., p$, are the unknown constant coefficients of the model and ϵ is the approximation errors vector. By means of a given sample $(Y, \mathbf{X})$, where $\mathbf{X} = \{X_k\}_{k=1}^p$, it is possible to train the model and to compute an estimate of the coefficients, $\hat{\beta}_k$, with $k = 1, ..., p$, obtaining

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + ... + \hat{\beta}_p x_p. \tag{2.3}$$

Substituting in (2.3) $\{x_k\}_{k=1}^p$ with new observed values of the features we obtain an estimate $\hat{y}$ of the target variable. Furthermore, the coefficient estimates give us insights on the relationship between the respective variable and the one of interest. Indeed, β_k indicates by how much the value of the target variable increases in average, when x_k rises by one unit and all the other variables remains unvaried.

An issue arises when the features are measured in different units, as it usually is, because each coefficient is expressed in the relative variable unit. Hence, it is not possible to compare them directly in order to analyse the relative effects of the independent variables. One questionable solution is to standardise the regression coefficients [2] as

$$b_k = \hat{\beta}_k \frac{s_k}{s_y}, \tag{2.4}$$

where s_k and s_y are the standard deviations of x_k and y, respectively. The interpretation is similar to the previous one, but units are replaced by standard deviations. Thus, the standardised coefficients represent the change in standard deviations in the target variable, when x_k varies of one standard deviation. Therefore, larger is the absolute value of b_k, more is the importance and influence of x_k on y.

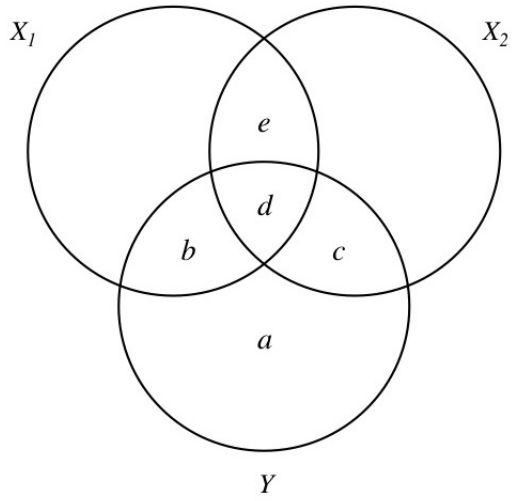


Figure 2.1: Variance distribution over two independent variables and a target variable. (a) is the unexplained variance in the target variable Y. (b) is the variance in Y predicted only by X_1. (c) is the variance in Y predicted only by X_2. (d) is the variance in Y explained by both variables X_1 and X_2. (e) represents covariance between X_1 and X_2 not related to Y. It is noted that (d) and (e) are not void only if X_1 and X_2 are correlated.

This approach has been largely criticised for different reasons, but in particular, because standardised coefficients combine standard deviation with estimated effects, making it very difficult to interpret them. Alternatively, other coefficients are often taken into account to quantify the relative importance of the features [11]. Among them there are partial and semi-partial correlation coefficients.

For the sake of clarity, suppose to have just two independent variables x_1 and x_2. The Venn diagram in Figure 2.1 represents how the variance is shared between them and the response variable. The partial correlation coefficient indicates the correlation between y and x_i without taking into account any contribution by x_j:

$$r^2_{yx_i \cdot x_j} = \left(\frac{r_{yx_i} - r_{x_i x_j} r_{yx_j}}{\sqrt{1 - r^2_{x_i x_j}} \sqrt{1 - r^2_{yx_j}}} \right)^2 . \tag{2.5}$$

The same measure can be calculated by means of the coefficient of deter-

mination R^2. Let $R^2_{y|x_i x_j}$ be the coefficient of determination of the model $y = \beta_0 + \beta_i x_i + \beta_j x_j$. Then, (2.5) is equivalent to

$$r^2_{yx_i \cdot x_j} = \frac{R^2_{y|x_i x_j} - R^2_{y|x_j}}{1 - R^2_{y|x_j}}. \tag{2.6}$$

Nevertheless, this coefficient neglects the variance in y explained by x_j. Instead, semi-partial correlation coefficients consider it and remove just the influence of x_j over x_i, obtaining

$$r^2_{y(x_i \cdot x_j)} = \left(\frac{r_{yx_i} - r_{x_i x_j} r_{yx_j}}{\sqrt{1 - r^2_{x_i x_j}}} \right)^2, \tag{2.7}$$

equivalent to

$$r^2_{y(x_i \cdot x_j)} = R^2_{y|x_i x_j} - R^2_{y|x_j}. \tag{2.8}$$

It actually measures the proportion of R^2 associated to x_i, aside from the other independent variable taking part to the model. For this reason semi-partial correlation coefficients are more suitable indicators of variable importance compared to partial correlation coefficients.

Going back to the general case with p independent variables, for each variable x_i its relative influence can be computed, extending (2.8) to

$$r^2_{y(x_i \cdot \mathbf{x}_{-i})} = R^2_{y|\mathbf{x}} - R^2_{y|\mathbf{x}_{-i}}, \tag{2.9}$$

where $\mathbf{x}$ concisely denotes $\{x_k\}_{k=1}^p$ and $\mathbf{x}_{-i}$ denotes the whole set of independent variables except for x_i.

The main issue of linear regression is that it is based on several assumptions, which usually are not satisfied by real data - above all the linearity hypothesis stands out. For these reason often is better to reach for other methods and to abandon the use of r^2-like measures.

2.2 Regression trees

Another very simple regression models are the regression trees. The idea is to split the feature space in a partition $\mathcal{R} = \{R_w\}_{w=1}^W$ by means of a set of rules (as illustrated in Figure 2.2), and in each region R_w the target variable

is approximated by $\hat{y}_w$, that often is the average value of the y observations belonging to the region itself. The rules are ranked and each one fixes a value that splits a feature domain. Each rule correspond to an internal node of the tree, also called splitting node, and the total number of rules defines the tree depth. A tree can be formally defined as

$$T(x; \mathcal{R}) = \sum_{w=1}^{W} \hat{y}_w \chi_{R_w}(x) \tag{2.10}$$

where χ_{R_w} is the characteristic function (also called indicator function) of R_w, defined as,

$$\chi_{R_w}(x) := \begin{cases} 1 & \text{if } x \in R_w \\ 0 & \text{if } x \notin R_w \end{cases} \tag{2.11}$$

Regression trees are easily interpretable models and variable importance is stated by the set of rules. Indeed, if a variable is not present in any of the rules, then it has not much influence on the target one, and, among the variables that participate in the rules, the higher is the rule ranking, the more important is the feature.

On the other hand, this model is not very robust: a small variation in the given sample can lead to very different fitted trees. To overcome this issue, methods have been developed, that fit not just a tree, but a collection of them, in order to gain robustness: this approach is called ensemble learning.

2.3 Ensemble learning methods

Ensemble methods combine a set of simple base models to boost their performances, at the expense of interpretability. In the following sections we will consider regression trees as base model.

2.3.1 Boosting

The boosting approach consists in stagewise methods that at each step learn from the errors of the model trained at the previous step, adding to it a new regression tree to improve its performances. Formally, at each step j, a new

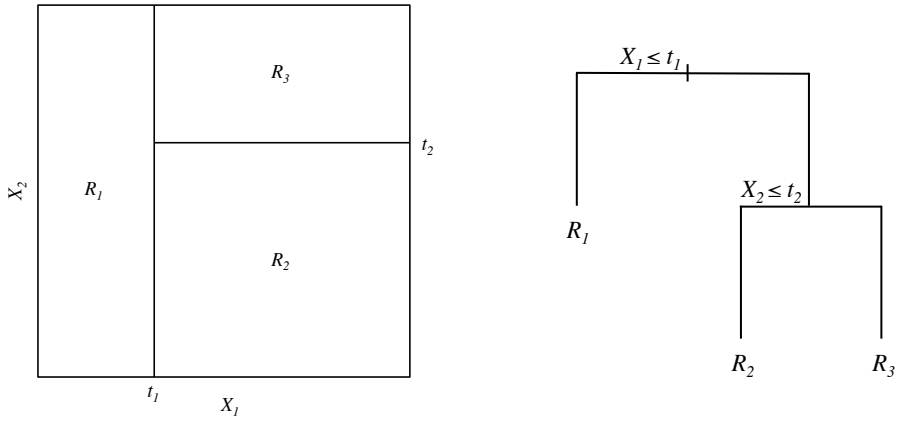


Figure 2.2: Two-dimensional regression tree example

tree is computed, solving the following optimization problem (equivalent to (2.1)), which aims to minimise the overall prediction error:

$$\hat{\mathcal{R}}_j = \arg\min_{\mathcal{R}_j} \sum_{i=1}^{N} L(y_i, f_{boost}^{j-1}(x_i) + T(x_i; \mathcal{R}_j)). \tag{2.12}$$

Usually, to solve this minimisation problem the gradient descent method is applied, and in such case the model is called gradient boosted trees.

Essentially, the first grown tree aims to predict the response value, then the second one aims to compensate for the errors of the former. Supposing that the loss function is the squared error,

$$L(y_i, f(x_i)) = \sum_{i=1}^{N} (y_i - f(x_i))^2,$$

then the second tree achieves his goal simply being fitted to the first tree residuals. Therefore, at the second iteration of the algorithm, the model consists in the sum of two trees, one predicting the target variable value, the other predicting the error committed by the former. Iterating the second step, every new tree is fitted to the residuals of the model developed up to that iteration. At the end we will obtain a model that is a sum of J trees:

$$f_{boost}^{J}(x) = \sum_{j=1}^{J} T(x; \mathcal{R}_j), \tag{2.13}$$

hence, the predicted values are given by the sum of the singular values predicted by each tree.

It should be noted that if each tree is composed of just one internal node, the model is additive and no interaction between variable is considered. Variable interactions can be represented by deeper trees, specifically, a model made up of trees with maximum depth equals to k can describe up to k-way interactions.

Differently from regression trees, gradient boosted models are hard to interpret, but a variable importance measure is available, that gives insights on the relationships between the features and the target variable. For each tree the importance of an independent variable x_k can be measured as

$$\zeta_k^2(T) = \sum_{m=1}^{M-1} \hat{\imath}_m^2 \chi(x^m = x_k), \tag{2.14}$$

where $M-1$ is the number of the tree internal nodes and $\hat{\imath}_m^2$ is the squared estimated improvement, expressed as squared error risk, gained in node m by splitting over a variable, such variable is denoted as x^m. In particular,

$$\hat{\imath}_m^2 = \min_t \imath_m^2(t) = \min_t \frac{w_{t_-} w_{t_+}}{w_{t_-} + w_{t_+}} (\bar{y}_{t_-} - \bar{y}_{t_+})^2, \tag{2.15}$$

where $\bar{y}_{t_-}$, $\bar{y}_{t_+}$ are the response means for the instances which have $x^m \leqslant t$ and $x^m > t$ respectively, and w_{t_-}, w_{t_+} are the corresponding total weights [3]. Indeed, it is possible to assign different importance, or weight, to the observations, such that instances with higher importance are more influential in the fitting process. When the weights are not assigned, each observation has same importance equals to 1. Hence, (2.14) sums for each internal node the squared estimated improvement given by x_k, that is different from zero only when $x^m = x_k$. This measure is not very useful, nor robust, for a single tree, but averaging it over the whole set of trees of a boosted model, it gains reliability and become a very useful interpretation instrument for boosting methods:

$$\zeta_k^2 = \frac{1}{J} \sum_{j=1}^{J} \zeta_k^2(T_j), \qquad \text{with} \quad T_j = T(x; \mathcal{R}_j). \tag{2.16}$$

2.3.2 Random forests

Another ensemble method are random forests. Differently from boosting, random forests build a set of trees that are decorrelated. Indeed, whereas

the goal of boosted models is to reduce bias, learning from the already grown trees, random forests grow each tree separately leaving the bias unchanged. However, the main difference between these two approaches is that boosted models are constituted by the sum of each tree outcomes, whereas random forests average the results of the grown trees:

$$f_{rf}^{J}(x) = \frac{1}{J}\sum_{j=1}^{J} T(x; \mathcal{R}_j), \tag{2.17}$$

where J is the number of trees generated.

The collection of trees is built by mean of the bootstrap method, that is a resampling technique that creates new datasets of the same size as the original one, just sampling it with replacement. Hence, each tree is fitted to a different dataset, in order to obtain different trees.

To introduce further variability, the tree growing algorithm is slightly modified. At each step a random features subset of size $m \leqslant p$ is sampled and the splitting variable must belong to it. In this way, when the most influent features do not belong to the random sample, the algorithm is forced to choose variables that otherwise would not be considered, because they are less influent.

Thanks to these characteristics, this method builds trees very different from each other, learning from all the variable available and broadening its knowledge. Therefore, averaging the tree outcomes, we obtain a model that overcomes the lack of robustness, intrinsic to singular regression trees.

Due to their structure similar to boosted methods, random forests are difficult to interpret as well, but the same measure (2.16) can be applied to gain insights on variable relationships. It is noted that, since the splitting variables are chosen every time over a different limited set of features, it is likely that every independent variable will be eventually picked for the splitting. Therefore, hardly some variable will have importance equal to zero, differently from boosted models, where such occurrence is not unusual.

Moreover, another measure that quantifies the variable prediction strength can be calculated for random forests. When the algorithm build a tree, some observations from the original sample are left out, since they were not picked in the bootstrapping process. Such observations constitute the so called out-of-bag sample (OOB). Each tree is tested twice, first with the OOB sample and then with the same sample but randomly permuting the values for a variable x_k. The resulting accuracy variation is averaged over all trees and

normalised by the standard deviation of such variations. The more the accuracy decays, the more the variable in exam is considered important.

Chapter 3

Opening the black box

The ensemble models presented in the previous chapter belong to a group of models called black box models. This name is due to their characteristic of not being self-explanatory. It is difficult to trace back which features dynamics brought to the obtained output. In order to understand such dynamics we studied the ideas implemented by the R package `'iml'`; thereafter we implemented an alternative method more suitable for our case study.

3.1 `iml` package

The best way to understand how a single variable influences the response variable is visualising this relation with a plot, such that it is readable also by non-experts. The `'iml'` package makes it possible in three different ways, which are presented below.

3.1.1 Individual conditional expectation curves and partial dependence plots

Individual conditional expectation (ICE) curves and partial dependence plots (PDPs) are the first two options [3, 10], very simple and intuitive, but reliable only if the input variables are uncorrelated.

They evaluate the target variable for each observation multiple times, varying the value of the feature x_i whose effects are going to be plotted. Therefore, a distinct analysis is carried out for each instance and the response evolution for different values of x_i is obtained as result.

Individual conditional expectation curves plot a line per observation showing how the prediction is affected by the variation of x_i. Instead, partial dependence plots summarise all the lines in one representing the average:

$$\bar{f}_i(x_i) = E_{-i}\left[\hat{f}(X)\right] = E_{-i}\left[\hat{f}(x_i, \mathbf{x}_{-i})\right] = \int \hat{f}(x_i, \mathbf{x}_{-i}) p_{-i}(\mathbf{x}_{-i}) d\mathbf{x}_{-i}, \quad (3.1)$$

with $p_{-i}(\mathbf{x}_{-i})$ marginal probability density of $\mathbf{x}_{-i}$, defined as

$$p_{-i}(\mathbf{x}_{-i}) = \int p(\mathbf{x}) dx_i, \quad (3.2)$$

where $p(\mathbf{x})$ is the joint probability density of all features. Hence, to detect the relationship between the feature in exam x_i and the response variable, the model response is marginalised over the distribution of the other features $\mathbf{x}_{-i}$. The result is a function that depends only on x_i, including its interactions with the others variables.

Instead of the integral, the following averaging is implemented:

$$\bar{f}_{i,PDP}(x_i) = \frac{1}{N} \sum_{j=1}^{N} \hat{f}(x_i, x_{j,-i}), \quad (3.3)$$

where N is the number of observations and $x_{j,-i}$ is the vector collecting the feature values of the j-th observation except for the i-th variable. For large datasets this formula can be computationally expensive, due to the fact that for each distinct value of x_i the model is evaluated N times.

Since averaging could hide some non-negligible effects and individual conditional expectation plots potentially deliver more information, it could be a valid alternative to plot both solutions together, as shown in Figure 4.10.

The main problem with these two approaches is that they are both based on the assumption that there is no interaction or correlation between the features, which is not so obvious. Taking this for granted carries the risk of evaluating feature value combinations that are unrealistic, introducing bias into the analysis. Indeed, the conditional density probability of the feature in exam is neglected, and equal consideration is given to every value combination whether it is likely or not. For example, let assume we are analysing a dataset collecting physical information about a population, such as height and weight, which are strongly correlated. Evaluating ICE curves and PDPs, we would consider unlikely value combinations such as height 1.95 m and weight 45 kg, and those would contribute to the results as much as realistic combination, that have been actually observed.

3.1.2 Accumulated local effects plots

Accumulated local effects (ALE) plots overcome this problem [1, 10]. Instead of carrying out an instance-wise analysis, they perform a neighbourhood-wise analysis. Specifically, the analysed feature domain is split in intervals and the variable influence is evaluated locally in each of them. Furthermore, differently from PDPs, they do not plot the average response values, but rather the average variations of the response variable, delivering the pure effects of the variable in exam, without mixing them with the ones of all the other features.

Let x_i be the analysed variable, $\mathcal{I} = \{I_k\}_{k=1}^{K}$ a partition of x_i domain, $\mathfrak{J}_i(k) = \{j\}$ the set of indexes corresponding to the observations for which $x_{j,i} \in I_k$ and $n_i(k)$ its cardinality. For each instance x_j, with $j \in \mathfrak{J}_i(k)$, the model is applied twice, setting first $x_{j,i}$ equals to the right endpoint of I_k, $z_{k,i}$, and then to the left one, $z_{k-1,i}$. Thereafter, the difference between the two predicted values is calculated as

$$\delta_{I_k,j} = f(z_{k,i}, x_{j,-i}) - f(z_{k-1,i}, x_{j,-i}). \tag{3.4}$$

Averaging the obtained differences $\{\delta_{I_k,j}\}_{k,j}$ over each interval I_k, we obtain the set of local average effects $\{\delta_{I_k}\}_k$.

$$\delta_{I_k} = \frac{1}{n_i(k)} \sum_{j \in \mathfrak{J}_i(k)} \delta_{I_k,j}. \tag{3.5}$$

Finally, the accumulated local effects are calculated for each interval accumulating the local average effects of the previous intervals.

$$\tilde{f}_{i,ALE}(x_i) = \sum_{k=1}^{K_i} \delta_{I_k}, \tag{3.6}$$

where K_i is the index of the interval I_{K_i} to which the value x_i belong.

Thereafter, to have mean effect zero, (3.6) is centred as

$$\tilde{f}^*_{i,ALE}(x_i) = \tilde{f}_{i,ALE}(x_i) - \frac{1}{N} \sum_{j=1}^{N} \tilde{f}_{i,ALE}(x_{j,i}). \tag{3.7}$$

The main advantages of this method are two:

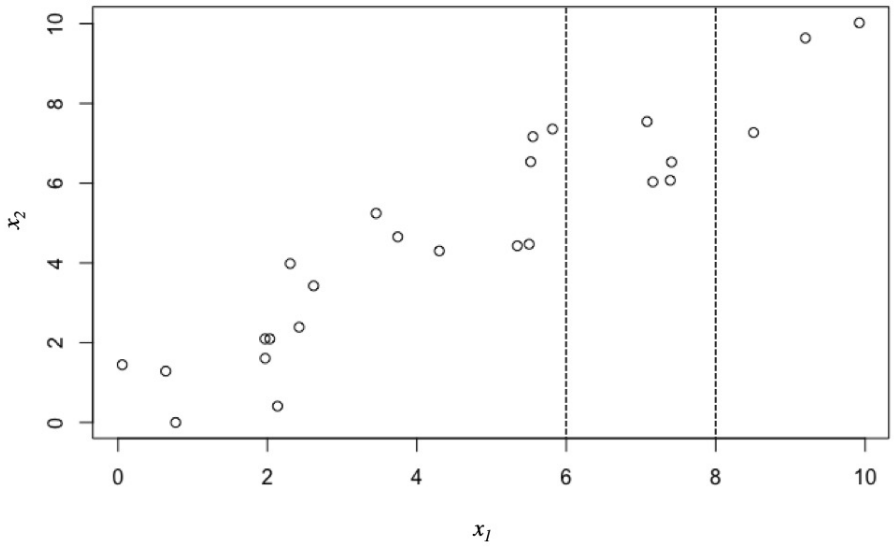


Figure 3.1: For example, to evaluate the local effect in the interval $[6, 8)$, only the observations belonging to it are considered, predicting for each of them the value of y with $x_1 = 8$ and $x_1 = 6$. Then the differences are computed, averaged and accumulated with the results obtained in the previous intervals. Doing so we avoid to consider unrealistic data combinations such as $(x_1, x_2) = (1, 7)$.

- the risk of considering unrealistic data instances is neutralized by shifting the analysis from the entire domain of the variable for each instance to local evaluations within neighbourhoods;
- the plotted effects are pure and not mixed with potentially correlated features, due to the fact that the average is computed for the variations of the predicted values, and not for the values themselves as in partial dependence plots.

3.2 Partial cumulative differences plots

We picked some characteristics of the plots presented above and rearranged them, designing partial cumulative differences (PCD) plots. The intuition of ALE plots consisting in shifting the focus from the predicted values to their variations has been preserved. Nevertheless, their local analysis could result quite limiting in presence of small and imbalanced datasets. When it is reasonable to assume that the risk to introduce unrealistic instances is negligible, an approach similar to the one introduced by PDPs could be more efficient. For these reasons we developed a new tool that allows displaying partial cumulative differences in different ways, computing mean, median, standard deviation and percentiles as best suits the needs.

A pseudo-code of the function is shown in Algorithm 1 and an implementation in R is reported in Appendix A. The inputs to the function are:

- `dataset`: the analysed data frame;
- `variableName`: a string containing the name of the variable in exam;
- `fittedModel`: an object of the class `randomForest` or `gbm`;
- `meanMedian`: a string specifying whether the mean or the median of the effects should be plotted;
- `percentile`: the lower percentile that we want displayed together with its symmetrical with respect to the median, if `NULL` (default value) the standard deviation is plotted;
- `categorical`: a boolean variable specifying whether the variable in exam is categorical or continuous (default).

Algorithm 1 pcdPlot pseudocode

```
1: function pcdPlot(dataset, variableName, fittedModel,
   meanMedian, percentile = NULL, categorical = FALSE)
2:     if categorical==TRUE then
3:         domain ← distinct observed values of variableName
4:     else
5:         domain ← equispaced array over variableName domain
6:     j ← 1
7:     for i in domain do
8:         dataset[variableName] ← i
9:         predicted ← apply fittedModel to dataset
10:        if not first iteration then
11:            difference ← predicted - predictedOld
12:            m ← get meanMedian of difference
13:            effect[j] ← effect[j-1] + m
14:            if percentile==NULL then
15:                sd ← get standard deviation of difference
16:                upper[j] ← effect[j-1] + sd
17:                lower[j] ← effect[j-1] - sd
18:            else
19:                p1 ← get percentile of difference
20:                p2 ← get percentile of difference
21:                upper[j] ← upper[j-1] + p1
22:                lower[j] ← lower[j-1] + p2
23:        predictedOld ← predicted
24:        j ← j + 1
25:    plot effect, upper, lower and variableName distribution vs.
   domain
```

The model is applied over each instance multiple times, varying the value of the analysed feature over all its domain. Every time the value of the variable in exam changes from η to $\eta + \Delta$ the difference between the new predicted values and the ones calculated at the previous step, with $x_i = \eta\mathbb{I}$, is calculated for every instance (line 11). At lines 12-13 the mean (or the median) of these variations is evaluated and accumulated with the previous values. In the same way, at lines 14-22, percentiles (or standard deviation)

are computed. Finally, before restarting the cycle, the predicted values are stored in order to be compared to the ones evaluated in the next step.

For a more complete evaluation, in order to avoid misleading interpretations, along the x axis it is provided a boxplot, showing the distribution of values of the variable in exam in the original dataset.

When the analysed variable is categorical, it takes every value belonging to its domain and, differently from continuous ones, a barplot is displayed instead of a boxplot. In the following chapter some examples are reported (Figures 4.11, 4.12, 4.13, 4.14).

Chapter 4

Case study

4.1 Tetra Pak

In 1951 Ruben Rausing founded Tetra Pak, an innovative food packaging company destined to become a market leader company. The very first prod-

Figure 4.1: Tetra Pak logo

uct developed was a tetrahedric carton for milk packaging, but over time the company broadened its horizons, developing not only packaging, but also processing and distribution solutions for a huge variety of beverages and food products. Today Tetra Pak is the world's leading food packaging company delivering end-to-end solutions to thousands of customers worldwide.

Figure 4.2: A set of different Tetra Pak packages

4.1.1 Case study description

Due to confidentiality reasons only a simplified version of the real problem is reported in this thesis.

For its production needs, Tetra Pak relies on a worldwide network of suppliers, buying from one or more of them in each country where it operates. The variable of interest is the number of successfully delivered orders from suppliers to Tetra Pak. In particular we want to know which variables affect it, in which way, and whether the socio-economic and geographic framework of the different countries plays any role. In this simplified version of the problem, the assumption done on the response variable is that the higher the better.

4.2 Data collection and variables selection

4.2.1 Internal data

The first source of data was a company's internal database, containing suppliers data. Tetra Pak collects data from its suppliers, but not all of them are sharing information and sometimes data are not accurate enough to guarantee reliability. For these reasons, the amount of missing data is considerable and part of the data available is based on estimations. Futhermore, some information are manually collected from people, hence human errors can occur. Therefore, it was necessary a prior cleansing process delivered by means of SQL scripts.

In the following, a summary description of these data is provided. For

each supplier we know:

- country;
- production capacity;
- percentage of capacity dedicated to Tetra Pak supplies;
- company type (a categorical variable which can assume five different values);
- production equipment level;
- number of raw material suppliers;
- number of successfully delivered orders to Tetra Pak.

The last one is the variable of interest and from now on we will refer to it as *sales*.

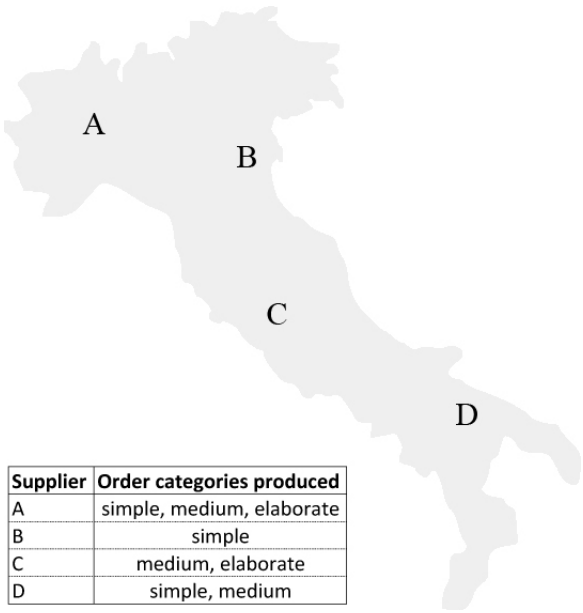
Furthermore, the orders belongs to different categories and for each supplier is known which categories of orders produces. Thanks to these information a co-supply index for each supplier s has been calculated:

$$\xi(s) = \sum_{c \in \mathbb{C}_s} (\mathcal{N}_c - 1)$$

where $\mathbb{C}_s$ is the set of order categories produced by the supplier s and $\mathcal{N}_c$ is the number of suppliers, operating in the same country of the supplier s, delivering the order category $c \in \mathbb{C}_s$. We subtract a unit from $\mathcal{N}_c$ to avoid counting the supplier s itself. Furthermore, must be noted that a supplier s' can be counted multiple times in the co-supply index of the supplier s if it shares with s more than one order category. An example is shown in Figure 4.3.

4.2.2 External data

In order to understand if the culture, the socio-economic framework and the geography of each country influence somehow the supplier sales, it was necessary to retrieve data describing them. All these data have been collected in a table in which each record corresponds to a country.



Supplier	Order categories produced
A	simple, medium, elaborate
B	simple
C	medium, elaborate
D	simple, medium

$$\xi(A) = (3-1)_{simp} + (3-1)_{med} + (2-1)_{elab} = 5$$

$$\xi(B) = (3-1)_{simp} = 2$$

$$\xi(C) = (3-1)_{med} + (2-1)_{elab} = 3$$

$$\xi(D) = (3-1)_{simp} + (3-1)_{med} = 4$$

Figure 4.3: Toy example of the calculation of co-supply indexes for suppliers in a country.

Starting from the climate, five logical variables (*equatorial, arid, warmTemperate, snow, polar*) has been created and the values has been assigned referring to the five main climates of the Köppen-Geiger climate classification [9]. In case of four or more climates present in the same country, just the main three has been considered.

All the other external variables come from the dataset of the World Bank named World Development Indicators. Among these indicators we selected the ones pertaining to the analysis and available for all the countries where Tetra Pak suppliers are present:

- *totTax17*: percentage of commercial profits that businesses has to pay as taxes overall in 2017;
- *laborTax17*: percentage of commercial profits that businesses has to pay as taxes on labor in 2017;
- *profitTax17*: percentage of commercial profits that businesses has to pay as taxes on profits in 2017;
- *otherTax17*: percentage of commercial profits that businesses has to

pay in 2017 as taxes that are not on labor or profits, such as vehicle and fuel taxes, property taxes, municipal fees and so on;

- *taxTime17*: hours spent in 2017 preparing and paying taxes, taking into account only labour taxes, sales taxes and the corporate income tax;
- *distFront17*: distance to frontier score in 2017, that indicates how far an economy is from the best performance recorded among all countries since 2005, evaluating several topics, such as starting a business, getting credit, registering property, etc.;[1]
- *easyBus17*: ease of doing business in 2017, that is a ranking based on the distance to frontier score, lower ranks corresponds to economies that facilitates more business operations [1]
- *last10yDeaths*: count of the battle-related deaths in the last ten years;
- *popGrowth16*: percentage that expresses the exponential growth rate[2] of population in 2016 compared to 2015;
- *compEduc16*: number of years of compulsory education in 2016;
- *PPP16*: Purchasing Power Parity conversion factor for 2016, that is the amount of local currency necessary to purchase the amount of services and goods in the local market that 1 U.S. dollar would purchase in U.S.A.;
- *gni16*: gross national income in 2016;
- *gniGr16*: annual percentage of growth of gross national income in 2016;
- *gnipc16*: gross national income per capita in 2016;
- *gds16*: gross domestic savings in 2016 expressed as percentage of gross domestic product;

[1] A more detailed description can be found at `http://www.doingbusiness.org/content/dam/doingBusiness/media/Annual-Reports/English/DB18-Chapters/DB18-DTF-and-DBRankings.pdf`

[2] $G = \ln\left(\frac{p_{2016}}{p_{2015}}\right)$, where p_t is the midyear population for year t.

- *gdp16*: gross domestic product in 2016;
- *gdpGr16*: annual percentage of growth of gross domestic product in 2016;
- *gdppc16*: gross domestic product per capita in 2016;
- *gdppcGr16*: annual percentage of growth of gross domestic product per capita in 2016;
- *gdpPPP16*: gross domestic product in 2016 converted by means of purchasing power parity rates to international dollars;
- *gdpPPPpc16*: gross domestic product per capita in 2016 converted by means of purchasing power parity rates to international dollars;
- *infl16*: inflation in 2016;
- *undernour15*: percentage of population whose nourishment does not reach the minimum amount of dietary energy requirements in 2015.

Variable selection

Overall we had 32 variables with no insights on eventual redundancy shared by some of them, especially the economic ones. To avoid useless computational complexity, we investigated whether we could reduce the number of features, selecting the more significant. To achieve this goal we consulted subject matter experts and applied an algorithm that studies the correlation between the variables using the mutual information index [14], revealing which are more significant for our variable of interest.

The selected variables are:

- *Company.Type*;
- *Equipment.Level*;
- *Production.capacity*;
- *Dedicated.capacity*;
- *Raw.material.suppliers*;
- *Co.supply.index*;
- *taxTime17*;
- *totTax17*;
- *laborTax17*;
- *distFront17*;

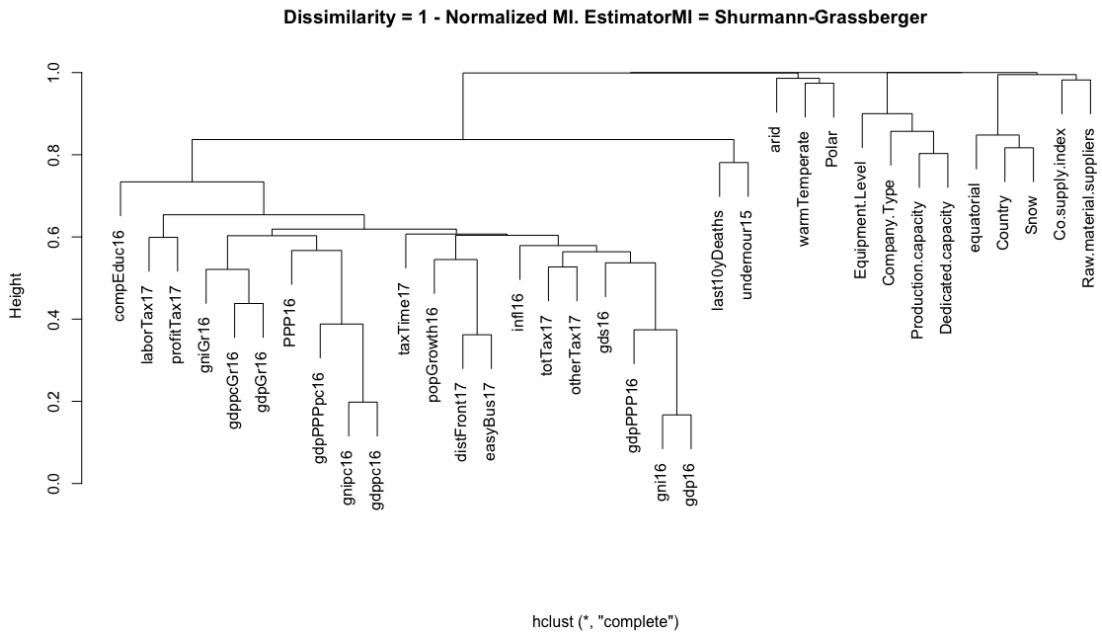


Figure 4.4: Correlation map

- *gnipc16*;
- *gdpPPP16*;
- *infl16*.

4.3 Data analysis

4.3.1 Regression analysis

Once the data were ready to be analysed, we assessed which regression model suited better the problem in exam. Linear regression models are not appropriate for a problem of such complexity, where the relationships between variables are unlikely to be linear. Whereas ensemble methods could be a useful resource.

Before fitting any model it is recommended to understand the distribution of the variables in exam. In particular it is important to analyse the response variable, in order to know its order of magnitude and to be able to assess whether the error committed by the model is negligible or not.

```
> summary(custData$Sales)
   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
  1.000   4.000  6.000  6.889   9.000 43.000
```

As we can see, *Sales* takes mostly values of the order of units, hence, an error of few units is quite high, but, in the same way, if a feature influences the target variable just by a unit it is quite significant.

Starting from boosting, there exists an R package, called `'gbm'`, that implements gradient boosted models. By mean of the function `gbm` it is possible to fit a model, giving as inputs:

- `formula:` a regression formula describing the model to fit;
- `distribution:` a string specifying the target variable distribution;
- `data:` the data frame to analyse;
- `n.trees:` number of trees to fit;
- `interaction.depth:` maximum depth of trees allowed, namely maximum interaction level allowed among variables, (e.g., when equal to 1 no interaction is considered and the model is additive);
- `cv.folds:` if greater than 1 the algorithm perform a cross-validation at each iteration and the number of folds is equal to `cv.folds`.

This function takes as inputs also many other variable that we are not going to discuss, but a complete explanation is available on CRAN website.[3].

In our case, we are interested to regress `sales` on all the other variables and the most suitable choice as distribution is the Poisson. As a first try, we stick to the default values for `n.trees` and `interaction.depth`, that are respectively 100 and 1. Running the following line we obtain a gbm object named `saleBoost`:

```
1 > saleBoost <- gbm(Sales~.,data=saleData, distribution ="poisson",
      n.trees =100, interaction.depth =1)
```

To understand whether a hundred trees are enough or not for a good fit, we can use the function `gbm.perf` that calculates the optimal number of iterations, taking as inputs a gbm object and the method used to compute the estimate. The methods available are three, out-of-bag estimate, out-of-sample estimate and cross-validation, but the last one can be considered the most reliable, as shown in [13]. Hence, if we want to run `gbm.perf` with cross-validation, we first need to run again the above code setting `cv.folds` different from the default value 1. The loss function used by the algorithm to assess the error depends on `distribution` value. When it is set as "poisson" the error is expressed as Poisson deviance [13].

```
2 > saleBoost <- gbm(formula = Sales~., data = saleData,
      distribution = "poisson", n.trees = 100, interaction.depth = 1,
      cv.folds = 10)
3 > gbm.perf(object = saleBoost, method = "cv")
4
5 [1] 37
```

The function `gbm.perf` along with the optimal number of iteration, give as output a plot as Figure 4.5 by default. It is possible to disable it by setting the input `plot.it` equal to `FALSE`.

Figure 4.5 shows the error evaluated on the training set (in black) and on the test set (in green) at each iteration. In blue it is shown the iteration that reports the lowest error on the test set, corresponding to the output displayed in console at line 5.

Since the estimated optimal number of iteration is well below 100, it is not necessary to increase `n.trees`. Furthermore, it is neither useful to reduce

[3]https://cran.r-project.org/web/packages/gbm/gbm.pdf

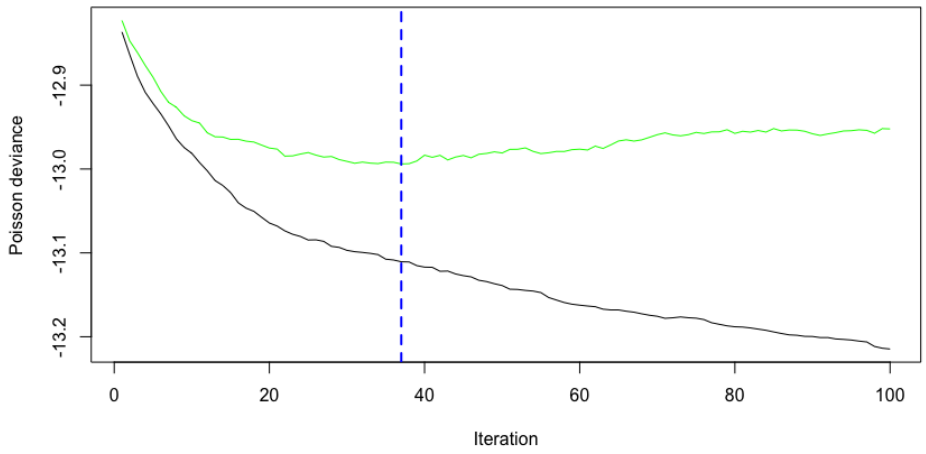


Figure 4.5: GBM performance plot

it, because we would fit again the same model, simply stopping the algorithm early, not growing the trees that provide no added value to the model. Hence, we would execute the same code two times without any further benefit.

Regarding the tuning of the variable `interaction.depth`, we decided to allow at most 3-way interaction, therefore, by means of cross-validation we assessed which value assign to it. A vector of size `n.trees`, collecting the error computed at each iteration as Poisson deviance, is stored as `cv.error` in any gbm object that has been obtained by running `gbm` with cross-validation. Calling the function summary on such vector we can compare the performances of different models.

```
> set.seed(1369)
> saleBoost1 <- gbm(Sales~., data=saleData, distribution
    ="poisson", n.trees =100, interaction.depth =1, cv.folds = 10)
>
> set.seed(1369)
> saleBoost2 <- gbm(Sales~., data=saleData, distribution
    ="poisson", n.trees =100, interaction.depth =2, cv.folds = 10)
>
> set.seed(1369)
> saleBoost3 <- gbm(Sales~., data=saleData, distribution
```

```
    ="poisson", n.trees =100, interaction.depth =3, cv.folds = 10)
>
> summary(saleBoost1$cv.error)
   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
 -12.99 -12.98 -12.97 -12.96 -12.96 -12.82
> summary(saleBoost2$cv.error)
   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
 -12.98 -12.97 -12.94 -12.94 -12.93 -12.83
> summary(saleBoost3$cv.error)
   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
 -12.98 -12.96 -12.93 -12.93 -12.91 -12.84
```

It can be noted that the performances do not vary significantly, changing the value of `interaction.depth`. However, comparing the quartiles and the mean, it results that the error is slightly smaller when `interaction.depth` is equal to 1.

The code executed by the `gbm` function also computes the variable importance defined as (2.16). To display these values we just need to call the function `summary` on the gbm object and it will returns both the variable list with their relative importance and a barplot, see Figure 4.6.

```
> summary(custBoost1)
                                     var   rel.inf
Production.capacity     Production.capacity 21.338838
Dedicated.capacity       Dedicated.capacity 17.780234
Co.supply.index             Co.supply.index 12.032458
Equipment.Level             Equipment.Level 7.531387
Raw.material.suppliers Raw.material.suppliers 7.522901
gnipc16                             gnipc16 6.165835
gdpPPP16                           gdpPPP16 5.427797
laborTax17                       laborTax17 5.179042
Company.Type                   Company.Type 4.678017
taxTime17                        taxTime17 3.567275
totTax17                          totTax17 3.387600
infl16                              infl16 2.713270
distFront17                    distFront17 2.675347
```

Thereafter, we fitted a random forest model, by means of the function `randomForest`, belonging to the homonymous R package. As `gbm`,

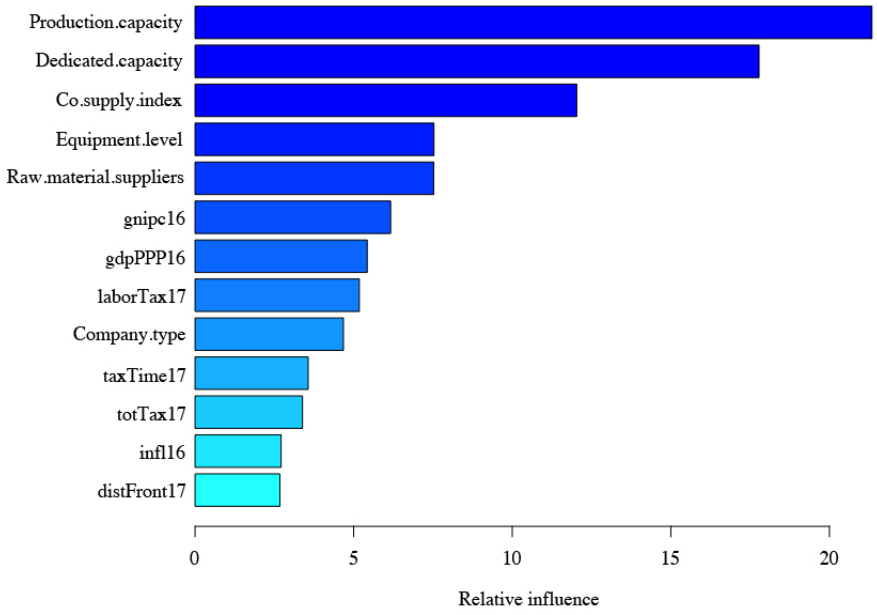


Figure 4.6: Feature relative importance in gradient boosted model.

`randomForest` take several variables in input[4], but we are going to discuss just few of them. In particular:

- `formula:` a regression formula describing the model to fit;
- `data:` the data frame to analyse;
- `ntree:` number of trees to fit;
- `mtry:` size of the features random sample from which the splitting variable is chosen;
- `importance:` logical variable specifying whether the variable importance should be computed or not.

Since the aim is to understand the behaviour of the variables, `importance` will be set to `TRUE`. Instead, `ntree` and `mtry` can be tuned, respectively, with cross-validation and a specific function, `tuneRF`, implemented in the package itself.

Specifically, a cross-validation function, computing the mean squared error, has been implemented (see Appendix B for the R code) and executed for random forests built with different values of `ntree`. Then, the performances have been plotted as shown in Figure 4.7.

```
> ntrees <- c(10, 20, 30, 40, 50, 70, 90, 120, 150, 200, 250, 300,
    400, 500, 600, 700, 800, 900, 1000)
> crossVal <- 0
> j <- 1
> for (i in ntrees) {
+   crossVal[j] <- crossValidation(data = custData, y = "Sales",
    method = "RandomForest", ntree = i)
+   j <- j+1
+ }
> plot(ntrees,crossVal)
```

Repeating this cross-validation test multiple times, we assessed that 200 is a good choice as `ntree`, since compared to smaller values the error drops remarkably, but adding further trees does not bring any significant improvement to the model.

[4]randomForest package documentation available at https://cran.r-project.org/web/packages/randomForest/randomForest.pdf

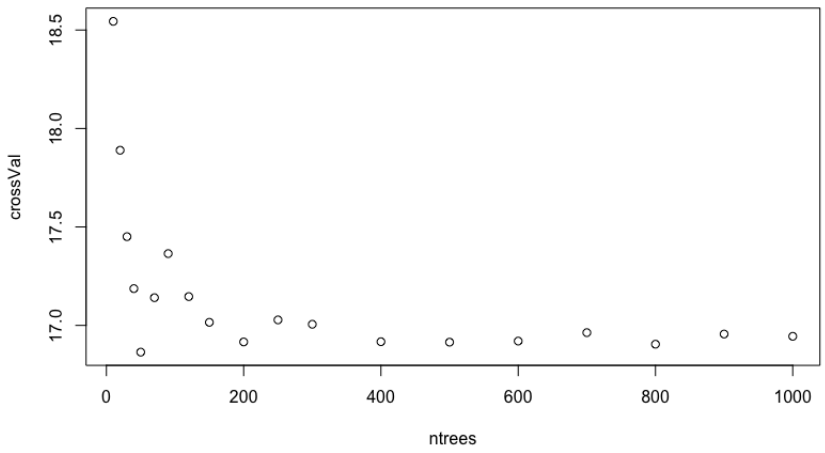


Figure 4.7: `ntree` tuning.

To establish which value is more suitable for `mtry` we can simply apply the function `tuneRF` giving as inputs[5]

- `x:` matrix or data frame of features;
- `y:` target variable vector;
- `ntreeTry:` number of trees to grow;
- `stepFactor:` parameter by which `mtry` is inflated or deflated, at each step;
- `improve:` the minimum improvement in OOB error necessary to continue the research.

It calculates for different values of `mtry` the OOB error of the model and display it both in console and with a plot, see Figure 4.8.

```
47 > saleFeatures <- saleData
48 > saleFeatures$Sales <- NULL
```

[5]Also for this function more inputs are available, see https://cran.r-project.org/web/-packages/randomForest/randomForest.pdf.

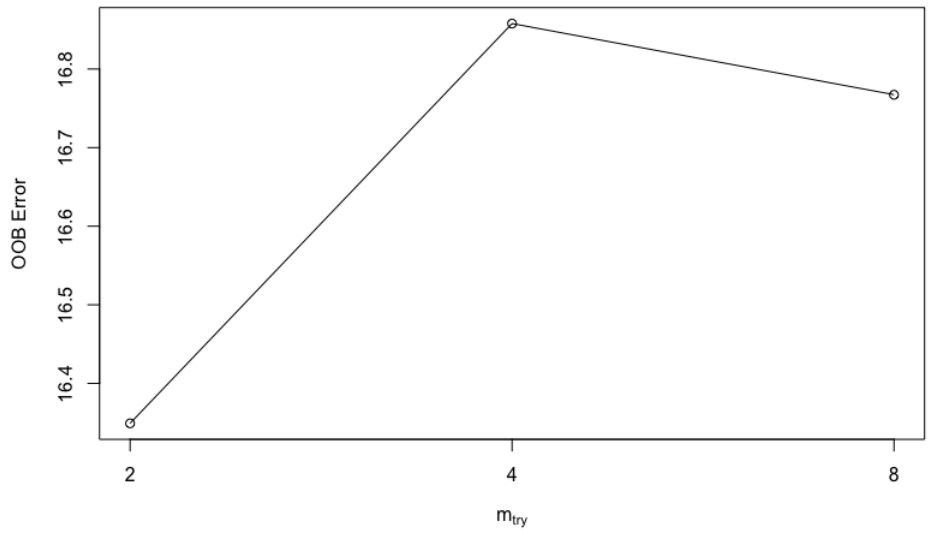


Figure 4.8: `mtry` tuning.

```
> mtryTune <- tuneRF(x = saleFeatures, y = saleData$Sales,
    ntreeTry = 200)
mtry = 4 OOB error = 16.85785
Searching left ...
mtry = 2   OOB error = 16.34912
0.03017736 0.05
Searching right ...
mtry = 8   OOB error = 16.76721
0.005376473 0.05
```

We executed the function with the selected value of `ntree`, the default value of `improve`, that is 0.05, and with different values of `stepFactor`. The only value of `mtry` that seems to perform slightly better than the default value 4, is 2.

As a double check we tuned again `ntree` setting `mtry = 2` and the value previously selected has been confirmed.

Once all the parameters have been assessed, we fitted the random forest:

```
> sale.rf <- randomForest(formula = Sales ~ ., data = saleData,
```

```
    importance = TRUE, ntree = 200, mtry = 2)
```

`sale.rf` is a random forest object containing several information about the model. Among them we can find the variable importance calculated as discussed in 2.17, stored as `importance`. Furthermore, it is possible to plot these values similarly as for boosted trees models , with the function `varImpPlot`, that takes as input a random forest object (see Figure 4.9).

```
> sale.rf$importance
                          %IncMSE IncNodePurity
Co.supply.index        0.52425623    1295.6443
Production.capacity    1.60869976    1657.5130
Dedicated.capacity     2.01710216    1799.0010
Company.Type           0.19936245     703.3986
Equipment.Level        1.01524422    1254.2139
Raw.material.suppliers 0.35104074    1133.1120
taxTime17              0.04327482     614.4901
totTax17               0.18871698     649.8098
laborTax17             0.18591600     712.9181
distFront17            0.42057431     685.3919
gnipc16                0.40651533     679.2631
gdpPPP16               0.25153233     674.2369
infl16                 0.25871958     605.5216

> varImpPlot(sale.rf)
```

It should be noted that the results of the two measurements are pretty consistent. Indeed, the top four is the same, with the only difference in *Equipment.Level* and *Co.supply.index* that are inverted, but, actually, it is almost a tie when they are calculated with (2.16). In general this measure tends to distribute the importance quite uniformly over the features, indeed, we could split the variables in three group: extremely influent, very influent, ordinarily influent. If we need a more detailed ranking, we can consult the other measure that delivers a less uniform influence distribution. The only difference between this two measures that stands out regards *Raw.material.supplier*, that results very influent on one side, but not so much on the other, where it switches its position with *gnipc16*.

Similar remarks may be made confronting the results computed with (2.16) over the two different models. Indeed, they detect as most impor-

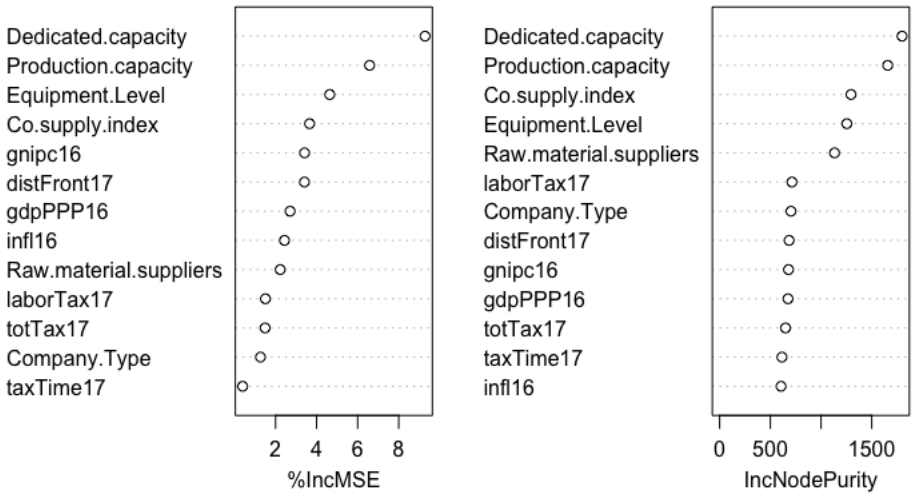


Figure 4.9: Feature relative importance in random forest model. The measurements are computed permuting the OOB sample on the left and evaluating the improvement gained by splitting on the variable on the right.

tant the same variables and the random forest does not deliver many other information about the others, differently from boosted trees.

However, the results obtained with the boosted trees model cannot be trusted much. Performing a cross-validation analysis, we assessed by comparison of mean squared errors, that this model perform much worse than the random forest.

```
> crossValidation(custData, "Sales", "RandomForest", ntree = 200)
[1] "folds= 10"
[1] 16.82437

> crossValidation(custData, "Sales", method = "Boosting",
    ntree=100)
[1] "folds= 10"
[1] 44.84724
```

One might fairly object that both of them lack in accuracy, but it is important to remember that the purpose of this analysis it is not to predict the target variable, but instead, to acquire knowledge about the underlying

system dynamics.

4.3.2 Visualisation

Thanks to the importance measures we assessed which features are more significant to the variable of interest, but we still did not know which relationship bounds them and how the target variable varies when a feature value increases or decreases. We wanted to graphically illustrate it in order to complete the task in the clearest possible way.

Since the analysed sample is quite limited and imbalanced, probably accumulated local effects plots are not a good choice. Indeed, our dataset imbalance does not correspond necessarily to the reality, but rather to a limited availability and quality of data. In this case taking into account not observed feature values combinations, could help broaden our vision, leading to a more general analysis. After consulting subject matter experts, in order to make sure there were no risks to introduce unrealistic instances, we concluded that in this case partial cumulative differences plots are the best option. However, for the sake of completeness, the following shows how to obtain all the plots presented in Chapter 3.

Since we assessed that the random forest model performs better than the gbm one, all the plots are generated exploiting the first.

The command `FeatureEffect$new()`, belonging to the `iml` package, allows computing all the plots discussed in 3.1 just specifying:

- `predictor:` a Predictor object, that can be obtained with the command `Predictor$new()` that takes as input the predictive model, the features sample and the respective target variable observations (see line 83);
- `feature:` a string containing the name of the variable in exam;
- `method:` a string specifying which plot should be generated ('ale', 'pdp', 'ice', 'pdp+ice').

For example, it is possible to obtain the plots in Figure 4.10 with the following code:

```
82 > X = saleData[which(names(custData) != "Sales")]
83 > predictors = Predictor$new(sale.rf, data = X, y =
     saleData$Sales)
```

```
>
> aleProd = FeatureEffect$new(predictor, method = 'ale', feature =
    "Production.capacity")
> aleProd$plot()
>
> pdpProd = FeatureEffect$new(predictor, method = 'pdp', feature =
    "Production.capacity")
> pdpProd$plot()
>
> iceProd = FeatureEffect$new(predictor, method = 'ice', feature =
    "Production.capacity")
> iceProd$plot()
>
> pdpiceProd = FeatureEffect$new(predictor, method = 'pdp+ice',
    feature = "Production.capacity")
> pdpiceProd$plot()
```

The `pcdPlot` function, that plots the partial cumulative differences of a variable, is already been explained in details in 3.2. To execute this function the R package `plotly` is required.

To verify the results obtained by the computation of variable importance delivered by the random forest model, we report here the plots for three variables, each belonging to a different level of influence individuated in the previous section: *Production.capacity* as extremely influent feature, *Equipment.Level* as very influent and *totTax17* as ordinarily influent. Furthermore, to show all the potentialities of the function we plot the mean and standard deviation for the former and the median with the percentiles for the others. In Appendix C are reported the partial cumulative differences plots for all the variables.

```
> library(plotly)
> pcdPlot(dataset = saleData, variableName =
    "Production.capacity", fitted = sale.rf, meanMedian = "mean")
>
> pcdPlot(dataset = saleData, variableName = "Equipment.Level",
    fitted = sale.rf, meanMedian = "median", percentile = 0.35)
>
> pcdPlot(dataset = saleData, variableName = "totTax17", fitted =
    sale.rf, meanMedian = "median", percentile = 0.35)
```

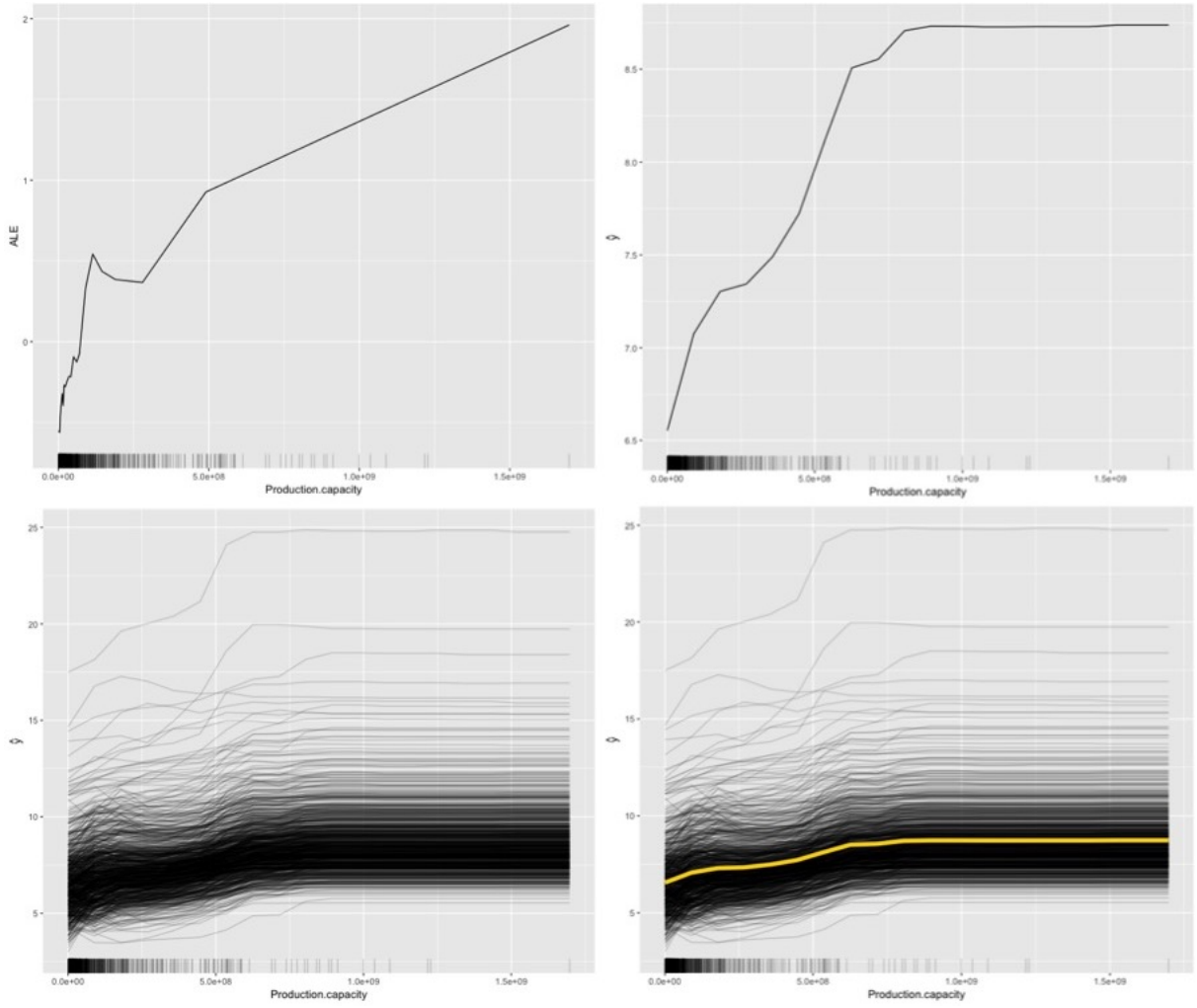


Figure 4.10: ALE plot (top left), PDP (top right), ICE plot (bottom left), PDP+ICE (bottom right) for *Production.capacity*.

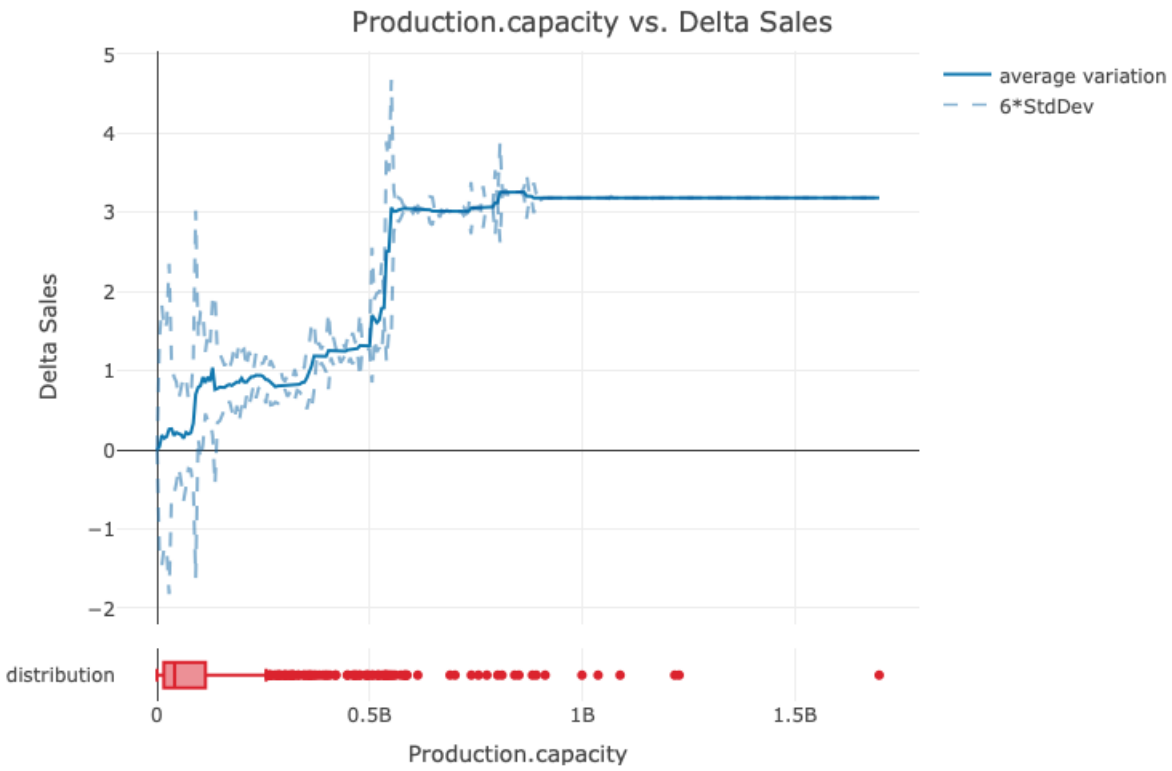


Figure 4.11: `Production.capacity` partial cumulative differences plot. The standard deviation is multiplied by 6 for a more readable visualisation.

Considering the order of magnitude of *Sales*, the variation delivered by *Production.capacity* is remarkably large, as assessed previously. Also the other two plots confirm the previous results. Indeed, in the PCD plot of *Equipment.Level* the *Sales* median value varies overall by about 1 unit, and by about 0.3 units in relation to *totTax17*. Furthermore, it can be noted that in both these plots the range delimited by the percentiles, and therefore the variance, increases with the variable in exam.

Such plots confirmed expected relationships, but also revealed unsuspected dependencies, deepening our knowledge about the system. Whereas it seems clear that suppliers with larger production capacity delivers an higher number of order, it is not that obvious that relying on a lower number of raw material suppliers implicates a smaller amount of delivered orders, as shown

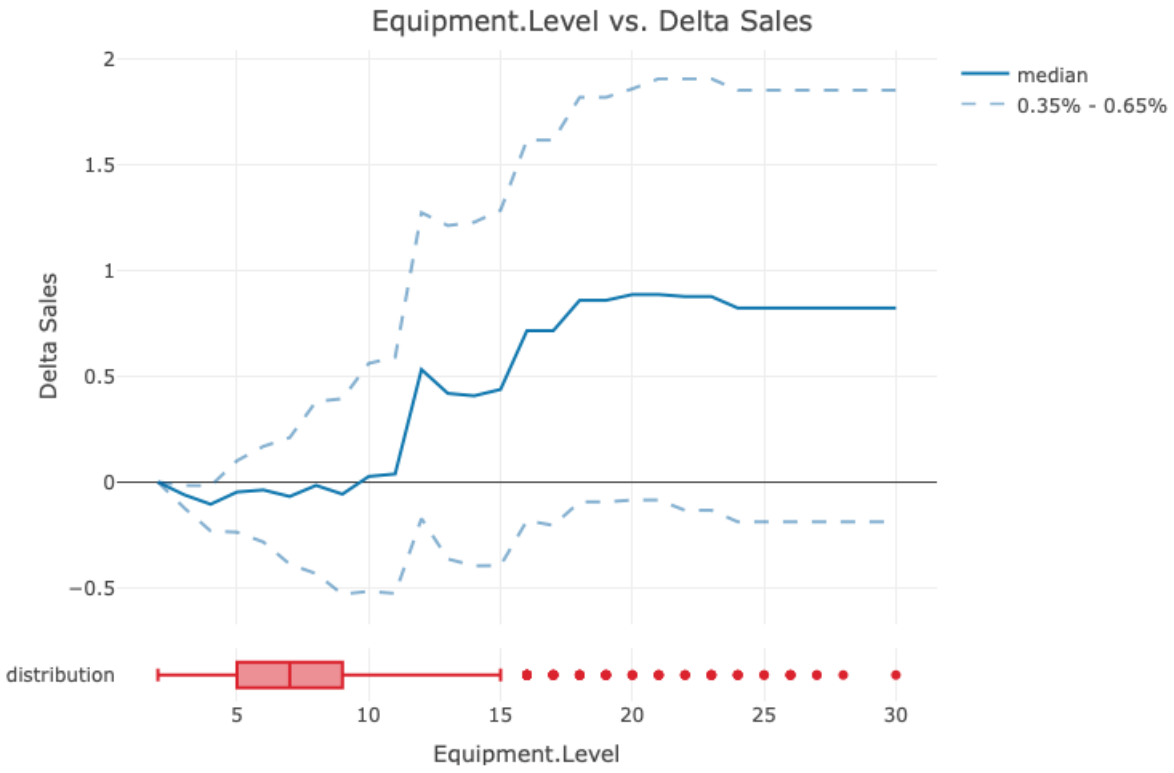


Figure 4.12: *Equipment.Level* partial cumulative differences plot.

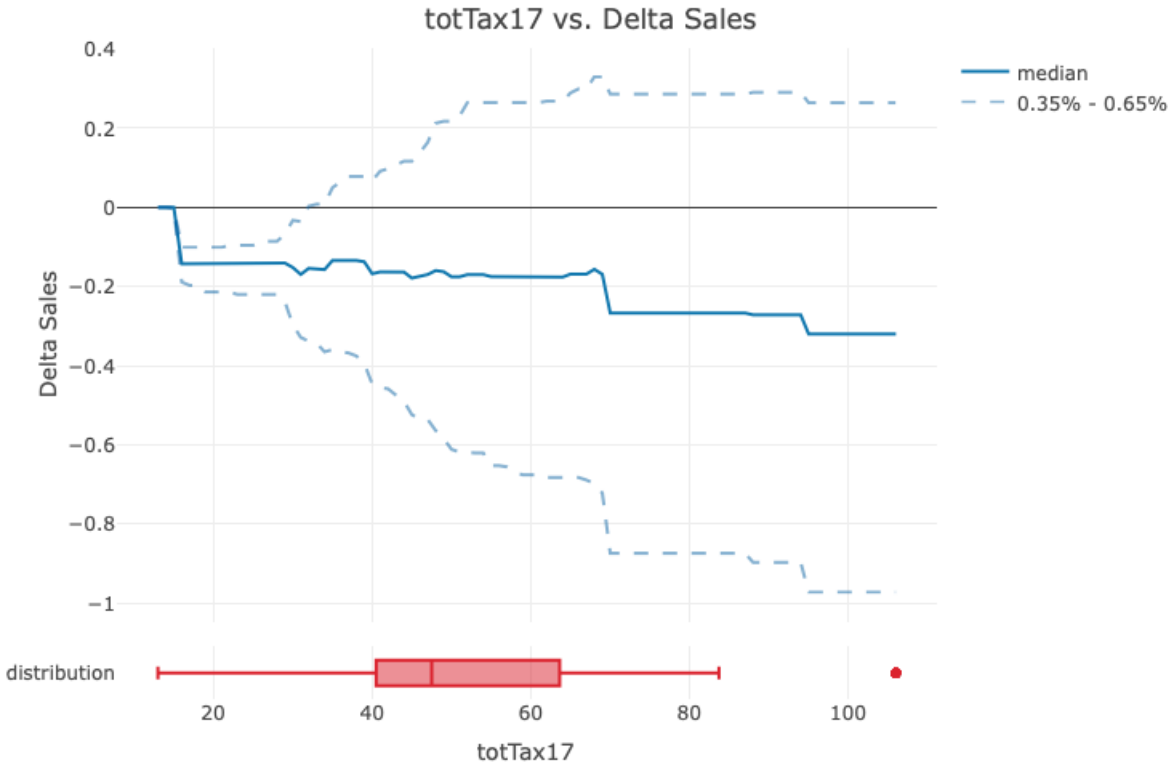


Figure 4.13: *totTax17* partial cumulative differences plot.

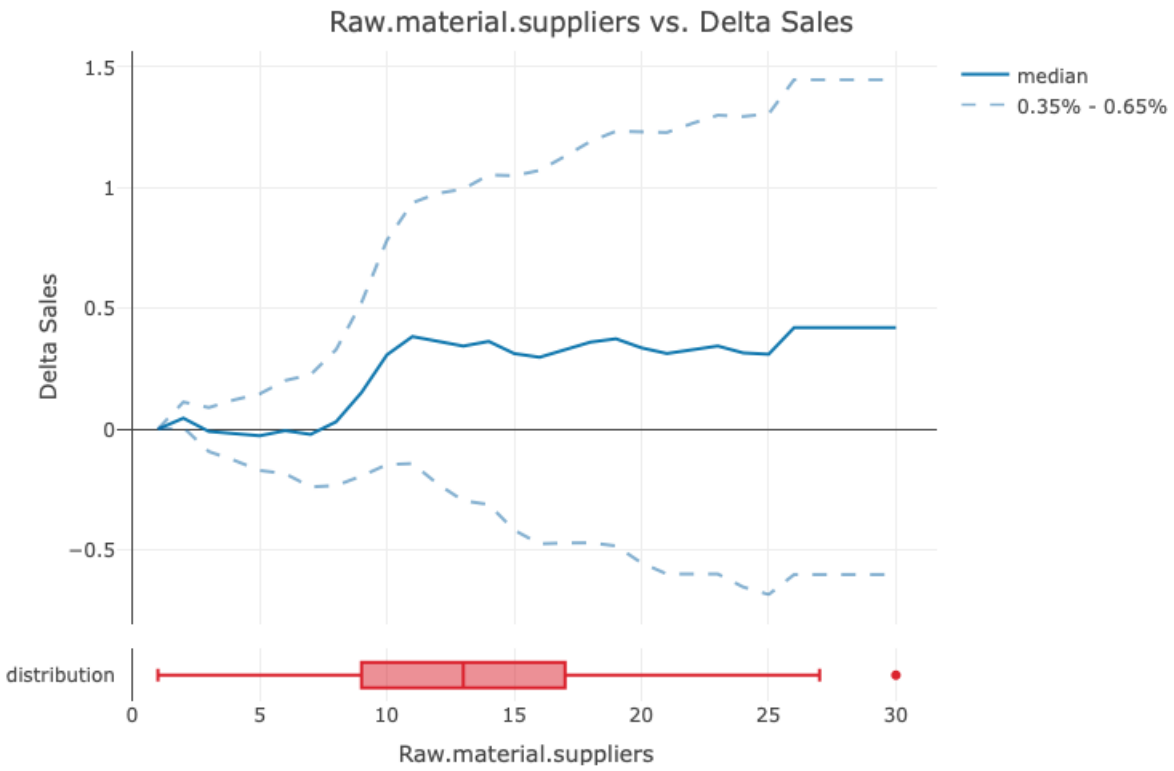


Figure 4.14: *Raw.material suppliers* partial cumulative differences plot.

in Figure 4.14.

Chapter 5

Conclusions

Since the amount of data available nowadays is increasing exponentially, it is possible to model and analyse always more complex systems. Many models allow accomplishing regression analysis delivering accurate results also for complex problems, but retrieving their dynamics is not as simple.

The aim of this work was to design a method to quantify variable importance and open the black box, delivering results understandable also by non-experts in the data analysis field. For this reason the specific results of the case study analysis are not discussed in details.

To achieve our goal we made use of regression models, trying to reveal how they interpret and elaborate the information provided by the features when predicting the target variable value. We focused on their influence over the response variable both quantitatively and qualitatively.

Carrying out this analysis for a company, it was crucial to produce results clearly interpretable by laymen. For this reason a specific attention has been paid to graphical tools.

Analysing the case study, we have assessed that random forest models performed better and partial cumulative differences plots were more suitable to achieve our goal. However, it is very important to understand that the choice of the methods to apply must be carried out on the basis of the dataset to analyse.

Furthermore, it should be kept in mind that limiting the interpretation to partial effects analysis is not advised, because features interactions could play a crucial role in the fitted model. However, as expected, detecting variable interactions behind a black box model is even more challenging than detecting the relative effects of the single features.

An interesting method to achieve this goal has been developed and implemented in the `astrid` package [5, 6], but it is suitable only for classification problem and it is extremely computationally demanding. We tried to apply it to our case study, discretizing the response variable, but the results were not significant. The development of a tool for the analysis of variable interaction in regression models will be the aim of further studies.

The next step for Tetra Pak will be to automate the type of analysis discussed in this work at the maximum extent, in order to successfully model highly complex systems.

Appendices

Appendix A

R code for partial cumulative differences plot

```
pcdPlot <- function(dataset, variableName, fittedModel,
    meanMedian, percentile= NULL, categorical = FALSE){
  #dataset: the analysed data frame
  #variableName: a string containing the name of the variable in
     exam
  #fittedModel: an object of the class randomForest or gbm
  #meanMedian: a string specifying whether the mean or the median
     of the effects should be plotted
  #percentile: the lower percentile that we want displayed together
     with its symmetrical with respect to the median, if NULL
     (default value) the standard deviation is plotted
  #categorical: a boolean variable specifying whether the variable
     in exam is categorical or continuous (default)

  #extract an array with the variableName values
  varPlot <- dataset[[variableName]]

  #array that collects all observed values of variableName in
     ascendent order, (removing duplicates)
  variable <- sort(unique(dataset[[variableName]]))

  #if the model is fitted with the gradient boosted trees method
     the optimal number of trees in order to predict y is retrieved
  if(class(fittedModel)=="gbm"){
```

```
17    ntree <- gbm.perf(fittedModel, method = "cv")
18  }
19
20  #when the variable analysed is continuous, an equispaced array is
      generated with extremes equal to the observed domain
      boundaries of the variable in exam, and increment equals to
      the average gap between the observed values. If the array
      size would be greater than 300, then the increment is not
      set, instead the size is set to 300
21  if(!categorical){
22    gap <- diff(variable)
23    avgGap <- mean(gap)
24    if(avgGap >= 1){
25      avgGap <- round(avgGap)
26    }
27    minimum <- min(variable)
28    maximum <- max(variable)
29    if((maximum - minimum)/avgGap>300){
30      variable <- seq(from = minimum, to = maximum, length.out =
          300)
31      avgGap <- variable[2]-variable[1]
32    }
33    else{
34      variable <- seq(from = minimum, to = maximum, by = avgGap)
35    }
36  }
37
38  test <- dataset
39  j <- 1
40
41  #declare arrays in which mean/median and percentiles/standard
      deviation values will be stored
42    var <- array(0, dim = length(variable))
43    upper <- array(0, dim = length(variable))
44    lower <- array(0, dim = length(variable))
45
46  #loop over the different values of the variable in exam
47  for (i in variable){
48
49    #set analysed variable value to i for all observations
```

```
50      if(categorical){
51        test[variableName] <- factor(i, levels = variable)
52      }else{
53        test[variableName] <- i
54      }
55
56      #predict y for each observation with gbm or randomForest
57      if(class(fittedModel)=="gbm"){
58        predicted <- predict(fittedModel, test, n.trees = ntree)
59      }
60      else{
61        predicted <- predict(fittedModel, test)
62      }
63      if(j>1){
64        #from second iteration on calculate difference respect
              previous step
65        diff <- predicted-prev
66          #calculate mean/median of the differences and accumulate it
                with the previous values
67          if(meanMedian=="mean"){
68            var[j] <- var[j-1]+mean(diff)
69          }else if(meanMedian=="median"){
70            var[j] <- var[j-1]+median(diff)
71          }else{
72            stop("meanMedian value is wrong: nether median nor mean")
73          }
74          if(is.null(percentile)){
75            #calculate standard deviation of the differences and add
                  it to and subtract it from
76            #the mean/median to obtain a range
77            stddev <- sd(diff)
78            upper[j] <- (var[j]+6*stddev)
79            lower[j] <- (var[j]-6*stddev)
80          }else{
81            #calculate given percentile and its symmetrical with
                  respect to the median and accumulate them with the
                  previous values
82            upper[j] <- upper[j-1]+quantile(diff, percentile)
83            lower[j] <- lower[j-1]+quantile(diff, 1-percentile)
84          }
```

```
85      }
86      #save values predicted to compute differences in the next
            iteration
87      prev <- predicted
88      j <- j+1
89    }
90    #plot results
91    if(categorical){
92      if(meanMedian=="mean"){
93        p1 <- plot_ly( x=~variable, y=~var, type = "scatter", name =
              'average variation') %>%
94          add_trace(y=~upper , color='rgb(22, 96, 167)', line =
                list(color='rgb(22, 96, 167)', dash="dash"), name =
                '6*StdDev', opacity=0.5) %>%
95          add_trace(y=~lower , color='rgb(22, 96, 167)', line =
                list(color='rgb(22, 96, 167)', dash="dash"),
                opacity=0.5, showlegend = FALSE) %>%
96          layout(title= paste0(variableName, ' vs. Delta Sales'),
                xaxis= list(title=variableName), yaxis=
                list(title='Delta Sales'))
97      }else if(meanMedian=="median"){
98        p1 <- plot_ly( x=~variable, y=~var, type = "scatter", name =
              'median') %>%
99          add_trace(y=~upper , color='rgb(22, 96, 167)', line =
                list(color='rgb(22, 96, 167)', dash="dash"), name =
                paste0(percentile, '% - ', 1-percentile, '%'),
                opacity=0.5) %>%
100         add_trace(y=~lower , color='rgb(22, 96, 167)', line =
                list(color='rgb(22, 96, 167)', dash="dash"), name =
                paste0(1-percentile, '%'), opacity=0.5, showlegend =
                FALSE) %>%
101         layout(title= paste0(variableName, ' vs. Delta Sales'),
                xaxis= list(title=variableName), yaxis=
                list(title='Delta Sales'))
102     }
103     p2 <- plot_ly(x=~varPlot, name = "frequence", showlegend =
            FALSE)
104   }else{
105     if(meanMedian=="mean"){
106       if(is.null(percentile)){
```

```
107         p1 <- plot_ly( x=~variable, y=~var, type = "scatter", mode
                = "lines", name = 'average variation') %>%
108           add_trace(y=~upper , line = list(color='rgb(22, 96,
                167)', dash="dash"), name = '6*StdDev', opacity=0.5)
                %>%
109           add_trace(y=~lower , line = list(color='rgb(22, 96,
                167)', dash="dash"), name = paste0(1-percentile, '%'),
                opacity=0.5, showlegend = FALSE) %>%
110           layout(title= paste0(variableName, ' vs. Delta Sales'),
                xaxis= list(title=variableName), yaxis=
                list(title='Delta Sales'))
111       }
112       else{
113       p1 <- plot_ly( x=~variable, y=~var, type = "scatter", mode =
              "lines", name = 'average variation') %>%
114         add_trace(y=~upper , line = list(color='rgb(22, 96, 167)',
              dash="dash"), name = paste0(percentile, '% - ',
              1-percentile, '%'), opacity=0.5) %>%
115         add_trace(y=~lower , line = list(color='rgb(22, 96, 167)',
              dash="dash"), name = paste0(1-percentile, '%'),
              opacity=0.5, showlegend = FALSE) %>%
116         layout(title= paste0(variableName, ' vs. Delta Sales'),
              xaxis= list(title=variableName), yaxis=
              list(title='Delta Sales'))
117       }
118     }else if(meanMedian=="median"){
119       if(is.null(percentile)){
120         p1 <- plot_ly( x=~variable, y=~var, type = "scatter", mode
                = "lines", name = 'median') %>%
121           add_trace(y=~upper , line = list(color='rgb(22, 96,
                167)', dash="dash"), name = '6*StdDev', opacity=0.5)
                %>%
122           add_trace(y=~lower , line = list(color='rgb(22, 96,
                167)', dash="dash"), name = paste0(1-percentile, '%'),
                opacity=0.5, showlegend = FALSE) %>%
123           layout(title= paste0(variableName, ' vs. Delta Sales'),
                xaxis= list(title=variableName), yaxis=
                list(title='Delta Sales'))
124       }
125       else{
```

```
126         p1 <- plot_ly( x=~variable, y=~var, type = "scatter", mode
                = "lines", name = 'median') %>%
127           add_trace(y=~upper , line = list(color='rgb(22, 96,
                167)', dash="dash"), name = paste0(percentile, '% - ',
                1-percentile, '%'), opacity=0.5) %>%
128           add_trace(y=~lower , line = list(color='rgb(22, 96,
                167)', dash="dash"), name = paste0(1-percentile, '%'),
                opacity=0.5, showlegend = FALSE) %>%
129           layout(title= paste0(variableName, ' vs. Delta Sales'),
                xaxis= list(title=variableName), yaxis=
                list(title='Delta Sales'))
130       }
131     }
132     #add distribution boxplot
133     p2 <- plot_ly(x=~varPlot, type = "box", name = "distribution",
            showlegend = FALSE)
134   }
135   x <- list(title = variableName)
136   y <- list(title = 'Delta Sales')
137   subplot(p1, p2, nrows = 2, heights = c(0.9, 0.1), shareX = TRUE)
        %>%
138     layout(title= paste0(variableName, ' vs. Delta Sales'),
          xaxis=x, yaxis=y)
139 }
```

Appendix B

R code for cross-validation

```
crossValidation <- function(data, y, method, dimBlocks = 0.1,
    ntree=1000){
  #data: the analysed data frame
  #y: a string containing the name of the variable in exam
  #method: a string specifying the method to validate, options:
      "RandomForest", "Boosting"
  #dimBlock: folds size expressed as percentual of the dataset size
  #ntree: number of tree to grow in both methods

  #n number of obs per fold (last fold will have also the remaining
      obs)
  #k number of folds

  dimension <- nrow(data)
  n <- as.integer(dimension*dimBlocks)
  k <- 1/dimBlocks
  extra <- dimension%%k
  # print(paste0("folds= ", k))
  err <- 0
  if (method == 'RandomForest'){
    #loop over first k-1 folds
    for (i in 1:(k-1)){
      #definition of train set and test set
      test <- data[((n*(i-1))+1):(n*i),]
      train <- data[-(((n*(i-1))+1):(n*i)),]
      #fit the model
```

```
      fmla <- as.formula(paste0(y, "~."))
      rf <- randomForest(fmla, data = train, ntree=ntree, mtry = 2)
      #predict y for test set and compute mean squared error
      pred <- predict(rf, test)
      errVect <- (test[y]-pred)^2
      err <- err + sum(errVect)/n
    }
    #repeat the code in the previous loop for the last fold
    test <- data[((n*(k-1))+1):dimension,]
    train <- data[-(((n*(k-1))+1):dimension),]
    fmla <- as.formula(paste0(y, "~."))
    rf <- randomForest(fmla, data = train, ntree=ntree, mtry = 2)
    pred <- predict(rf, test)
    errVect <- (test[y]-pred)^2
    err <- err + sum(errVect)/(n+extra)
    err <- err/k
  }
  else if (method == 'Boosting'){
    #apply the same algorithm to the boosting case
    for (i in 1:k){
      test <- data[((n*(i-1))+1):(n*i),]
      train <- data[-(((n*(i-1))+1):(n*i)),]
      fmla <- as.formula(paste0(y, "~."))
      custBoost <- gbm(fmla, data=train, distribution ="poisson",
          n.trees =ntree, interaction.depth =5, cv.folds = 10)
      pred <- predict(custBoost, test, n.trees=gbm.perf(custBoost,
          method = "cv"))
      errVect <- (test[y]-pred)^2
      err <- err + sum(errVect)/n
    }
    test <- data[((n*(k-1))+1):dimension,]
    train <- data[-(((n*(k-1))+1):dimension),]
    fmla <- as.formula(paste0(y, "~."))
    custBoost <- gbm(fmla, data=train, distribution ="poisson",
        n.trees =ntree, interaction.depth =5, cv.folds = 10)
    pred <- predict(custBoost, test, n.trees=gbm.perf(custBoost,
        method = "cv"))
    errVect <- (test[y]-pred)^2
    err <- err + sum(errVect)/(n+extra)
    err <- err/k
```

```
60    }
61    else{
62      print("Error: invalid method")
63    }
64    return(err)
65  }
```

Appendix C

Case study partial cumulative differences plots

The variables are plotted in order of importance as calculated by (2.16) with the random forest model in Chapter 4. The standard deviation is plotted multiplied by 6 in order to make the plots more readable.

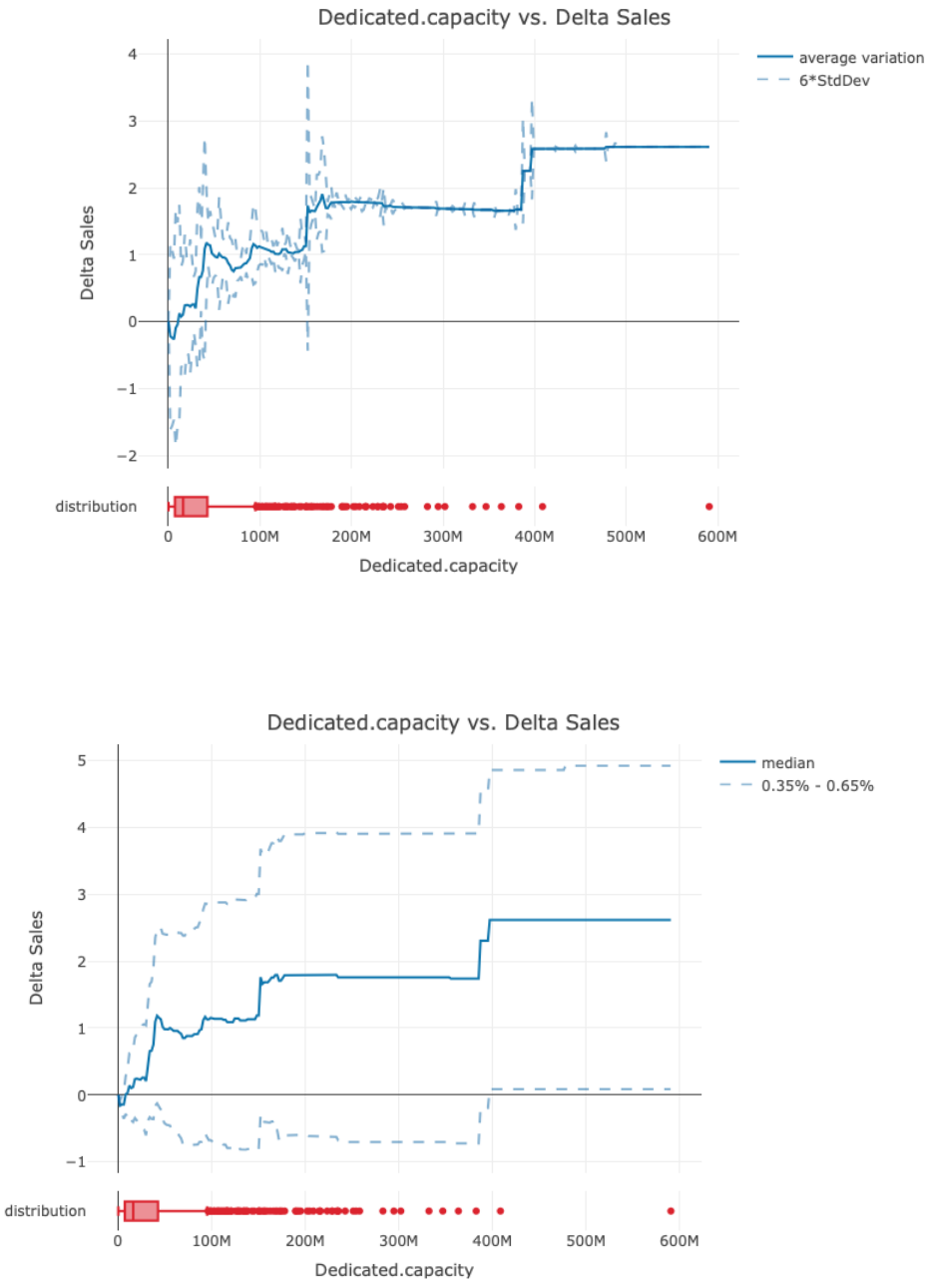
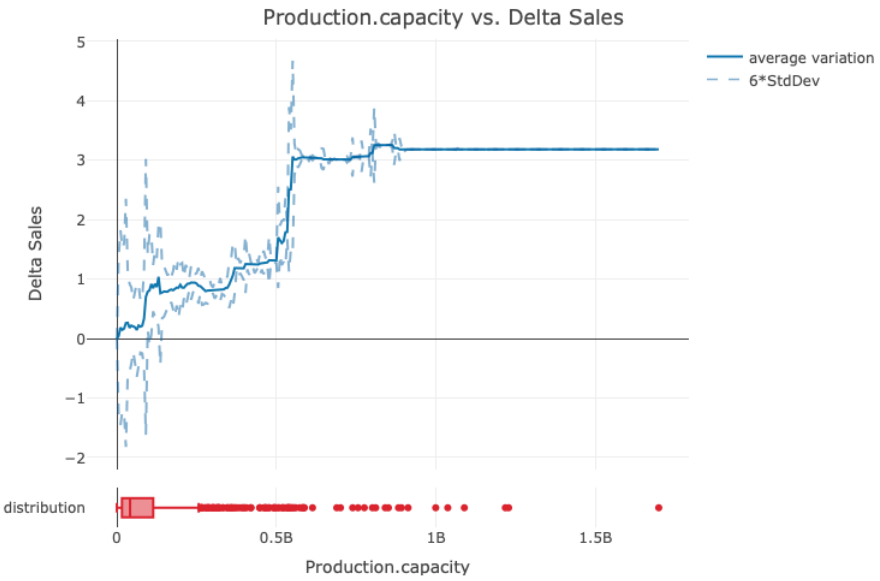


Figure C.1: *Dedicated.capacity* partial cumulative differences plot.



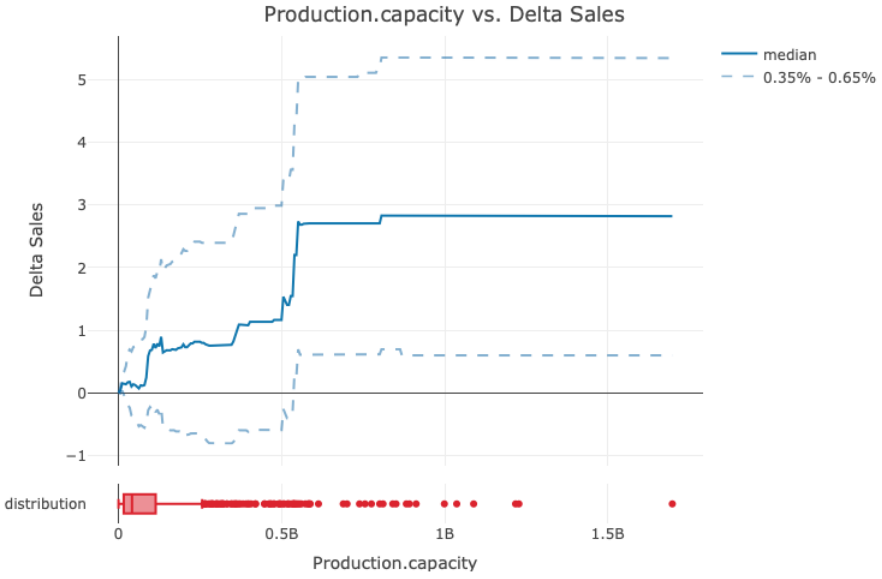


Figure C.2: *Production.capacity* partial cumulative differences plot.

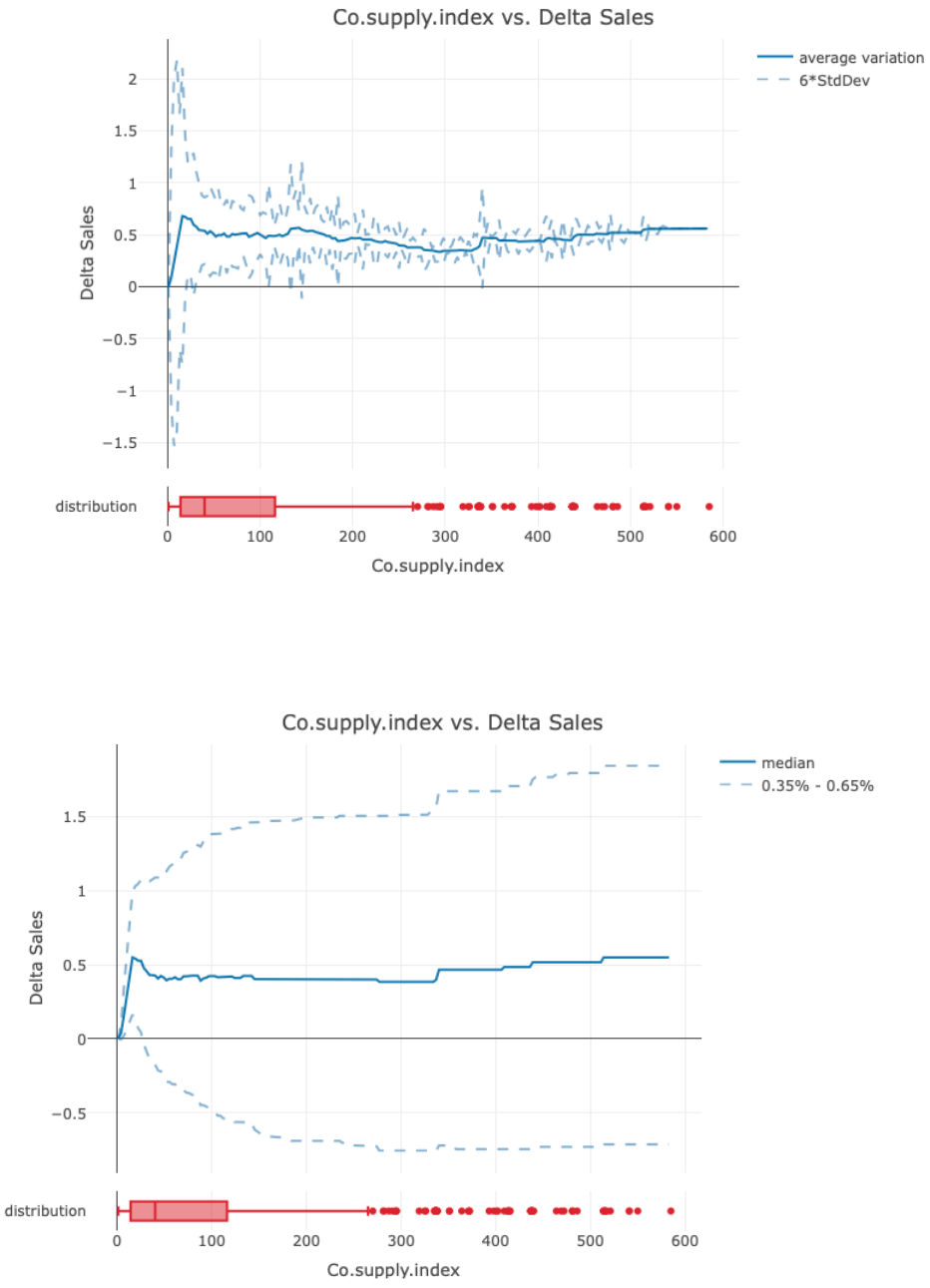
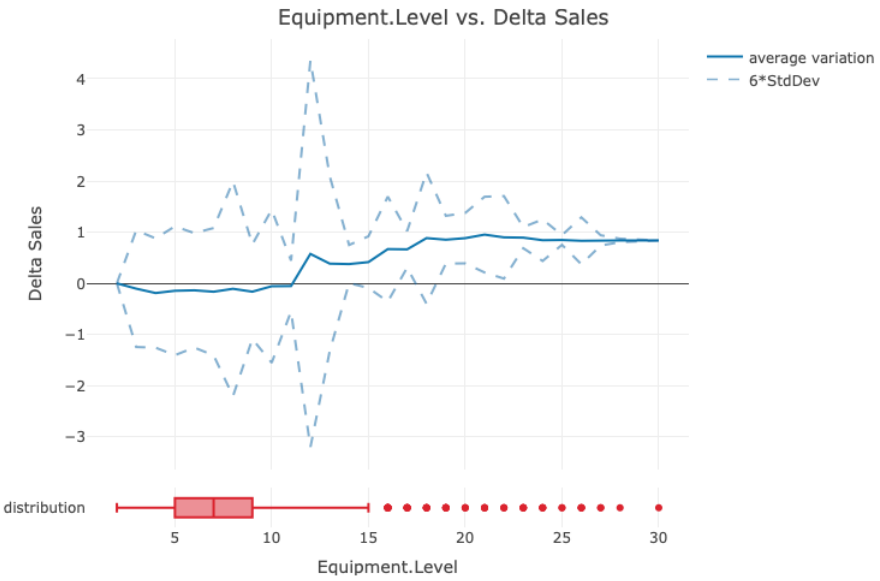


Figure C.3: *Co.supply.index* partial cumulative differences plot.



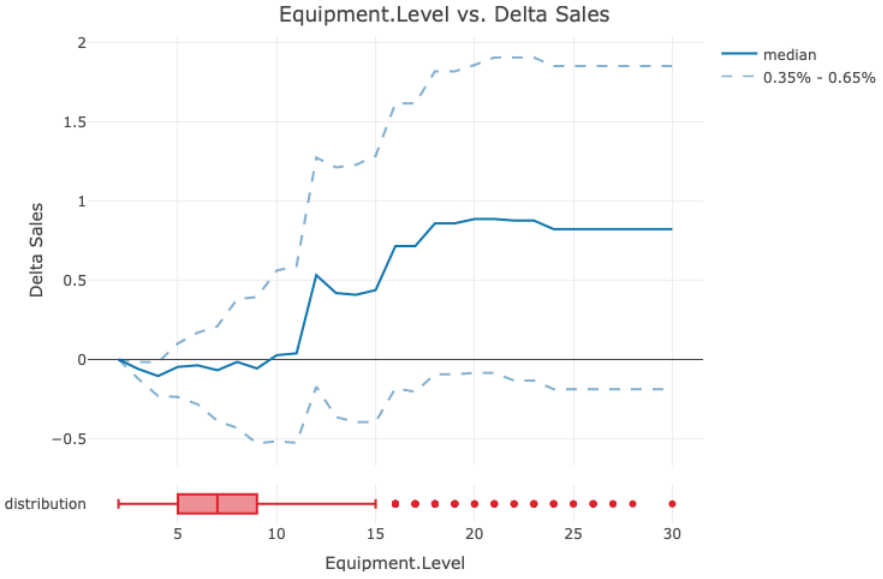


Figure C.4: *Equipment.Level* partial cumulative differences plot.

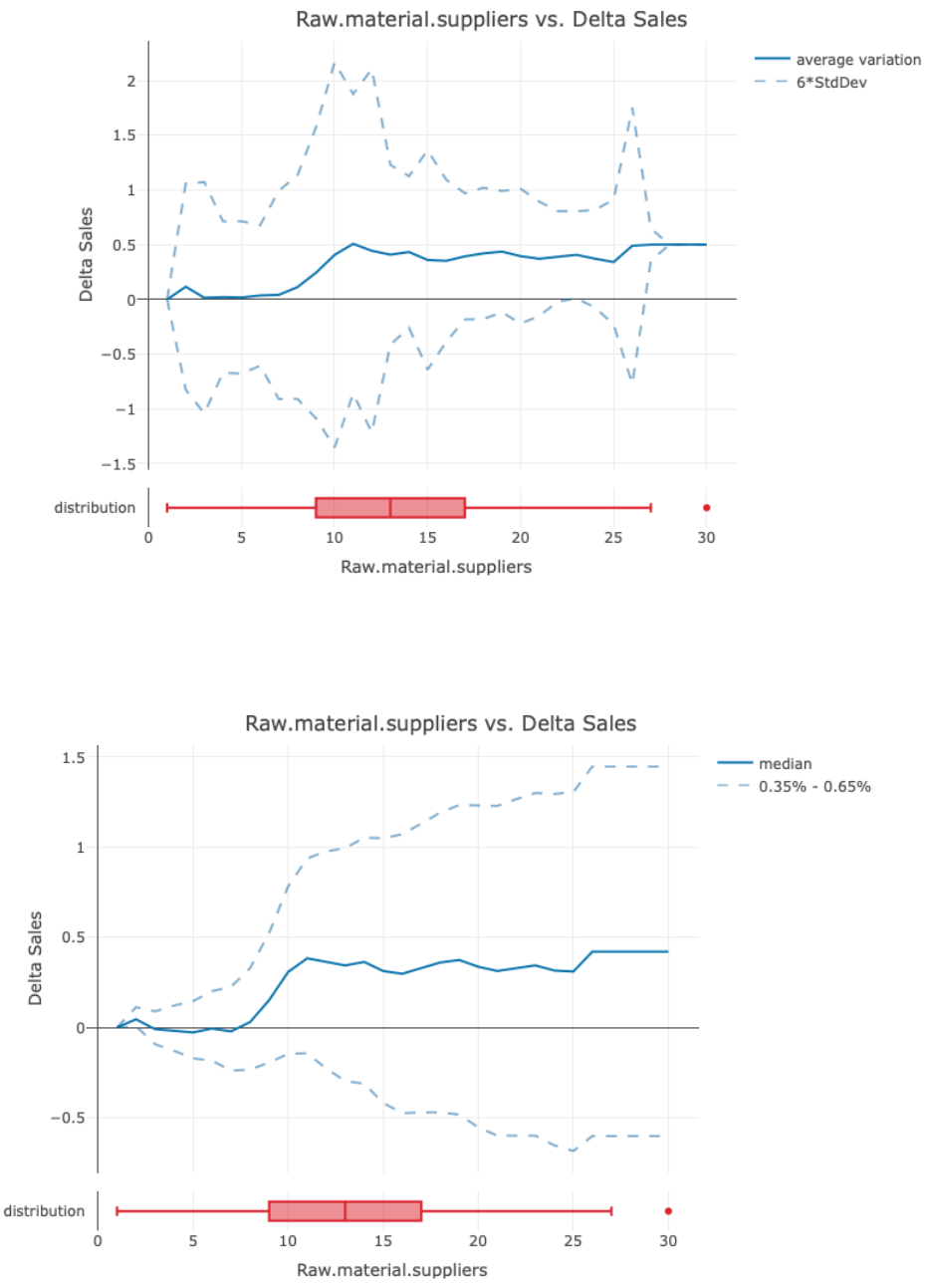


Figure C.5: *Raw.material.suppliers* partial cumulative differences plot.

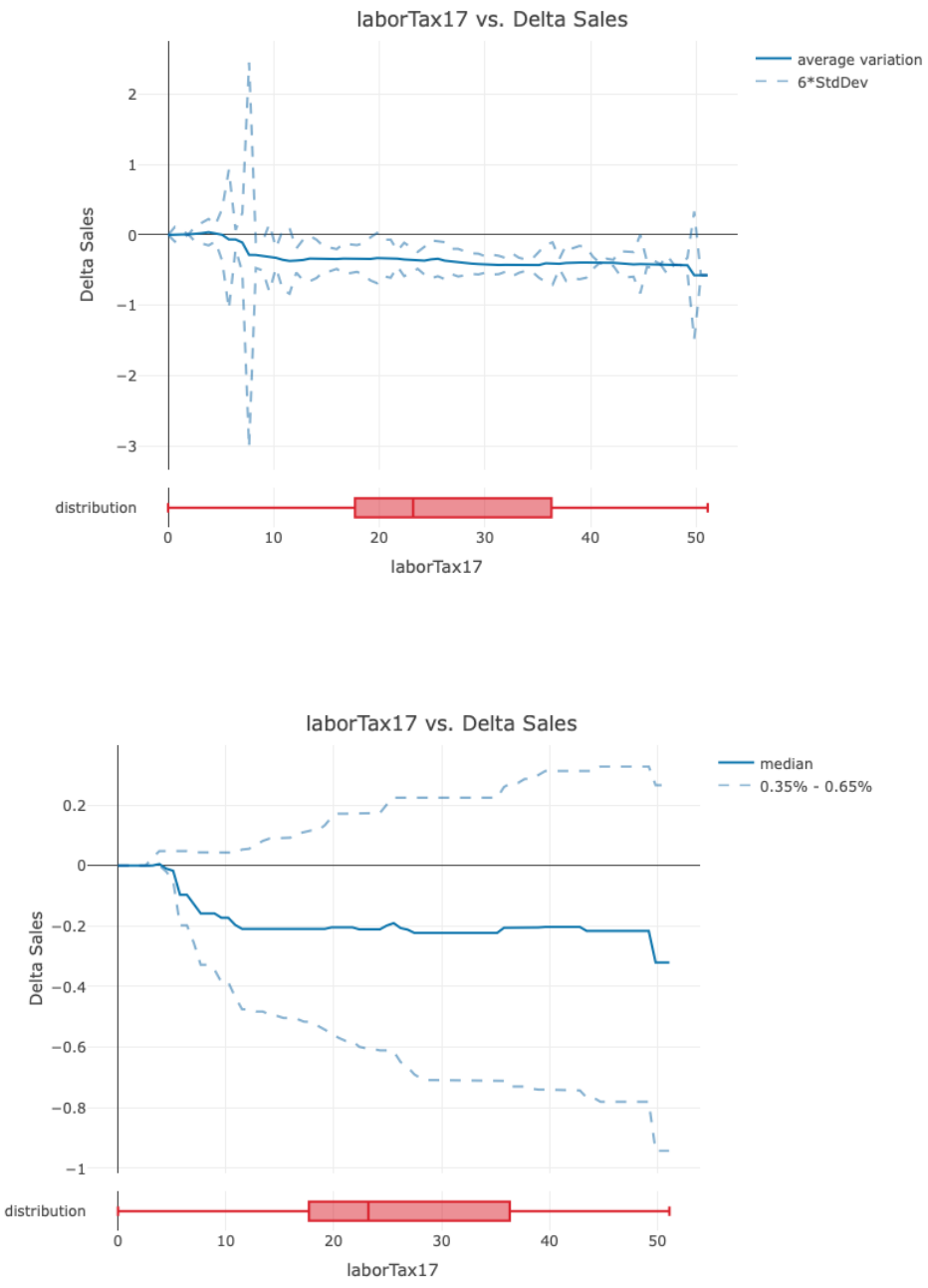


Figure C.6: *laborTax17* partial cumulative differences plot.

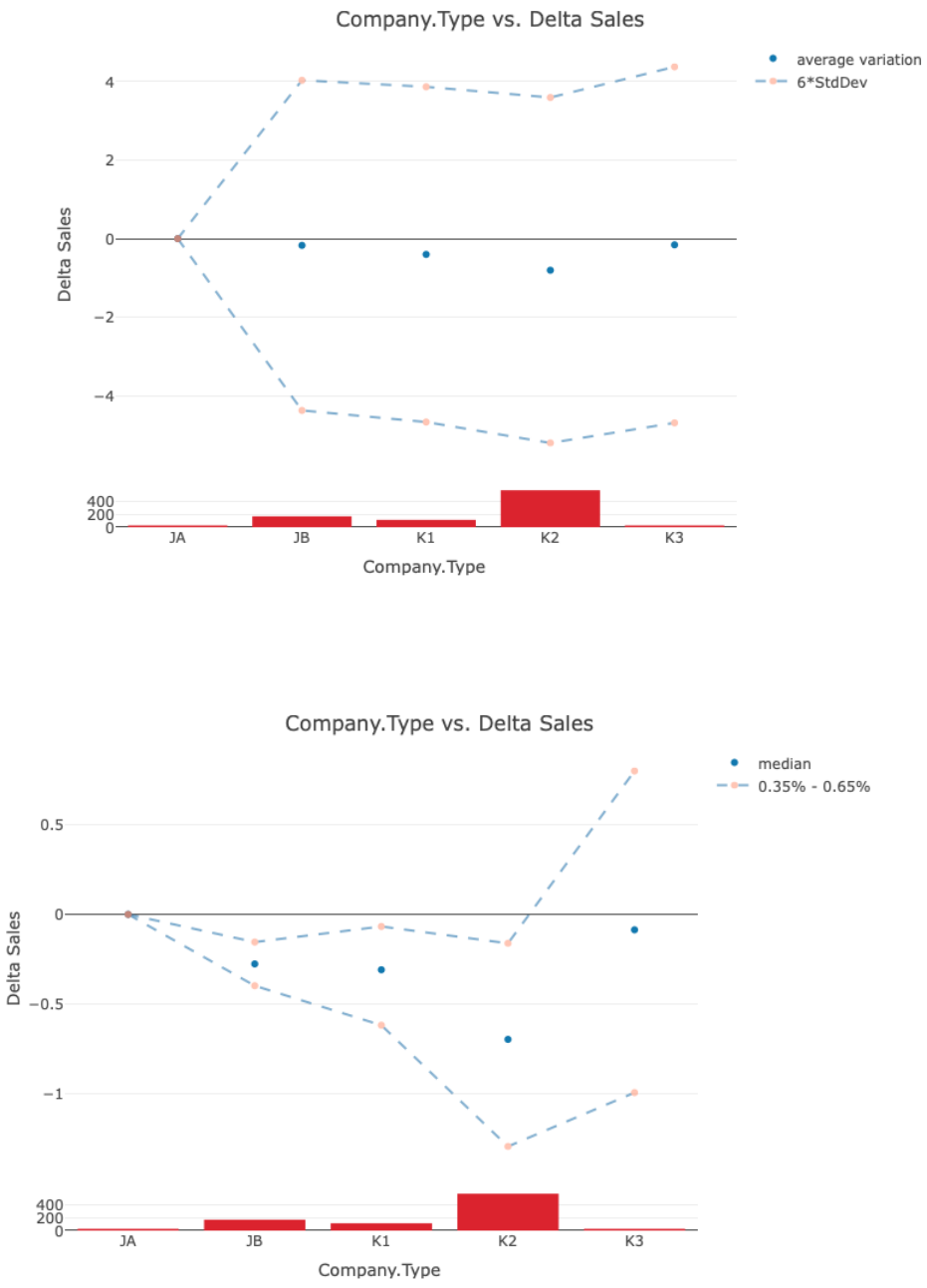


Figure C.7: *Company.Type* partial cumulative differences plot.

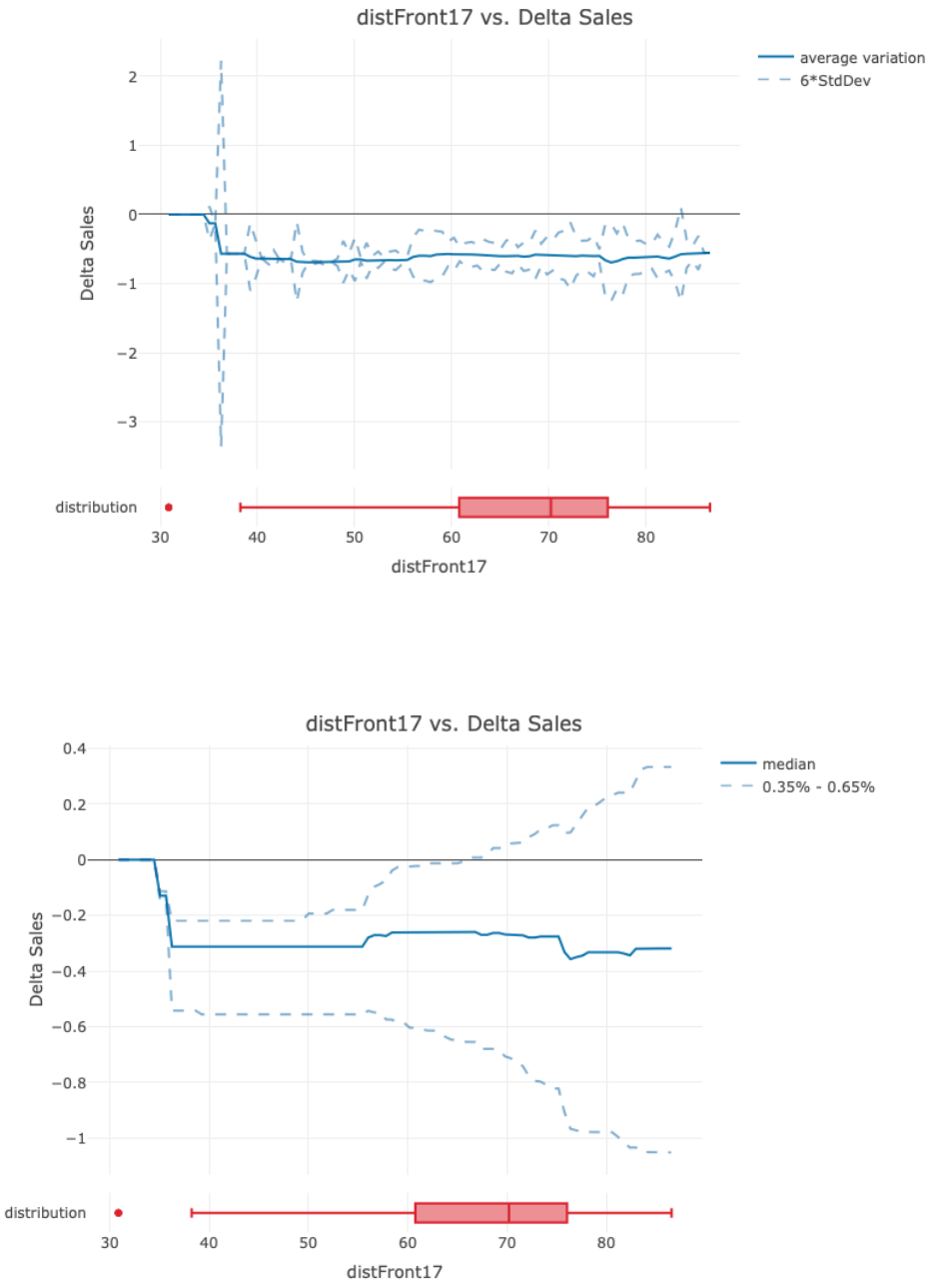


Figure C.8: *distFront17* partial cumulative differences plot.

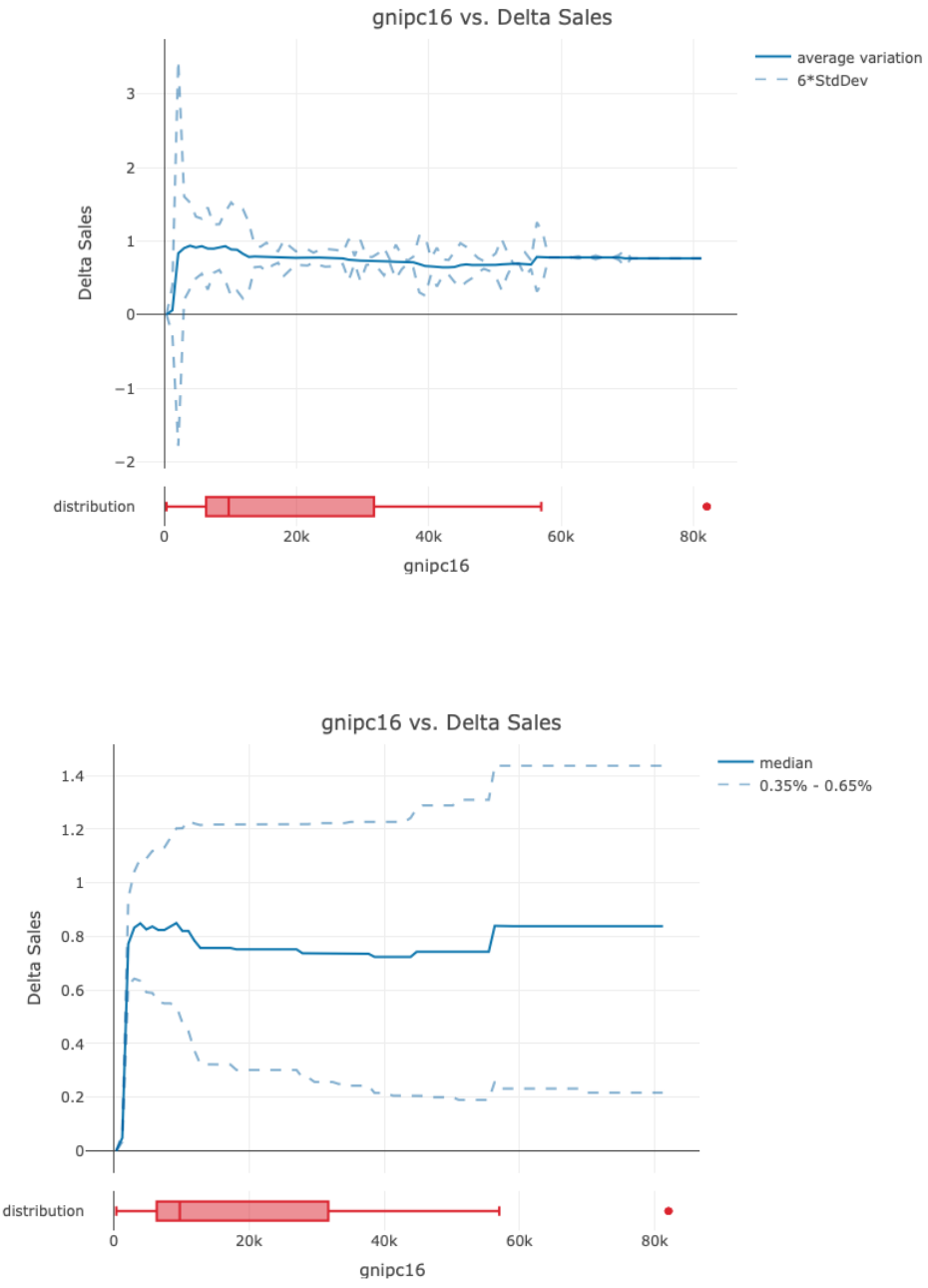


Figure C.9: *gnipc16* partial cumulative differences plot.

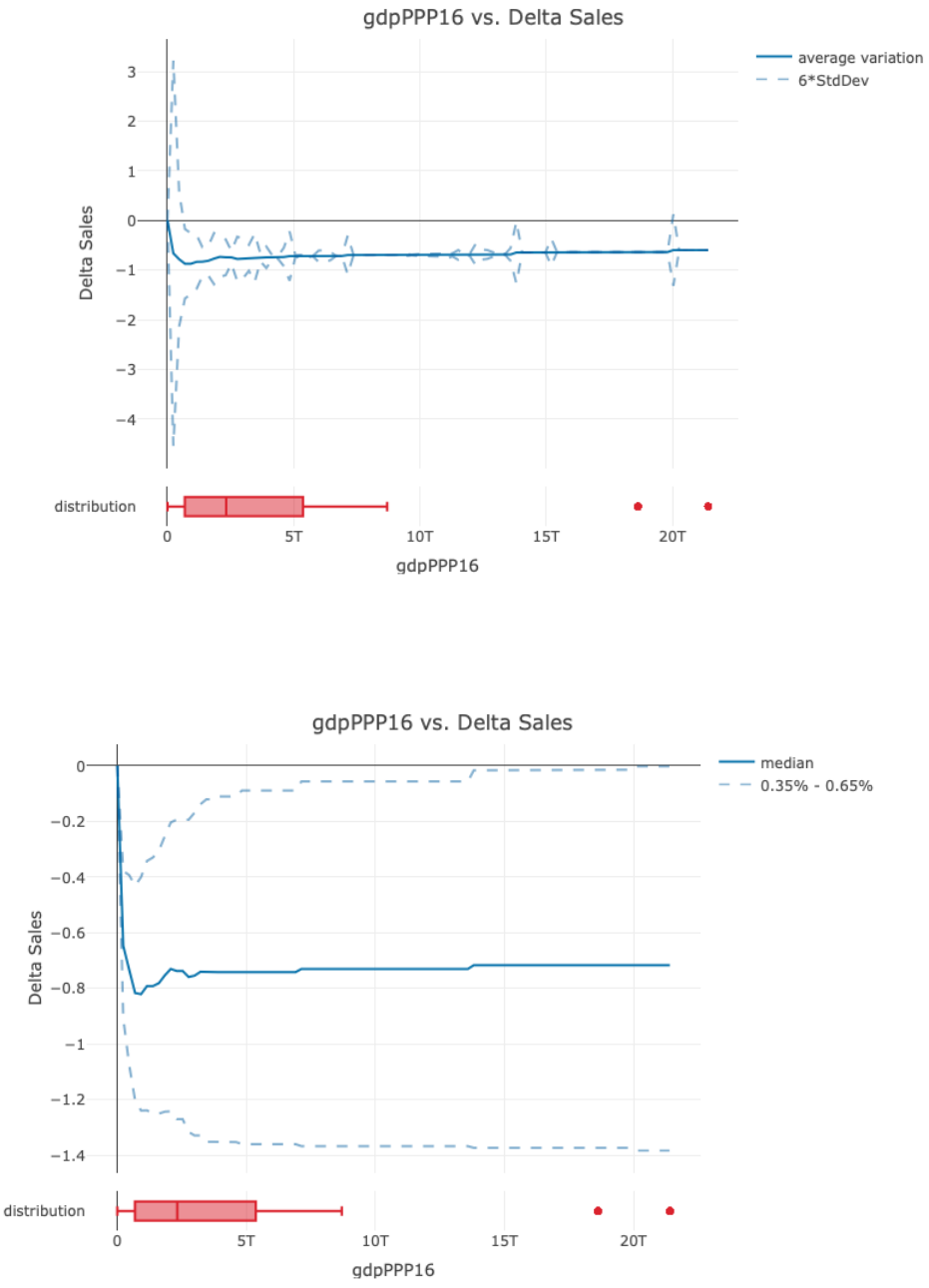


Figure C.10: *gdpPPP16* partial cumulative differences plot.

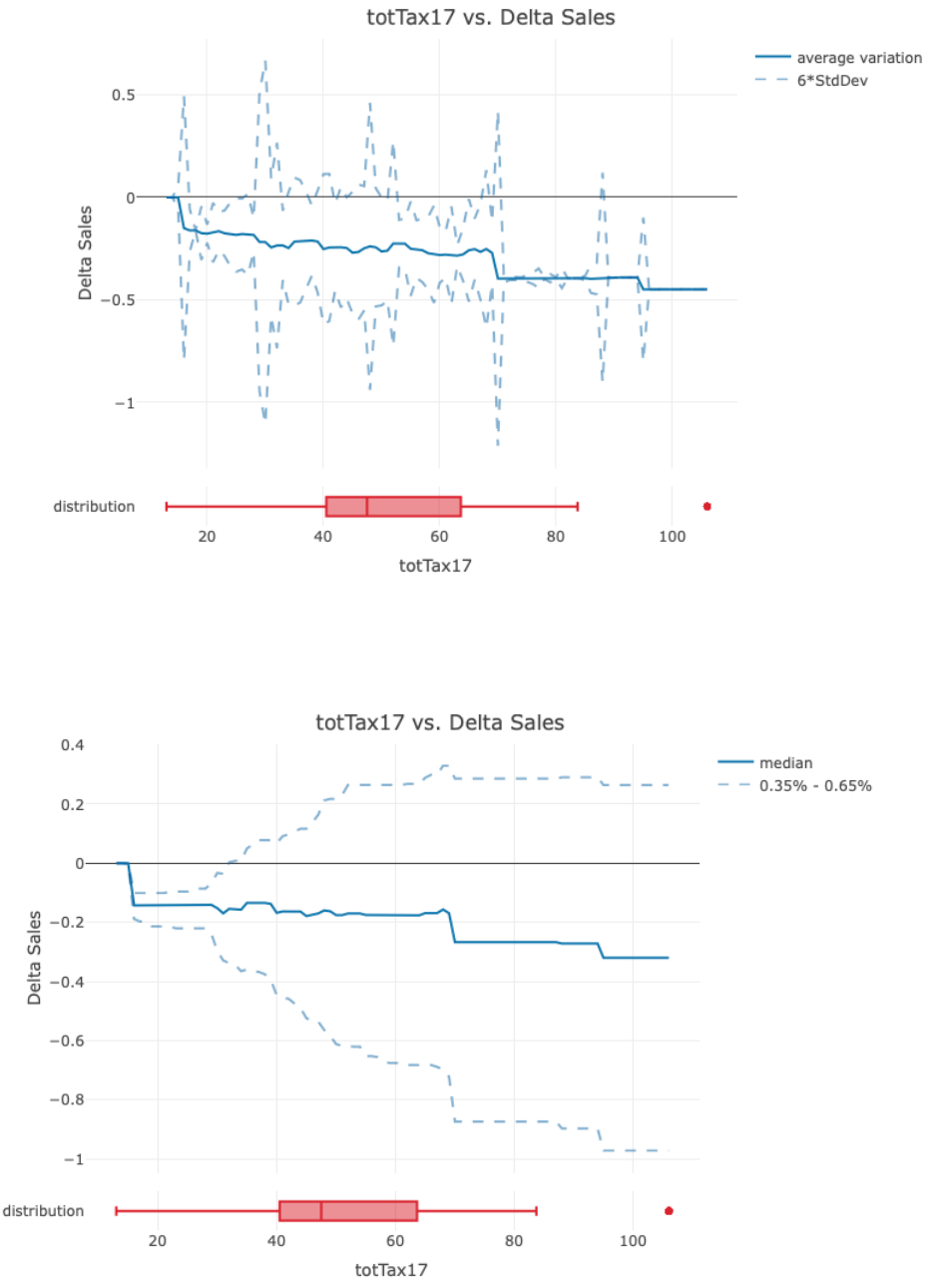


Figure C.11: *totTax17* partial cumulative differences plot.

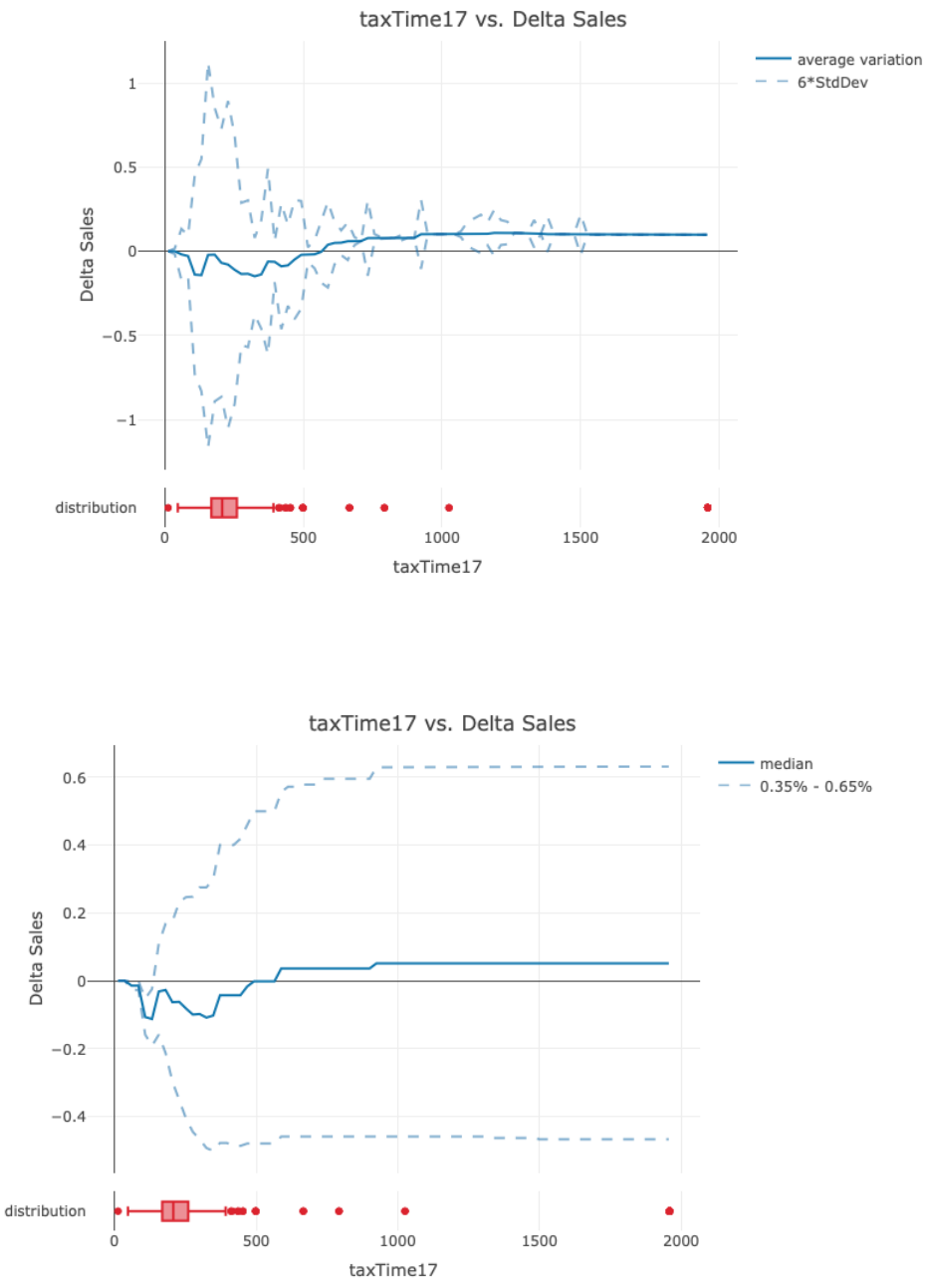


Figure C.12: *taxTime17* partial cumulative differences plot.

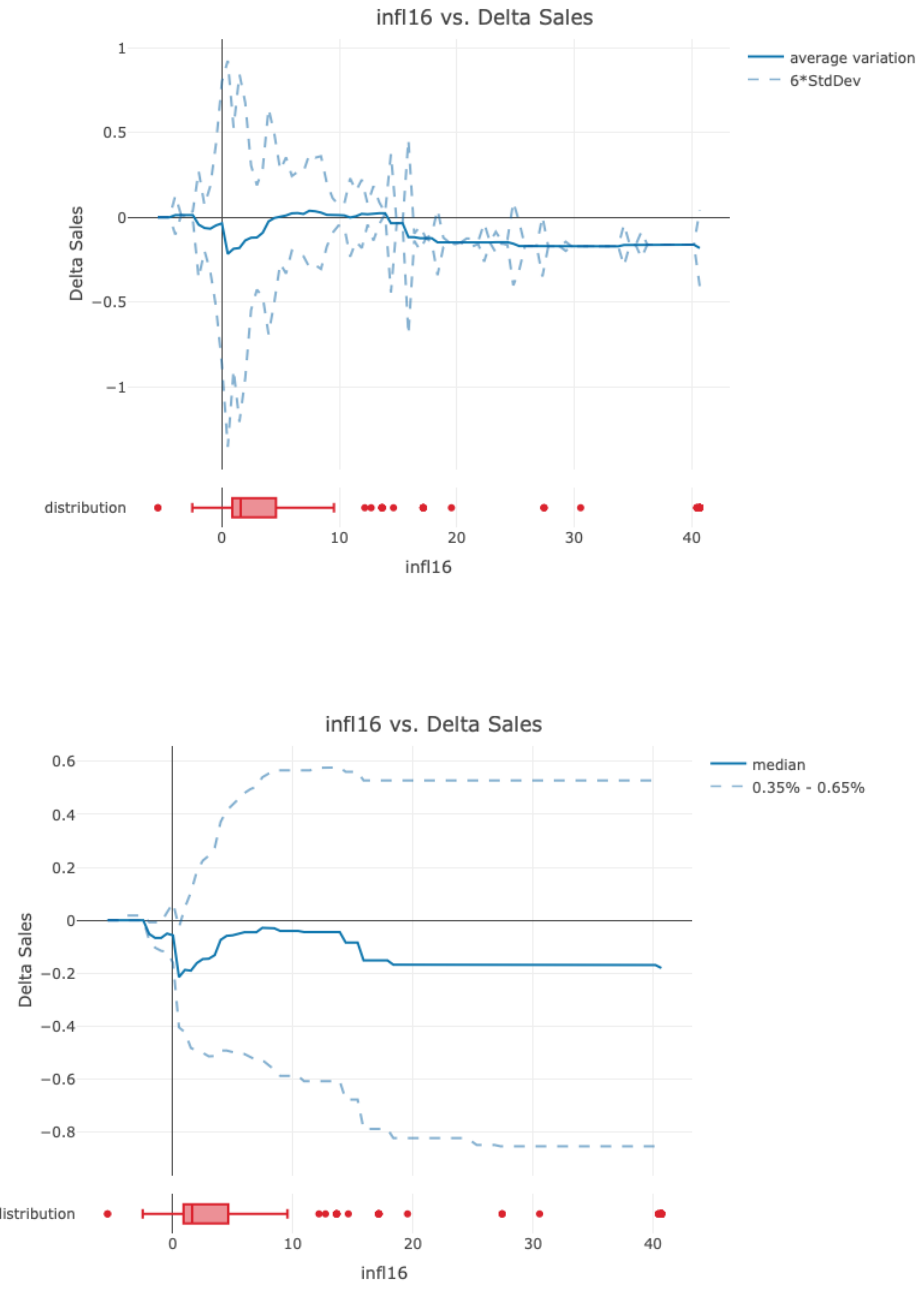


Figure C.13: *infl16* partial cumulative differences plot.

Bibliography

[1] D. W. Apley. Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models. *arXiv e-prints, arXiv:1612.08468*, 2016.

[2] J. Bring. How to Standardize Regression Coefficients. *The American Statistician*, Vol. 48, No. 3, pp. 209-213, 1994.

[3] J. H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, Vol. 29, No. 5, pp. 1189-1232, 2001.

[4] T. Hastie, R. Tibshirami, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2009.

[5] A. Henelius, K. Puolamäki, and A. Ukkonen. Finding Statistically Significant Attribute Interactions. *arXiv e-prints, arXiv:1612.07597*, 2017.

[6] A. Henelius, K. Puolamaki, and A. Ukkonen. Interpreting Classifiers through Attribute Interactions in Datasets. *Proc. 2017 ICML Workshop on Human Interpretability in Machine Learning (WHI 2017)*, 2017.

[7] A. Henelius, K. Puolamäki, I. Karlsson, J. Zhao, L. Asker, H. Boström, and P. Papapetrou. GoldenEye++: A Closer Look into the Black Box. *Proceedings of the Third International Symposium on Statistical Learning and Data Sciences*, pp. 96105. Springer, 2015.

[8] G. James, D. Witten, T. Hastie, and R. Tibshirami. An Introduction to Statistical Learning: with Applications in R. Springer, 2013.

[9] M. Kottek, J. Grieser, C. Beck, B. Rudolf, and F. Rubel. World Map of the Kppen-Geiger climate classification updated. *Meteorol. Z.*, 15, pp. 259-263, 2006. DOI: 10.1127/0941-2948/2006/0130.

[10] C. Molnar. Interpretable Machine Learning. `https://christophm.github.io/interpretable-ml-book/`, 2018.

[11] K. Murrey, and M. M. Conner. Methods to quantify variable importance: implications for the analysis of noisy ecological data. *Ecology*, Vol. 90, pp. 259-263, 2009. DOI:10.1890/07-1929.1

[12] D. Pedreschi, F. Giannotti, R. Guidotti, A. Monreale, L. Pappalardo, S. Ruggieri, and F. Turini. Open the Black Box Data-Driven Explanation of Black Box Decision Systems `https://doi.org/0000001.0000001`, 2018.

[13] G. Ridgeway. Generalized Boosted Models: A guide to the gbm package. `http://127.0.0.1:28243/library/gbm/doc/gbm.pdf`, 2018.

[14] A. Serra. Exploring associations of several variables using mutual information. *Master's Thesis*, 2018.

[15] The World Bank. World Development Indicators. `http://databank.worldbank.org/data/download/archive/WDI_excel_2018_04_18.zip`, 2018.