Dynamic Deal Scoring

LUMEN Data Science 2021.

Project Documentation

Contents

1 Business Objectives

Discounting strategies are used by companies in order to establish themselves on the market. However, data suggests that most discounts are too large, resulting in the loss of significant amounts of potential revenue. To make matters worse, many agents are incentivised to close deals that will bring harm to their company in the long run. Overpricing, on the other hand, can lead to loss of volume, and ultimately also reduce the profits and harm the company.

In other words, one of the key factors determining the profits and the success of companies is how well they set the prices of their products or services. Optimizing the price by 1% can lead to profits increasing by over 8%. Underpricing and overpricing are therefore very important mistakes to avoid.

The optimal price depends on a variety of circumstances and facts about the deal to be made and the parties involved. Taking this into account is the job of agents making the deals. Much of the dependence, however, cannot be figured out by common sense and experience alone. This is where data analysis can be extremely useful.

The solution to these problems requires a sophisticated data driven model. Such a solution would compare a new deal to be made with deals with similar parameters. With this kind of a model, the agent can dynamically compare current deals, in real time, with similar deals from the past, as well as similar deal scored by their peers. The solution would grade the quality current deal (A-F), with A meaning that deal can be automatically closed, and lower grades implying that it should be reconsidered.

1.1 Business goals

The purpose of the data analysis is to assist sales teams with setting the price of their products and specific deals. The deals made so far, recorded within the provided dataset, can give insight into the quality of prices through the comparison with similar deals and their quantity.

Our goal is to create a model that helps predict the quality of new deals to be made, depending on the price, and thus inform the decision of sales agents.

1.2 The data

Data was provided by *Atomic Intelligence d.o.o.*, a company with headquarters in Zagreb. Data used for implementation of the project is property of *Atomic Intelligence d.o.o.* and is available for use only during this project. Any unauthorized use of data is illegal and is punishable by Croatian laws.

1.3 Data analysis goals

Through studying the provided data we want to increase understanding of how the given properties of deals affect the possible prices. Based on these insights we construct a model which, after having learned about the dataset, is able to take a new entry of properties (features) of a deal, and return the price range judged to be reasonable, while also dividing it into five segments which determine the quality of the eventual deal. The five segments represent the five grades, from A to F.

1.4 Project plan

Here we lay out the steps of how our project was carried out.

The first step is studying the dataset. Here we try to gain better insights into what the features mean, how they interact (e.g. whether there are correlations) and how we can identify mistakes and omittances within the data.

The second step involves cleaning and repairing data, and ultimately organizing it in a usable way.

Thirdly we look for correlations and mutual predictability of the features that could be relevant for the solution of the problem. We use *pandas, seaborn, matplotlib, numpy, scipy,* etc. to analyze and visualize what the data can tell us about the optimal pricing of items. We conclude which assumptions about the data can be used to make predictions and which models might suit our needs.

As the fourth step, we choose the model to be used to create the solution in the form of determining the ranges of the five grades from A to F for each new dataset entry used for evaluation. We do this by trying with several models we judge to be applicable for our goal and selecting the most promising and best working one.

The fifth step is final evaluation, where we use our model to judge the deals in the dataset it was trained on, and show that the solution makes sense.

The last step is deployment – we organize our results and our solution and prepare it for use by the grading committee.

2 Data Understanding

2.1 Data description

The provided dataset given in the *CSV* format consists of entries representing successfully carried out deals that a manufacturing company (the identity of which is unknown to us) made to sell their items. Each entry has 33 features, which are described in the following list:

- *Manufacturing Region* – region of manufacturing location on a global scale, e.g. the

continent

- *Manufacturing Location Code* – A code representing the specific manufacturing location
- *Intercompany* – a YES/No feature saying whether the given deal was a sale to an intercompany or to an end customer
- *CustomerID* – a unique identifier of the customer
- *Customer industry* – a code representing the industry the customer belongs to
- *Customer Region* – global scale region of the customer
- *Customer First Invoice Date* – the date on which the customer of this deal had their first invoice
- *Top Customer Group* – important customers across locations are grouped
- *Item Code* – a code unique to the item sold
- *Product family* – rough grouping of the product
- *Product group* – much finer grouping of the product
- *Price last modified date in the ERP* – date when the price of the item was last changed
- *Born on date* – date when the item was first introduced/manufactured
- *Make vs Buy* – signifies whether the item was manufactured or bought and resold
- *Sales Channel - Internal* – Name of the internal Sales representative
- *Sales Channel - External* – Name of the external Sales representative
- *Sales Channel* - Grouping – Sales representative channel
- *Invoice Date* – date when the invoice took place (when the item was shipped/delivered)
- *Invoice #* – unique invoice number
- *Invoice Line #* – Line item number for the invoice
- *Order Date* – date when the item was ordered
- *Order #* – unique order number
- *Order Line #* – line item number for the order

- *Invoiced qty (shipped)* – quantity of the item that was invoiced
- *Ordered qty* – quantity that was ordered
- *Invoiced price* – unit price used on invoice
- *Invoiced price (TX)* – unit price on invoice accounting for tax schemes of certain regions
- *Cost of part* – unit cost used on the invoice
- *Material cost of part* – segment of the unit cost related to the material used
- *Labor cost of part* – segment of the unit cost related to the labor
- *Overhead cost of part* – segment of the unit cost related to the overhead costs
- *GM%* – gross margin percentage - uplift of price over cost
- *# of unique products on a quote* – the number of items sold together

2.2 Gross margin %

The gross margin percentage is our most important feature, since it is what we're trying to predict and grade based on other features. It is defined as the uplift of price at which the company sold it over cost of item (cost of the parts used, the process of manufacturing etc.):

$$GM\% = \frac{\text{price} - \text{cost}}{\text{price}} \tag{1}$$

Since both the cost and price are non-negative values, the maximal *GM%* is 1. It will be negative if the item was sold at a price lesser than its cost for the company, which is a situation that should be avoided. The *GM%* signifies what percentage of the price of an item is the profit of the company, and therefore the higher its value, the better the deal. Of course, deals of a too high *GM%* will not be able to be made since no customer is willing to pay an arbitrary amount of money. Therefore the realistic yet good enough *GM%* will depend on the features of the deal. Our objective is to determine which range of the *GM%* is reasonable based on the features of a deal, and to divide it into five grades that are equally likely to occur based on the dataset.

2.3 Data exploration

2.3.1 Feature distribution

In following figures we can see distributions of various, features.

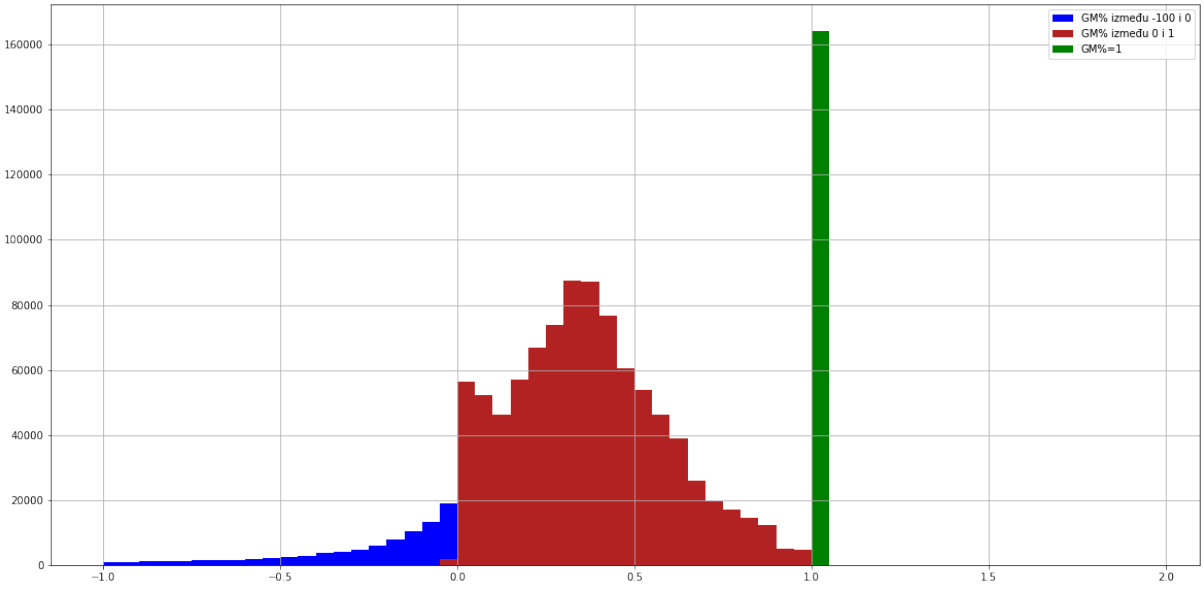

Figure 1: Histogram, representing $GM\%$ distribution across whole dataset. It is obvious that $GM\%$ should follow log normal distribution. Problematic regions correspond to $GM\% = 1$ and $GM\% < 0$

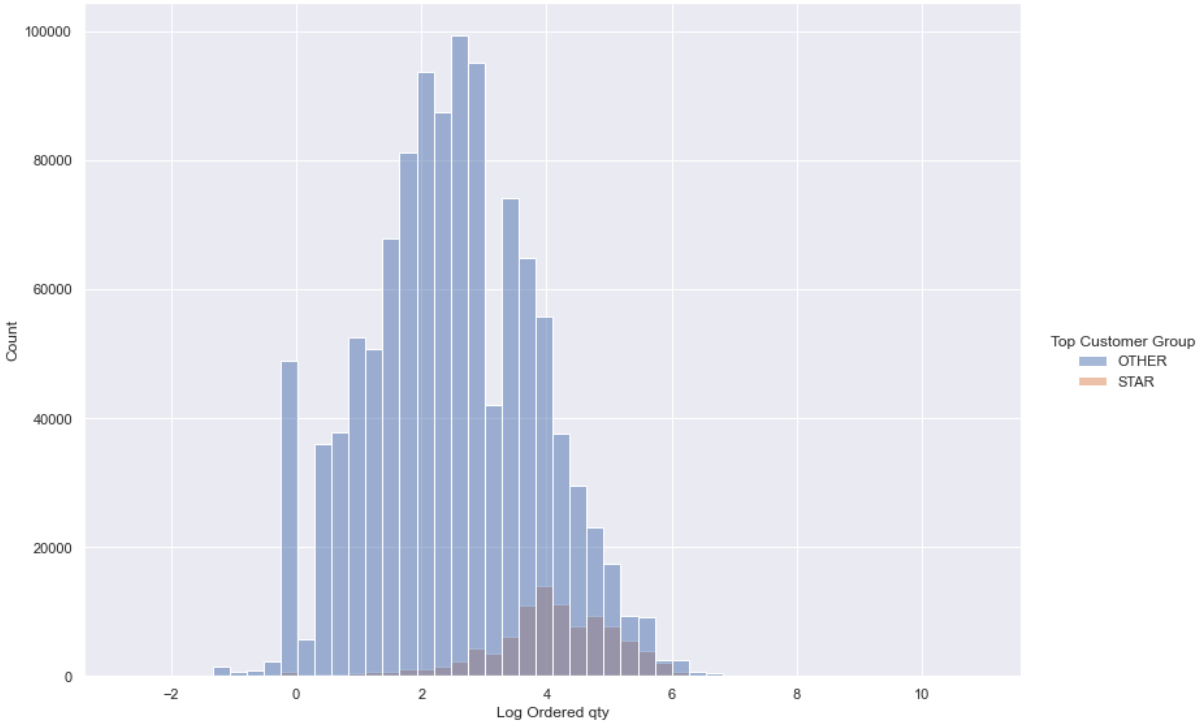

Figure 2: Histogram, representing*log ordered qty* distribution across whole dataset. As we can see customers from *Top Customer group* = STAR tend to order more. And *Ordered qty* is log normally distributed

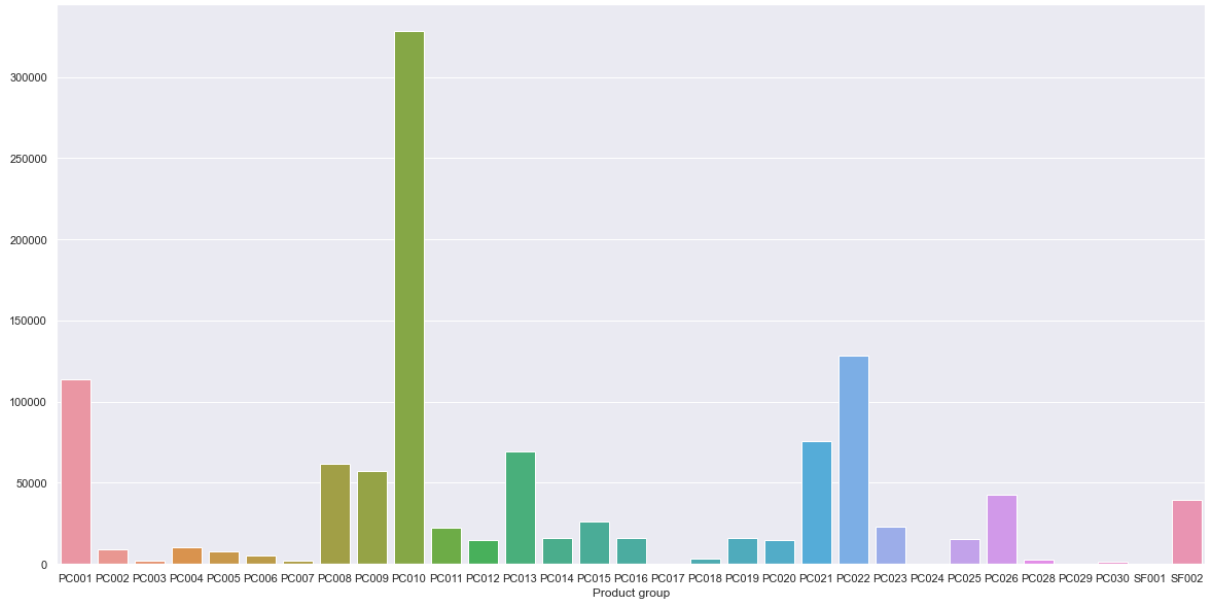

Figure 3: Histogram, representing proportion of each *Product group* in dataset

2.3.2 Natural clustering

As we can see in 4. Log ordered quantity and log cost of part form natural clusters, provided *Manufacturing Location Code* is fixed.

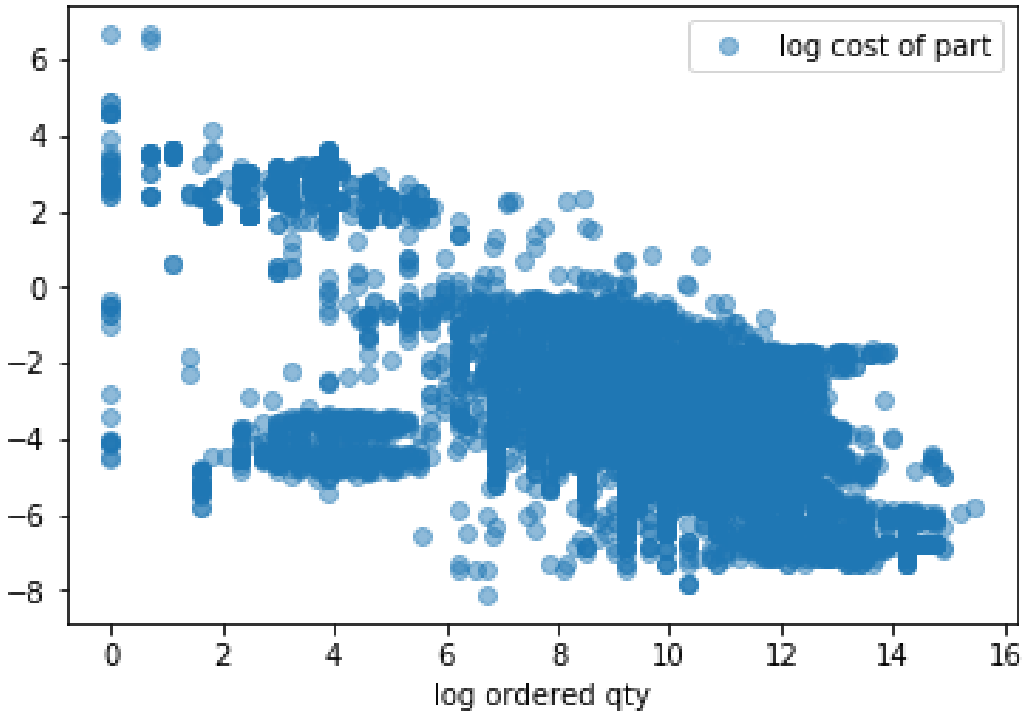

Figure 4: The 2D scatter plot showing the "natural" clustering of *log ordered qty* and *log cost of part* for particular Manufacturing Location Code

2.4 Data quality report

Several problems with the data were noticed.

The biggest problem is a large amount of missing data. Figure 5 shows the amount of missing data for each feature with corresponding numerical percentages. Many columns have a large percentage of missing values, some even as big as 70%. If we dropped all entries which contain any missing values the data would be too diminished to provide any useful insight.

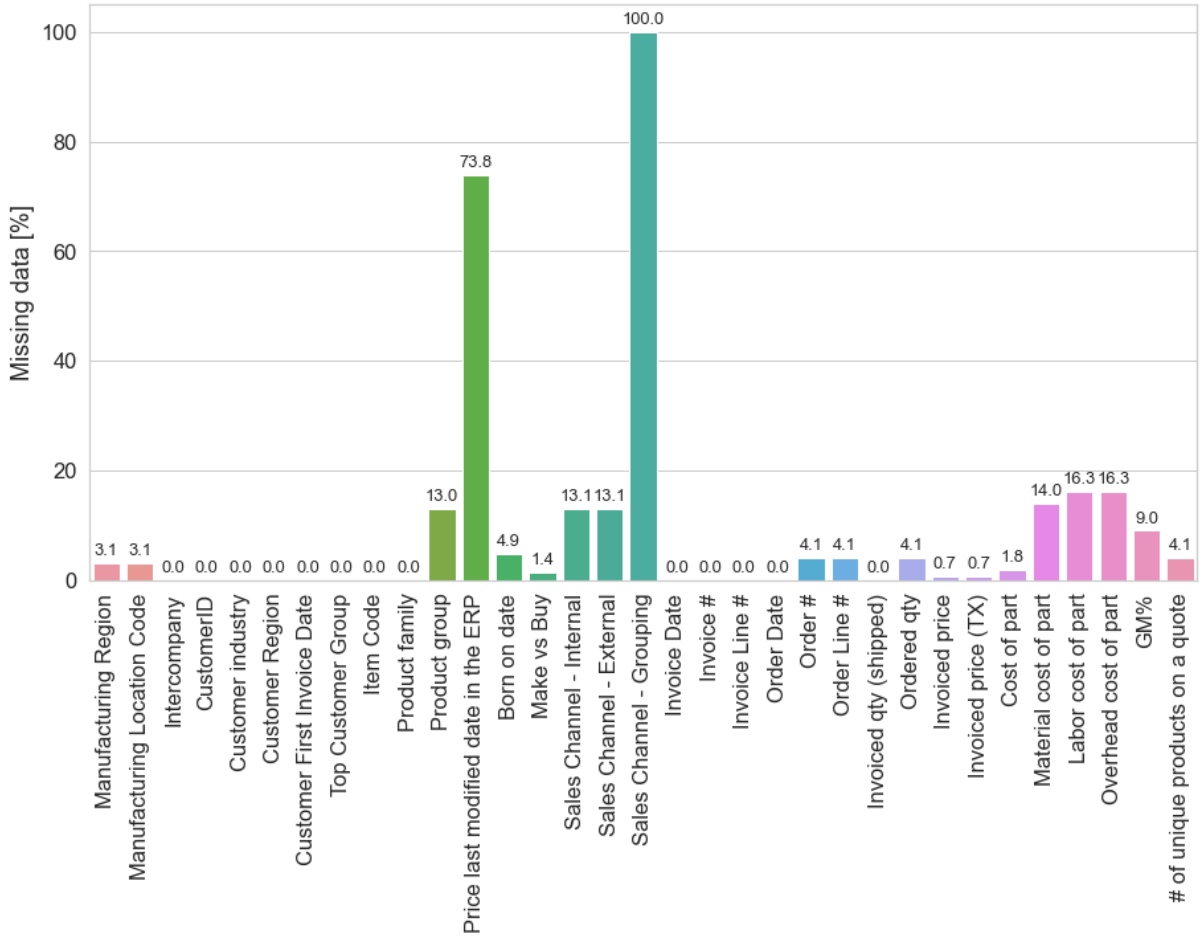

Figure 5: The percentage of missing values for each column

The second problem are the values which are not explicitly missing but which don't make sense for the column in which they are in. For example, negative or zero prices and dates in the future. All of those nonsensical values should be removed and treated as missing.

The data is was collected over several offices of a large company therefore it contains some inconsistencies. The given parameters are somewhat correlated and that is expected if they all describe the same item or the same client.

Based on what they represent, we expect a several relationships between the features. For example, all the partial costs should sum up to give the total cost of part and the same

item should always belong to the same product group. Sometimes there are inconsistencies in those relationships and they indicate errors in the data. These inconsistencies will be further explored and fixed in 3 section.

Additionally, the data contained some miscellaneous rough errors which needed to be located and fixed.

3 Data preparation

We explored the distribution of missing and nonsense data on a column by column basis in order to find a rational and preferably unbiased way to fill in the missing values. Filling in the data was not always possible so some data entries had to be dropped but the aim was to keep that to a minimum. In this section we go through the different approaches for finding the nonsense data and filling in the missing. We will also explain the inconsistencies we found and how we fixed them.

3.1 Rough errors

A small number of entries had a wrong separator between *Item Code* and *Product family* which caused the values to be displayed in the same column. Therefore, all the subsequent column values for those entries were shifted to the left. We found the shifted entries, separated *Item Code* from the *Product family* and moved the shifted values to the right columns.

One entry for the *Item Code* "652181" has the *Invoiced price* equal to $1.089758 \cdot 10^9$ which is clearly a mistake because other prices for that item are around 14. We removed that entry.

Items "065291" and "3002-84" have extraordinary high standard deviation of *Invoiced price* and when we looked into it further we discovered that there are entries with negative prices which are exactly equal to another positive price. We removed both the positive and the negative price entries. This might be because an item was returned by the customer. We attempted to find other similar entries for other items but we were not able to find a consistent way to do that without risking removing valid entries.

Sales Channel - External and *Sales Channel - Internal* contain the exactly same values and we dropped one of them because it was double information. *Sales Channel - Grouping* has no values and it was just dropped.

3.2 Imputing categorical features based on *Item Code*

It was reasonable to assume that each item (identified by *Item Code*) belongs to a unique *Product group* and we checked and confirmed that it is true. For this reason we can use the *Item Code* to fill some missing *Product group* entries. We will explain the idea for

filling those values in a general way because it can be used in for several other pairs if variables. We can fill a target variable with a reference variable if we assume that the target is constant for each reference value. The idea is to first find all the unique reference variables for the entries which are missing the target variables. After that we determine which of those unique reference variables have at least one entry where the target variable is not missing - for those values of the reference variables it is possible to fill in the target variable. We iterate over the possible values of reference variable and for each value fill in the target variable with the value which it has on a place where it is not missing.

We applied the method we just explained to fill in *Product group* based on *Item Code* and it reduced the percentage of missing *Product group* values form 13% to 7.8%.

All entries have a specified product family "PF000", "PF001" or "PF002", but we had reason to suspect that "PF000" actually signifies that the product family is missing. Let us explain our reasons. After we fixed *Product group* based on *Item Code*, we were checking if each product group belongs to a unique product family. We discovered that some product groups contain items belonging to multiple product families, which is fine. However, if we group by product group, the number of items belonging to product family "PF000" was much smaller compared to the number of items belonging to other product families. We then went a step back and checked if each product group belongs to a unique product family before the product group filling. We discovered that all the entries with "PF000" and a valid product group were the ones we filled and therefore should probably have a different product family. The other reason for our suspicion is that entries where *Manufacturing region* and *Manufacturing location code* are missing are exactly the ones where the *Product family* is "PF000". And finally, for all of the items (prior to any corrections) with "PF000" the product group is missing. Our suspicion was confirmed by the organisers so we proceed to fill in the "PF000" entries with the correct ones based on *Item code*. Filling reduced the percentage of entries with "PF000" from 3.1% to 1.9%.

The next feature we wanted to fill in based on *Item Code* is *Manufacturing region*. Most items are manufactured only in one region but not all. Before we filled the *Manufacturing region* we checked if any items, for which it is possible to do the filling, are produced in multiple regions. We discovered only three ("FAI", "FREIGHT" and "OTESC1") and we skipped filling based on those *Item Codes* because the *Manufacturing region* was ambiguous. According to the name "FREIGHT" is probably some kind of transportation which is being performed on and between multiple continents but we were unable to find any other irregularities for the skipped *Item Codes*. We were able to fill 1.2% of *Manufacturing region* values, taking the percentage of missing values from 3.1% to 1.9%.

We also filled *Manufacturing Location Code* based on *Item Code* because most items were produced in only one factory To avoid inconsistencies we skipped items: "FAI", "OTESC1", "/C" and "*250001". We lowered the number of missing *Manufacturing Location Code* values from 3.1% to 2.5%.

Finally we filled *Make vs Buy* column missing values based on *Item Code*. We confirmed that items belong to a unique *Make vs Buy* category. Filling reduced the percentage of missing *Make vs Buy* values from 1.4% to 1.1%.

3.3 Processing price and cost columns

Negative and zero prices and costs are not valid entries so we removed them and we will attempt to fill them correctly. The new percentages of missing values are shown in figure 6. *GM%* values are consistent with *Invoice price* and *Cost of part* which means we will not be able to use them to fill the cost or price.

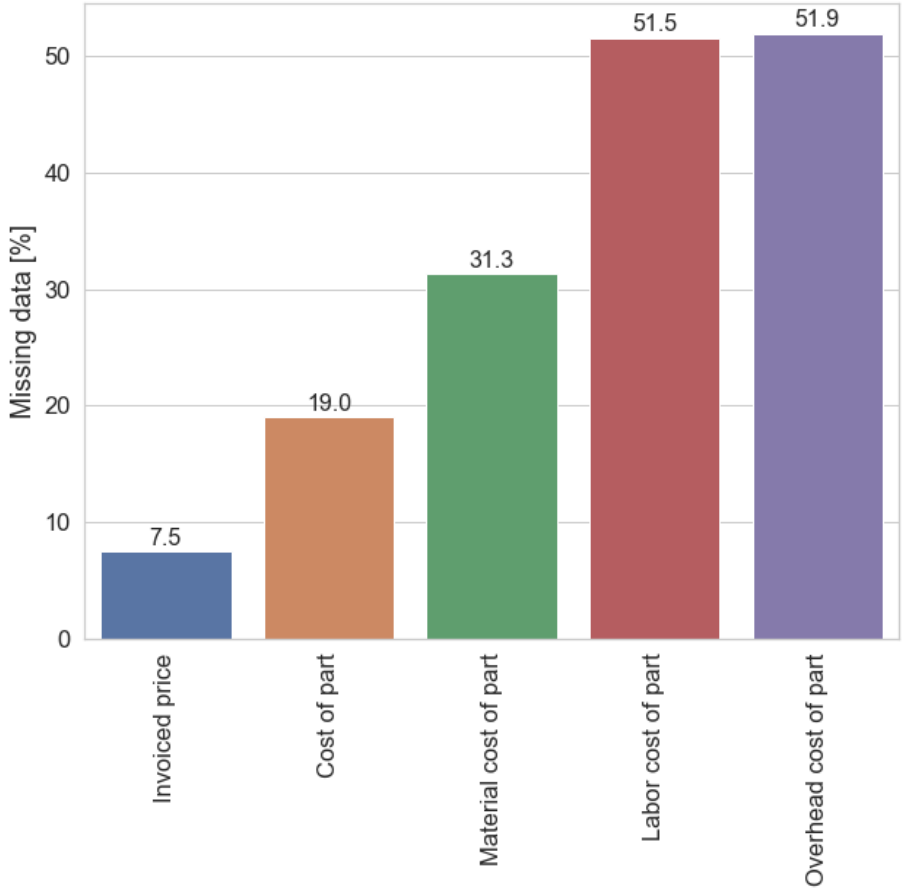

Figure 6: The percentage of missing cost and price values after removing zero and negative entries

Our goal is to create a model which will help in the formation of prices and we are using known GM-s for that. If we attempt to fill in the missing prices we have to do the same thing we want the model to do and that wouldn't be legitimate. However, we can try to fill in the *Cost of part*.

First, we are trying to fill the *Cost of part* based on *Item Code* because we expect the same item to cost approximately the same every time. Of course there are exceptions and we need a way to find them and quantify if it is alright to use them. We will put a

limit on the relative standard deviation of cost (M_r) for each item and fill only based on *Item Codes* for which have M_r below the limit. Explicitly the relative standard deviation of a set of values $x_1, x_2, ..., x_N$ is defined as:

$$M_r = \frac{\sigma}{\bar{x}} = \frac{\frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{x})^2}{\frac{1}{N}\sum_{n=1}^{N} x_n} \tag{2}$$

where $\bar{x}$ denotes the mean value of the set.

The histogram on figure 7 shows the distribution of relative standard deviation of the cost of part over the whole dataset. The green plot shows the percentage of cost which can be filled if we put the limit on a certain relative standard deviation.

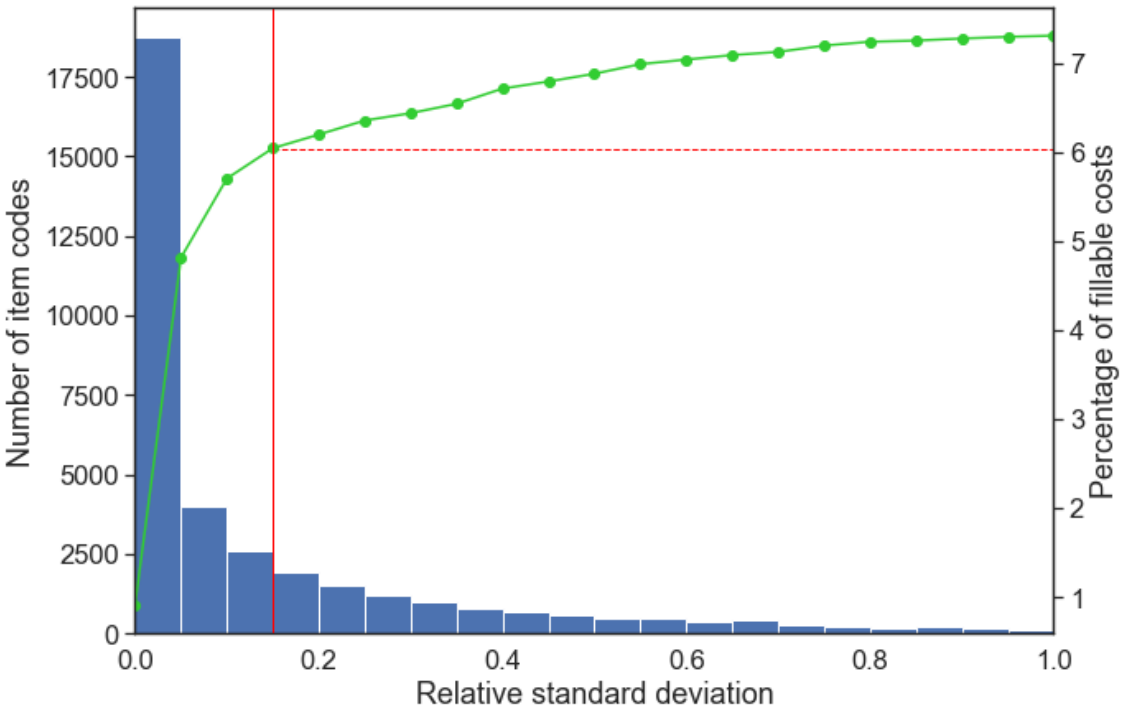

Figure 7: The histogram of relative standard deviation of the cost of part over the whole dataset in blue and the percentage of cost which can be filled if we put the limit on a certain relative standard deviation in green. The full red line represents the chosen limit of 0.15 for relative standard deviation and the dashed red line marks 6% of the costs which can be fixed with a chosen limit.

We have to chose a limit for the relative standard deviation which will enable us to fix a good portion of costs but still not include the items where the existing cost varies too much. We chose the limit of 0.15, which means that for filling we only used items for which the average variation of cost is lower than 15%. Just for reference this condition is satisfied by 40% of items which have more than one cost present. This limit enables us to decrease the percentage of missing item costs by 6%. For the items which satisfy the condition we filled the cost with the mean value for that item. The percentage of missing *Costs of part* decreased to 13%.

After filling by *Item code* we looked for other variables to be used in the same way to fill the *Cost of part*. We decided to use *CustomerID* and *Order #* because if we group by those features the relative standard deviation is small for a large part of the dataset. The corresponding plots are on figure 8. We also checked other features but they either did not have a good relative standard deviation or would not fill a significant percentage of costs. Filling by *CustomerID* makes sense if we assume that some customers only purchase similar items which means that the costs are similar. Filling by *Order #* might be good because similar costing items are being ordered together. Of course this is not the rule but we avoid the non favorable cases by setting a limit on relative standard deviation.

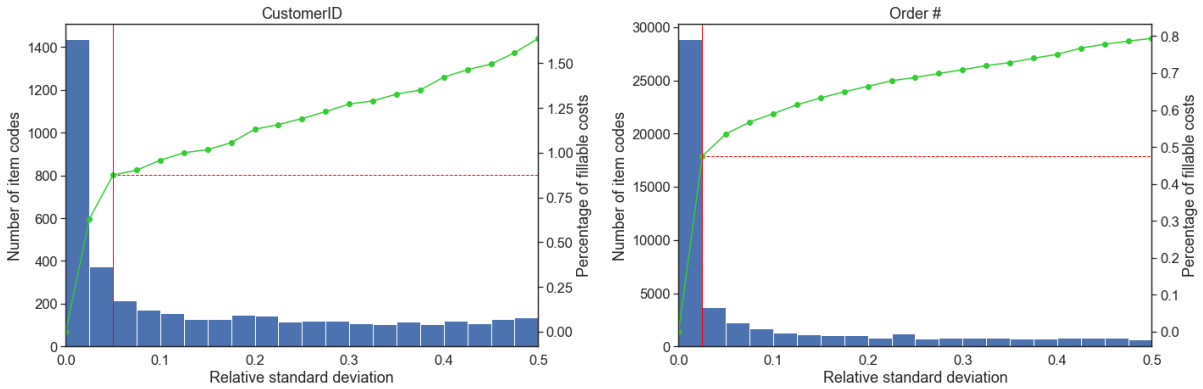

Figure 8: The same as figure 7 for filling based on *CustomerID* and *Order #*, note the change in x axis. The limits for *CustomerID* and *Order #* were set to 0.05 and 0.025, respectively.

For the items which satisfy the conditions we filled the cost with the mean value for that *CustomerID* or *Order #*. We filled 0.9% of costs using *CustomerID* and 0.5% using *Order #*. This leaves us with 11.6% of missing *Cost of part* values.

3.4 Filling *Labor* and *Overhead cost of part*

All the partial costs should sum to give the *Cost of part*. In the dataset there are differences which are shown in figure 9. The small differences in the figure are clearly the result of rounding but there are other differences which might be errors.

When the *Cost of part* was smaller than the sum of partial costs we replaced it with the sum because we assume that some partial cost were imputed after the total cost was calculated.

For filling the partial costs we looked for correlations and we plotted them on figure 11.

Labor and overhead cost of part are highly correlated and we can use this relationship to fill them for instances where we know the Material and total cost.

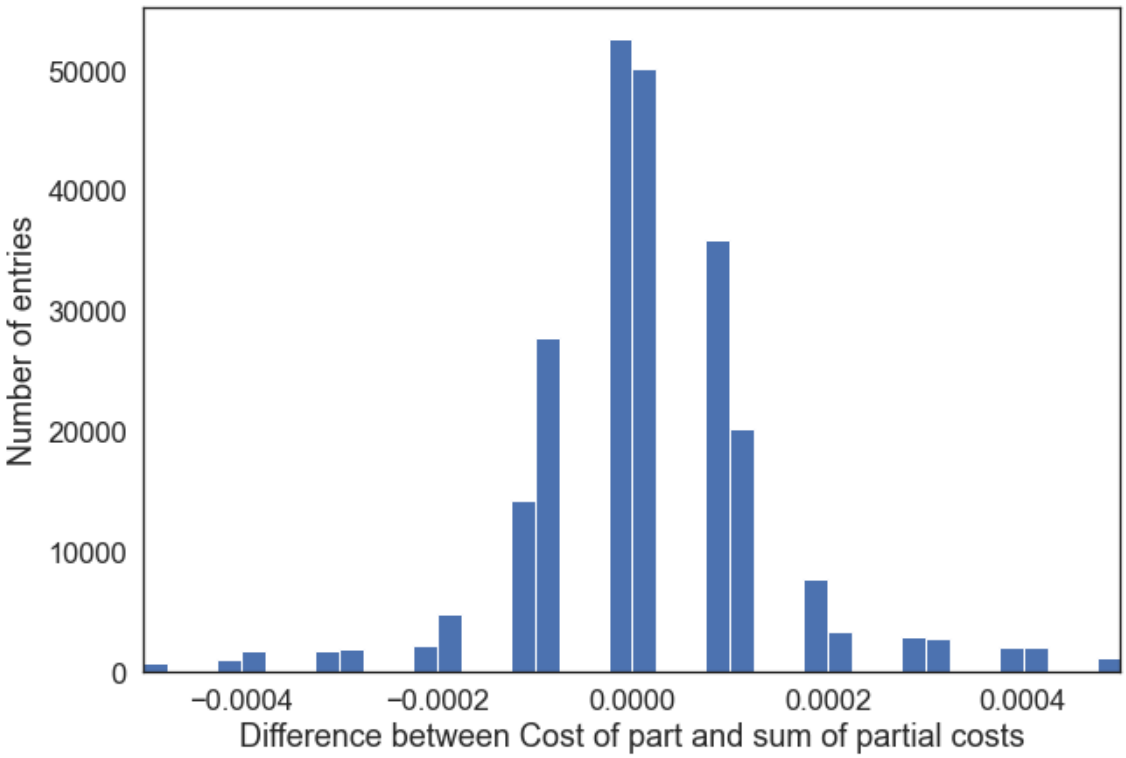

Figure 9: Differences between total cost and sum of partial costs

The correlation is especially strong if we separate by *Manufacturing Location Code* which is shown on figure 11.

For each *Manufacturing Location* we made a linear fit on log-log plot of labor and overhead cost of part and used it to fill the costs. A few examples are on figure 12

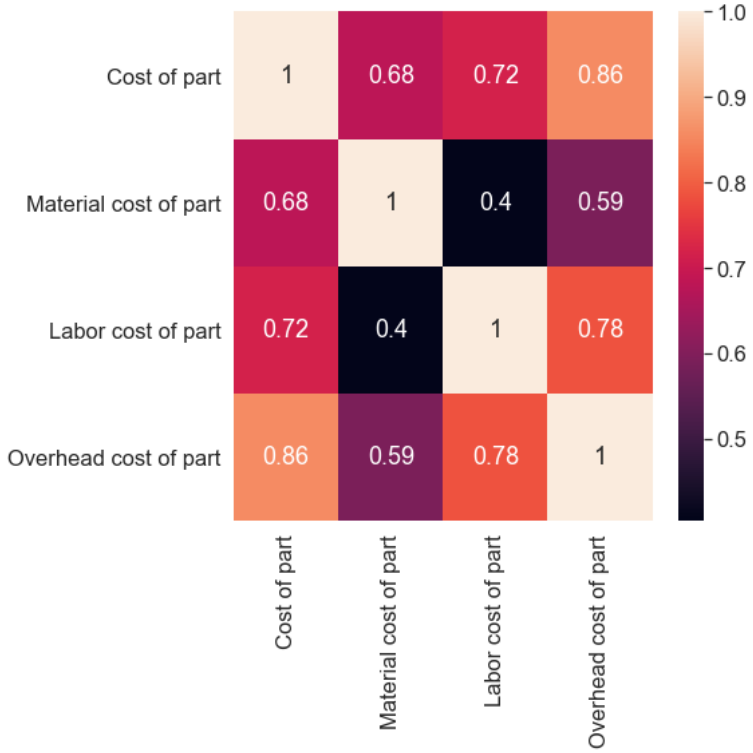

Figure 10: The correlation of costs

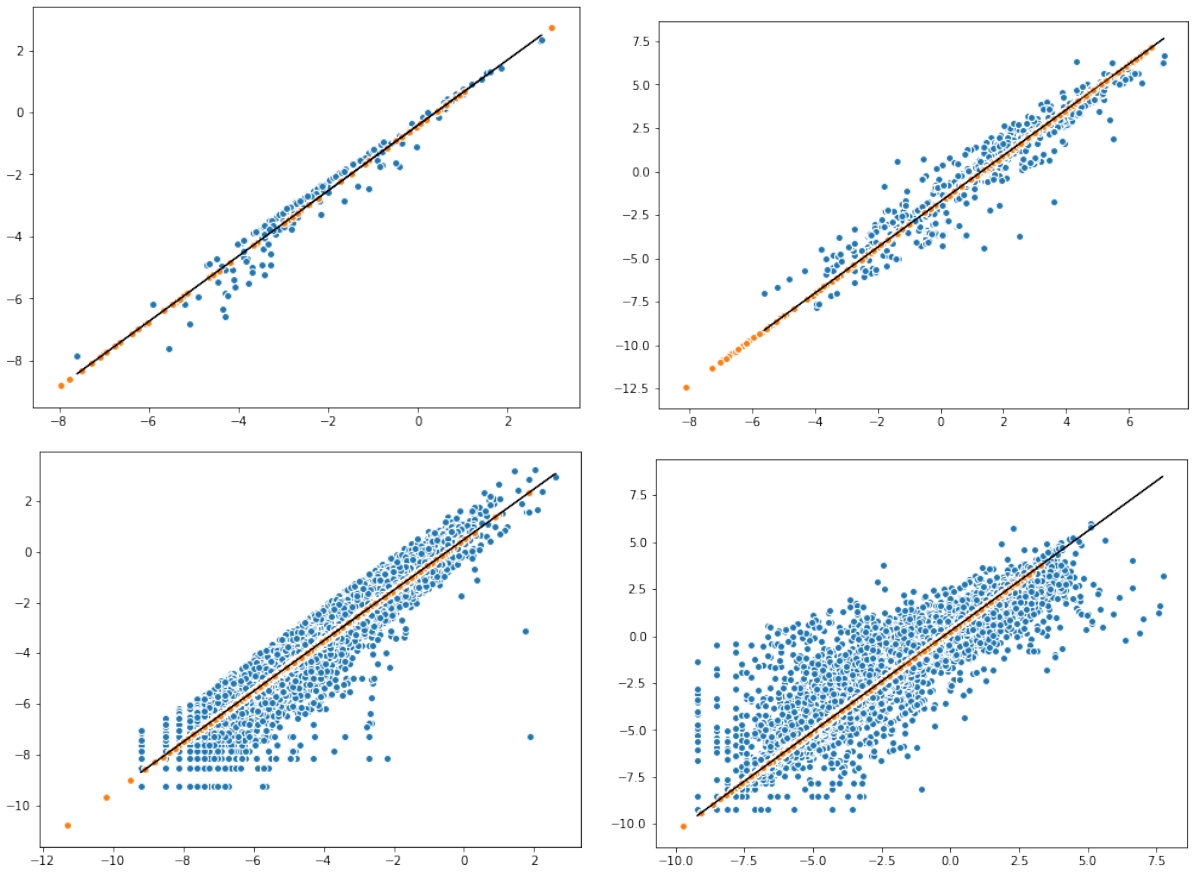

Figure 12: The relationship between logarithms of Labor (x-axis) and Overhead (y-axis) cost of part. The blue points are the existing values while the orange points are the imputed. The black line is the linear fit on the existing values. Manufacturing Location Code from left to right and top to bottom: N9, N12, B77, B6

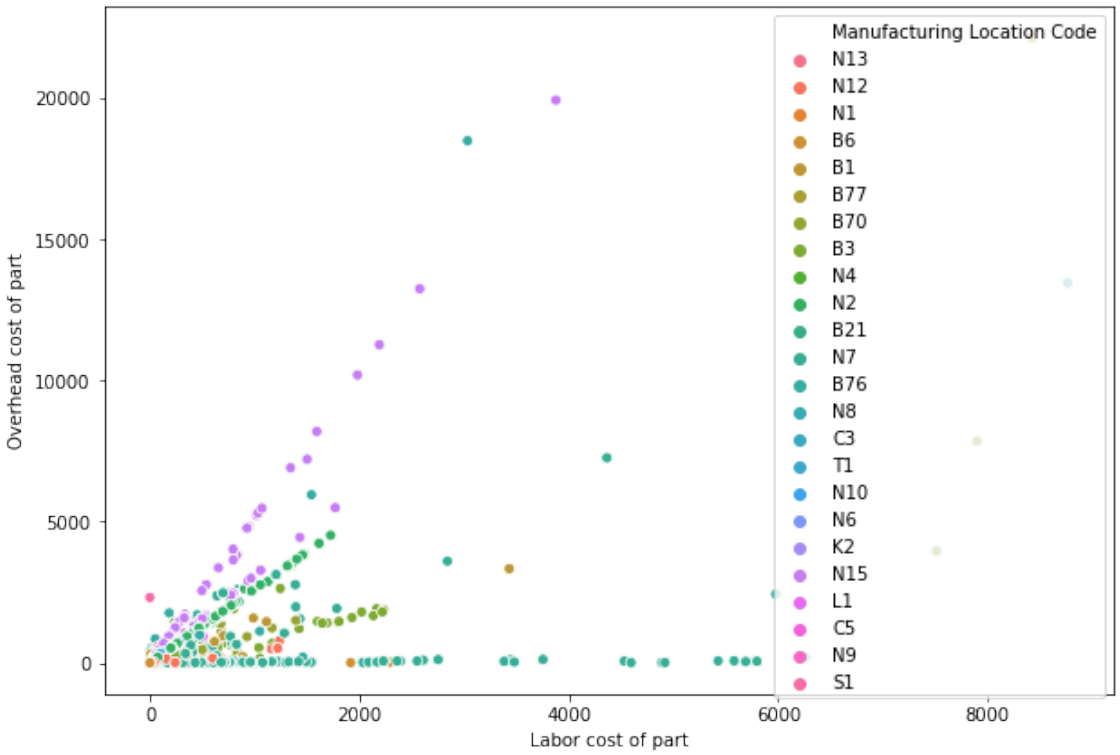

Figure 11: The relationship between labor and overhead cost of part grouped by *Manufacturing Location Code*

There were a couple of locations (B21 and N15) where the linear relation was not good and they are on figure 13. Those locations were not used for data filling.

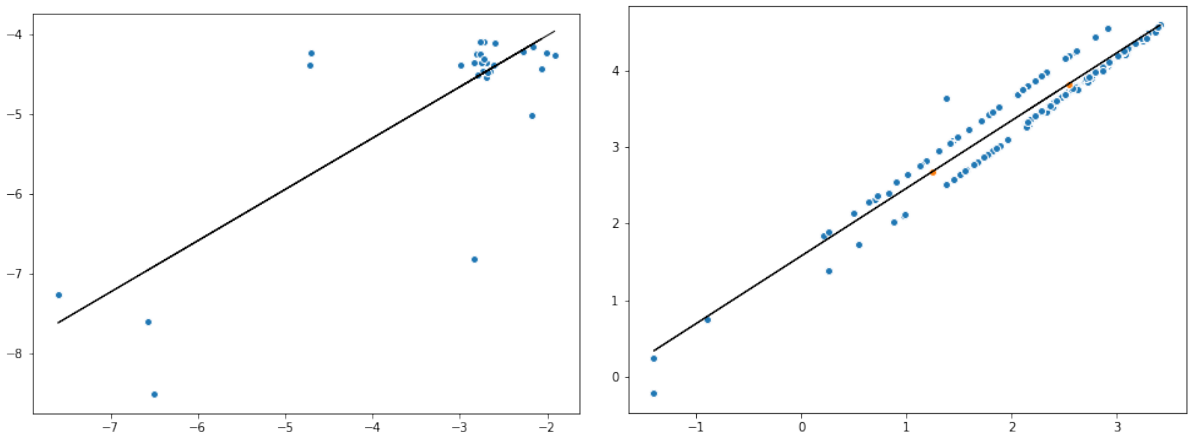

Figure 13: The relationship between logarithms of Labor (x-axis) and Overhead (y-axis) cost of part for the locations B21 and N15 which were not used. The labels are the same as figure

For location N7 we had a special situation because the relationship between labor (x-axis) and overhead (y-axis) cost of part show two different trends. We were able to separate them by fitting a line on all data and then separating the points on either side

of the line (figure 14 left). We discovered that the clusters are most clearly separated by *Invoice Date* (figure 14 right) and we used it to do separate input curves for the two clusters. Separation by date probably occurs due to the automation of the manufacturing process in the factory because on the later dates labor costs decreased and the overhead increased. The factory needed fewer people but had more other expenses.

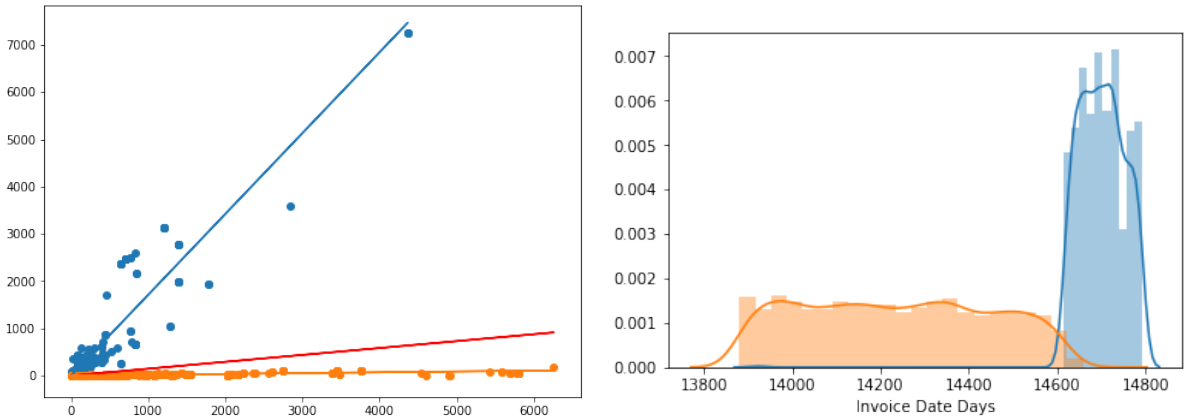

Figure 14: **Left:** The separation of two trends in the relationship between labor (x-axis) and overhead (y-axis) cost of part for the locations N7. The two trends are colored blue and orange, and the red line is the fit which separates them. **Right:** The histogram of invoice dates for manufacturing location N7 where two trends are marked orange and blue the same as on the left part.

In the case where the *Material cost* is equal to the *Cost of part* we fill the other partial costs with zeros. The original dataset had many zero values and for some items it could actually be equal to zero.

3.5 Processing date columns

There are five features of the dataset representing dates – *Customer First Invoice Date, Price last modified date in the ERP, Born on date, Invoice Date* and *Order Date*. We checked them for consistency among themselves and with other features, as well as for whether they make sense in general. Many of them didn't, in ways explained below, and many were missing.

It was firstly useful to turn all these date features into datetime type variables for easy access to years and months, since all the entries were written in the appropriate format to begin with. Some of the dates were automatically removed such as those with year 9999. Next, we created new features representing the dates in days past since 1980-01-01. This helped the process of repairing as we could work with pure numbers. We manually checked for nonsense dates and discovered entries of *Born on date* in 1900 and 2021 which were clearly not correct and we removed them. Other date columns contained no values

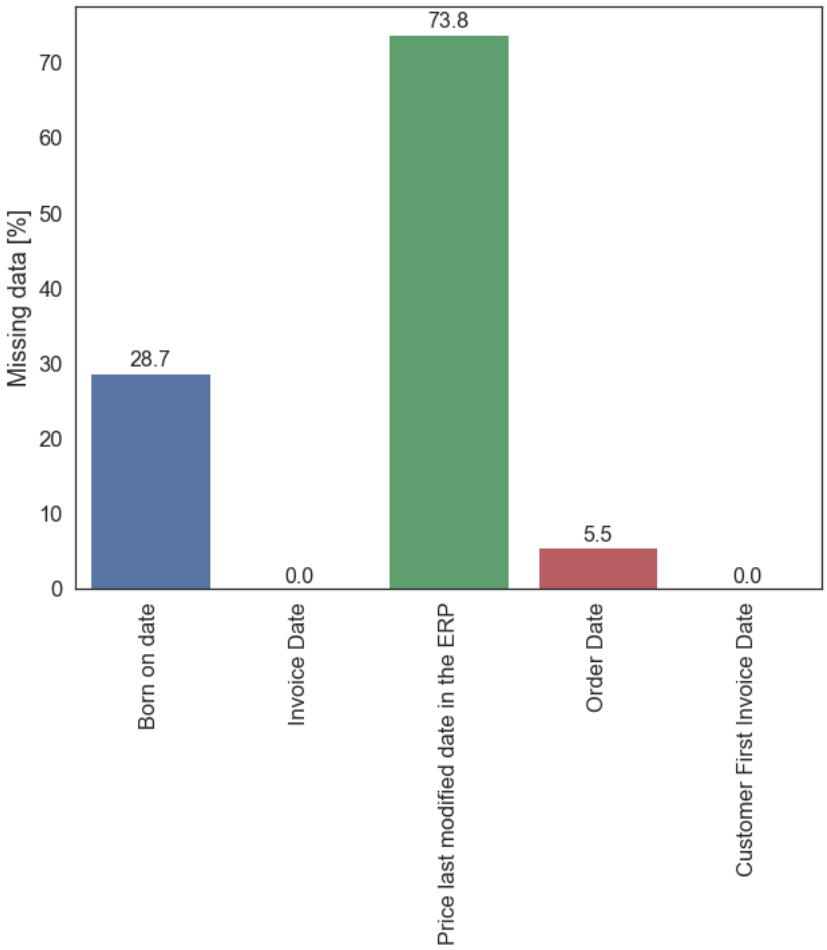

Figure 15: The percentage of missing date values

which were clearly wrong.

The first repair we made was by taking valid *Order Dates* and filling the missing ones that have the same Order # as the entries with valid ones, since equal Order # means they were part of the same order and therefore should have the same Order Date.

Next, we switched the *Order Date* and the *Invoice Date* in entries where the invoice allegedly happened earlier, since that is clearly nonsense, and this was the simplest assumption we could make. In places where Order Date was still missing, we filled it statistically based on the rest of the data and its Invoice Dates. We did this by plotting the *Order Dates* and *Invoice Dates* (leaving out the 10% of the entries with the two dates being the most distant) and doing a linear fit (which worked very well), which represented the most likely Order Date as a function of the *Invoice Date*. The plot and fitted curve are shown on figure 16. We used the fit to imput the missing values of *Order Dates* for the known *Invoice Dates*

Born on Date is characteristic of each unique item, meaning it has to be consistent for all entries with the same *Item Code*. When it wasn't, we replaced all the *Born on Date*s with the earliest one for that *Item Code*. We also replaced the *Born on Date* with

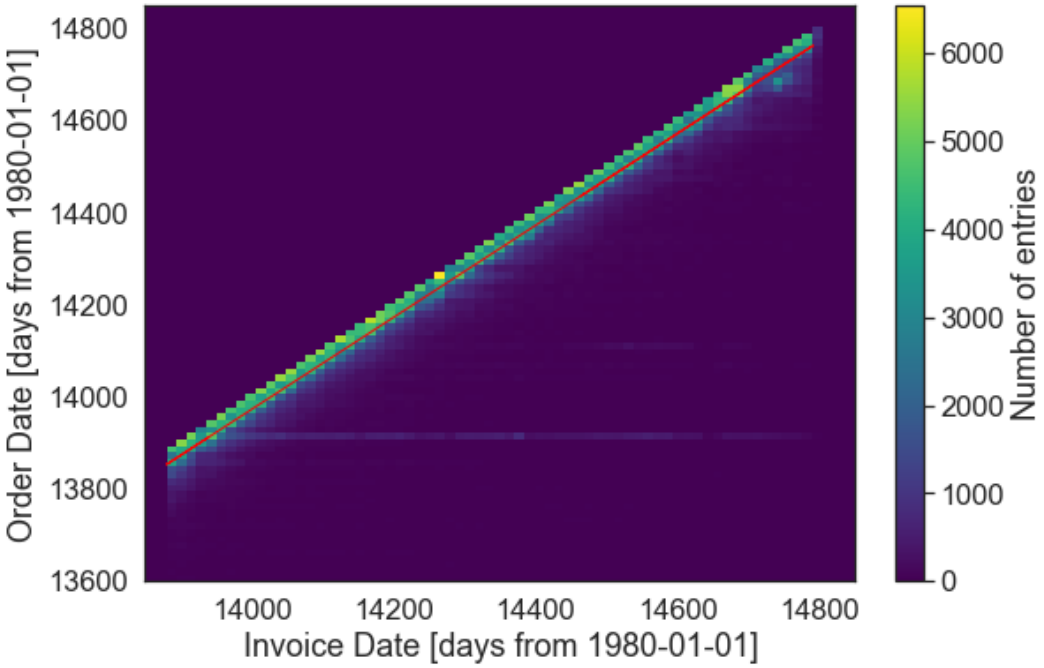

Figure 16: The 2D histogram showing the relationship of *Invoice Dates* and *Order Dates*. The red line is the linear fit.

the *Order Date* in places where the *Order Date* was earlier.

Last modified date (*Price last modified date in the ERP*) necessarily has to be later for an item than its *Born on Date*, so we replaced the two where it wasn't so. We also filled Last modified dates where they were missing on items that appear in other entries and weren't missing them.

Where either *Born on date* or *Price last modified date* was missing we filled it with the other. Finally, where both dates were missing we filled them in with the earliest order date for that item.

The final form of the date features has no missing values. While some of them might be off compared to the actual dates when the events took place, they are statistically speaking well approximated, which is a significant improvement from the original data, the missing dates in which are shown in Figure 15.

3.6 Item Code string clustering

A pattern among *Item Codes* was observed, there were many items written in the same format, i.e having the same number of characters and using the same separators. So a simple algorithm was designed to group items into finer groups. However, we did not use these groups in our solution, due to the fact that we might have created artificial noise that would interfere with results.

3.7 Final data processing

The percentage of missing data after removing nonsense and filling where possible is shown in figure 17. It looks like the percentage of missing data increased in some columns, but this is because we removed the zero costs.

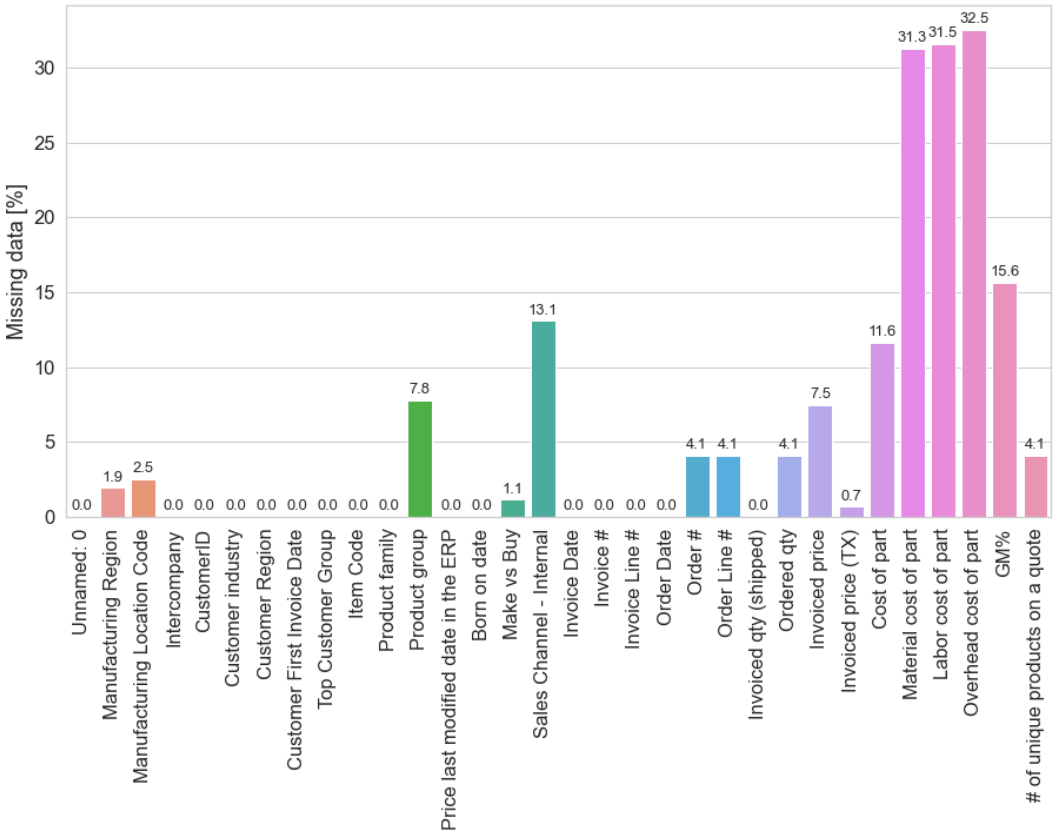

Figure 17: The percentage of missing values for each column after removing nonsense and filling.

The amount of entries missing partial costs could be reduced by using a random forest to predict the *Material cost of part* and than using the linear relationship between the *Labor* and *Overhead cost of part*. The partial costs were not very important for the final model.

Finally all the columns which contained missing values in the variables which were used for the model were dropped. Dropping the rows where *GM%* was missing was the most significant step, since it is the dependent variable to assess.

Additionally, months were extracted from dates to give information about seasonal changes.

4 Modelling

4.1 Hypothesis

The main objective is to find a way to group similar deals. After grouping them, a grading of deals can be made. Those deals that are on the lower end of the *GM%* are graded as worse, while bigger margins signal better deal execution by the agent. Our task was to find a model which can group similar deals effectively.

Target variables are deal scores (A,B,C,D,F). As they are not a priori given in the dataset, the problem was open to solutions from unsupervised learning as well as supervised learning (given the appropriate target variable is found).

An obvious approach would be to perform customer-item collaborative filtering, but there are problems with that idea. Firstly, for many items, there is only one customer, so our similarity matrix would be too sparse to make predictions. An alternative approach would be to make a collaborative filtering style metric, but with "courser" features, e.g. to find similar manufacturing location codes, where similarity is defined through customers' willingness to pay premium.

Another approach would be to cluster deals using an unsupervised learning algorithm, such as k-modes or DBSCAN. But the problem with this approach is that there is no objective way to measure distances between different categorical variables as well as the difference between categorical and numerical variables.

Our solution is based on the observation that the *GM%* distribution changes as we split the dataset using various parameters. Finding such features and splits will be the main objective of our model. Random-forest feature select might seem like good idea, but it will yield model specific results, which are not interpretable. Also, such a solution would rely on how the dataset is labeled. Since we have to figure out the correct way to label the dataset, such a robust solution would be redundant. Moreover, continuous features, such as order qty, are obviously huge factors when it comes to deals, but their impact is not obvious right away due to numerous confounding factors. Once the dataset is split on the basis of categorical features, clusters of continuous ones will emerge.

We attempted to preform evolutionary algorithm that would split the dataset in such way that it maximizes difference between final *GM%* distributions (using Kolmogorov-Smirnov test). Unfortunately, we failed at that attempt due to heavy computational demands. So we decided to go with combined solution of CHAID trees for splitting on basis of categorical features and k-means for clustering numerical features.

4.2 Modeling technique

Features were divided into the numerical and categorical class. For categorical variables, CHAID will be preformed with *GM%* as target variable. Afterwards, a k-means clustering

will be performed on numerical features, for refined clustering. When the clusters are formed, a grading is calculated based on *GM%* quantiles. Categorical features considered include:

- *Manufacturing Location Code*
- *Product group*
- *Customer Region*
- *Customer industry*
- *Top Customer Group*
- *Make vs Buy*
- *Intercompany*
- *Order month* (this feature is constructed with the idea that it would capture seasonality of certain products.)

Numerical features include:

- log *order qty*
- log *cost of part*

Both of these are constructed with the idea that they would capture the magnitude of a deal, in contrast to the non log-ed features. Additionally, both are normally distributed.

The main idea behind this approach is that CHAID will find a meaningful way to cluster the categorical features without the need to rely on some black box algorithm with spurious metric. As we have seen in data exploration, log cost of part naturally clusters with log ordered qty. This fact would be utilized via a k-means algorithm. And finally, refined clusters will be formed. Using said clusters, *GM%* cut-offs would be deduced using the pandas qcut function. There will be uniform distribution of deals across all grades, that will grant our model robustness.

4.2.1 Assumptions

The main assumption is that after the data is split by CHAID using categorical features, it will be straightforward to find numerical feature clusters, and that those clusters will be good enough for deal grading.

4.2.2 Method

After data was cleaned as described in previous chapters, python library CHAID was used to create a CHAID tree for categorical features.

CHAID (Chi-square Automatic Interaction Detector) builds a tree for predicting the dependent variable, in our case the *GM%*. It splits the data into groups of approximately equal number of entries in the best possible way, using the chi-squared test. It keeps splitting until the best outcome is achieved and no further splitting can be done. The resulting categories determine groups of possible deals (or entries of the same type as in the dataset) that best predict the dependent variable, the *GM%*. As such, the CHAID tree divides our space of possible deals (in the categorical features) into groups of similar ones. To create grade ranges for a new entry, we use the group that the entry belongs to, that is, we use the *GM%* distribution of that group.

K-means is an unsupervised clustering algorithm. Given n points in d-dimensional space it finds a best way to split the n points into k clusters. Formally it computes:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \frac{1}{2|S_i|} \sum_{\mathbf{x},\mathbf{y}\in S_i} \|\mathbf{x}-\mathbf{y}\|^2$$

(3)

The algorithm is straightforward, it consists of an assignment step and an update step. During the assignment step, each point is assigned to a cluster whose center has the minimal Euclidian distance from the point. In the update step, centers are adjusted so that the total distance is minimized. Iterations are preformed, until it converges i.e. assignments no longer change.

There are various ways to calculate how well the clustering was done. We will be using the *kneelocator.elbow* function from *kneed* library on scikit inertia score.

Inertia is nothing but the sum of square distances inside clusters. It suffers from some drawbacks, mainly that it assumes clusters are convex and isotropic. But, since dimensionality of our clustering is low, we can inspect by eye whether clustering was good or not. Example of k-means algorithm applied on log cost of part - log ordered qty plot is shown in 18

Kneelocator.elbow finds the point in the convex set of maximal derivative (so called elbow point). In our case that's the inertia score plot as a function of the number of clusters. Lower inertia $\implies$ better clustering, up to a point after which each increase in the number of clusters results in lesser improvements of inertia score. This is nothing else than a point of maximal derivative, for a convex set. Knee locator in action is shown

in 19.

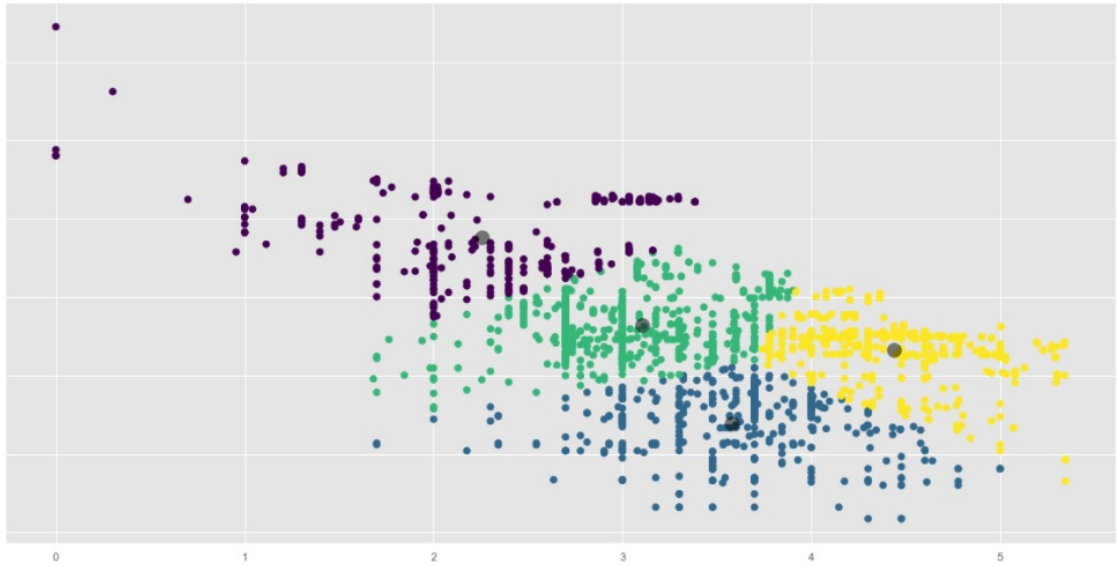

Figure 18: 2-d scatter plot showing log cost of part - log ordered qty dependence, clustered via k-means

To wrap it all up, simple classification tree (DecisionTreeClassifier from scikit-learn) is used to classify new deals into preexisting buckets. The decision tree has the classic tree structure consisting of inner nodes (those with children) and external nodes (those without children, they re also commonly referred to as leaves). Each inner node contains rules for splitting (based on some feature) into children nodes. Leaves represent final buckets. Decision trees are easily interpretable, but tend to overfit easily. Thankfully, since our buckets are defined by splitting rules, decision trees are just a means to an end of finding the right bucket in a way that is consistent and fully interpretable.

5 Evaluation

Our project will be evaluated by looking at real gross margins from actual scored deals and see how they compare to our predictions.

6 Deployment

6.1 Deployment

Our solution is deployed as a REST-API. In a nutshell, the end user is able to input a single entry, consisting of deal features. Those features are passed to our model, which then places the deal into its corresponding bucket. End user is then given a single result, the bucket with exact *GM%* cutoffs for each grade (A-F). This process is depicted in 20.

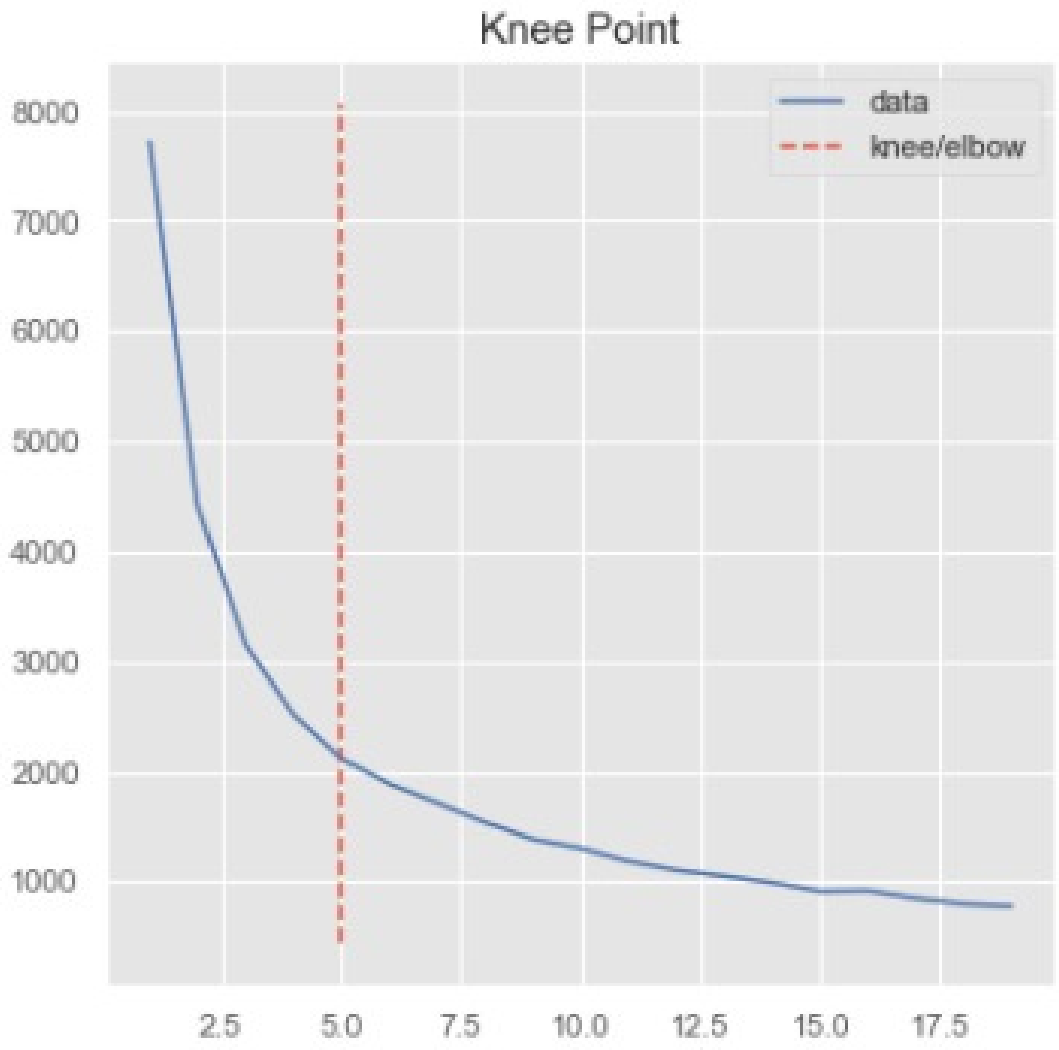

Figure 19: Graph representing inertia score as a function of the number of clusters

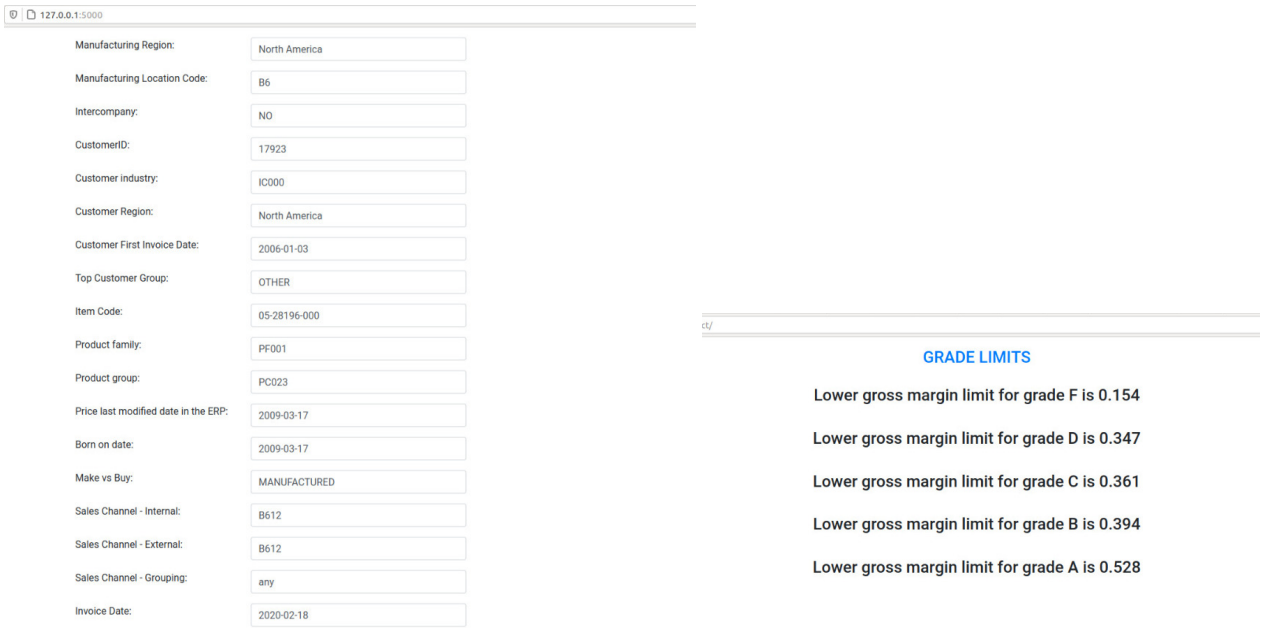

Figure 20: User interface, on left we have deal features, and on right we can se live time calculated margines of each grade.

6.2 Final report

After thorough data exploration and data cleaning, we were clean most of impurities in data. Using new (clean) data we developed deal grading system using a combination of CHAID trees and k-means clustering. Each new deal is easily evaluated using our interface.

6.3 Monitoring

It should be noted that our model should be monitored closely and frequently evaluated to see whether it's making sensible predictions or not. When a substantial amount of new data is collected it can be retrained and/or readjusted.

6.4 Project review

This project was very educative, there were many data cleaning challenges, and the desired way of solving the problem was very ambiguous, so we had to explore many different paths and possibilities, during which we learned a lot about various data cleaning procedures as well as machine learning algorithms. On top of that we had an opportunity to solve a real life business problem. An improved version of such a solution could be used to generate revenue for companies.