SNOOPY QUARTET

JOSIP DOMAZET

MARINO VOĆANEC

DAVID DUKIĆ

IVAN LANDEKA

DYNAMIC DEAL SCORING

LUMEN DATA SCIENCE 2021

Contents

This document was submitted to Lumen Data Science 2021, Croatia's biggest student data science competition organised by eSTUDENT.

Zagreb, 18th of May, 2021

Business understanding

Dynamic deal scoring is an important aspect of modern-day pricing strategies. With the development and popularization of the internet a new possibility emerged for salespeople to offer their products online and for customers to get access to them within a few seconds. To this day, the system works so well that customers can go online and buy almost anything in just a few clicks. This opened a possibility for salespeople to offer different products to different customers at diverse prices depending primarily on the demand for the product. At first, when there were a few trusted customers, salespeople would offer them prices for products manually by looking at a customer's purchase history and previous successful deals. The goal was always to maximize gross margin. So why not always offer the highest possible price for every product? Although this indeed maximizes the gross margin, it does not guarantee that customer is going to accept the offered price. This means that we want to offer the highest possible gross margin that the customer is willing to accept. Gross margin for a particular deal can change depending on a customer's financial status. As the network of customers and available products became larger, it became more difficult for salespeople to make deals manually *and* know how good is every deal they are providing to the customer since the size of the business grows rapidly.

The problem for salespeople is that they do not have objective comparison points for deals they are making, i.e. they do not have much information about how do their colleagues price similar deals[1]. This is how dynamic deal scoring was born. The idea is to have a model which will grade deals made by salespeople. Grades can be represented as A, B, C, D or F, with A being the best grade. If the deal ends up having a grade that is good enough, it can be automatically accepted. If the grade is *not* good enough, then manual input from companies can help in order to modify the deal with the utmost goal to maximize the revenue. To sum up, dynamic deal scoring can help in scaling up the business with automatic approval of good enough deals and by giving salespeople a reference point for their deals. The final goal is to win as many deals as possible.

[1] Walter Baker, Michael Kiermaier, Paul Roche, and Veronika Vyushina. Advanced analytics in software pricing: Enabling sales to price with confidence. https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/advanced-analytics-in-software-pricing-enabling-sales-to-price-with-confidence, June 2018

Data understanding

In order to understand the underlying data, we first need to understand the structure of the data set and the meaning of each column. After we know what we are dealing with, we can perform extensive exploratory data analysis in order to find valuable insights. This chapter will present

- Data set structure
- Column summaries
- Exploratory data analysis
- Data quality discussion.

Data set structure

The data set has almost 1.3 million rows[2] and 33 columns. Each row represents one invoice line. The available column names along with corresponding data types are shown in Table 1.

[2] To be exact, it has 1294962 rows

Almost all of the columns are self-explanatory. Several columns might not be obvious at first:

- `top_customer_group` has value `STAR` for special customers and `OTHER` for the rest of the customers
- `product_family` segments products in three product families
- `product_group` is more granular than `product_family` and it segments products in multiple product groups.

Column name	Data type
manufacturing_region	categorical
manufacturing_location_code	categorical
customer_id	categorical
customer_industry	categorical
customer_region	categorical
customer_first_invoice_date	date
top_customer_group	categorical
item_code	categorical
product_family	categorical
product_group	categorical
price_last_modified_date_in_the_erp	date
born_on_date	date
make_vs_buy	categorical
sales_channel_internal	categorical
sales_channel_external	categorical
sales_channel_grouping	categorical
invoice_date	date
invoice_num	categorical
invoice_line_num	categorical
order_date	date
order_num	categorical
order_line_num	categorical
invoiced_qty_shipped	decimal
invoiced_price_tx	decimal
invoiced_price	decimal
cost_of_part	decimal
material_cost_of_part	decimal
labor_cost_of_part	decimal
overhead_cost_of_part	decimal
gm	decimal
num_of_unique_products_on_a_quote	integer

Table 1: All columns in the data set with corresponding data types.

In order to understand the data set structure, we need to understand how it was created. After customer places the order, company can complete the order by one or multiple deliveries. For each delivery, the company issues an invoice.

Customer ID	Date	Order line num.	Item code	Ordered Qty.	...
25172	2019-04-10	1301607	1001754	585800	...
25172	2019-04-10	1301608	1002192	90000	...

Table 2: `order_num` 717525

Table 2 shows the data about an order example from the data set. We only included columns relevant for discussion about the data set structure. We can observe the following:

- an order (`order_num`) consists of one or multiple order lines (2 in this example)
- an order line contains information about which item was ordered and how many units of it were ordered
  - `item_code` denotes which item was ordered
  - `ordered_qty` denotes how many units of the item were ordered
  - other item-related columns like `product_family`, `product_group`, `manufacturing_region` were obtained by joining customer order table with item table using `item_code` column as a key
- an order contains customer ID (`customer_id`) and date (`order_date`)
  - other customer-related columns like `top_customer_group`, `customer_industry`, `customer_first_invoice_date` were obtained by joining customer order table with customer table using `customer_id` column as a key

We need to emphasise the data set granularity is at invoice line level because that means we can have multiple data set rows, i.e. invoices lines for the same order line. That can happen because, as we already said, one order line can be delivered through multiple invoices.

There are a few things to note about invoice lines:

- each invoice line has corresponding order line
- columns related to shipped quantity, unit price and unit cost are defined on invoice line level, not order line level

Invoices should inform us how the order was delivered to the customers. Each invoice contains shipped items from the order, how many units were shipped, at what unit price, and at what unit cost. This means that if we track how the order was processed by invoices and we

add up shipped quantity for each invoice line that refers to the same order line, we must get exactly the ordered quantity.
Let's show an example from the data set which will be useful to understand the relationship between invoice line and corresponding order line.

Two invoices for order in Table 2 are shown below.

Invoice date	Order line num.	Invoice line num.	Item code	Shipped qty.	Ordered qty.	Unit Price	Unit Cost	...
2019-04-26	1301607	6899873	1001754	4500	585800	0.394	0.1	...

Invoice date	Order line num.	Invoice line num.	Item code	Shipped qty.	Ordered qty.	Unit Price	Unit Cost	...
2019-05-05	1301607	6900027	1001754	54000	585800	0.394	0.250	...

Invoices shown above have only one invoice line. Furthermore, their invoice lines refer to the same *order line*. Since they refer to the same order line, that also means that they refer to the same item. If we sum up shipped quantity, we will get exactly ordered quantity.
The final data set was obtained by joining invoice-related tables with order-related tables using `order_line_num`, since that is the column that connects invoice line to order line.

Data set is at invoice level which means that we will need to be careful when counting values across data set since data set has repeating values of order-level data, for example `ordered_qty`, `order_date`, etc. Counting of values should not be performed by counting rows. For example, if someone wants to count the number of orders and does that by counting all rows, the result will be wrong since one order will have as many rows in the data set as there are invoice lines related to that specific order.

Note that columns `item_code`, `manufacturing_location_code`, `customer_industry`, `product_group`,`product_family` do not reveal us what their values mean since they are encoded. This will make our analysis a bit more difficult interpretation-wise.

Column summaries

The first thing to do in order to familiarize ourselves with the data set is to look at summaries of every column. Since the data set has a lot of categorical variables, the summaries will mostly be bar plots, while histograms and density plots will be used for decimal variables. Date columns will also use bar plots. This section will not go into too much detail with analysis since that is what exploratory data analysis part is for. The idea is, as already stated, to familiarize ourselves with the data set and also to see the typical values for columns.

Invoice and order columns

First type of columns we will analyse are invoice and order columns. Since we are working with data set related to orders and invoices, it is a good idea to count total number of orders and invoices.

Column Name	Number of Unique Values
`order_num`	772640
`invoice_num`	992110

Table 3: Total number of orders and invoices.

We can see from Figure 3 that the number of invoices is greater than the number of orders. This is due to the fact that there are orders which were processed using multiple invoices. Invoice line has 1294933 unique values which is 29 less than the total number of rows in the data set. This is actually a data quality issue because there are 29 duplicate invoice line rows in the data set.

Customer columns

There are 12214 customers in total. They are from three distinct regions:

- North America
- Europe
- Asia.

If we group customers by `customer_region`, we can see that most of customers (∼60%) are from North America, as shown in Figure 1.

We can now incorporate `top_customer_group` column to see whether the distribution of `STAR` customers by region matches the distribution of `OTHER` customers. However, it needs to be emphasised that there are only 87 STAR customers. This means that STAR customers make up less than 1% of total customers [3]. Figure 2 shows us that STAR

[3] 0.71% to be exact

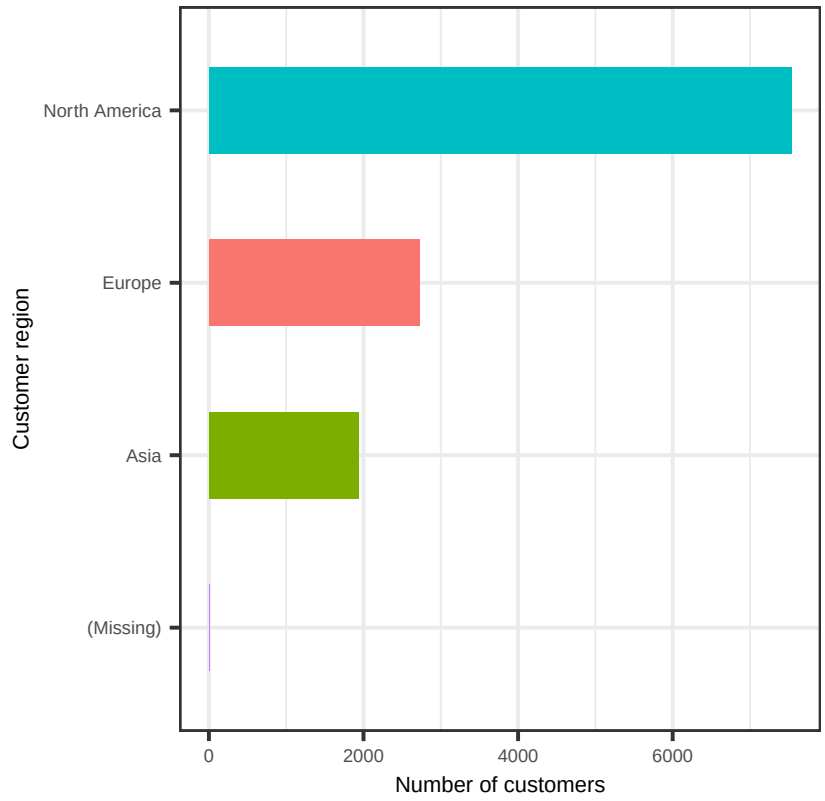
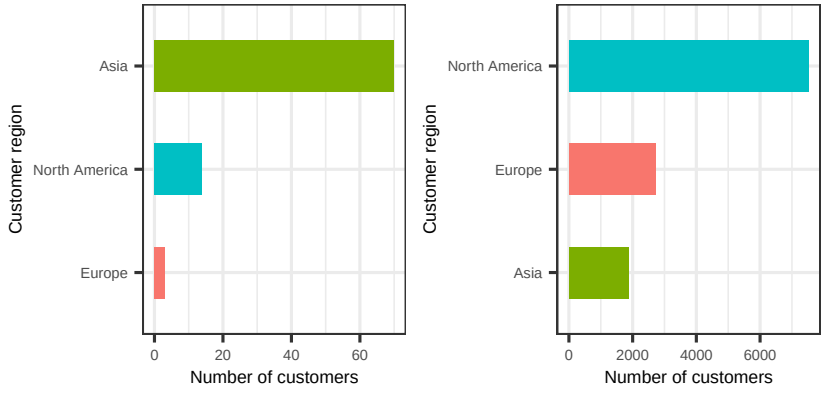


Figure 1: Number of customers by `customer_region`. *Only 13 customers do not have customer region defined.*

customers are mostly found in Asia, in contrast to OTHER customers which are mostly found in North America.



(a) STAR customers

(b) OTHER customers

Figure 2: Number of customers by `customer_region` *and* `top_customer_group`.

Another customer-related column is `customer_industry` which segments customers into 15 different industries. Figure 3 shows us the

number of customers by customer industry. Notice that industry name is encoded so we do not know what each industry code actually stands for. Almost 70% of customers are in IC008 or IC000 customer industries. We can only assume that IC008 and IC000 represent widespread industries.

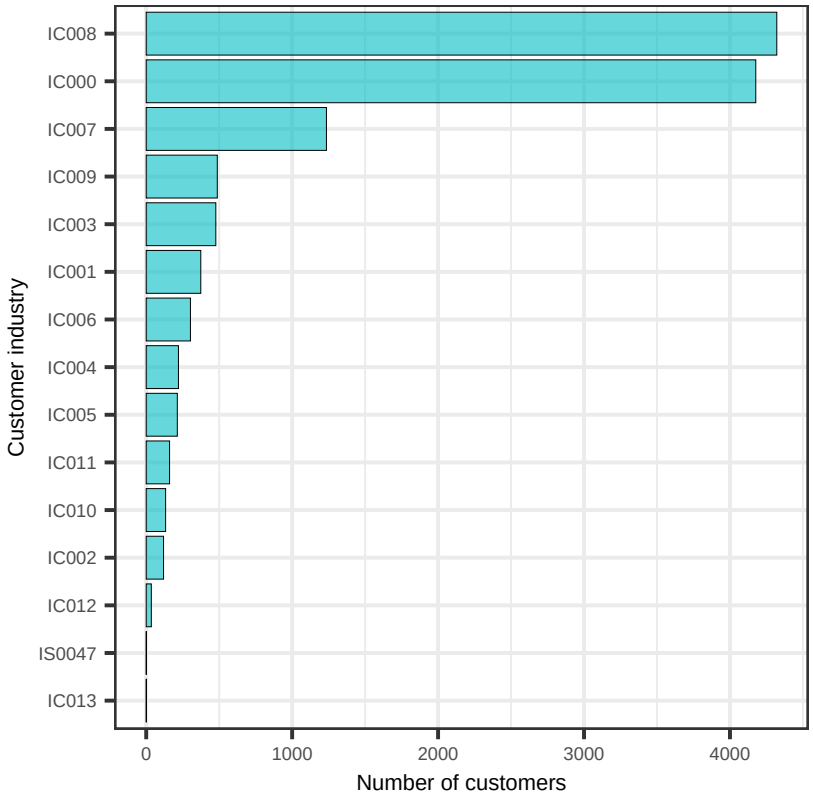


Figure 3: Number of customers by customer industry. *Only 13 customers do not have customer industry defined.*

Next column to look at is `customer_first_invoice_date`. Figure 4 shows the distribution of customers by the year of their first invoice. Around 50% of customers are from 2015 to 2020, with the biggest peak in 2018. The oldest customers are from 1993. This can help us if we want to divide customers into two groups:

- old customers
- new customers.

For example, we could say that old customers are customer whose first invoice date is in red color (before 2015), while new customers are in blue color.

Furthermore, there is a column `intercompany` which denotes whether the invoice was between a parent company and its subsidiaries or other

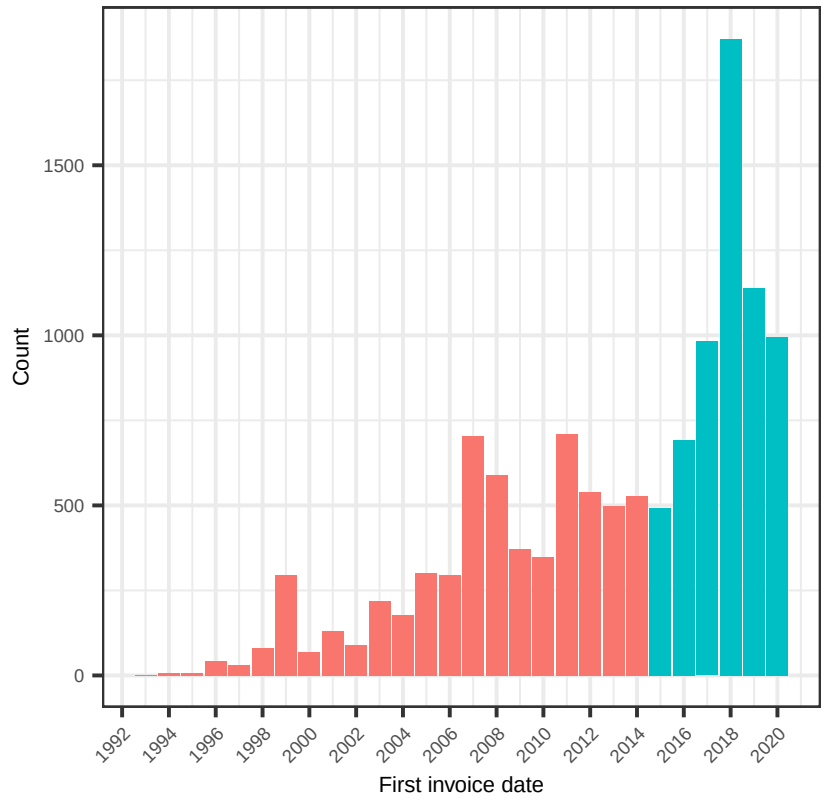


Figure 4: Number of customers by the year of first invoice. First invoice years colored red have about the same percentage of users as the ones colored blue.

related entities. When that column is equal to `YES`, corresponding `customer_id` for those rows represents entity related to parent company. Selling a product to other customers is not the same as selling it to some entity related to the parent company. The number and proportion of rows by `intercompany` is shown in Table 4.

Intercompany	Number of rows	Pct. of rows
YES	190969	14.7%
NO	1103993	85.3%

Table 4: Number of data set rows by `intercompany` column.

Let's see how customer regions are distributed for `intercompany` invoices. The results are in Table 5. Most of the `intercompany` rows belong to Asian customers.

Customer region	Number of rows	Pct. of rows
Asia	180502	94.5%
North America	7142	3.74%
Europe	3318	1.74%

Table 5: Number of `intercompany` data set rows by customer region.

Item and manufacturing columns

The analysis continues with columns corresponding to item and manufacturing columns. Our company of interest sells items which are either manufactured or bought from another company. Each item has its own code, the day is was first introduced, product group, and product family. Since each item is stored as item code, we do not know what each item code really represents. The difference between product group and product family is in granularity. Product family is a more general grouping of items in 3 product families (PF000, PF001, and PF002). Product groups define more granular grouping of items in 31 product groups. There are around 63 thousand unique items in data set [4]. If we group items by product family, we get the distribution shown in Table 6. PF001 is by far the biggest product family, while PF000 is the smallest one.

[4] 63020, to be exact

Product Family	Number of unique items
PF000	2602
PF001	39319
PF002	21506

Table 6: The number of unique items by `product_family`.

Figure 5 shows us that each of product families PF001 and PF002 have product groups that stand out with the number of unique items in them.

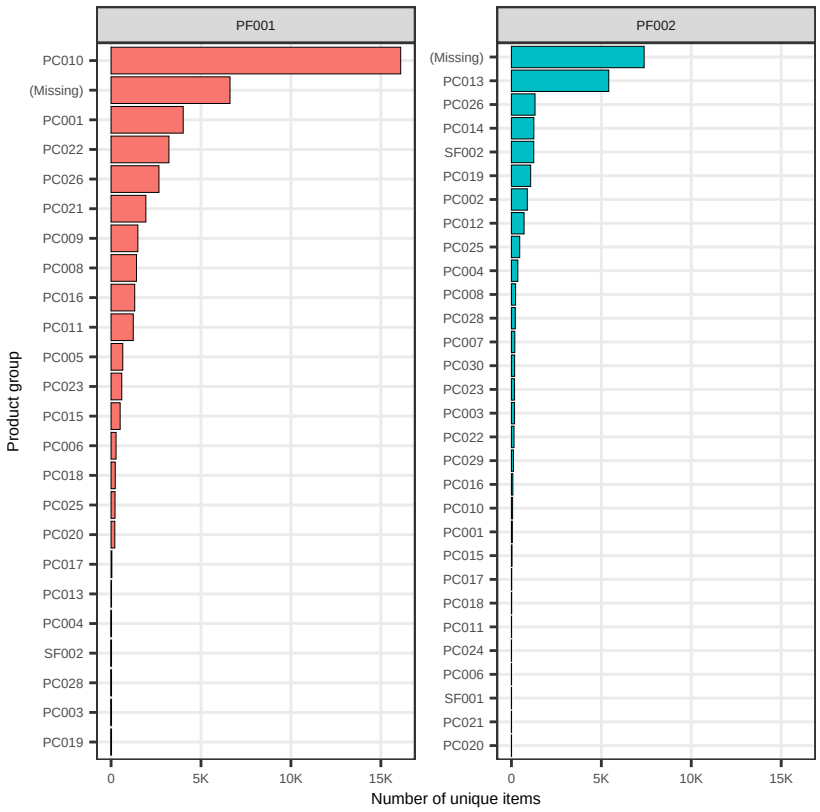


Figure 5: Number of unique items for each product group in product family. *Notice that product family PFooo is missing since it does not have any product group defined.*

However, notice that both of them also have a significant number of items without any product group defined. Moreover, product family PFooo does not have any product group defined.

To see which manufacturing regions and locations are the most common in the data set, we can group the data set by aforementioned columns and count the number of rows for each group. The number of rows by manufacturing region is shown in Table 7.

Manufacturing region	Number of rows	Pct. of rows
North America	734815	56.7%
Asia	464038	35.8%
Europe	55600	4.29%
(Missing)	40509	3.13%

Table 7: The number and percentage of data set rows by `manufacturing_region`. North America and Asia have the most rows by far.

Figure 6 shows us:

- the most frequent location codes for each manufacturing region
- how many different manufacturing locations exist
- there are rows with both missing manufacturing region and location.

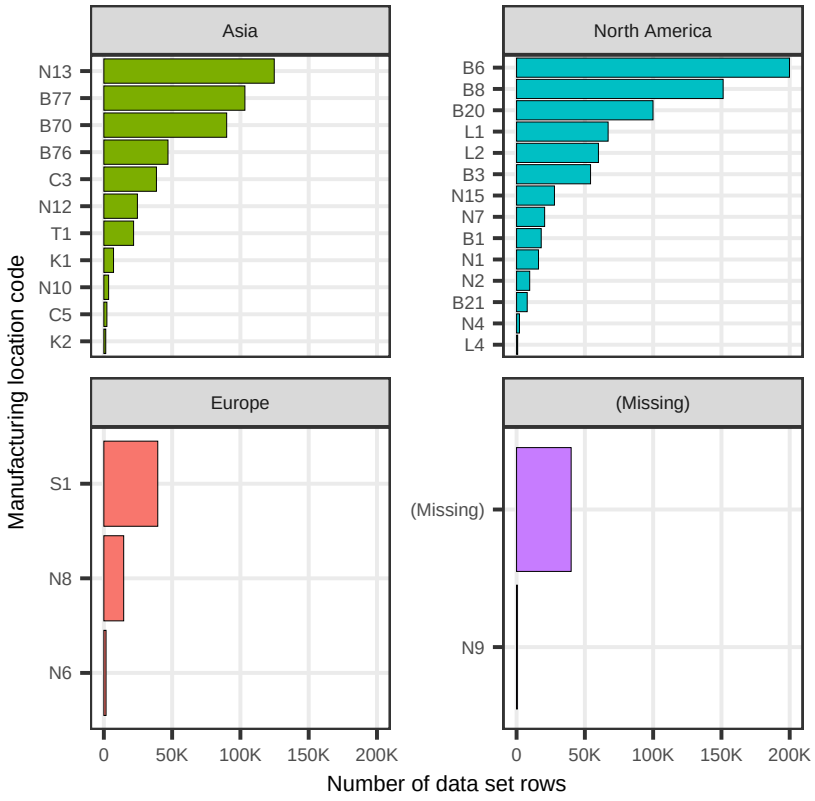


Figure 6: Number of data set rows by manufacturing region and location. *Notice that quite a few order lines have missing manufacturing location. This will be discussed in the rest of this document.*

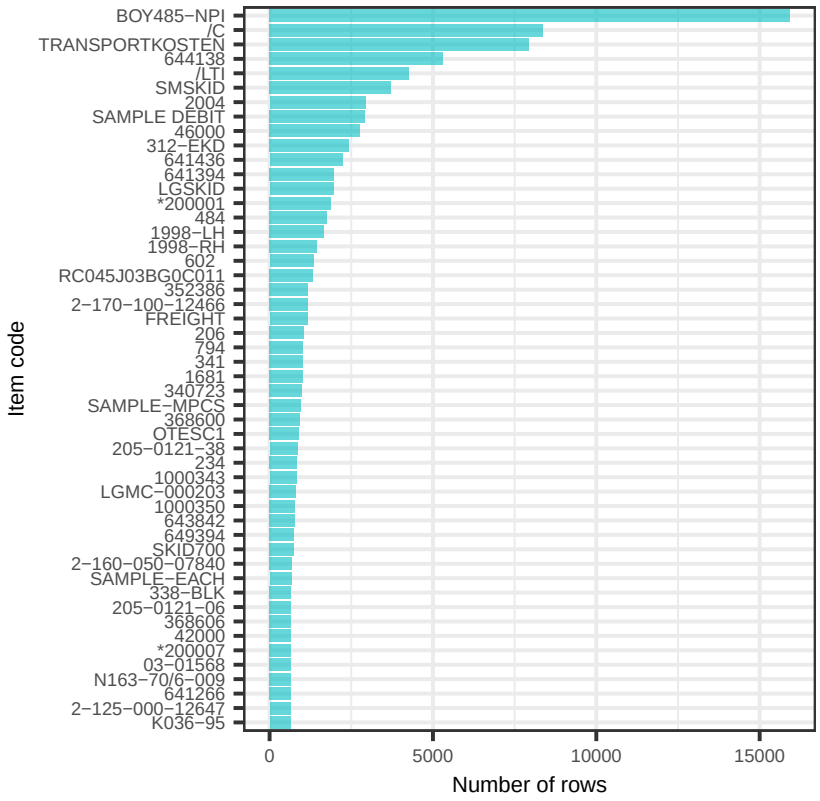


Figure 7: Top 50 item codes by frequency in the data set.

Furthermore, it makes sense to see which items are the most frequent in the data set [5].

Figure 7 shows us the most frequent items. We can see that the top item codes have much more rows than other item codes. Moreover, we observe that third item code is `TRANSPORTKOSTEN` which refers to the transport/delivery that the company was charging through invoices. However, transport is not a typical item in a sense that the company is manufacturing it or selling it. We will need to check whether other frequent item codes, such as `BOY485-NPI` and `/C` stand for transportation. It makes sense that transportation occurs so frequently in the data set because every customer order has to be delivered.
Since we have different customer and manufacturing regions, we can expect that different items will be relevant for each of those groups. In addition to this, we also have `STAR` and `OTHER` groups of users. Furthermore, we need to take `intercompany` into account. The reasons we will get different most popular items when segmenting the rows are:

- the company manufactures different items by region

[5] Note that this does not include the ordered quantity. This is only item frequency at data set level.

- customers from different regions buy different items
- intercompany invoices are different than *normal* invoices because those invoices are between related entities
- `STAR` customers order different things than `OTHER` customers.

If we try to find top 3 items by `customer_region`, we will get results shown in Table 8.

Manufacturing region	Top three items
Asia	BOY485-NPI, 644138, SAMPLE DEBIT
Europe	TRANSPORTKOSTEN, 644138, 641436
North America	/C, /LTI, SMSKID

Table 8: The most popular items by `customer_region`.

However, if we omit rows with `intercompany` equal to `YES`, we get results in Table 9. Notice that top three values for Asia have changed.

Manufacturing region	Top three items
Asia	644138, 641436, 641394
Europe	TRANSPORTKOSTEN, 644138, 641436
North America	/C, /LTI, SMSKID

Table 9: The most popular items by `customer_region` with `intercompany` equal to `YES`.

Since we mentioned that the transportation cannot be manufactured, we also need to mention that there are other items that are not manufactured, but rather bought and resold. That is exactly what the `make_vs_buy` column stands for. Figure 8 shows us all values for `make_vs_buy` column and corresponding number of rows. Notice that there are many similar values. For example, `BUY`, `BUY - IMPORTED`, and `BUY - LOCAL` all refer to `BUY` value, but they are split into multiple categories. However, we also observed values with word `PURCHASED`. It would be instructive to lump all `make_vs_buy` values into two corresponding values.

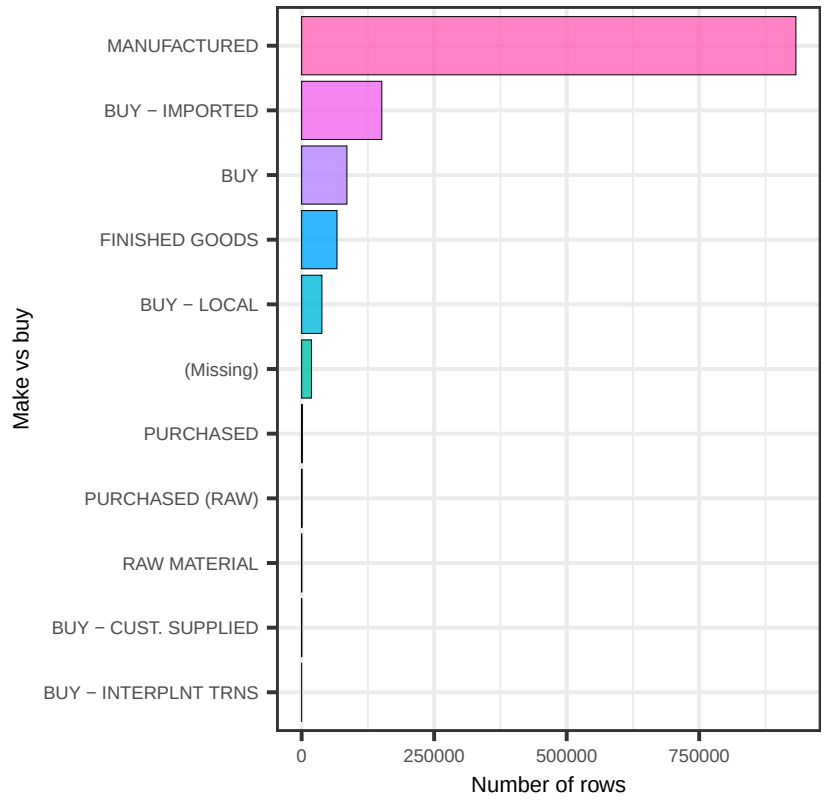


Figure 8: `make_vs_buy` categories.

We can also look at `born_on_date` column to see how old are the items that the company is selling. That column denotes when the item was first introduced or manufactured.

Figure 9 shows histogram of `born_on_date` column. Most of the items were introduced in the last few years. However, there are items which have 1901 as the year of `born_on_date`. This seems like a data quality issue. Additionally, some items do not have that value specified. If we recall that there are items which stand for transportation, then it would make sense that those items do not have that column specified.

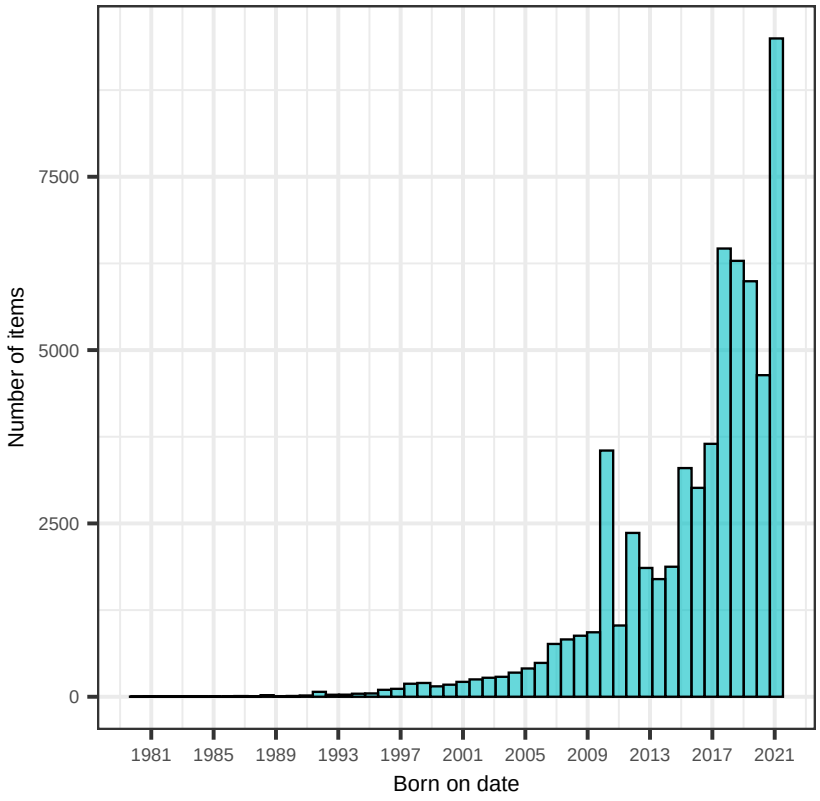


Figure 9: The distribution of number of items depending on corresponding `born_on_date` column value.

Sales channels

There are 661 sales channels in total. Sales channel is defined on invoice level. The data set has `sales_channel_internal` and `sales_channel_external` columns. Even though one would think that they would be different, they are actually the same across the entire data set. Furthermore, there is also a column `sales_channel_grouping` which does not have any values specified so we cannot make any comment about this column.

Price and quantity columns

As already stated, each *invoice* line has corresponding item code, shipped quantity, unit price, and unit cost.

Let's start with unit price and show its distribution.

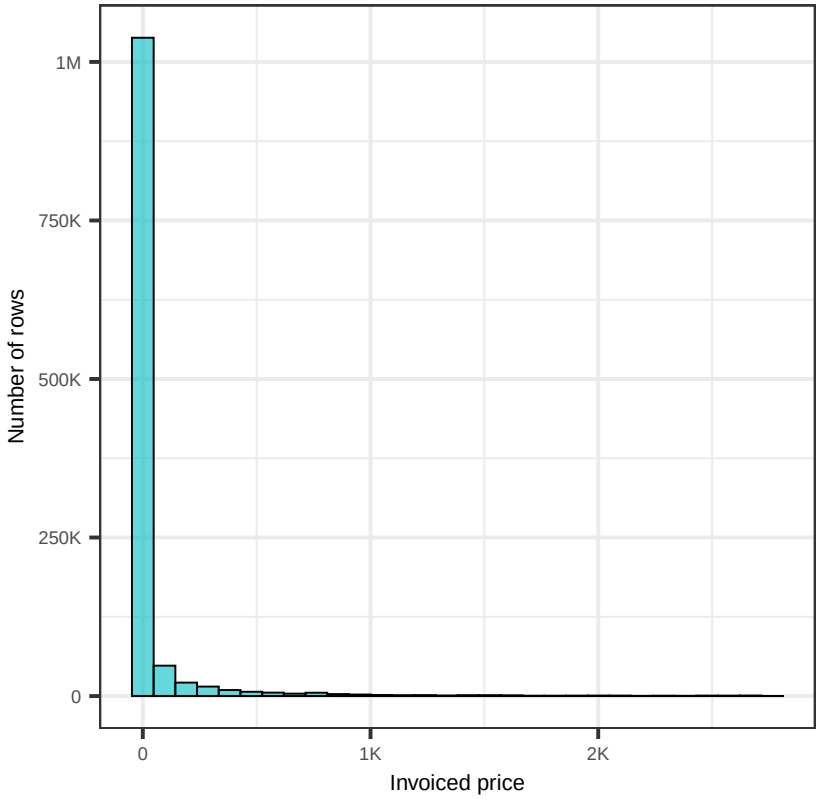


Figure 10: Histogram of `invoiced_price` values look very skewed so linear x-axis does not seem very suitable

Figure 10 shows the histogram of unit `invoiced_price`. We can notice that there are a lot of rows with invoiced price equal to zero. This will be discussed in Section Data quality. Summary statistics for `invoiced_price` before and after filtering of extreme outliers is shown in Table 10. We can notice that the mean changed a lot after the filtering,

but median did not. This is because median is a more robust measure of central tendency than mean. Filtered data has coefficient of variation of 4.19 which indicates high variability. Let's plot the data on log x-axis to mitigate the effect of outliers and big variation in data. The obtained histogram is shown in Figure 11. Now, we can observe that log x-axis suits the data better.

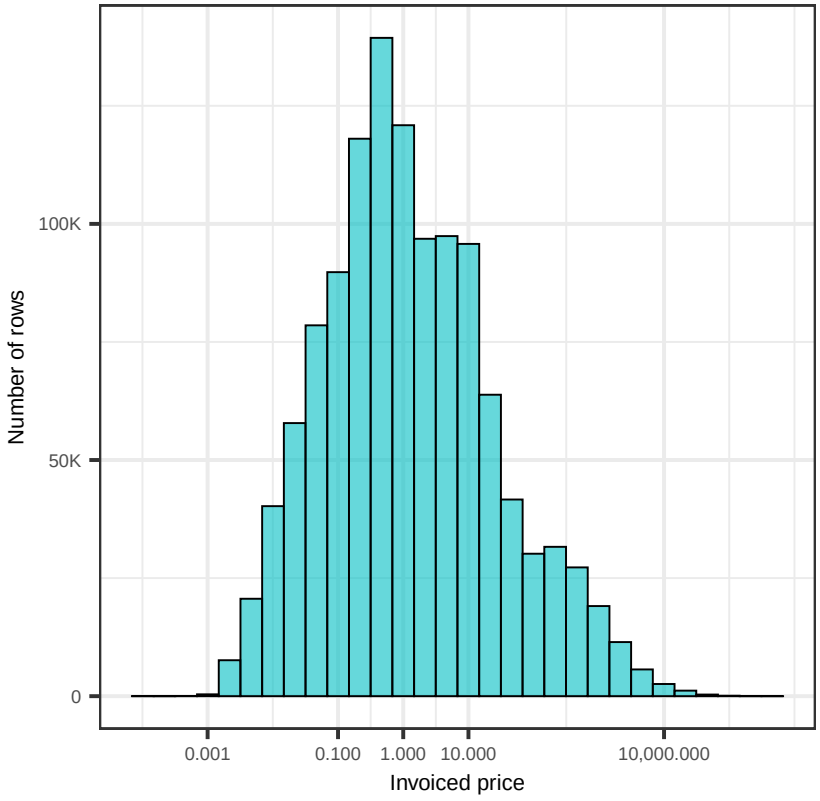


Figure 11: After changing to log scale, the histogram of unit `invoiced_price` row counts looks more informative.

Data	Min	Max	Mean	Stddev	Median
raw	-99999	1089758016	984.1517	960859.6	0.69
filtered	0.0001	449521.5	147.274	1798.451	0.884

Table 10: `invoiced_price` summary statistics for non-filtered data and filtered data.

Apart from unit price, each item has a corresponding total cost which is segmented into three components:

1. labor cost
2. material cost
3. overhead cost (all remaining costs).

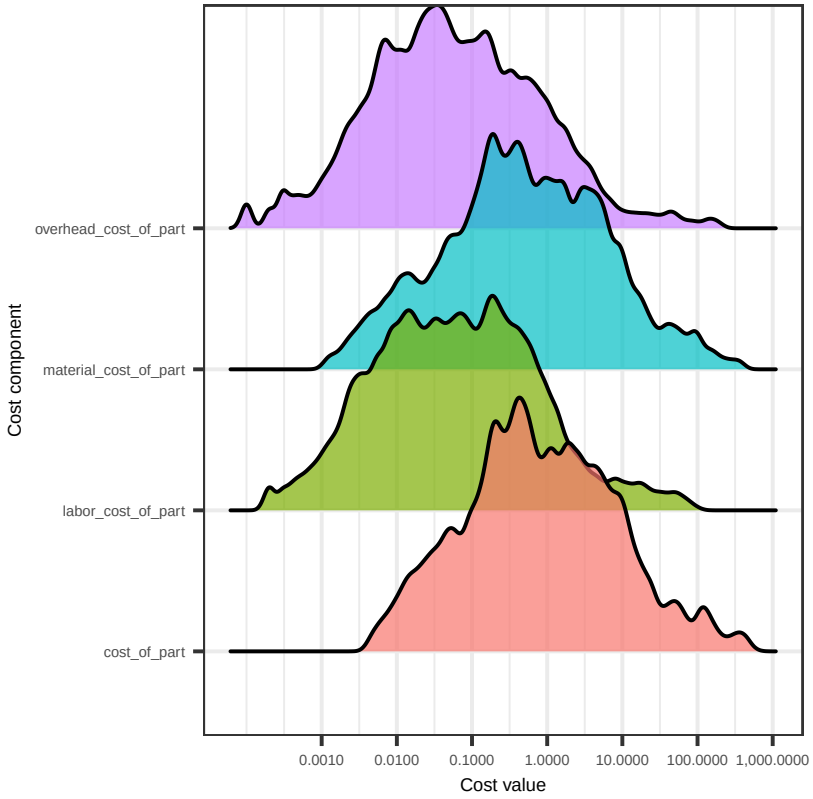


Figure 12: The distributions of cost components. `cost_of_part` stands for total unit cost and should be equal to the sum of other cost components.

Figure 12 shows the distribution of cost values for cost related columns. Notice that we are again using log x-axis. It looks like `labor_cost_of_part` is on average greater than `material_cost_of_part`. This plot was generated by taking into account only rows with costs greater than zero. Around 17% of data set rows has `cost_of_part` equal to zero which is not a small number and will be deeper analyzed in the sections that follow.

Now it is time to look at distribution for one of the most important columns: `gm` (gross margin).

Gross margin in normal conditions takes values between 0 and 1. However, when looking at summary statistics for `gm` across the entire data set, we get something different. Table 11 shows summary statistics for `gm`.

Min	Max	Mean	Stddev	Median
-4528710	6882.41	-9.61	5707.93	0.37

Table 11: Summary statistics for gross margin across the entire data set.

We can immediately see that there are values that are not expected for gross margin. For example, min value is around -4.5 million which definitely does not make a lot of sense. Moreover, around 9% of the data set has missing `gm` value [6]. Since `gm` spans a vast interval, we might want to bin the rows by gross margin into four bins: one for negative values, one for expected/normal gross margin values, one for values above 1, and one for missing values.

[6] This will be inspected in Section Exploratory data analysis

	$\texttt{gm} < 0$	$0 \leq \texttt{gm} \leq 1$	$\texttt{gm} > 1$	gm is NA
# of rows	110971	1066773	96	117122
% of rows	8.569 %	82.379 %	0.007 %	9.044 %

Table 12: Binning rows by gross margin values.

We can see from Table 12 that the majority of rows have normal value for gross margin. Very few rows have gross margin greater than 1. However, almost 18% of rows have either negative gross margin or a missing value. They will be analyzed in further sections. For the time being, we will show the histogram for values between 0 and 1.

The histogram is shown in Figure 13. Notice that gross margin values look normally distributed around 0.35, but there is a big number of observations with gross margin exactly equal to 1. This will also be investigated in the following sections.

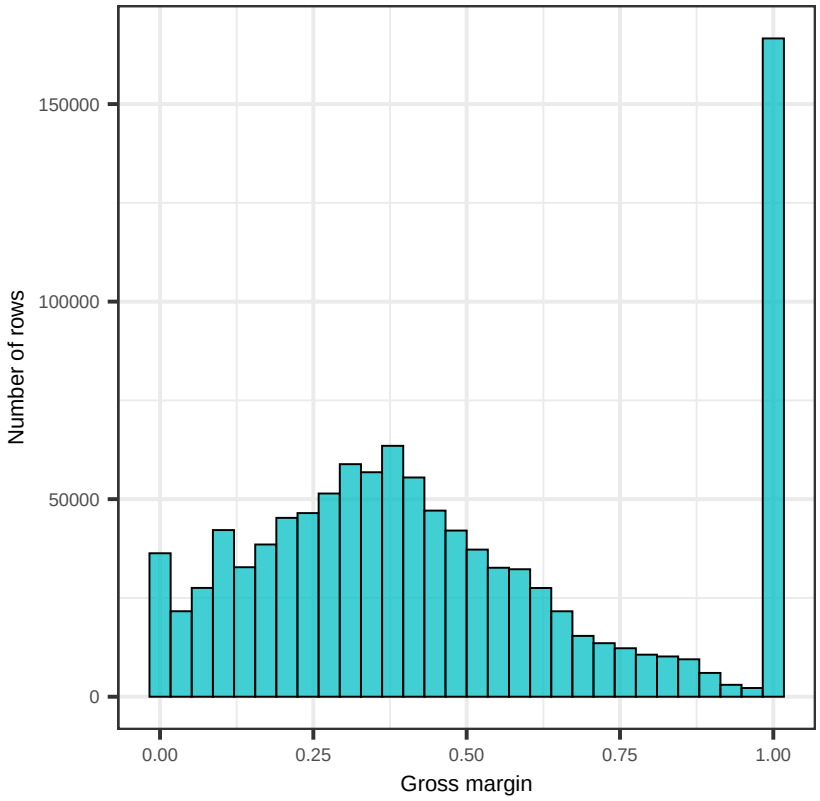


Figure 13: Immense number of rows have gross margin exactly equal to 1.

Now let's see the `ordered_qty` column. This column denotes the ordered quantity for each order line. Note that this column is at order level while `invoiced_qty_shipped` is at invoice level. Since one order line can be processed by multiple invoices, `ordered_qty` can be repeated for each invoice line. Sum of `invoice_qty_shipped` values for the same order line should be equal to `ordered_qty` value.

To get a feel of what is the order of magnitude for ordered quantity, we can take a look at Table 13.

Min	Max	Mean	Stddev	Median
-81000	7800100000	95795.19	78109368	275

Table 13: Summary statistics for `ordered_qty` across the entire data set.

Numbers look extreme since minimum value is -81000, while maximum is 7800100000. Furthermore, there are even negative values which do not make sense in this context[7].

[7] More on this in Section Data quality

Let's now split ordered quantity into a few bins in order to get a feel of its distribution. We have few extreme ordered quantity values.

`ordered_qty`	< 0	0	$\langle 0, 10\rangle$	$[10, 100]$	$\langle 100, 1000]$	≥ 1000
% of rows	0.002 %	1.21 %	11.85 %	22.47 %	25.38 %	38.73 %

Table 14: Binning order lines by `ordered_qty`.

Figure 14 shows the histogram of ordered quantity on a linear scale.

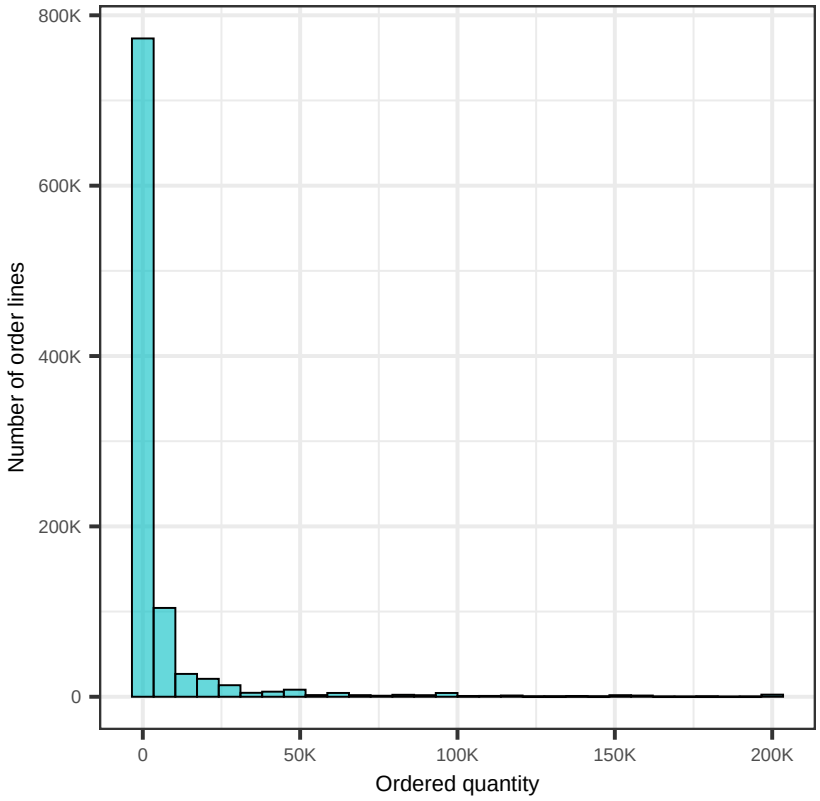


Figure 14: Histogram of ordered quantity.

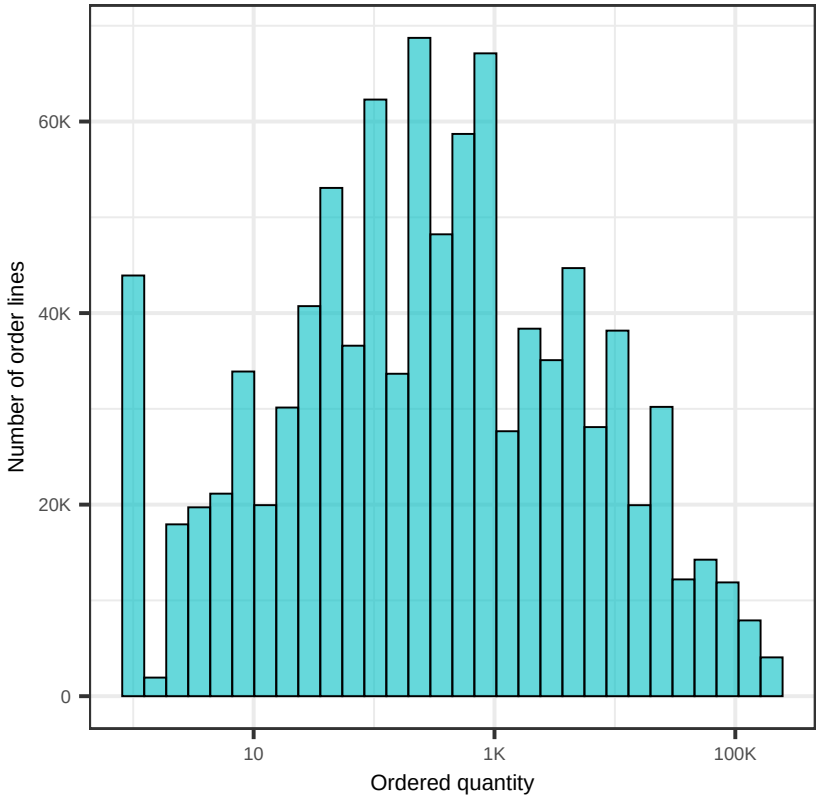


Figure 15: Histogram of ordered quantity after filtering out quantities greater than zero and filtering by percentile. *The leftmost column shows number of order lines for ordered quantity equal to one.*

However, linear scale is not ideal since ordered quantity spans over a very wide range of values, even after the removal of outliers. Henceforth, it is better to show its histogram on a log scale. We also see a lot of values close to zero. In fact, there is a substantial number of order lines with ordered quantity equal to zero.

Modified histogram with filtered data by percentile and by ordered quantity greater than zero and with applied log scale on x-axis is shown in Figure 15.

Exploratory data analysis

This is the most important section in this chapter. Here we will try to identify and visualize the most interesting insights within or between columns. Let's kick off with in-depth analysis of `gm` column.

Gross margin by customer data

To get a feel of gross margin values across the data set, let's show the distribution of gross margin by customer region. We can ask questions like: *Do we make better deals in a specific region?*[8]

[8] Of course, high gross margin alone does not make a deal great, unit price and quantity also have to be included

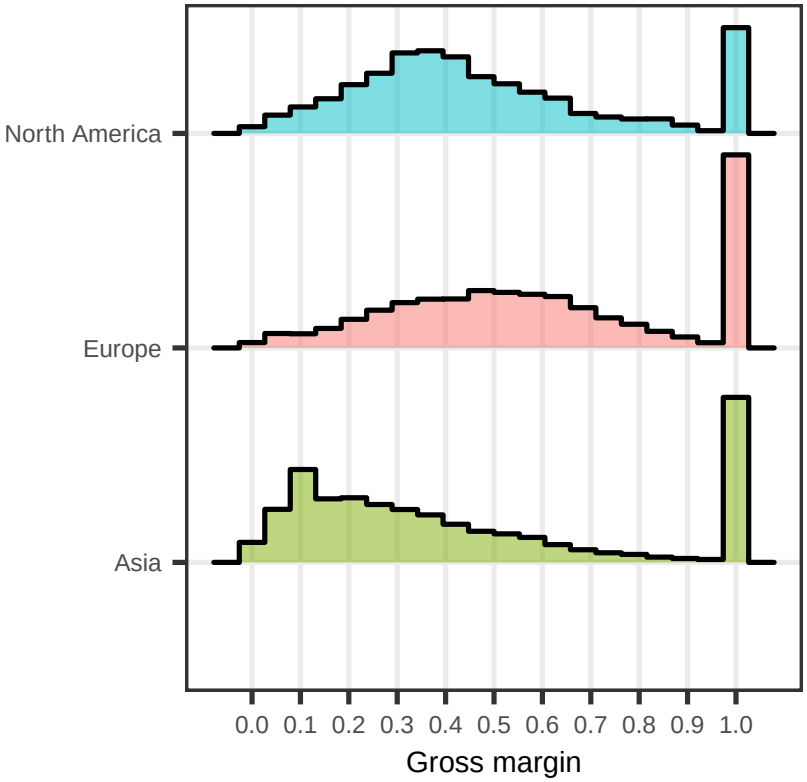


Figure 16: Gross margin by customer region. *Notice that all regions have a lot of values with gross margin exactly equal to one.*

We can see from Figure 16 that gross margin differs by `customer_region`. Median gross margin values are shown in Table 15. Europe has the highest median gross margin.

Customer region	Median gross margin
Europe	0.569
North America	0.420
Asia	0.310

Table 15: Median gross margin by customer region.

Furthermore, we can notice that all of the customer regions have a significant number of invoice lines with gross margin equal to one. This is definitely something that needs further attention and analysis because gross margin can be equal to one if and only if `cost_of_part` column is equal to zero[9]. However, we need to ask ourselves why we have invoiced lines with `cost_of_part` equal to zero? This is crucial since there are quite a few of such invoice lines and they could affect the model we will be building. We need to ask ourselves: *For which items would it make sense to have unit cost equal to zero?* If we count which items had the most invoice lines with `cost_of_part` equal to zero, we get the following items:

[9] $gm = \frac{invoiced\_price - cost\_of\_part}{invoiced\_price}$

1. `/C`
2. `TRANSPORTKOSTEN`
3. `/LTI`.

Item code `TRANSPORTKOSTEN` reveals us that there are invoice lines which stand for transportation costs. For each of those items, all invoice lines have `cost_of_part` equal to zero. Furthermore, all of their invoice lines also have `invoiced_price` equal to zero.

We will assume that these three item codes represent transportation and delivery items. However, we also have items which have `cost_of_part` sometimes positive and sometimes equal to zero.

To keep things simple, we will filter out invoice lines with `cost_of_part` equal to zero because we suspect those cases are special invoice lines which are part of implemented business logic. Figure 17 shows us gross margin density distribution by `customer_region` after removing invoice lines with zero unit cost. Now North America and Europe start to resemble normal distribution, but Asia does not so much because it has a peak at values 0 and 0.1. Europe and North America also have a visible increased gross margin distribution at value of 0.0. Gross margin can be 0.0 if and only if the unit price is exactly equal to the unit cost.

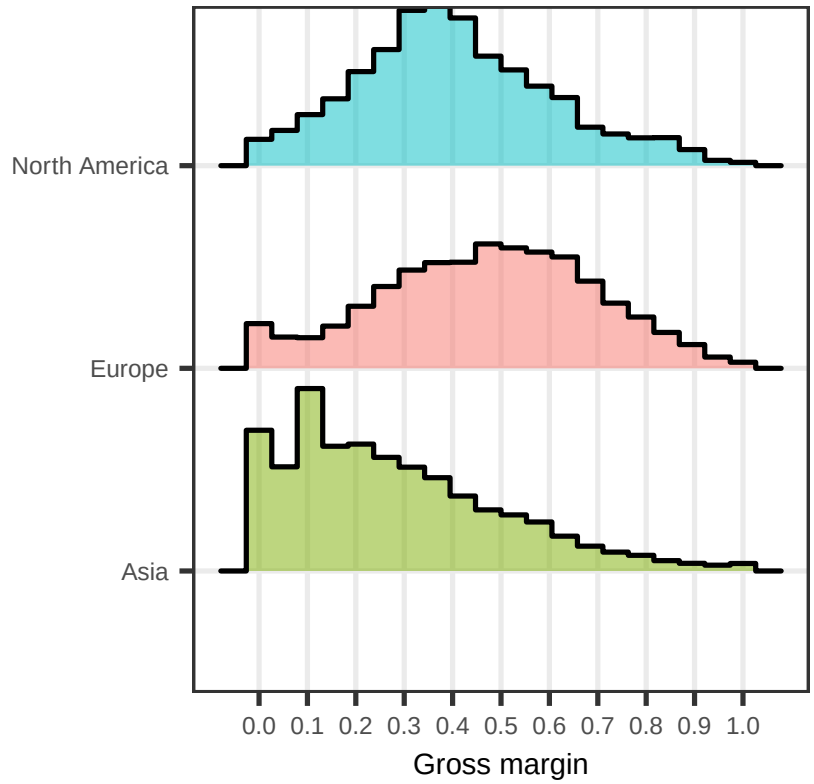


Figure 17: Gross margin by customer region after removing invoice lines with zero unit cost.

Let's try to see gross margin distribution by `customer_region` after the additional removal of rows with gross margin exactly equal to zero. Figure 18 shows resulting distributions.

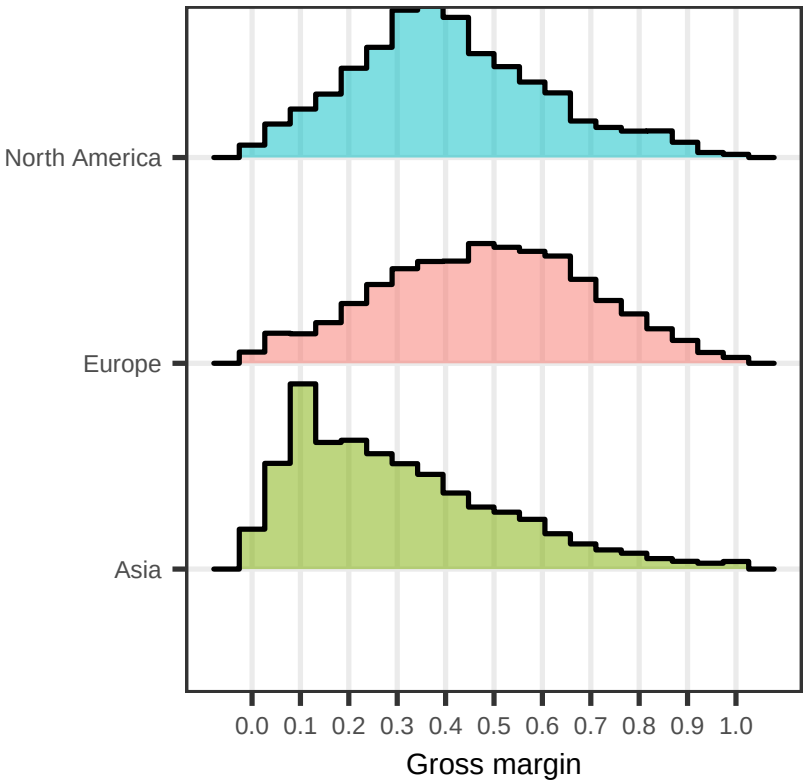


Figure 18: Gross margin by customer region after removing invoice lines with zero unit cost *and* gross margin equal to zero.

As we can see from Figure 18, Europe and North America now look rather normally distributed, while Asia does not.

Let's take a closer look at Europe's gross margin distribution in Figure 19. Mean value is around 0.48, with standard deviation around 0.21.

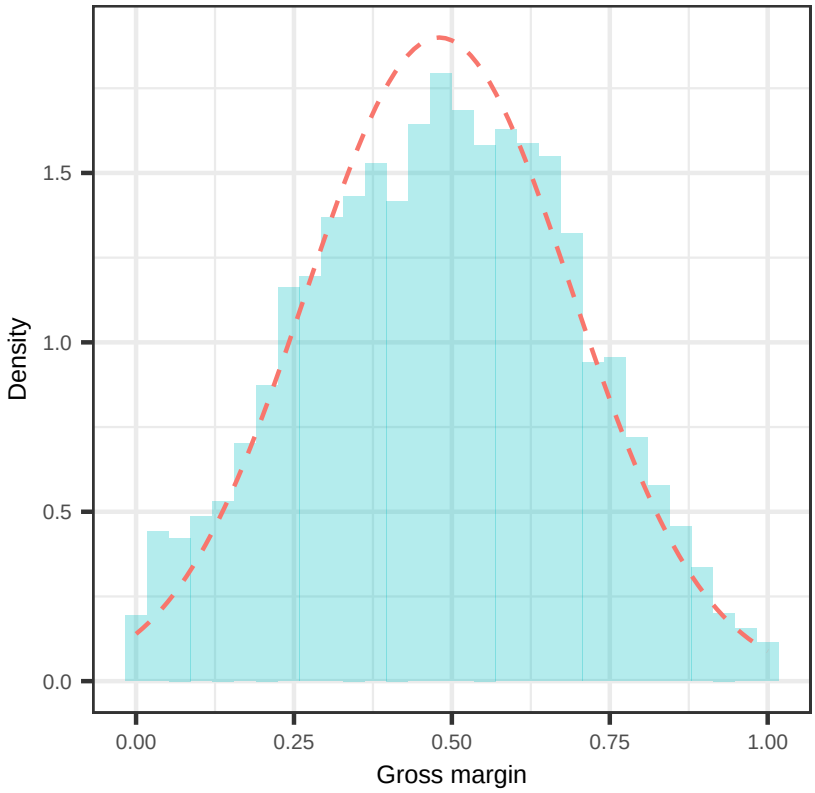


Figure 19: Europe's gross margin distribution after removing invoice lines with zero unit cost compared with normal distribution.

For North America, the distribution is shown in Figure 20. Mean value is around 0.41, with standard deviation around 0.20. Again, we get that the average gross margin for customers in Europe is higher than average gross margin for North American customers.

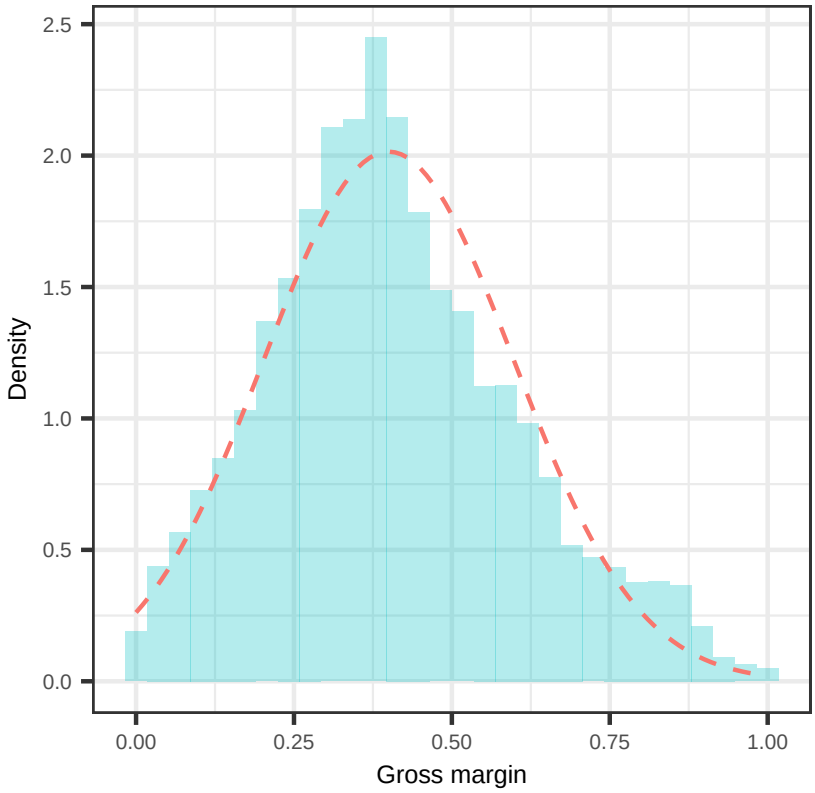


Figure 20: North America's gross margin distribution after removing invoice lines with zero unit cost compared with normal distribution.

However, for Asia the situation looks a bit different. Gross margin distribution looks smaller and it has significant number of invoice lines with value around 0.1.

A closer look into invoice lines with gross margin around 0.1 shows that many of those invoice lines have `intercompany` column equal to `YES` or `top_customer_group` equal to `STAR`. Figure 21 shows the difference in gross margin distribution for intercompany and non-intercompany invoice lines.

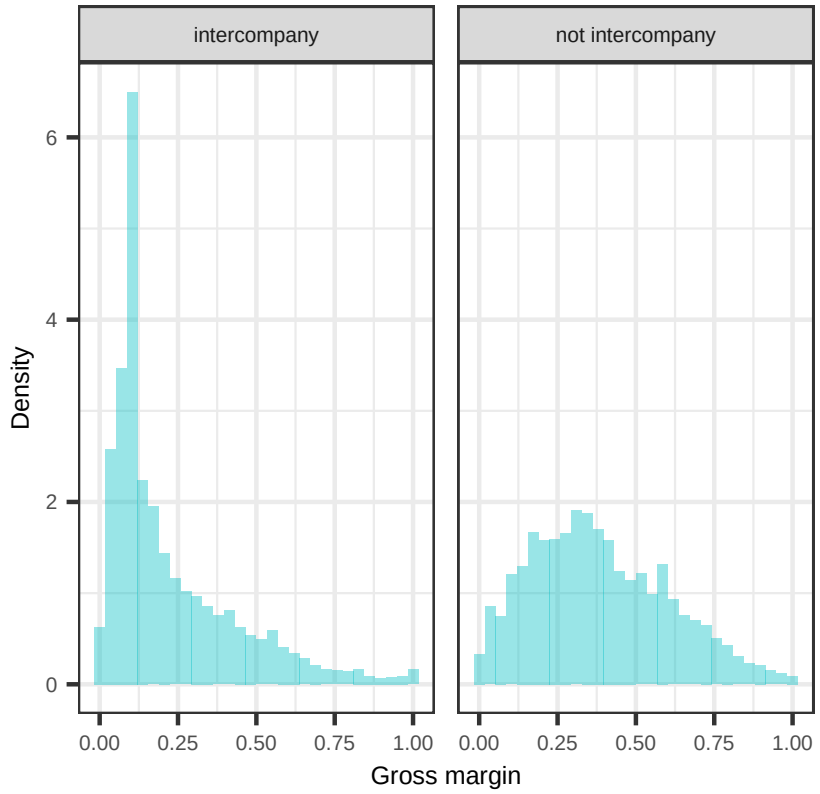


Figure 21: Asia's gross margin distribution for intercompany and non-intercompany invoice lines.

If we filter out invoice lines regarding star customers and intercompany invoices, we get the distribution that looks much more like normal distribution, The distribution is shown in Figure 22. Mean value is around 0.38, with standard deviation around 0.21.

Columns like `intercompany` and `top_customer_group` will be in-depth analyzed in the rest of the document. It is obvious from Figure 21 that intercompany invoice lines were responsible for the spike around gross margin value of 0.1 We can ask ourselves why was Asia problematic with getting the distribution to look normal? This answer is rather simple: 95% of intercompany invoice lines belongs to the Asian customer group and 98% of STAR invoice lines likewise belongs to the

Asian customer group.

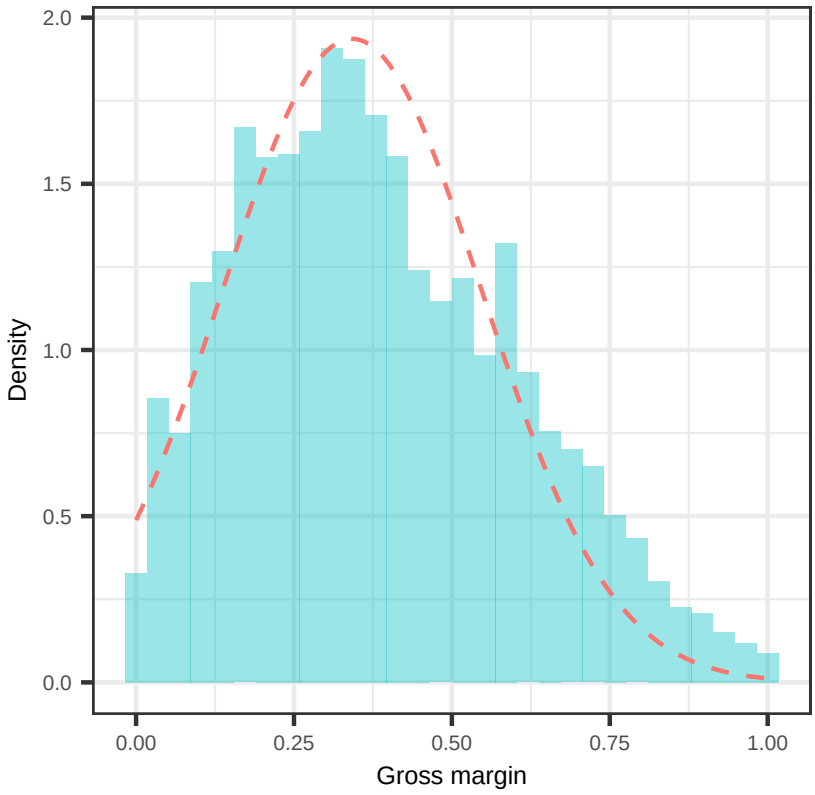


Figure 22: Asia's gross margin distribution after removing invoice lines with zero unit cost and filtering intercompany invoices compared with normal distribution.

To sum up:

- gross margin differs by `customer_region`
- gross margin exactly equal to 0.0 and 1.0 is suspicious
  - gross margin 0.0 means no profit
  - gross margin 1.0 means zero unit cost which is strange if we consider commercial processes of producing and (re)selling items to customers
  - `intercompany` and `top_customer_group` can greatly influence distributions, especially in Asia because there is a vast majority of intercompany invoices and `STAR` customers.

Differences between gross margin of different customer regions could exist due to multiple possible reasons:

- customers in different regions buy different items
- customers in different regions have more money so they are willing to buy items at a non-discounted price

- customers in specific regions buy same items in larger quantities in order to get a quantity discount.

Now that we know how gross margin differs across customer regions, let's continue by taking `customer_industry` into consideration.

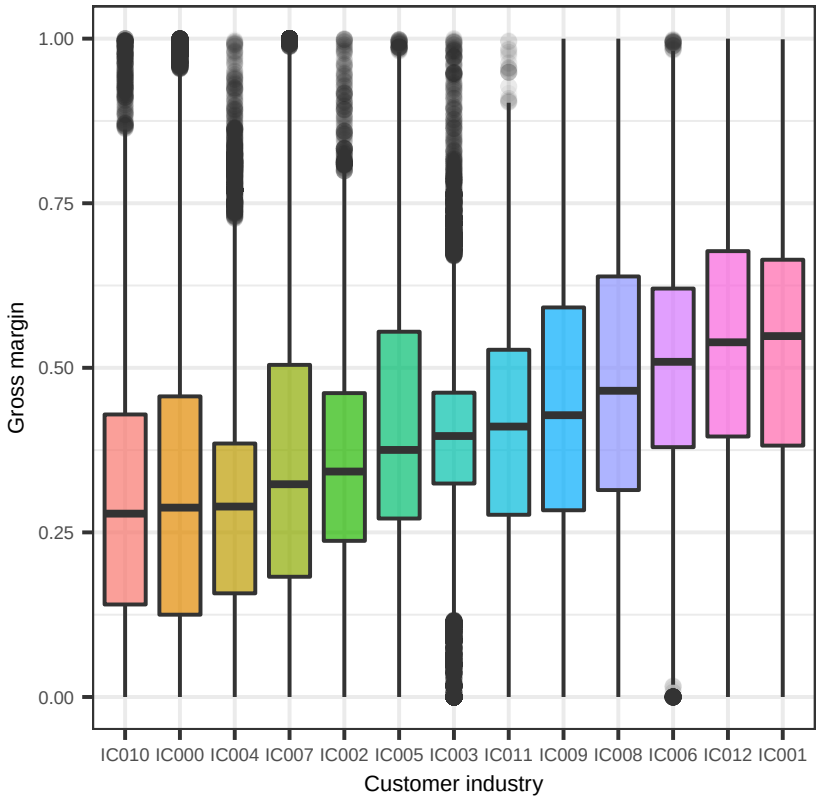


Figure 23: Gross margin by `customer_industry`.

We can see from Figure 23 that the company has different gross margin for different customer industries. However, if we filter out `intercompany` invoice lines, we will end up with something different.

Now we can take a look at Figure 24. Let's compare, for example, customer industry ICooo. After filtering out the intercompany rows, median gross margin increased by 16% percent.

Since we have already talked about how intercompany rows and `STAR` customers can influence results a lot, they will be examined individually.

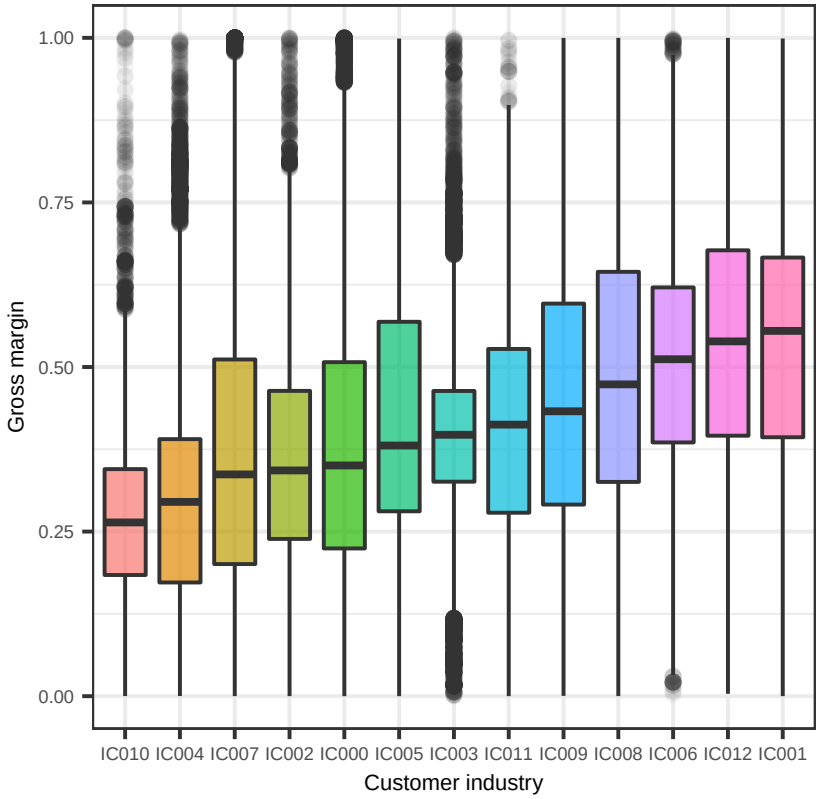


Figure 24: Gross margin by `customer_industry` without intercompany rows.

We can now incorporate both customer industry and customer region to see how will gross margin for specific `customer_industry` vary by `customer_region`. We can observe from Figure 25 that there are customer industries that have consistent ranking by customer regions, such as `IC001` by looking at the same colors of box plots. However, note that some of the box plots on the figure are skewed since they do not have many invoice lines in the data set, e.g. `IC010` for North America. Furthermore, we can notice that the company has lower gross margin for customers from, for example, `IC000` industry in North America where median of gross margin is near 0.375, while in Europe and Asia the same industry has median around 0.5. Therefore, we can conclude that there is an interaction between customer's industry and region.

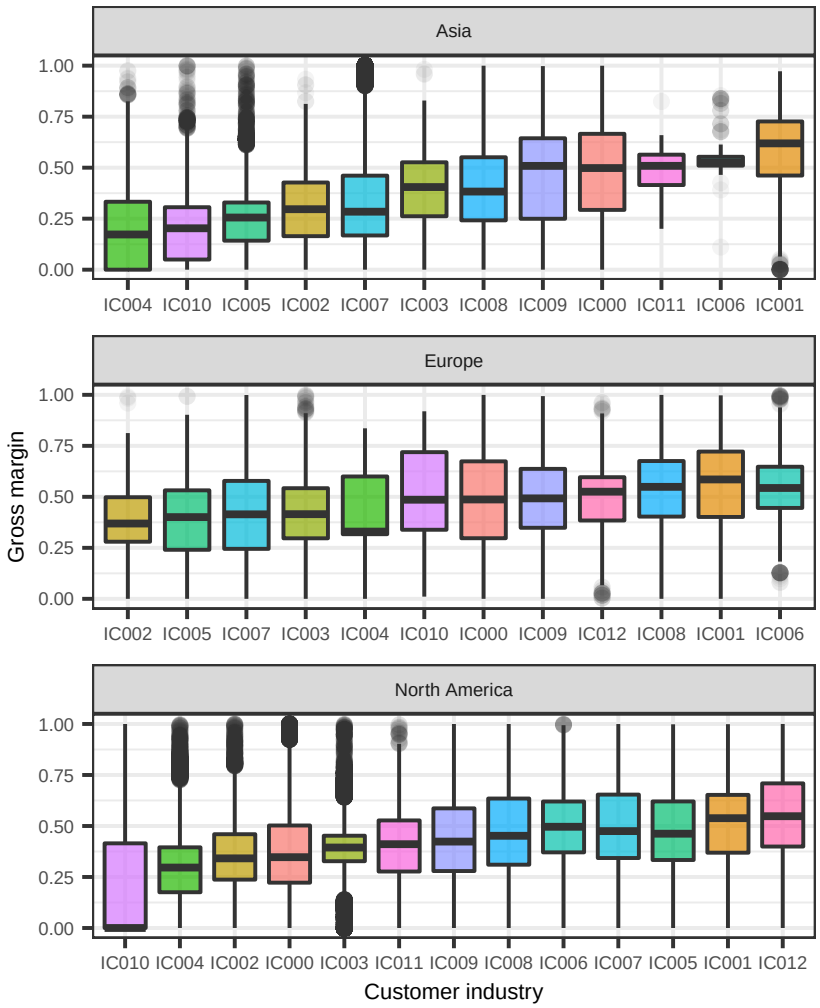


Figure 25: Gross margin by `customer_industry` *and* `customer_region` (gross margin was filtered to have values in range $[0, 1]$).

Gross margin with price, cost, and quantity

Gross margin is calculated directly from `invoiced_price` and `cost_of_part`, but both variables are also dependent on `ordered_qty`.

We can calculate company's revenue for each invoice line by using formula

$$revenue = gm \cdot unit\ price \cdot invoiced\ quantity$$

Let's plot the distribution of company's revenue by invoice line.

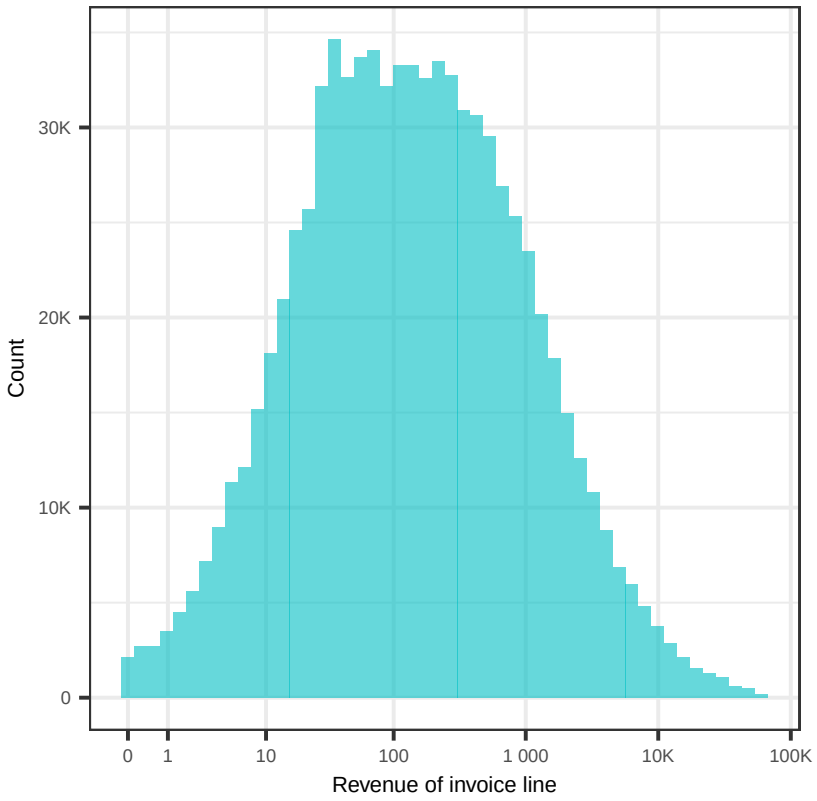


Figure 26: The distribution of revenue by invoice line.

Figure 26 shows us the distribution of invoice line revenue. Note that we are using *log* scale since the distribution of revenue is very skewed on a linear scale due to variety of invoiced prices and ordered quantities. The data looks log-normal.

Mean	Median	Stddev
1137.039	131.56	155152.9

Table 16: Summary stats for revenue of invoiced lines.

Now that we introduced another variable called `revenue`, let's calculate RPU [10] for each of the customer regions. The results are shown in Table 17.

[10] RPU = revenue per user

Customer region	RPU	Customer count
North America	$53954	6772
Asia	$200225	1391
Europe	$24644	2421

Table 17: RPU by customer region.

We see that Asia has by far the biggest RPU. However, we have already seen that Asia has the majority of `STAR` and intercompany invoice lines.

Thus, let's omit `STAR` customers and intercompany transactions for RPU since it would be useful to calculate RPU for ordinary customers. The results are in Table 18. North America and Asia have significantly higher RPU than Europe.

Customer region	RPU	Customer Count
North America	$51835	6706
Asia	$58639	1131
Europe	$23690	2393

Table 18: RPU by customer region. *Note that we omitted star customers and intercompany transactions.*

We can see how much did `STAR` customers and intercompany rows influence the results. Let's now investigate `STAR` customers.

Star customers

As we already said, STAR costumers can be found mostly in Asia (Figure 27).

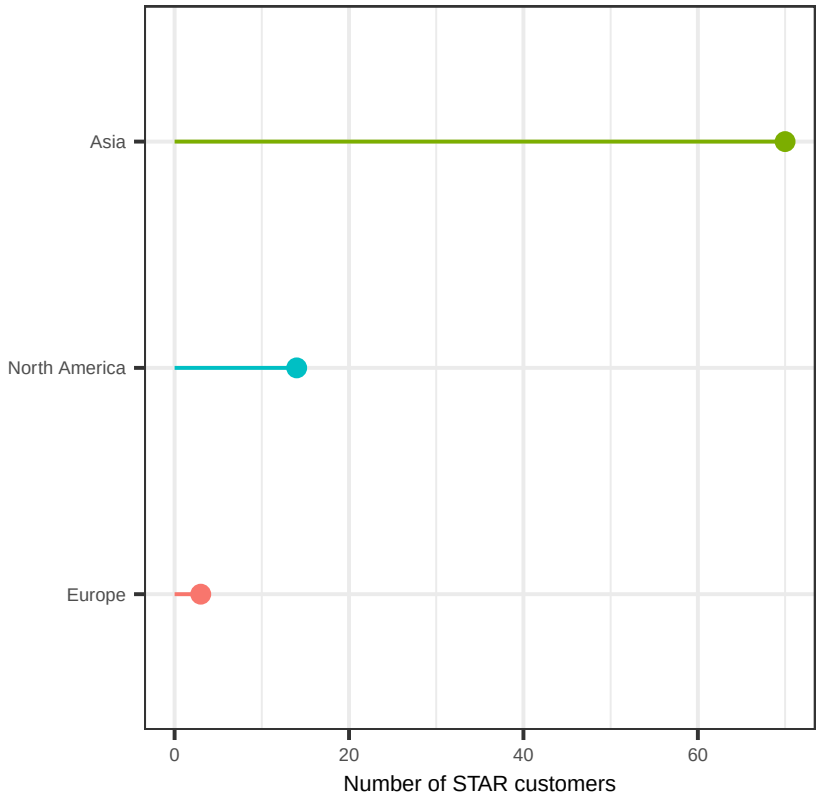


Figure 27: STAR customers are located mostly in Asia.

Moreover, STAR customers stand out in the terms of ordered quantity. Median ordered quantity for STAR customers is more than 50x larger than for OTHER customers. Box plot for both of the groups is shown in Figure 28.

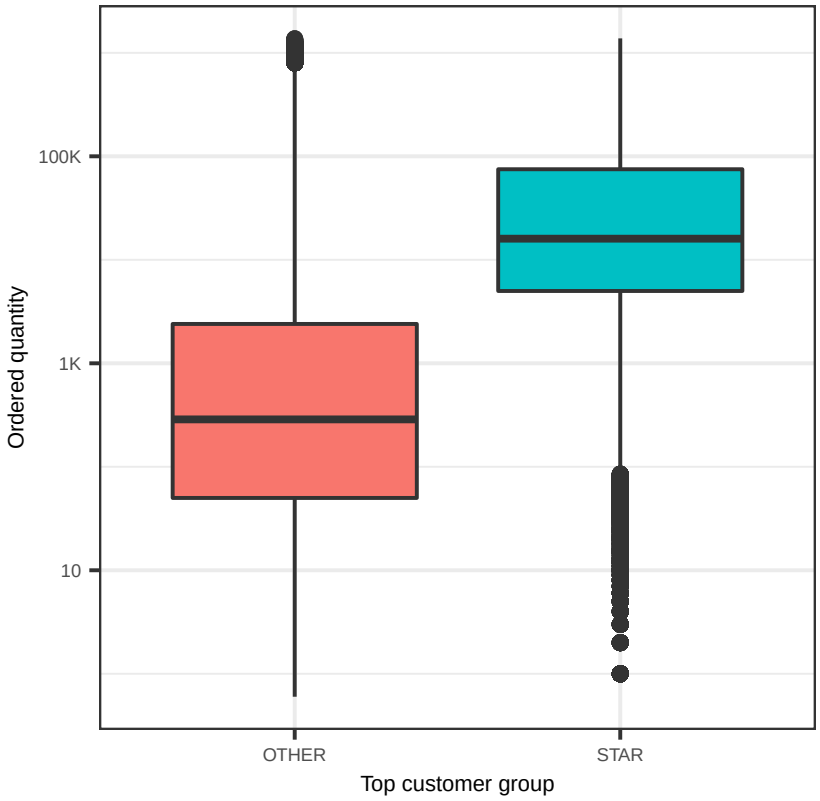


Figure 28: STAR customers have much larger orders than than OTHER customers.

Obviously STAR customers order a much bigger quantity of items than OTHER customers. This also means that they should be much more profitable for the company business. Even though the percentage of STAR customers is a little less than 1%, they had high percentage of invoice lines with the biggest revenue (Figure 29).

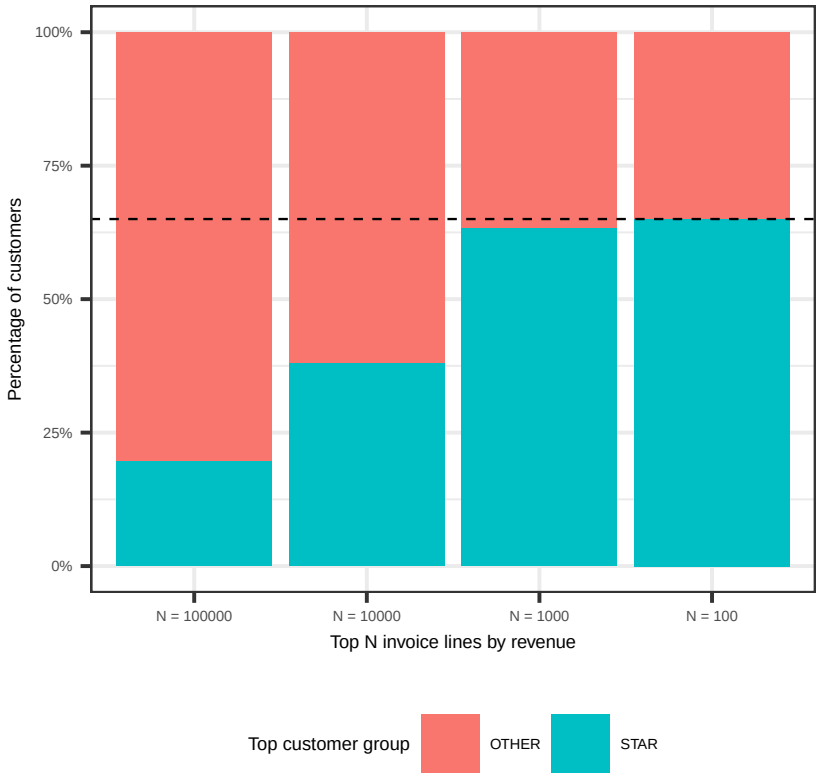


Figure 29: STAR customers are responsible for 65% of top 100 invoice lines by revenue.

Intercompany

`Intercompany` denotes whether the deal was a sale inside the company or end customer sale. Figure 30 shows the difference in gross margin for intercompany sales and end customer sales. Intercompany sales have lower gross margin due to lower invoiced prices.

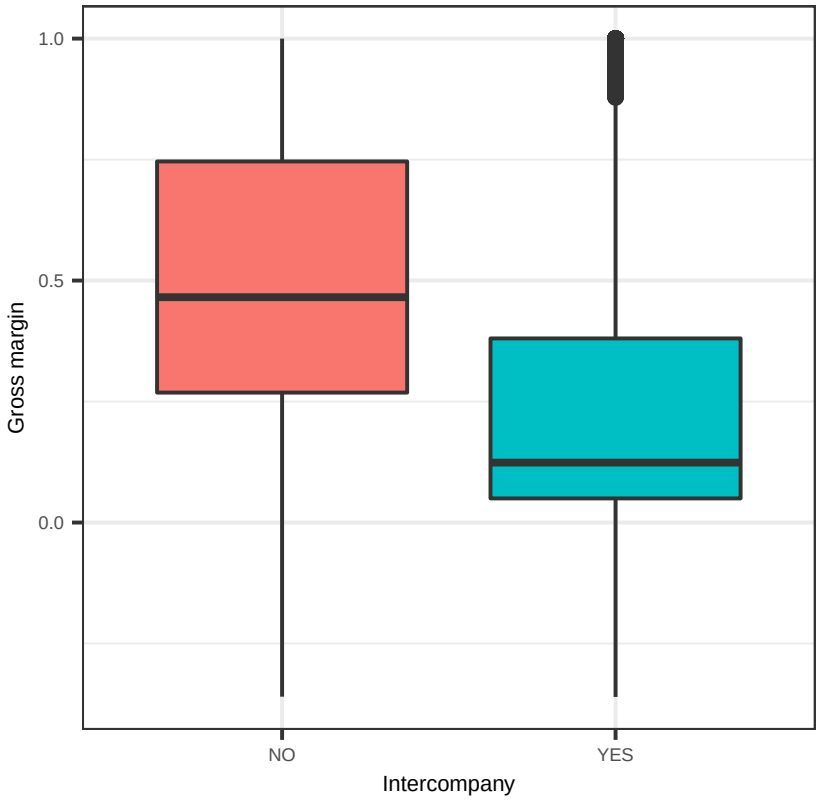


Figure 30: Gross margin by column `intercompany`.

Intercompany unit price was 44% lower than for non-intercompany. This was calculated by using invoice lines with items which appeared for both intercompany and non-intercompany transactions. So, generally speaking, intercompany unit price for some item is *lower* than non-intercompany price for the same item.

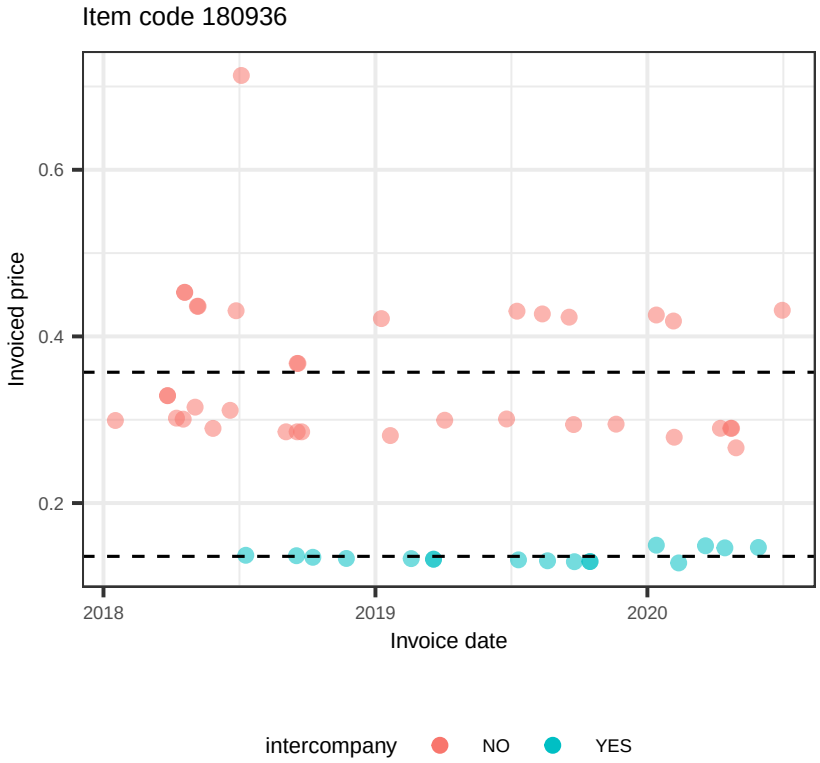


Figure 31: Intercompany sales have lower invoiced prices than end customer sales *This is an example for specific item.*

Revenue per user

Now that we know revenue per user (RPU) by `customer_region`, we can check out how much did each customer industry have for customer region. That is shown in Figure 32.

We can easily recognize from Figure 32 that customer industries are not uniformly distributed across customer regions. For example, customer industry `IC006` has significantly higher RPU percentage in Asia and Europe than in North America.

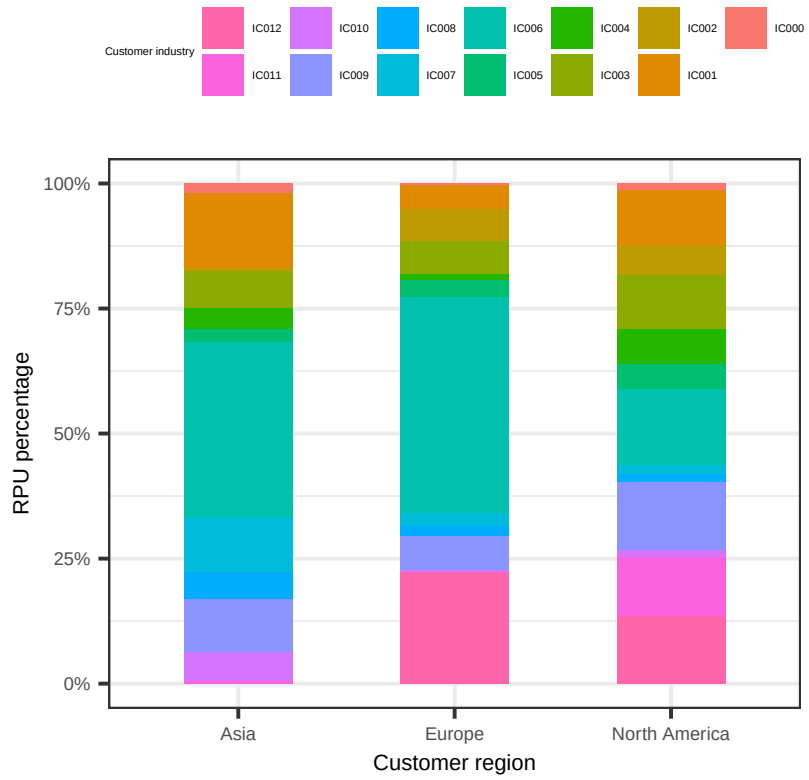


Figure 32: Different customer industries make up different percentage of RPU across customer regions.

Binning unit prices

Let's try to analyze the relationship between invoice price and gross margin. Since gross margin has unit price in the denominator, it can variate a lot for small unit prices. We give the following example:

$$unit\ price = 2, \quad unit\ cost = 0.5 \longrightarrow gm = \frac{2 - 0.5}{2} = 75\%$$

$$unit\ price = 2, \quad unit\ cost = 1.0 \longrightarrow gm = \frac{2 - 1.0}{2} = 50\%.$$

In order to make analysis between invoice price and gross margin more clear, we will do binning of invoice price into four bins. The question is: how to perform this binning? This can be done by dividing `invoiced_price` values into four bins with each bin spanning equal interval or with each bin spanning equal number of observations. However, another option is to do it manually using domain knowledge. Since we have seen in previous sections that invoiced price is log-normally distributed, it makes sense to make bins with logarithmically increasing borders. The bins will be:

1. *inexpensive*: $\langle \$0, \$10 \rangle$
2. *moderately priced*: $[\$10, \$100 \rangle$
3. *expensive*: $[\$100, \$1000 \rangle$
4. *very expensive* $\geq \$1000$.

After binning the `invoiced_price`, we get the results shown in Figure 33.

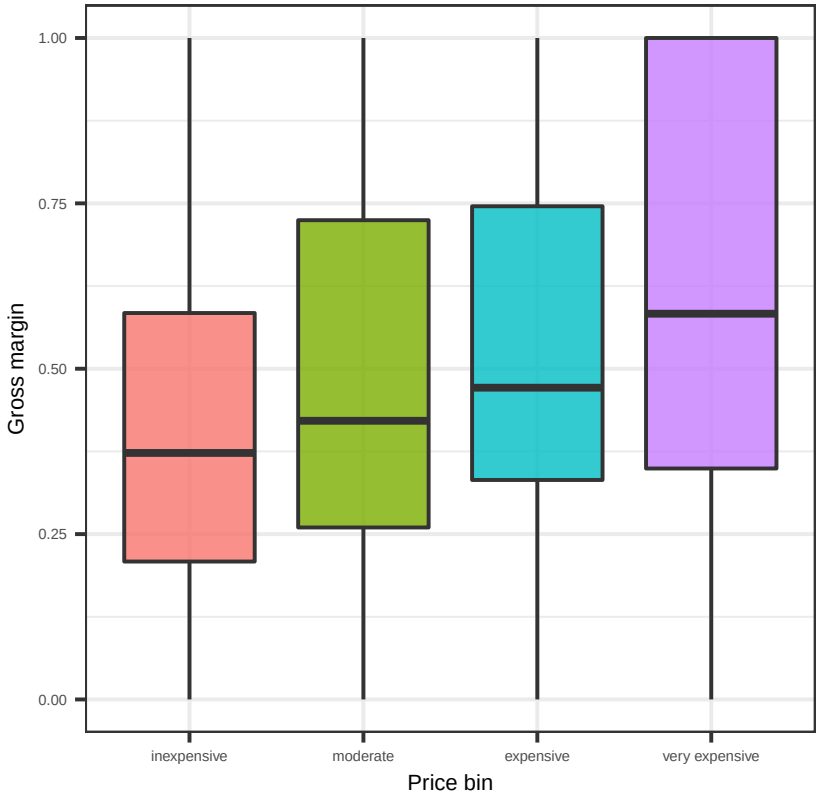


Figure 33: Gross margin box plots for different predefined price bins. At first glance, more expensive items (with higher invoiced price) have higher gross margin.

It seems like invoice lines with greater invoiced price get higher gross margin values. We can see an increase in gross margin median for less expensive bins to more expensive bins. However, the *very expensive* group has significantly higher gross margin. That price bin has gross margin of 1.0 in its interquartile range. Let's analyze why is that by plotting the density distribution in the form of ridgeline. The results are presented in Figure 34.

We can see that price bin of more than $1000 has much more invoice lines with gross margin equal to one. That is why box plots in Figure 33 shows that price bin of more than $1000 has significantly higher median than other price bins. Almost 30% of its invoice lines have gross margin equal to one[11].

Again, having gross margin at 0.0 and 1.0 is suspicious because:

- gross margin 0.0 means the company is making no revenue so we can assume that invoice lines with gross margin of 0.0 are special cases, i.e. they do not represent typical commercial processes of selling
- gross margin equal to 1.0 means that the company had zero cost in

[11] To repeat once again, gross margin equal to one means that the unit cost was zero

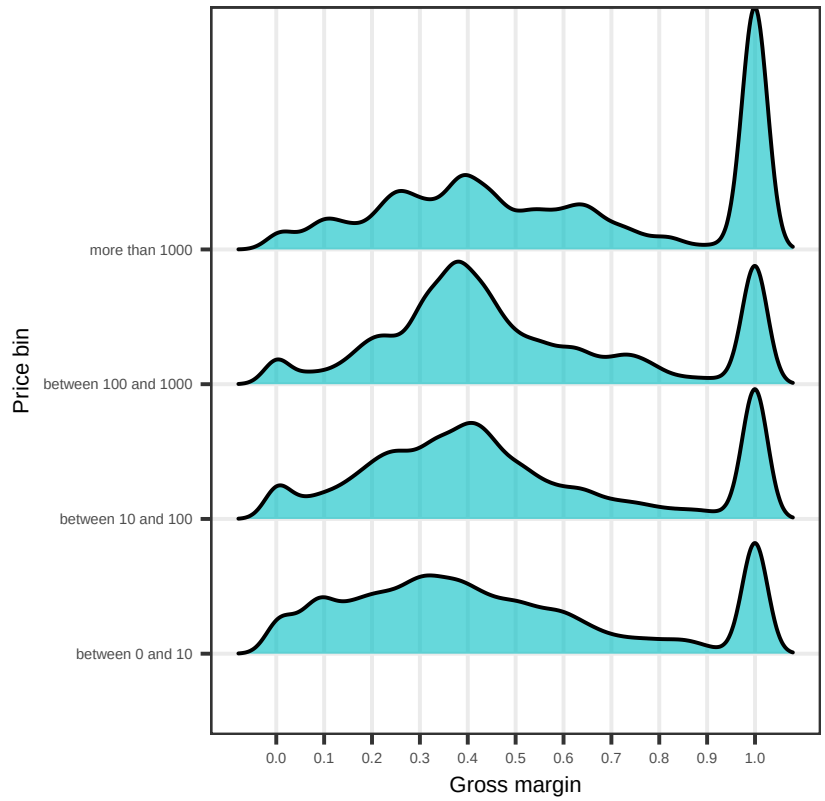


Figure 34: Each of these distributions has many invoice lines with gross margin equal to one, but most of them are in bin of more than $100.

the whole selling process which is not realistic from the business perspective.

After removing asymptotic values of gross margin (0.0 and 1.0), we get box plots shown in Figure 35.

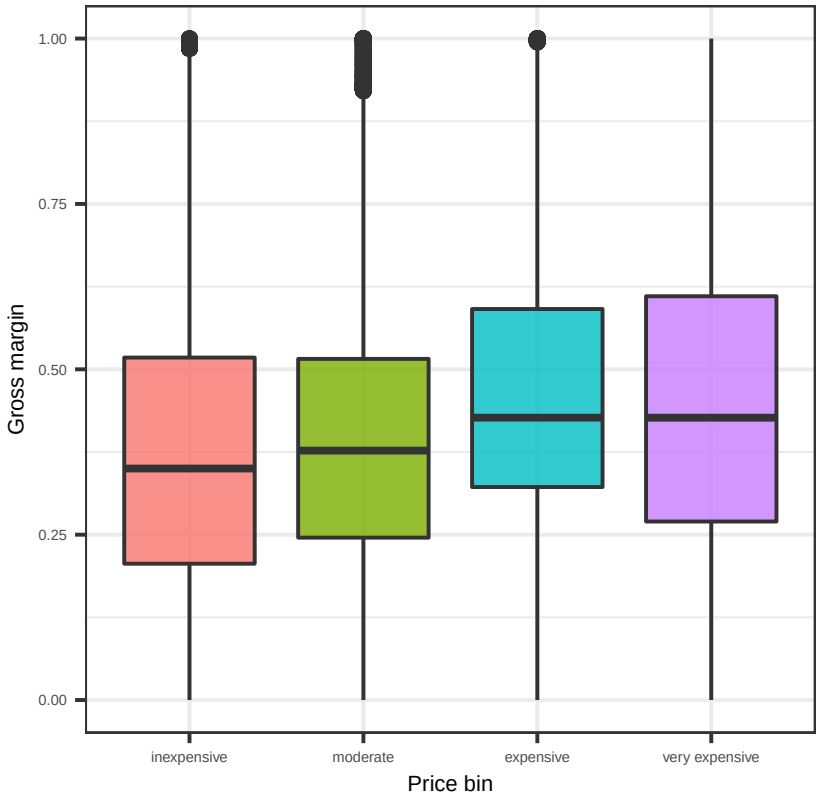


Figure 35: After filtering out asymptotic values of gross margin, price bins have smaller differences between median gross margin.

Let's now take a look at the relationship between unit price (`invoiced_price`) and unit cost (`cost_of_part`). We can assume that the item will have higher price if the cost of manufacturing that item was higher. Figure 36 tells us that, in general, the more expensive the item is, the bigger the cost of part is. Note that, again, we are using log scale since `cost_of_part` has a wide range of values, from zero to a few hundreds of thousands. We can also observe that for *very expensive* price bin, we have quite a few outliers that have high price, but rather low values for `cost_of_part`.

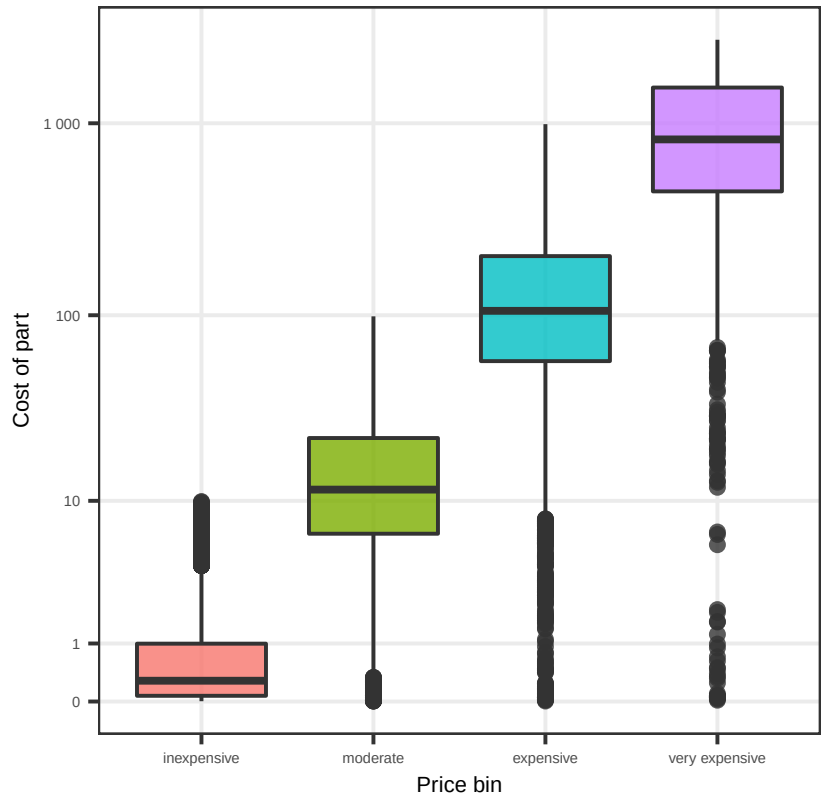


Figure 36: More expensive items have higher unit cost.

It is not strange to see the same item being charged less when ordering higher quantity. That is called *quantity discount* - an incentive offered to a buyer that results in a decreased cost per unit of goods or materials when purchased in greater numbers[12].

Figure 37 shows decrease of invoice price with increase in shipped quantity for two chosen items.

[12] Daniel Liberto. Quantity discount. https://www.investopedia.com/terms/q/quantity-discount.asp, March 2021

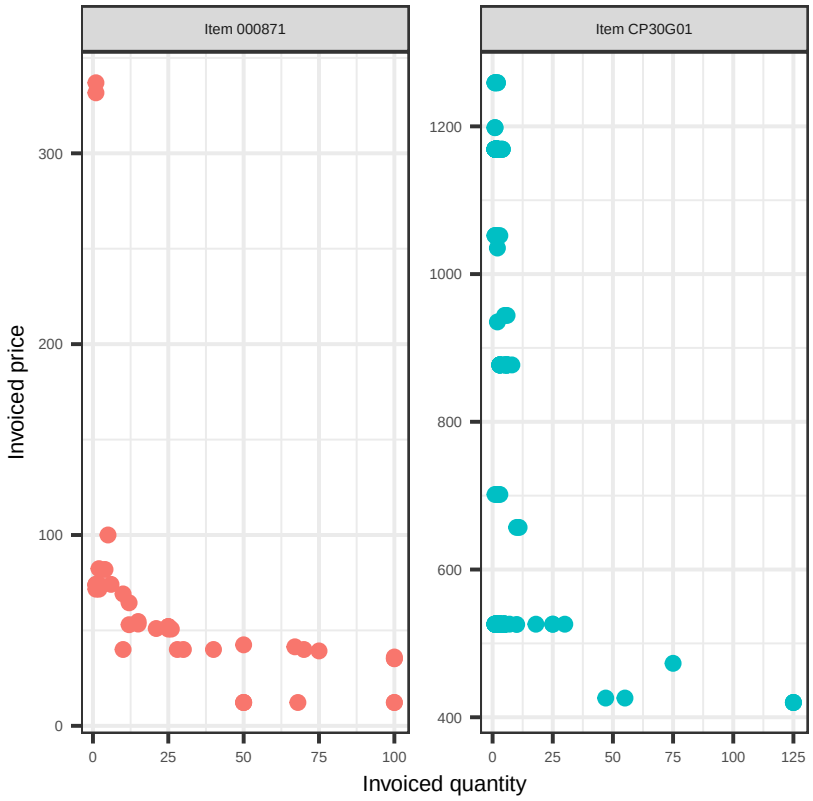


Figure 37: We can observe quantity discount for specific items.

Let's now calculate how the observed change in invoiced price relates to gross margin. To be specific, let's pick an item from the right side of Figure 37. If the cost of production does not change, decrease in invoiced price must result in decrease in gross margin. Figure 38 shows us the change in gross margin after increase of invoiced quantity for one particular item chosen as an example.

One must not conclude that smaller gross margin means bad business results. Let's take a look at Figure 39.

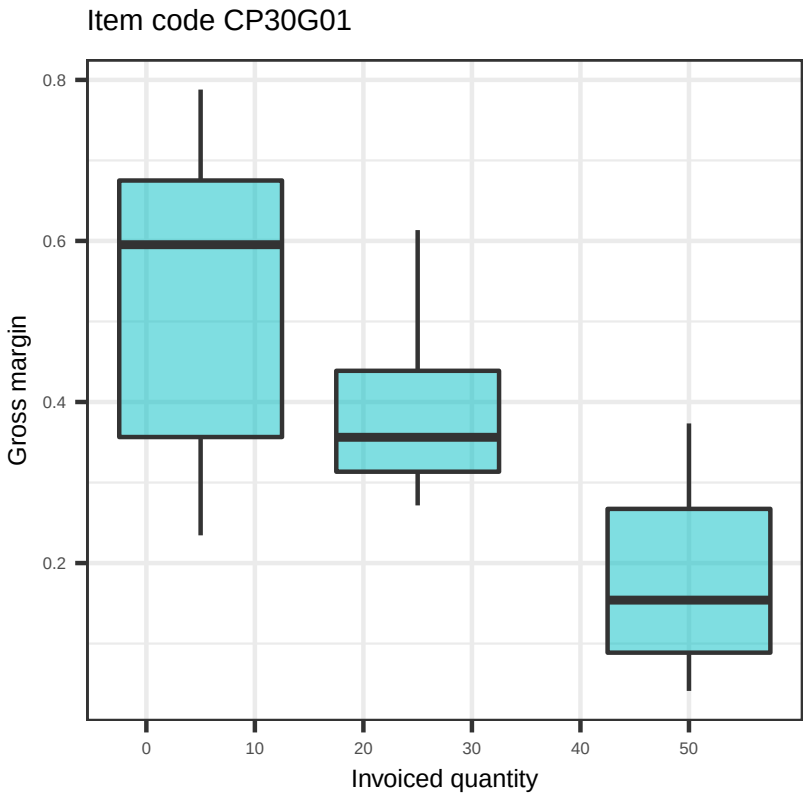


Figure 38: For this item the gross margin decreased with increase of invoiced quantity.

This item has smaller gross margin when bought at larger quantity, but total revenue is *increased*. Exact median values for gross margin and revenue are shown in Table 19. If we think about it, it makes sense. Gross margin alone does not tell us whether specific invoice line made company a lot of money. For example, if we have

$$cost = 2,\ price = 10 \implies gm = \frac{10-2}{10} = 80\%$$

the total revenue is 8. However, if we have

$$cost = 40,\ price = 50 \implies gm = \frac{50-40}{50} = 20\%$$

the total revenue is 10. In the latter case, the gross margin was smaller, but the revenue was larger.

Quantity bin	Gross margin median	Revenue median
low	0.595	814
medium	0.356	3972
high	0.154	4712

Table 19: Gross margin and revenue for Item `CP30G01` after splitting invoice lines into bins by quantity.

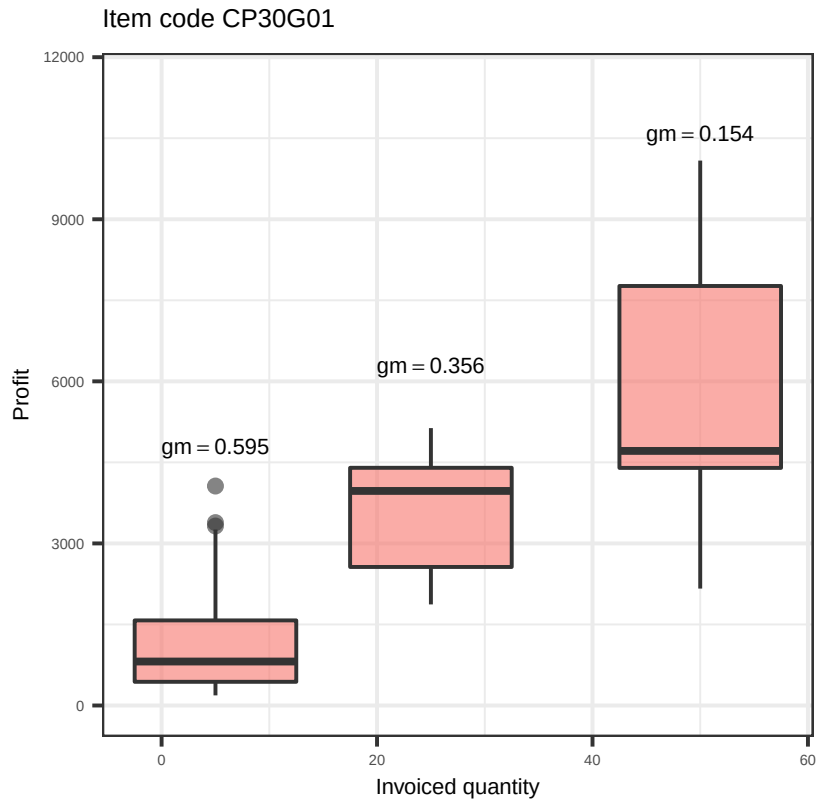


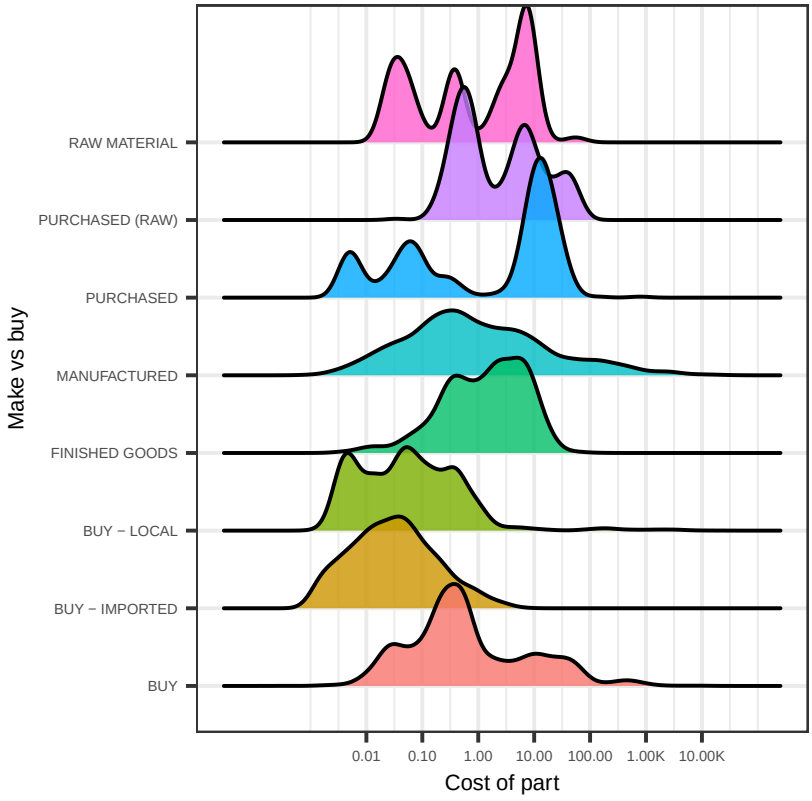
Figure 39: For this item gross margin decreased but total revenue increased.

Make vs buy discussion

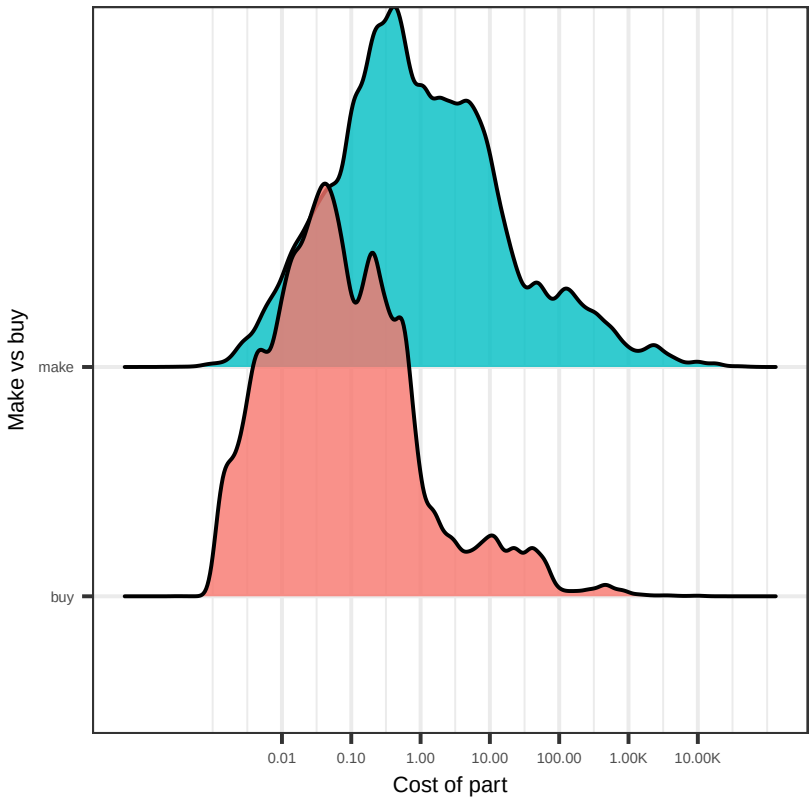
The aforementioned idea of possible merging[13] of `make_vs_buy` into only two categories needs to be further analyzed. It is instructive to observe how the new `MAKE` and `BUY` categories relate to other columns and each other.

Figure 40 shows how `cost_of_part` depends on different `make_vs_buy` categories. The plot before the merge confirms the problem of too fine granulation of the `make_vs_buy` column. It is quite inconvenient to make general inferences about a lot of separate distributions of costs of parts for 8 different `make_vs_buy` categories (two of the categories were filtered out since they had `cost_of_part` lower or equal to zero - `BUY - CUST. SUPPLIED` and `BUY - INTERPLNT TRNS`). However, it is quite straightforward to make conclusions about the plot after the merge into only two categories. For instance, the first thing that can be noticed is that bought items have generally lower `cost_of_part` than made ones.

[13] Merging details of initial categories are shown in Table 22



(a) Before merging



(b) After merging

Figure 40: Cost of part distribution for different `make_vs_buy` categories before and after merging into only two categories. Before plotting, $\log_{10} x$ was applied on x-axis. Negative and zero `cost_of_part` values were removed a priori.

Another interesting observation that comes into light after the merge is shown in Figure 41. Similarly to the `cost_of_part`, bought items have generally lower `invoiced_price`.

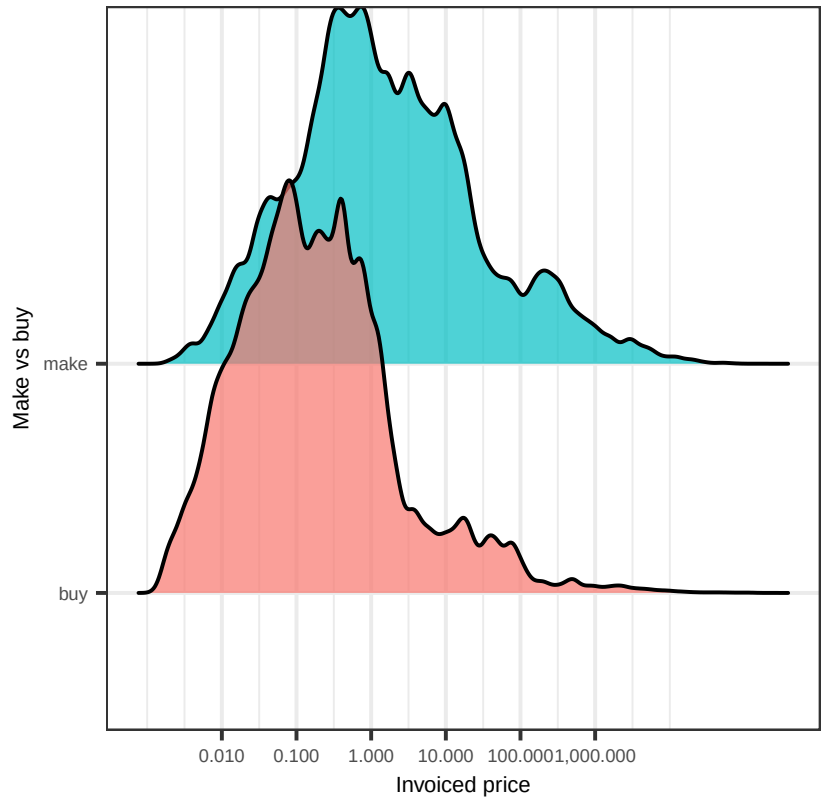


Figure 41: Distribution of values from `make_vs_buy` column with respect to the `invoiced_price` variable. Logarithm was applied to x-axis and only the values with `invoiced_price` higher than zero were included in calculation. Additionally, `invoiced_price` values were filtered between 1^{st} and 99^{th} percentile.

Let's now consider the proportion plot of the newly labeled `make_vs_buy` rows with relation to `customer_region`. This relationship is shown in Figure 42 which tells us that for every existing `customer_region` (`Asia`, `Europe`, and `North America`) most of the products marketed to customers are manufactured and not bought and resold.

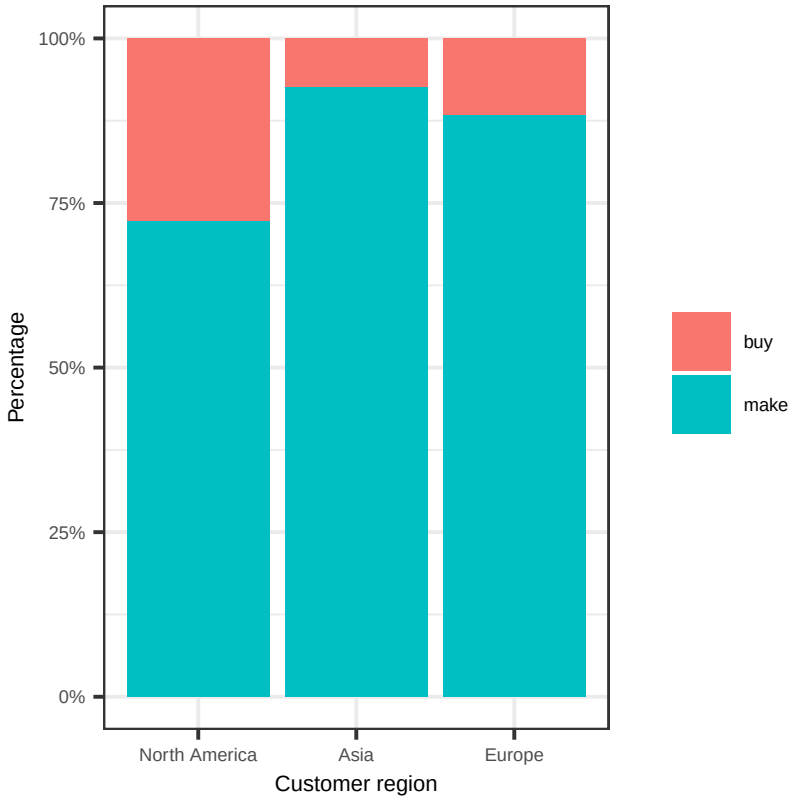


Figure 42: Proportion of rows by `customer_region` in relation to the merged values of the `make_vs_buy` column.

Customer behaviour

To achieve the goals of dynamic deal scoring, it is important to understand the customer behaviour so that we can distinguish high valued customers from those who do not have as much of an impact on the revenue. As it was already mentioned, there are 87 `STAR` customers in the data set, which we will analyze separately to identify differences in their behaviour compared to other customers in the data set. Additionally, as it was emphasized in Figure 2, most of the `STAR` customers are located in Asia. There is a significant difference between the industries which `STAR` customers belong to. From observing Figure 43, it is obvious that `STAR` customers belong to specific industries labeled with `IC007`, `IC005`, and `IC010` while ordinary customers mostly belong to the industry with code `IC000`.

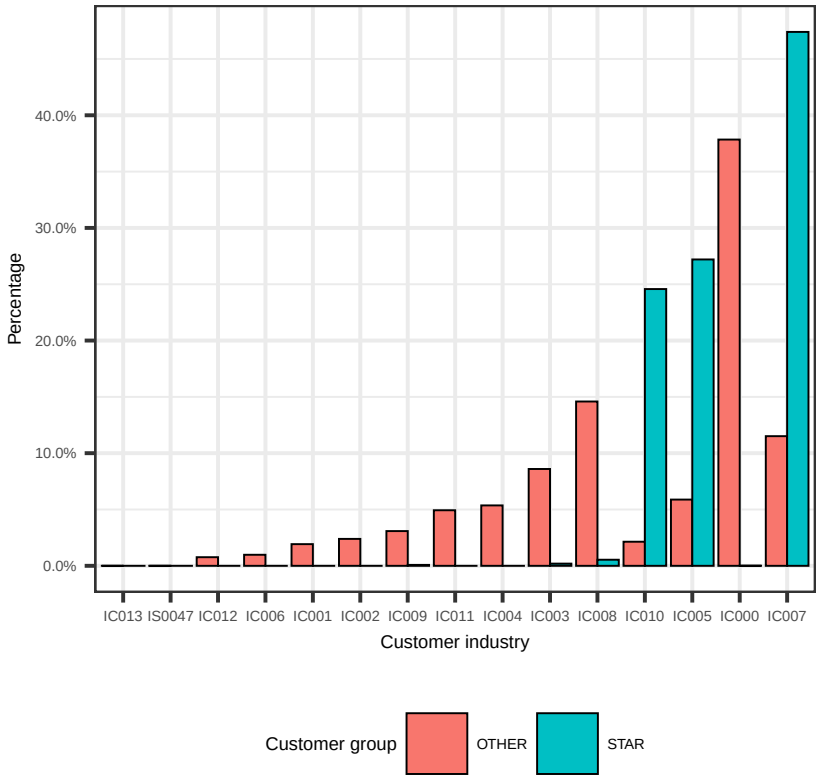


Figure 43: Industries that `OTHER` and `STAR` customers belong to, shown in percentages.

Another interesting point of view is observing the actions clients are making over time. In Figure 44 we can see that the number of orders that STAR customers make is significantly higher than the number of orders that OTHER customers make. Subsequently, the median of the revenue is also higher within the STAR customers.

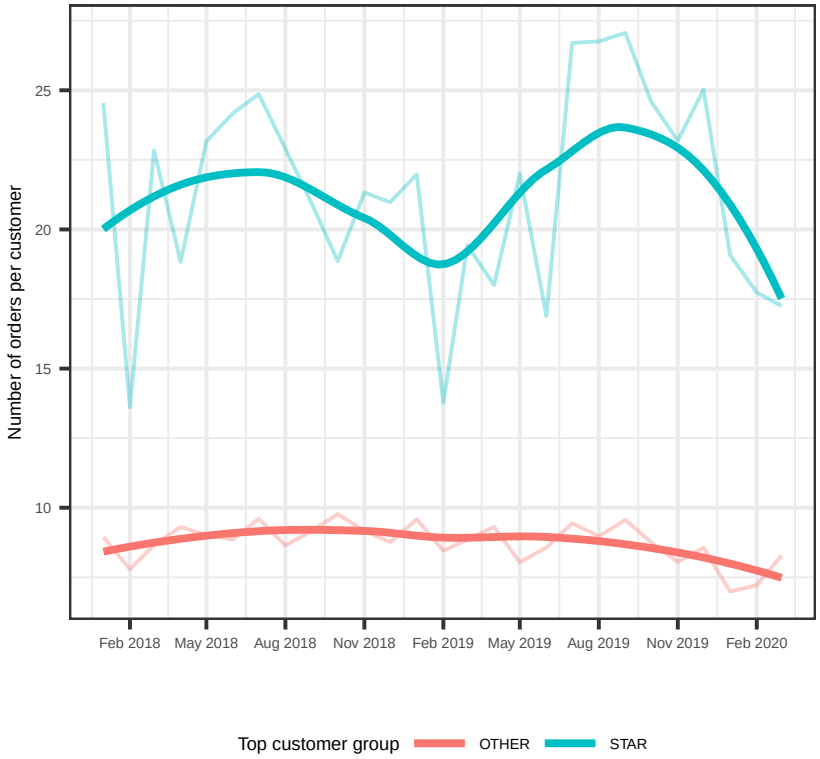


Figure 44: Number of orders in time for STAR and OTHER customers.

Besides the frequency of the orders, ordering quantity of users in the STAR group is much higher. This is presented in Figure 45 where it is clear that the values for the ordered quantity go much higher for STAR customers. On the other side, the invoiced price they receive is slightly lower than for OTHER customers. Figure 45 shows us these differences on a logarithmic scale, because `invoiced_price` and `ordered_qty` can have values with large order of magnitude.

One more thing here that could be analyzed in detail is how the customer behaves with respect to how long they have been a customer for the company, i.e. *customer lifespan*. By using values of column `customer_first_invoice_date`, it is possible to calculate the rounded number of customer lifespan in years. The distribution of customer lifespan is shown in Table 20:

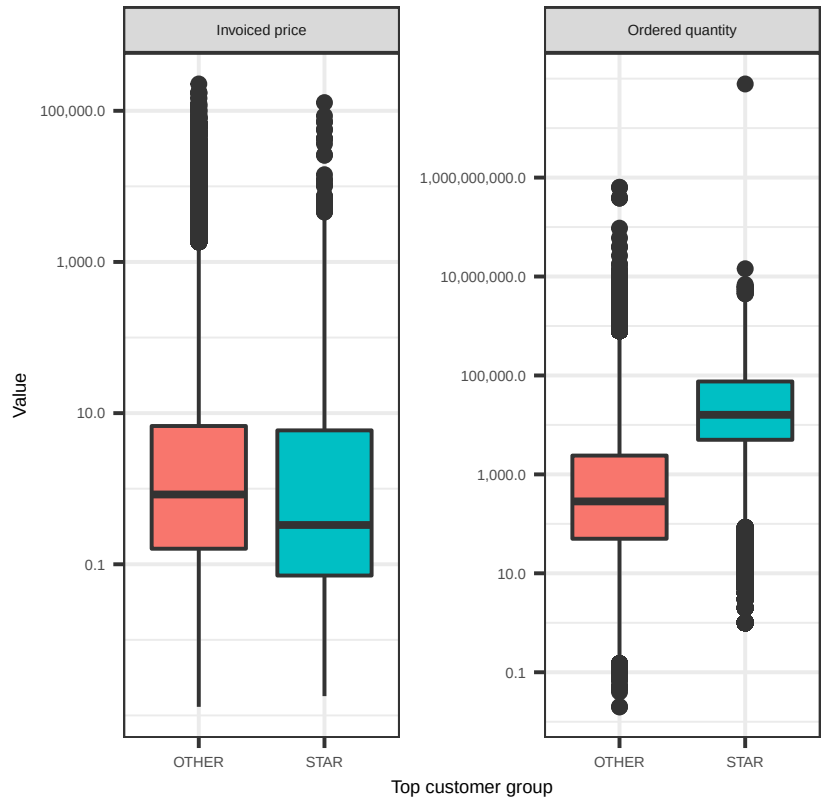


Figure 45: Relationship between `top_customer_group`, `invoiced_price`, and `ordered_qty`.

Quartile	Customer lifespan
min	0
q1	2
q2	5
q3	11
max	27

Table 20: Customer lifespan years divided into quartiles.

By looking at the aforementioned table, we can observe if all the customers were treated differently based on their lifespan. Substantial number of invoice dates in the data set goes back to the beginning of 2018, but a lot of the customers have a much earlier `customer_first_invoice_date`. For example, we have customers that had their first invoice over 20 years ago. Customers inside certain lifespan groups behave differently, but in general the number of orders made by new customers is lower than of those who are with the company for a longer time (Figure 46).

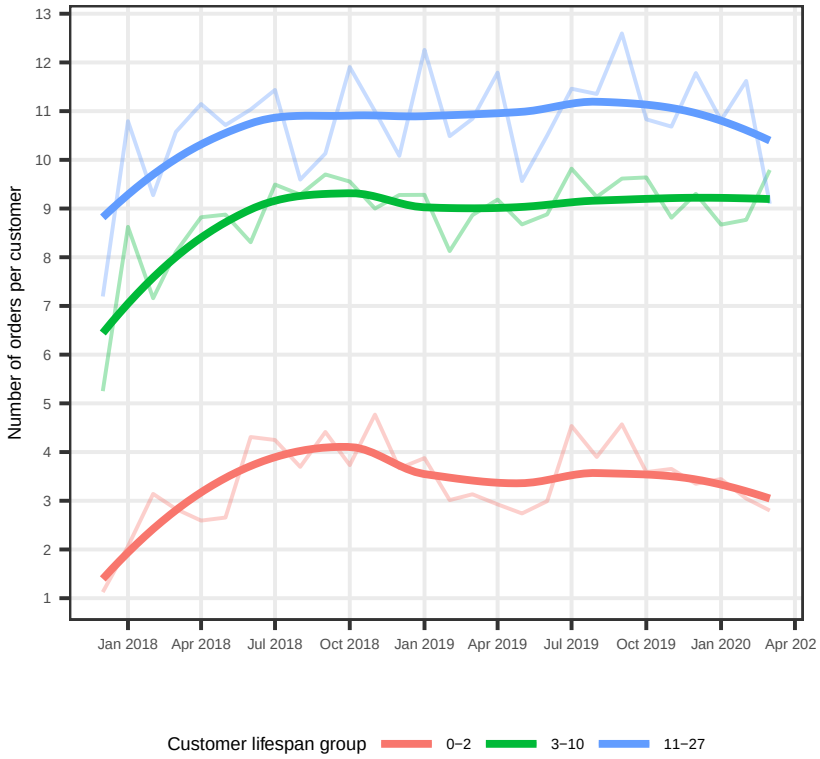


Figure 46: Pattern of customer behaviour over time with respect to number of orders.

Let's end this subsection with a simple proportion plot shown in Figure 47. It tells us that most items in the data set are sold to customers exactly in the region where they were manufactured. For example, most items manufactured in *Asia* are sold to customers in *Asia* etc. This is most accentuated for the case of North America.

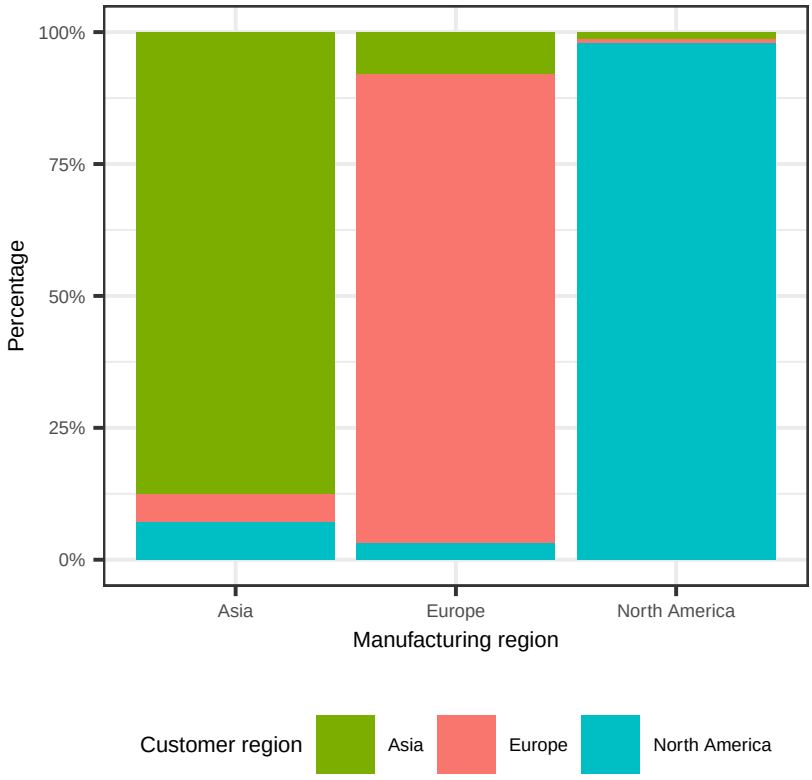


Figure 47: Proportion of rows labeled with `customer_region` in relationship with rows labeled with `manufacturing_region`.

Interesting invoice patterns

This part describes interesting patterns observed in the data set which are a result of businesses processes. They are particularly interesting since they can look like a data quality issue if one does not know how the company deals with each specific situation. Some interesting cases for invoice:

- presence of transportation, expedite, and delivery items
- negative price
- blanket orders.

As already stated, there are invoice lines for which we concluded that they represent transportation, delivery, expedite, and similar costs. They sometimes have revealing item codes, such as:

- `TRANSPORTKOSTEN`
- `/EXPEDITE-CUST`
- `FREIGHT`[14] etc.
- `/LTI`[15].

[14] *freight*: transport (goods) in bulk by truck, train, ship, or aircraft. (Oxford Dictionary definition)

[15] It is possible that LTI denotes American logistic company `https://www.ltitrucking.com/`

The listed item codes are the most frequent ones since every order has to be delivered. They can be recognised by zero unit cost. Moreover, company sometimes charges those items but more often does not. Possible reasons for mentioned behaviour are:

- company has contracts with delivery companies so it does not charge the customer
- company pays for delivery if the customer pays above predefined price threshold for the order
- free shipping except in some special cases.

To continue with invoice patterns, we also observed that some invoice lines have negative invoiced price. Initially, we thought that this is a data quality issue. However, that is not the case. This specific situation can happen if customer overpays the order. If the price for the next order he makes is smaller than the amount he overpaid the previous order, the next order will have invoices with negative invoiced price. Company implemented this logic to have its records in order.

Blanket orders are orders with multiple delivery dates scheduled over a period of time[16]. Customer make them in order to take advantage of predetermined pricing. Concrete example from the data set is shown in Table 21.

[16] Wikipedia. Blanket order. `https://en.wikipedia.org/wiki/Blanket_order`, February 2021

Order Num.	Order Line Num	Item Code	Ordered Qty	Delivered Date
731696	1257554	1000351	1500000	**2018-08-15**
731696	1257555	1000351	1500000	**2018-08-22**
731696	1257556	1000351	1500000	**2018-08-29**
731696	1257557	1000351	1500000	**2018-09-05**

Table 21: Example of a blanket order from the data set.

Data quality

Data set came in an unique format. Columns were separated with a vertical bar. Character encoding for the data set was UCS-2LE. There were also some minor mistakes in the data set that needed rectifying before we could successfully load the data set into memory. Extra quotes were found in some lines, while some contained peculiar characters that we eliminated.

Regarding the columns of the data set, almost every column had some irregularities. Let's start with the graphical representation of missing values in the data set by columns. Figure 48 shows that two columns stand out the most by missing values count. The first one is `sales_channel_grouping` and all the values from this column were missing. Hence, it was eliminated from further consideration. The second one is `price_last_modified_date_in_the_erp` with 74% of missing data which indicates that price was not modified for most of available data. Since the `gm` column is the most important indicator for dynamic deal scoring, it is necessary to give comment about its 117122 missing values.

Gross margin can be undefined only in three cases:

1. `invoiced_price` is zero
2. `invoiced_price` is missing
3. `cost_of_part` is missing

which follows directly from Equation 1.

Columns `sales_channel_internal` and `sales_channel_external` were exact duplicates of each other i.e. they had same values in all rows. These columns represent coded names of either internal or external sales representatives with no additional information. Moreover, another two columns were almost exact duplicates of each other. These are `invoiced_price` and `invoiced_price_tx` with 88% rows with same recorded invoiced price. Column `invoiced_price_tx` refers to the actual invoiced price from the transaction after accounting for exchange rates due to possible different currencies between the seller company and the customers (which were located on three different continents according to the values in column `customer_region`).

Next, we will discuss the illogical values that appeared in some columns. For example, there are 549 rows with negative `invoiced_price`.

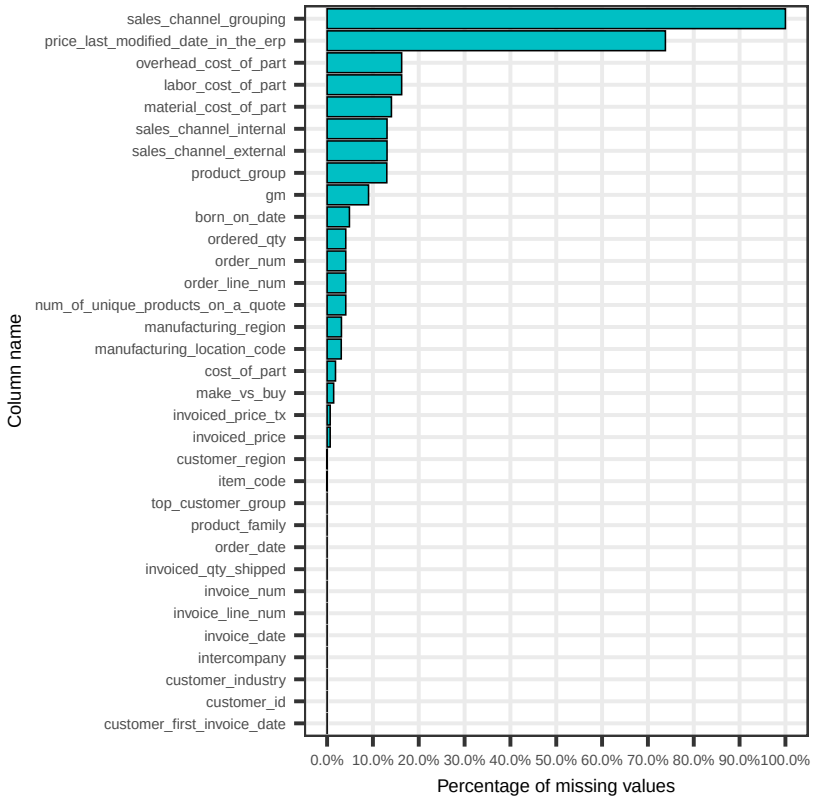


Figure 48: Counts of missing values for each column of the data set.

This could be explained by the fact that the company keeps track of overpaid orders so the next order, for a customer that overpaid the previous one, is recorded in the company database with negative `invoiced_price`. There are also 87441 rows with `invoiced_price` equal to zero which is the same as giving the invoiced item for free. However, this is probably due to the structure of the data set. Another case is when the item in the data set corresponds to the transportation cost. Most of these items with price set to zero have been manufactured and sold in Asia. Furthermore, there are 970 lines with negative `cost_of_part` and 221027 lines with its value set to zero (this leads to 164163 number of lines with `gm` equal to exactly 1). Consequently, lines where `invoiced_price` was lower than `cost_of_part` with positive `invoiced_price` gave 110971 negative `gm` values. Equation 1 gives insight why this makes sense. Gross margin can only be negative if fraction

$$\frac{cost\_of\_part}{invoiced\_price}$$

results in value higher than 1. There exists 96 lines with `gm` higher than 1. For this to happen, the mentioned fraction has to be lower than 1. Another peculiar situation that we observed was with the ordered and invoiced quantity columns. `ordered_qty` column had negative value in 22 and `invoiced_qty_shipped` in 79 lines.

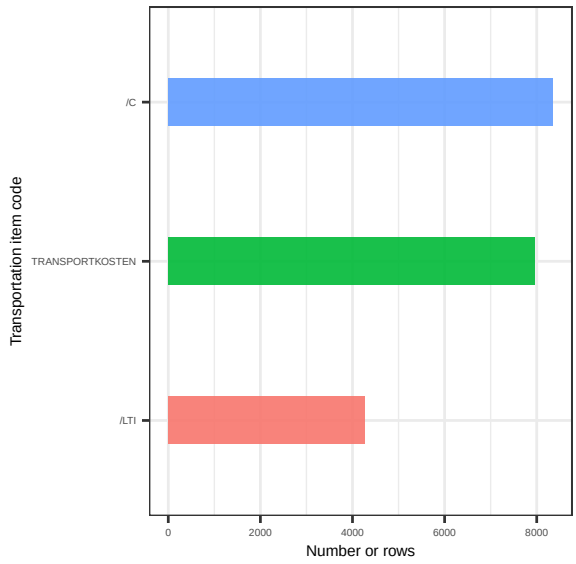


Figure 49: Row counts of the most frequent item codes with all values of `invoiced_qty_shipped` equal to zero.

For 15737 lines of column `ordered_qty` and 161091 lines of column `invoiced_qty_shipped` value was zero. The mentioned zero values can be explained by looking at the `item_code` column for the corresponding lines. Some of the items that appeared the most in these lines are the ones with item codes `TRANSPORTKOSTEN`[17], `/C`, and `/LTI`. Since the

[17] To be exact, `TRANSPORTKOSTEN` had one record with `invoiced_qty_shipped` different from zero (probably a labeling error)

items are coded, the only self-explanatory `item_code` that could lead us to this conclusion was `TRANSPORTKOSTEN` which had `customer_region` equal to `Europe`. We believe that this represents the shipping cost for customers in Germany. Using inductive reasoning, we inferred that most of other items with zero value in columns `ordered_qty` and `invoiced_qty_shipped` refer to lines that describe shipping costs. We show row counts of these three most interesting item codes with `invoiced_qty_shipped` equal to zero in Figure 49.

We decided to look at all of the mentioned irregularities as irrelevant for our dynamic pricing model and discarded them beforehand. More on these data cleaning decisions comes in Chapter Data preparation.

$$gm = \frac{invoiced\_price - cost\_of\_part}{invoiced\_price} \tag{1}$$

Worth mentioning are also the values in columns `ordered_qty` and `invoiced_qty_shipped` that are not integers. This might be unusual, to order a fraction of some item, but it is valid to represent ordered and invoiced quantities in that way. Furthermore, we found out there are items which sometimes do and sometimes do not have `cost_of_part` specified. We will discuss this in Chapter Data preparation. The `make_vs_buy` column should also be mentioned here. As Figure 8 presented, the granulation of its distinct values might be too verbose when we consider the name of the column. Thus, for the sake of the model we are developing, we can merge its categories as the Table 22 suggests. Furthermore, if we look at the number of rows in the data set that contain one of the make or buy categories we can see that there is a negligible number of rows labeled with `RAW MATERIAL`, `BUY - CUST. SUPPLIED`, `BUY - INTERPLNT TRNS`, and `PURCHASED (RAW)`. These rows could easily be discarded since they could lead to overfitting when building our model.

Another observation that puzzled us was that each data set row with non-missing `make_vs_buy` value had defined corresponding `manufacturing_region` regardless of whether the item was *manufactured* or *bought*. In other words, if the item was bought, why does it have defined where it was manufactured? The explanation in this case could be that the company that bought and resold items knew where the items were manufactured originally and thus the information ended in the data set.

Lastly, given the information that the data set was constructed manually, there exists a chance for labeling errors. One of the possible labeling errors is the item manufactured in Asia presented in Table 23. This is an extreme outlier when we consider that the mean `invoiced_price` of all the products in its `product_family` `PF002` equals to 496.618 (calculated without the outlier's `invoiced_price`). Additionally, its `cost_of_part`

Make	Buy
MANUFACTURED (932386)	BUY (85452)
RAW MATERIAL (184)	BUY - IMPORTED (151245)
FINISHED GOODS (66727)	BUY - LOCAL (38348)
-	BUY - CUST. SUPPLIED (155)
-	BUY - INTERPLNT TRNS (13)
-	PURCHASED (1174)
-	PURCHASED (RAW) (779)

Table 22: `make_vs_buy` categories classification. Numbers in brackets refer to row counts from the data set.

is equal to zero which results in `gm` equal to 1 which is also suspicious for item so absurdly expensive. Another example of obvious labeling errors are the 70777 rows with `order_date` set to 9999-12-31 and 32 rows with negative `customer_id` set to -99.

These might not be the only labeling errors, but we hope to eliminate all of the exceptions with common sense filters. These include rules like `cost_of_part` > 0, `invoiced_price` > 0 etc. Since `cost_of_part` of the outlier is zero, the first mentioned filter will rule out the first mentioned outlier from further analysis and the development of the machine learning model.

manufacturing_region	product_family	invoiced_price	cost_of_part	gm
Asia	PF002	1089758016	0	1

Table 23: Paramount columns and corresponding values for the obvious labeling error of record with `item_code` equal to 652181 and `invoiced_price` of roughly 1 billion dollars.

Data preparation

Through previous chapter, we have analyzed columns of the data set in great detail and stated ideas about which columns to keep for building our model, which columns to remove, which to merge etc. In this chapter, we systematically bring data preparation and cleaning decisions that emerged after data understanding was finished.

Regarding the missing values, the decision was made not to impute the missing values during the training phase. However, during the testing phase this is necessary to enable prediction for incomplete data input and we implemented a simple missing values imputation technique which we describe in the Chapter Modelling.

For some columns, we tried to impute missing values *by hand* before building the model. This was the case when two columns were describing the same feature (one more general and the other more specific) and the value was missing for the more general column of the two. Hence, we could infer the missing value by looking at other rows where the information was present. We tried this imputation on two pairs of columns. The first is pair consists of `manufacturing_region` (general) and `manufacturing_location_code` (specific). If manufacturing region was missing and manufacturing location code was not, the value for manufacturing region was inferred by looking at rows where both values were present. Unfortunately, the only case when `manufacturing_location_code` was present and `manufacturing_region` was not for the location code equal to `N9` and no other rows had defined `manufacturing_region` for that specific location code. Similarly, the second pair of columns was `product_family` and `product_group`. Again, this yielded no saved rows because `product_family` was defined for all the rows of the data set.

Since there were 221027 (17%) rows with `cost_of_part` equal to zero, we inferred the `cost_of_part` value using values from past or future orders with same `item_code`. Otherwise, we would lose a vast number of rows. Naturally, after the inference, `gm` was recalculated because `cost_of_part` changed. This procedure saved us around 11000 rows.

After making inferences about missing or illogical values, the follow-

ing columns were filtered on specific conditions to remove rows that would otherwise be responsible for potential overfitting of the machine learning model (noise in the data set):

- column `intercompany` was filtered with condition `intercompany == NO` because values with `intercompany == YES` would include bias into model since these rows refer directly to the deals inside the company and would bring more harm than good to the data distribution
- all rows that had `cost_of_part` == 0 after imputation were discarded because it does not make sense that the item was made or bought for free
- upon column `invoiced_price` the constraint was set to `invoiced_price` > 0 to get rid of rows with negative invoiced price
- values of column `gm` were limited to the range $(0, 1)$
- the constraint for `ordered_qty` and `invoiced_qty_shipped` was quantity > 0 to account for rows with negative or zero values in these columns
- `make_vs_buy` column was reduced by 184 rows of `RAW MATERIAL`, 155 rows of `BUY - CUST. SUPPLIED`, 13 `BUY - INTERPLNT TRNS`, and 779 rows of `PURCHASED (RAW)` values since these were too specific cases which could undermine model training and evaluation
- finally, the precondition that was applied to `customer_id` was `customer_id` > 0 since there was one customer with negative `id` equal to -99.

The following columns were removed and were not used for building the model:

- `customer_id` was removed to enable prediction for new customers with unknown `id`
- `manufacturing_location_code` was removed to enable prediction for new, unseen location codes
- the following date related columns were removed: `customer_first_invoice_date`, `born_on_date`, `invoice_date`, `price_last_modified_date_in_the_erp`, `order_date`
- `invoiced_price_tx` was removed since it is redundant for the case where gross margin is calculated using `invoiced_price` values
- `invoiced_price` and `cost_of_part` were removed due to the fact that we are building the model depending on gross margin and these two columns directly specify gross margin

- sales_channel_internal and sales_channel_external were removed because they are related to the company itself and bring no information that could help in making inferences about data segmentation
- sales_channel_grouping was removed because all of its values were missing
- top_customer_group was removed because we decided do merge this column with customer_region
- columns invoice_num, invoice_line_num, order_num, and order_line_num were discarded since they bring no information that would help the model with prediction
- columns material_cost_of_part, labor_cost_of_part, and overhead_cost_of_part were ignored because they do not have direct influence on gross margin
- item_code was removed to enable prediction for new items with new, unseen item codes
- product_group was removed due to its too fine granulation
- invoiced_qty_shipped was removed because we don not have that information when making prediction with model, we only have ordered_qty
- intercompany was removed since after filtering it was left with only one value
- lastly, number_of_unique_products_on_a_quote was also discarded.

Some columns were relabeled:

- As stated in the previous chapter, make_vs_buy column had high granularity in relation to its name (values in that column were relabeled to match either MAKE or BUY according to the Table 22)
- customer_region and top_customer_group were merged by considering STAR customers from Asia as a fourth customer region (due to their unique order history) and customer_region column was relabeled to match this idea (top_customer_group column was discarded).

At last, the following new variables were added as derived features for the model:

- customer_first_inovice_date was used to create a new feature which we call new_old_customer (old customers are the ones with year of customer_first_invoice_date < 2015 while new ones are from 2015 to this day)

- `ordered_qty` was turned into a categorical feature called `ordered_qty_bucket` by cutting it into following five buckets: $[1, 10]$, $(10, 100]$, $(100, 1000]$, $(1000, 10000]$, $(10000, inf)$.

To summarize, after the data preparation was finished, only the following columns were used for building our model:

1. `manufacturing_region` - categorical feature
2. `customer_region` after labeling of `STAR` customers - categorical feature
3. `make_vs_buy` after merging into two categories - categorical feature
4. `ordered_qty_bucket` - categorical feature
5. `product_family` - categorical feature
6. `customer_industry` - categorical feature
7. `new_old_customer` - categorical feature
8. `gm` - numerical feature

After all of the aforementioned imputing, filtering and discarding of starting data set rows, the model-ready version of the data set had 696357 rows which means that only 54% of the starting number of rows was kept.

Modelling

This chapter describes statistical models - proposed solution to dynamic deal scoring problem. Our main goal is to group similar invoice lines together in groups which will be referred to as *segments* in the rest of the document. We can claim that the solution is actually some sort of unsupervised segmentation of the data set. Final goal is to place every new invoice line in the segment where that invoice line fits bets. Once we know where invoice line should be placed, we can then give suggestions for the bounds of gross margin (GM) classes: A, B, C, D, F.

Baseline

First we will describe a tree-like baseline model. When trees are mentioned in data science, first things which come to mind are decision trees and random forests. However, one should bear in mind that our primary task is not supervised learning. We are not trying to predict some concrete label for a given input example. Therefore, we cannot use supervised algorithms for building trees such as *ID3* or *C4.5*. In the rest of this chapter, a custom algorithm inspired by the use of information entropy in *ID3* is proposed.

In order to explain the idea behind the algorithm we should first define what information entropy is. Let us introduce a discrete random variable X with possible outcomes: $x_1, x_2, x_3, ..., x_n$. We define the entropy of variable X as $H(X) := -\sum_{i=1}^{n} p(x_i) \log(p(x_i))$ where $p(x_i)$ represents the probability of outcome x_i. In our case that probability could be relative frequency of a specific feature value calculated from the data set. But, the question is what entropy $H(X)$ tells us? How can we use its value in order to build a decision tree? On the very high level, entropy is the measure of information which random variable X emits. For instance, let's say we have an unfair gambling dice which for some reason always lands on number one. This dice (that is a random variable) gives no information as we always know what is the outcome ($p(x_i) = 1 \implies H(X) = 0$). On the other hand, with completely fair dice we are uncertain of the outcome and that dice gives us much more surprise/information. One could also argue that entropy is bigger as the

distribution of the random variable X approaches uniform distribution. With this idea in mind we can build the decision tree by recursively splitting data set by the feature with the current biggest entropy. That feature will give us the most information and it will split the data set in a way which seems most fair for the segmentation. With the assumption that splits result in groups where similar invoices lines are clustered together, we can use the leafs of the tree as resulting segments of the given data set. Splitting ends when entropy of all features is zero or when there are no more features to split by or when the number of examples in some tree node drops below the hyperparameter N.

We can demonstrate the algorithm in action on the following example. Suppose we have a data set D with two features F_1 and F_2 and their corresponding values $F_1 = \{A, B, C\}$ and $F_2 = \{X, Y\}$. Next, suppose following distribution of examples by features:

- $|D(F_1 = A, F_2 = X)| = 20$
- $|D(F_1 = A, F_2 = Y)| = 80$
- $|D(F_1 = B, F_2 = X)| = 10$
- $|D(F_1 = B, F_2 = Y)| = 90$
- $|D(F_1 = C, F_2 = X)| = 0$
- $|D(F_1 = C, F_2 = Y)| = 10$

Then, we have:

- $|D(F_1 = A)| = 100$
- $|D(F_1 = B)| = 100$
- $|D(F_1 = C)| = 10$
- $|D(F_2 = X)| = 30$
- $|D(F_2 = Y)| = 180$

At last, we will fix $N = 15$. Algorithm starts from the root of the tree. First we determine which feature has the largest entropy and then we use that feature for splitting. In this case it is the feature F_1 (figure 50):

- $H(F_1) = -\frac{100}{210}\log(\frac{100}{210}) - \frac{100}{210}\log(\frac{100}{210}) - \frac{10}{210}\log(\frac{10}{210}) = 0.37$
- $H(F_2) = -\frac{30}{210}\log(\frac{30}{210}) - \frac{180}{210}\log(\frac{180}{210}) = 0.18$

Algorithm continues recursively for each child. At this point, we only have one feature so there is no need to calculate entropy. Each child gets split by feature F_2 expect for the child where $F_1 = C$. That child will become a leaf because its row count is 10 which is less than $N = 15$ (figure 51).

Figure 50: Tree after F_1 split.

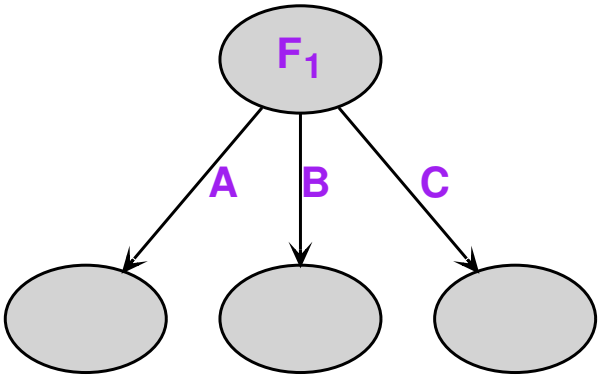
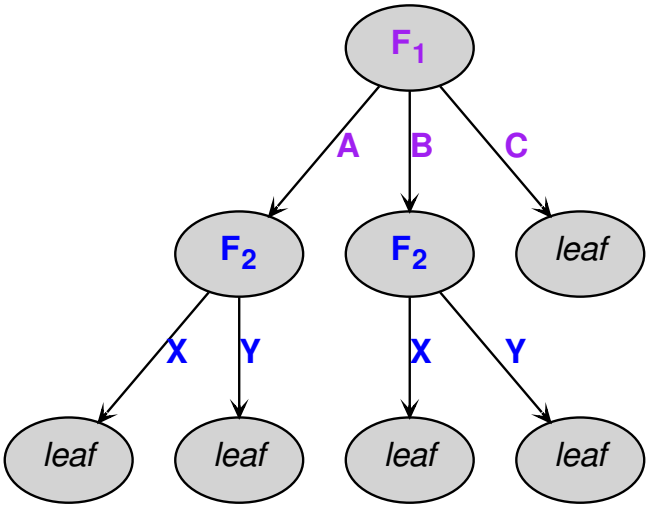


Figure 51: Tree after F_2 split.



After the tree is completely built, each of its leaves represents one segment of the data set. In our case, we ended up with the following segments:

- $D(F_1 = A, F_2 = X)$
- $D(F_1 = A, F_2 = Y)$
- $D(F_1 = B, F_2 = X)$
- $D(F_1 = B, F_2 = Y)$
- $D(F_1 = C)$

In order to obtain bounds of GM classes we will divide the data set into mentioned segments. Now, all we have to do is divide the segment into 5 bins such that each bin contains 20% of the total number of invoice lines in the segment. Such division will give us lower and upper bound for each of the GM classes: A, B, C, D, F. One might wonder why such subdivision in the segment was chosen. Surprisingly, the answer is rather easy to understand. Here we are trying to be completely unbiased and let the segment data itself to form the GM classes in a fair way by uniformly distributing invoice lines in GM classes.

Next, we showcase how this model can be used with our data set and for our dynamic deal scoring market segmentation problem. For instance, we can make substitutions $F_1 =$ `customer_region` and $F_2 =$ `intercompany`. Then, we have:

- $A =$ `Asia`
- $B =$ `Europe`
- $C =$ `North America`
- $X =$ `NO`
- $Y =$ `YES`

It should be noted how this choice of features is for demonstrational purposes only. Later we take the features that could really be used to get meaningful results. This choice leaves us with rather rough segmentation of the initial data set. Nevertheless, it is worth showing as it helps to understand the core idea behind this algorithm. Figure 52 shows segmentation of the filtered data set with hyperparameter $N = 250$.

In the end, main problems of this approach should be emphasized. First, this algorithm only works with discrete features. If continuous variables are to be used, they should be split into categories. Second,

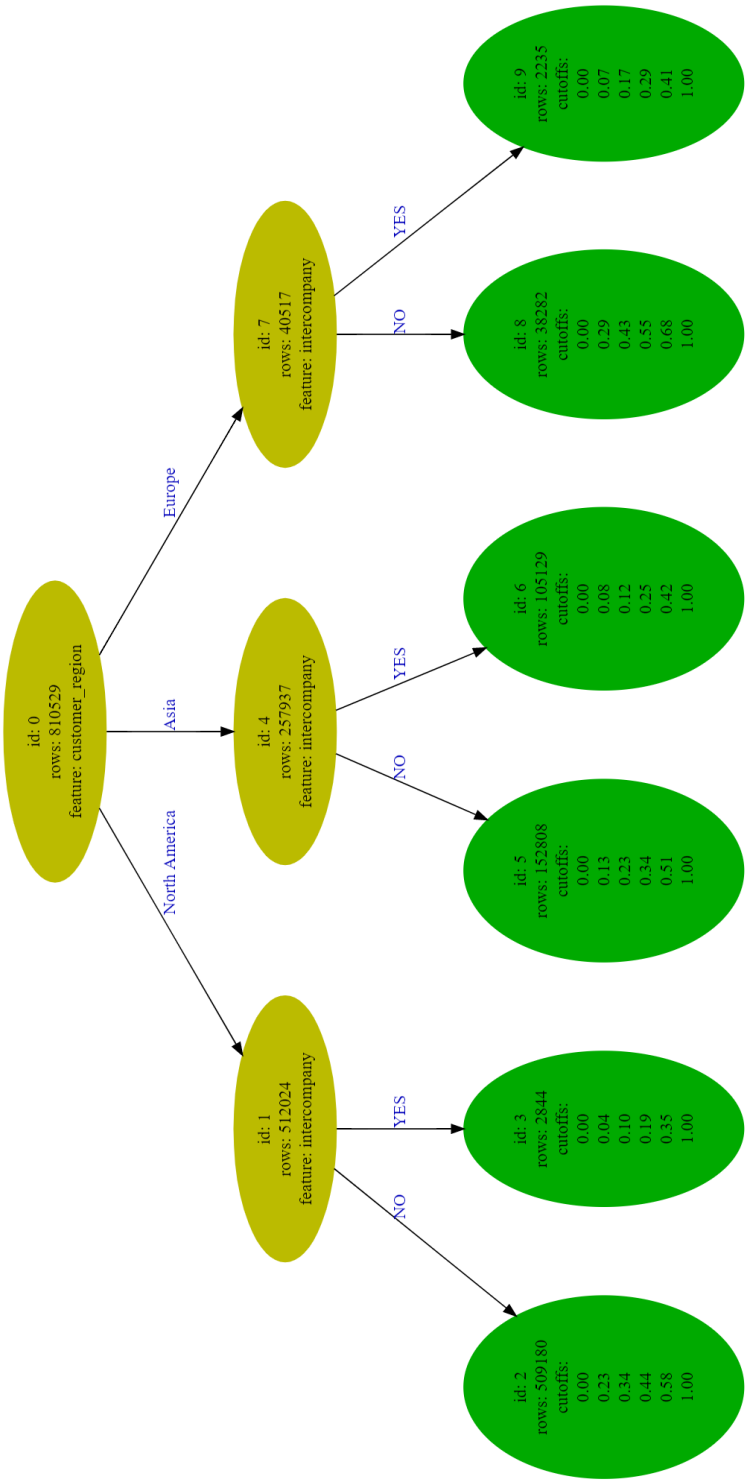


Figure 52: Example of a tree built using baseline algorithm.

GM (our main concern) is never taken into consideration. Splits occur based on the entropy of independent variables and nothing else. Third, hyperparameter N does not provide much control over the number of examples (rows) in the leaf nodes. For instance, even if $N = 10,000$, leaf nodes with very small rows count (< 100) may emerge in the resulting tree. These leaf nodes do not provide much information and should not be considered as segments. Rather than specifying when splitting of nodes ends, it would be much better to define the minimal number of examples in a leaf node. This way there is no possibility of getting unwanted leaf nodes in the tree.

CHAID tree segmentation

In this chapter, we will describe another similar approach for the segmentation of the given data set. Once again, we are dealing with a decision tree. However, this time we will not rely on entropy as a main metric for creating branches in the tree. Instead, we will use Chi-square statistic or F-test in order to detect relationship between variables. As a matter of fact, CHAID is an abbreviation for the *Chi-square Automatic Interaction Detector*.

First, we should explain what is the core idea behind this machine learning algorithm. As its name states, algorithm is detecting interaction between independent variables and target values and it does that by testing whether values of a concrete independent variable are uniformly distributed among target classes. Independent variable with the highest test score is chosen for the split in the next tree node. let's take a moment and describe why such procedure makes sense. Well, if independent variable values are equally distributed among target classes, then this variable cannot explain the distribution of the target variable. Such a variable is not biased at all toward some target classes and for that reason we cannot extract any valuable information from it. In other words, if such a variable is to be used in the exploitation phase of a model, its future predictions would not be very accurate since uniform distribution is not informative at all. Next, it is worth noting how CHAID can be used only if independent variables are discrete. However, dependent variable can be both discrete and continuous. In the case of a discrete dependent variable Chi-square test is used for interaction detection. In other scenario, when dependent variable is continuous, F-test (or Bartlett's or Levene's test) serves the same purpose.

For a better understanding let's explain CHAID on one example found online [18]. Table 24 shows data set for this example. It is rather simple - it considers the relationship between weather conditions and

[18] Sefik Ilkin Serengil. A step by step chaid decision tree example. https://sefiks.com/2020/03/18/a-step-by-step-chaid-decision-tree-example/, March 2020

decision for taking a trip in the nature. Independent variables are: *Outlook, Temperature, Humidity*, and *Wind* while the dependent variable is *Decision*.

Day	Outlook	Temperature	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Table 24: CHAID example - data set.

As before, we start from the root of the tree and first we have to determine which independent variable has the most impact on *Decision*. Since *Decision* is a discrete variable, we will use Chi-square test. Table 25 shows Chi-square values for all independent variables. Column *Expected* is calculated as *Total* / 2 while *Chi-square Yes* and *Chi-square No* values are obtained by using the following formula: $\sqrt{\frac{(y-y')^2}{y'}}$ where y is the actual and y' is the expected value.

Value	Yes	No	Total	Expected	Chi-square Yes	Chi-square No
Sunny	2	3	5	2.5	0.316	0.316
Overcast	4	0	4	2.0	1.414	1.414
Rain	3	2	5	2.5	0.316	0.316
Hot	2	2	4	2.0	0.000	0.000
Mild	4	2	6	3.0	0.577	0.577
Cool	3	1	4	2.0	0.707	0.707
High	3	4	7	3.5	0.267	0.267
Normal	6	1	7	3.5	1.336	1.336
Weak	5	2	7	3.5	0.802	0.802
Strong	3	3	6	3.0	0.000	0.000

Table 25: Chi-square values for CHAID root choice.

Summing all Chi-square values for each variable results in Table 26.

Variable	Chi-square value
Outlook	4.092
Temperature	2.569
Humidity	3.207
Wind	1.604

Table 26: Summed Chi-square values for CHAID root choice.

Now, we can observe that variable *Outlook* has the biggest impact on outcome *Decision* and for that reason we will choose it for the current branching in the tree. Figure 53 represents state of the CHAID tree after first step.

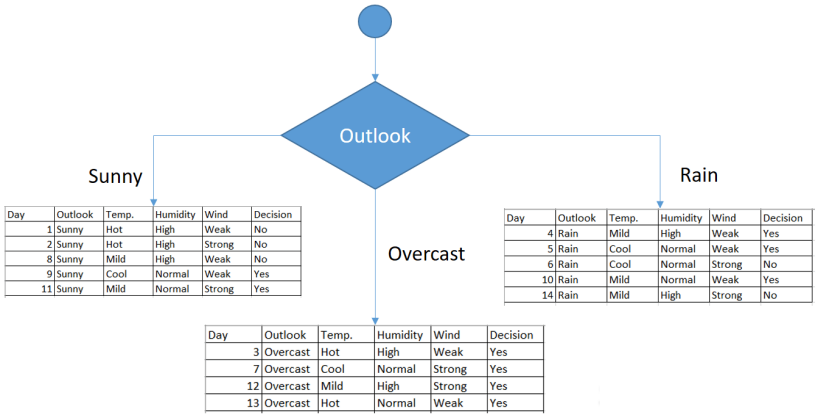


Figure 53: CHAID tree after the first step.

Algorithm continues recursively until there are no more variables to split by or when there is only one target value in a leaf node. Figure 54 shows final resulting tree. One could notice how variable *Temperature* is missing from the tree nodes. Why did this occur? Well, if we take a closer look on the data set segments in leaf nodes we can observe how there is only one target value (*Yes* or *No*) which means that no splitting is required as we already know the outcome.

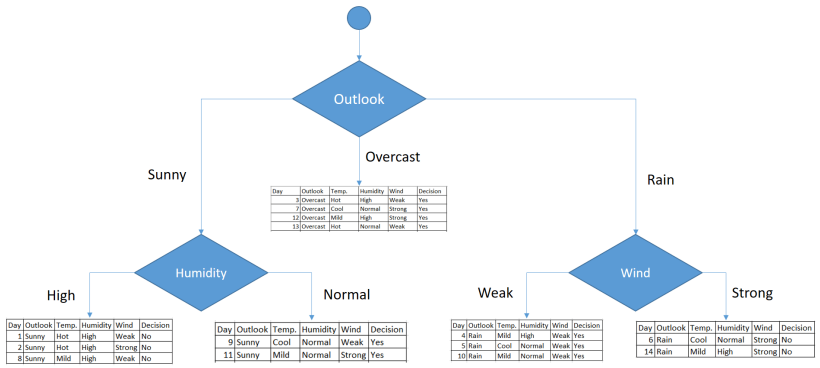


Figure 54: Final CHAID tree.

Before moving onto concrete solution to this problem, a couple of facts should be addressed. First, discrete target variable *Decision* was chosen exclusively for demonstration purposes. In our case target variable will be GM which is continuous. Second, up to this point, the proposed models were described without putting them in the context

of dynamic deal scoring. Our main intention for that was the desire to fully understand algorithms and principles before using them in a real world use case. Third and finally, the more careful reader would have noted how both models are decision trees. The answer to the question on why we explicitly chose trees for the solution to this problem is rather easy to grasp. We want our solution to be highly interpretable and decision trees allow us just that. With trees it is quite easy to follow the tree path and understand why certain segment of the data set was chosen for the prediction. Person left to interpret the output of the model does not have to understand the maths and statistics behind decision trees as the output is rather elegant and natural.

Decision Trees in Dynamic Deal Scoring

Now its time to discuss the use of described models in the context of dynamic deal scoring. As already mentioned, our main goal is the segmentation of the data set. With this thought, every leaf in a decision tree represents one segment of the data set. In this chapter we will mostly focus on model hyperparameters.

For baseline model we chose rather big $N = 1,000$ because of earlier mentioned problems with undesired size of segments. CHAID model is slightly complicated and has more hyperparameters[19]:

- `alpha_merge` $= 0.08$
- `max_depth` $= 3$
- `min_parent_node_size` $= 5,000$
- `min_child_node_size` $= 250$.

`alpha_merge` was set to a slightly higher value than default (0.05) in order to stimulate merges which then result in larger segments. For a `max_depth` 3 was chosen. Main motivation for this was interpretability. Trees with smaller depth are easier to understand. At last, `min_parent_node_size` and `min_child_node_size` were chosen empirically - focus was on getting somewhat larger segments with uniform distribution over GM classes.

Furthermore, instead of using just one monolithic CHAID model, we decided to build multiple models (CHAID trees) based on the `manufacturing_region` value. Beacuse of that, our CHAID model is actually a composition of 3 inner CHAID trees - one for each manufacturing region:

- `Asia`

[19] Mark Ramotowski. Chaid: A python implementation. https://github.com/Rambatino/CHAID#parameters, April 2021

- `Europe`
- `North America`.

Justification for such a decision is rather simple. We expect to achieve better results if we have separate models for distinct parts of a data set. However, one might wonder why manufacturing region was chosen in particular. Since the company has manufacturing regions on several continents, we developed a model for each of the regions so that every model can be tailored for manufacturing pattern specific for each of these regions.

What is more, as indicated in the Chapter Data preparation, we implemented two ways for imputing possible missing values in model exploitation phase. First, since CHAID library is transforming NA to a special value `'<missing>'`, branch with value `'<missing>'` could be chosen as a next step in tree traversal if some value is absent. Second method relies on the most frequent value in the current branching. For instance, if `customer_region` was missing in an input example and if there were 3 branches from the current node with the following number of rows in their corresponding child nodes:

- `customer_region = Asia` (rows = $1,000$)
- `customer_region = Europe` (rows = $3,000$)
- `customer_region = North America` (rows = 500)

then `Europe` would be chosen as it is the most frequent among them ($3,000 > 1,000 > 500$).

Finally, problem with missing GM bounds should be addressed. If there are not enough distinct GM values in the data set segment, then it is not possible to divide GM range into 5 classes such that each class contains roughly 20% of segment rows. Because of that, division can sometimes result in fewer than 5 classes. When making a prediction, such a result might be rather uninformative to the end user. We propose very simple yet quite effective solution to this problem. In order to illustrate the idea let's imagine we obtained a segment with only 4 bounds. For instance, $0.1, 0.2, 0.5, 0.7$. Since we want both upper and lower bound for each of the GM classes we need to somehow insert two more bounds. First we calculate the range between two consecutive bounds which results in:

- $0.2 - 0.1 = 0.1$
- $0.5 - 0.2 = 0.3$
- $0.7 - 0.5 = 0.2$

Then, we choose bounds which resulted in the widest range. In our case these are 0.2 and 0.5. Finally, we calculate their mean $\frac{0.2+0.5}{2} = 0.35$ and we insert the mean between them. This yields: $0.1, 0.2, 0.35, 0.5, 0.7$. Process is repeated recursively until there are exactly 6 values. For this example, final result would be: $0.1, 0.2, 0.35, 0.5, 0.6, 0.7$. It should be stated how this method works only if at least 2 bounds are present in the first place. However, it only makes sense to use it if 1 or at most 2 bounds are missing. Why do we have such a strong constraint? Well, this interpolation method might result in something extremely artificial if we were to use it when more bounds are missing.

Evaluation

At last, we should discuss model performance. Since we are not doing supervised learning, we cannot simply divide the data set into train and evaluation subsets and measure how the model performed on the evaluation subset. In our case there are no target values in the given data set. Because of that, we need to use alternate methods of model evaluation. For this problem we came up with two evaluation approaches which can be combined together. It is worth stating how both approaches use the traditional idea of a train-evaluation split with the assumption of distribution equality between the train and evaluation subset.

Proportional segmentation

The first idea is to observe how examples from the evaluation subset are distributed into obtained segments. If segmentation was done properly, we expect to observe same distribution of examples over segments for both train and evaluation subset. To clarify this idea, let's assume data set D with $1,000,000$ rows was split in two subsets: D_{train} with $800,000$ rows and $D_{evaluation}$ with $200,000$ rows. Furthermore, let's say that tree construction on subset D_{train} resulted in 4 segments:

- S_1 (rows = $150,000$, part = 18.75%)
- S_2 (rows = $200,000$, part = 25.00%)
- S_3 (rows = $100,000$, part = 12.50%)
- S_4 (rows = $350,000$, part = 43.75%)

If this segmentation really groups *similar* examples together then it is completely reasonable to expect the following:

- roughly $37,500$ examples from $D_{evaluation}$ ended in S_1
- roughly $50,000$ examples from $D_{evaluation}$ ended in S_2
- roughly $25,000$ examples from $D_{evaluation}$ ended in S_3
- roughly $87,500$ examples from $D_{evaluation}$ ended in S_4.

In order to get one scalar which represents how well the model performed, one could use mean square error or sum of absolute values of differences between parts (that is relative frequencies or percentages) on D_{train} and $D_{evaluation}$. For the sake of argument let's say Table 27 shows how examples from both train and evaluation data set are distributed among segments $S_i, i = 1, \ldots, 4$.

Segment	$\%(train)$	$\%(evaluation)$
S_1	18.75%	30.00%
S_2	25.00%	21.50%
S_3	12.50%	12.25%
S_4	43.75%	36.25%

Table 27: Train-evaluation segment relative frequencies.

In case we decide to use mean of absolute values, scalar which describes model quality would be:

$$|0.1875 - 0.3000| + |0.2500 - 0.2150| + |0.1250 - 0.1225| + |0.4375 - 0.3625| = 0.225$$

Observe how this summation should be divided with 4 (we have exactly 4 segments) in order to normalize the scalar value. This makes comparison between models with different number of output segments possible. Finally, we have: $0.225/4 = 0.05625$.

Uniform distribution over GM classes

Similar as with previous approach, the idea is to first divide given data set into train and evaluation subsets. After tree was fit on the train subset, segments are obtained. In order to evaluate how acceptable segmentation was, we could test whether examples from evaluation subset were uniformly distributed over GM classes in each segment. Reader might be puzzled with the choice of uniform distribution. To answer *why uniformly*, first we need to recall how GM bounds were calculated in the first place. GM range is split in such a way that 20% of examples fall in each class. In other words, splits are a direct consequence of equally distributing train examples in 5 distinct classes. Now, if the segmentation was correct, it is fair to assume how examples from the evaluation subset should also form uniform distribution over GM classes for each segment. If one concrete number (scalar) is needed as a measure of a model quality we could use the same metrics mentioned in the previous section, only this time we would look at the difference

between actual and uniform distribution. For example, let's pretend there are only two output segments S_1 and S_2. Furthermore, let's say that evaluation examples were distributed over GM classes with the following percentages (or relative frequencies):

- $S_1 \implies (0.2, 0.4, 0.2, 0.1, 0.1)$
- $S_2 \implies (0.1, 0.1, 0.1, 0.5, 0.2)$

If we were to use mean squared error, value for this example would be:

$$(0.2-0.2)^2 + (0.4-0.2)^2 + (0.2-0.2)^2 + (0.1-0.2)^2 + (0.1-0.2)^2 + (0.1-0.2)^2 + (0.1-0.2)^2 + (0.1-0.2)^2 + (0.5-0.2)^2 + (0.2-0.2)^2 = 0.18$$

Once again, take a note how whole sum expression should be divided with 2 as we have 2 segments. This allows us to compare models with different number of segments. After division, the final result is: $0.18/2 = 0.09$.

Decision Trees Evaluation

Ultimately, there is only one thing left to discuss and that is evaluation of two chosen models: baseline and CHAID. For the evaluation, both previously described metrics were used. Evaluation was conducted using typical shuffled k-fold cross-validation with number of data set folds $k = 5$. Table 28 shows baseline results, while table 29 shows CHAID results.

Fold #	Proportion error	Uniform GM error
1	0.006747	0.091116
2	0.006616	0.083642
3	0.006878	0.090195
4	0.007404	0.097978
5	0.006655	0.084824

Table 28: Baseline k-fold results.

Fold #	Proportion error	Uniform GM error
1	0.022862	0.002096
2	0.024781	0.005303
3	0.040810	0.007831
4	0.046603	0.007384
5	0.022905	0.005237

Table 29: CHAID k-fold results.

Calculated mean and sample standard deviation for both models and for both metrics is shown in table 30.

	Baseline	CHAID
mean (proportion)	0.006860	0.031592
std. (proportion)	0.000320	0.011273
mean (uniform GM)	0.089551	0.005570
std. (uniform GM)	0.005725	0.002272

Table 30: k-fold results: mean and sample standard deviation.

By just looking at the results, one might conclude how choice for the better model cannot be determined based on the given numbers. If we use *proportion error*, then baseline is better. However, if *uniform GM* is chosen then CHAID yields better results. For that reason resulting segments were additionally inspected by hand. Even though both models performed well, baseline had a significant problem whereas CHAID did not. Baseline yielded approximately 500 segments, 200 of which had less than 100 train rows. There are just far too many fine-grained problematic segments. Moreover, these segments often did not have enough distinct GM values in order to obtain 5 GM classes. On the other hand, CHAID resulted in roughly 80 larger segments. Very rarely, some segments were missing 1 GM class bound which was inserted with already described interpolation method.

Finally, when validation is considered we should also measure and discuss the model speed. Table 31 shows average times for training, evaluation and prediction (without taking network latency into account). Time was measured on a personal computer with Intel® Core™ i7-6820HQ and 16 GB of RAM. Evaluation is slower than training because of evaluation implementation in which each example from evaluation subset is predicted separately in a for loop pass.

	Baseline	CHAID
Training	9.156 s	12.883 s
Evaluation	51.613 s	75.969 s
Prediction (1 line)	0.371 ms	0.545 ms

Table 31: Average training, evaluation, and prediction times for baseline and CHAID.

Bibliography

Walter Baker, Michael Kiermaier, Paul Roche, and Veronika Vyushina. Advanced analytics in software pricing: Enabling sales to price with confidence. https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/advanced-analytics-in-software-pricing-enabling-sales-to-price-with-confidence, June 2018.

Daniel Liberto. Quantity discount. https://www.investopedia.com/terms/q/quantity-discount.asp, March 2021.

Mark Ramotowski. Chaid: A python implementation. https://github.com/Rambatino/CHAID#parameters, April 2021.

Sefik Ilkin Serengil. A step by step chaid decision tree example. https://sefiks.com/2020/03/18/a-step-by-step-chaid-decision-tree-example/, March 2020.

Wikipedia. Blanket order. https://en.wikipedia.org/wiki/Blanket_order, February 2021.