LAPIN AMK
Lapland University of Applied Sciences

Leveraging Machine Learning Insights to Optimize Marketing Strategies

Portfolio-based thesis

Emiliia Zemskova

Thesis
Degree Programme in Machine Learning and Data Engineering
Bachelor of Engineering

YEAR 2024

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

Name of Degree Programme
Degree

Author	Emiliia Zemskova	Year	2024
Supervisor	Kenneth Karlsson		
Commissioned by	Name of the Commissioner		
Title of Thesis	Leveraging Machine Learning Insights to Optimize Marketing Strategies		
Number of pages	44		

This thesis focused on the use of machine learning techniques, specifically logistic regression, to optimize marketing strategies for the Business-to-Consumer (B2C) sector. The study utilized a dataset from a Portuguese banking institution's direct marketing campaigns, which provided insights into customer responses to financial product offers. The main objective was to apply data-driven methods to enhance marketing outcomes, including customer segmentation, campaign timing, and personalized communication.

A logistic regression model was developed to classify customer behavior, with a focus on predicting whether clients would subscribe to a bank term deposit. Through this analysis, the thesis explored how machine learning can help marketers better understand customer preferences and tailor their strategies accordingly. One key finding was the importance of using OneHotEncoder to handle categorical data effectively, avoiding bias in model predictions. Additionally, the thesis examined the potential for optimizing marketing efforts by identifying the best times to contact customers based on historical engagement data.

The results of the logistic regression model indicated some success in predicting customer behavior, though challenges such as multicollinearity and model accuracy were noted. The conclusion suggests that further improvements could be made by exploring more advanced machine learning techniques. Overall, the thesis demonstrates the value of integrating machine learning insights into marketing strategies to drive better customer engagement and conversion rates.

Key words machine learning, logistic regression, marketing strategies, B2C, customer segmentation, data-driven marketing

CONTENTS

1 INTRODUCTION

In the modern business landscape, data-driven decision-making has become a cornerstone of successful marketing strategies. With the increasing availability of customer data and advances in machine learning, businesses now have the opportunity to better understand consumer behaviour and tailor their marketing efforts to maximize engagement and conversion. This thesis explores the application of machine learning, specifically logistic regression, to optimize marketing strategies using customer data.

The goal of this thesis is to analyze how machine learning techniques can be leveraged to provide deeper insights into customer behavior, which in turn can inform more effective and personalized marketing strategies. In particular, the focus is on the Business-to-Consumer (B2C) sector, where personalization and targeting are key factors in building strong relationships with customers. The logistic regression model is employed to classify customer responses to marketing campaigns, allowing for the identification of patterns and trends that can be used to optimize future marketing efforts.

The dataset used in this thesis originates from a banking marketing campaign, providing valuable information on customer interactions and responses. By applying machine learning techniques to this dataset, the thesis aims to uncover actionable insights that can enhance the efficiency of marketing strategies, particularly in terms of customer segmentation, campaign timing, and personalized messaging. Additionally, the thesis draws on my professional experience in B2C marketing, combining theoretical machine learning concepts with practical marketing applications.

2 BACKGROUND OF RESEARCH

2.1 Origins of the dataset

During the course of studies in Machine Learning and Data Engineering, the "Introduction to Machine Learning Methods" course was undertaken, taught by Senior Lecturer Tuomas Valtanen. A foundational understanding of various machine learning techniques was provided in the course, and these techniques were applied in the development of a comprehensive portfolio. The portfolio, showcasing a range of projects and analyses, is accessible on a personal website and GitHub repository. Among the projects developed, this thesis will primarily focus on one significant work: "Logistic Regression Using Banking Marketing Data." A detailed examination of the model development process will be presented, along with a thorough analysis of the outcomes derived from data analytics and logistic regression, and an exploration of potential marketing strategies informed by these insights.

To begin with, it is crucial to discuss the dataset employed in this study. The dataset originates from the UC Irvine Machine Learning Repository (Aha, 1987), a well-regarded collection of databases, domain theories, and data generators widely used in the machine learning community. Specifically, the data pertains to direct marketing campaigns carried out by a Portuguese banking institution. These campaigns were conducted via telephone, where multiple contacts with the same client were often necessary to determine whether the client would subscribe to a particular financial product, specifically a bank term deposit. The response variable in this context is binary, indicating whether the product was subscribed ('yes') or not ('no'). Understanding the nature and origins of this dataset is fundamental to comprehending the subsequent analyses and the insights drawn from the logistic regression model developed as part of this study.

With an understanding of Logistic Regression and its role in classification tasks, the necessary data engineering and model development phases were undertaken as integral components of the typical machine learning (ML) model life cycle (Figure 1). The complete life cycle of an ML model typically begins

with acquiring data, often from multiple disparate sources. This is followed by the data processing stage, which involves a series of crucial steps such as data cleaning, formatting, and quality checks, as well as feature transformations and feature selection. All of these tasks were meticulously carried out in this project and will be elaborated on in subsequent sections of this thesis.

The next phase in the ML life cycle is model training, where the selected algorithm is applied to the processed data. This stage encompasses various critical activities, including model evaluation, hyperparameter optimization, and versioning. These activities are designed to refine the model's accuracy and performance, ensuring it is well-tuned and capable of making reliable predictions.

The final stage of the ML life cycle is deployment, where the model is integrated into a production environment. This stage includes considerations such as workload type, monitoring, and continuous learning to ensure the model remains effective over time as it encounters new data. However, in this specific project, the deployment stage was deemed unnecessary and thus was not implemented. The focus of the project remained on data engineering and model development, as these stages were sufficient to achieve the research objectives without requiring the complexities of deployment.

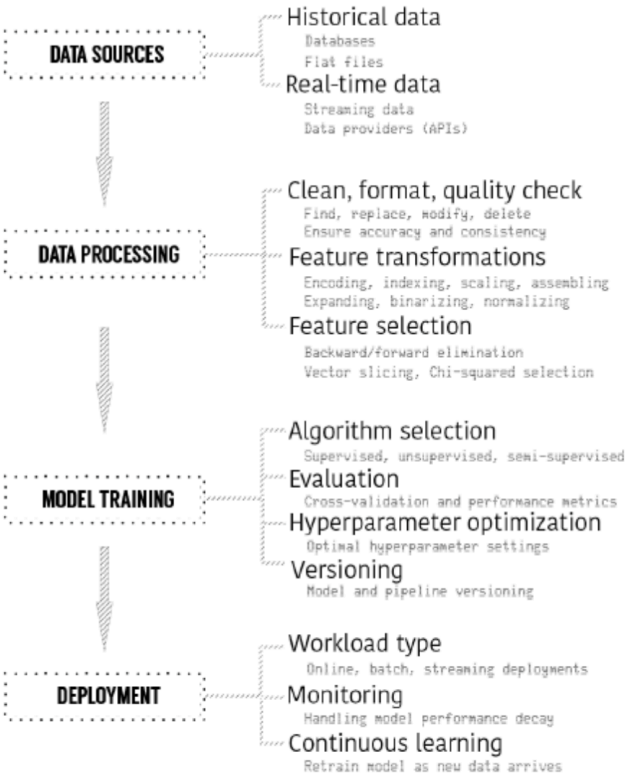


Figure 1. A typical ML model life cycle. Sibanjan, 2018.

As discussed by Rashmi (2021), Logistic Regression is a specific variation of Classification, a supervised learning process where the objective is to predict the class or category of given data points. In classification tasks, these classes are often referred to as targets, labels, or categories, and the machine learning model learns to assign new data points to these predefined categories based on patterns identified from the training data. For instance, in the context of email filtering, a classification model can be trained to differentiate between “spam” and “not spam” emails by analyzing existing data and then applying this learned knowledge to categorize incoming emails accordingly.

In the context of this thesis, the Logistic Regression model was employed as a classification tool to predict whether a client would subscribe to a term deposit, represented by the binary target variable y . This variable indicates the two possible outcomes: whether a client subscribes to the term deposit (‘yes’) or does not (‘no’). The Logistic Regression model, through the analysis of various predictor variables, attempts to classify new observations—i.e., potential clients—based on their likelihood of subscribing to the term deposit. This

approach is fundamentally aligned with the principles of supervised learning, where the model is trained on historical data with known outcomes and then used to make predictions on new, unseen data.

2.2 Tools and environments

For the data analysis, Jupyter Notebook was utilized, a cutting-edge, web-based interactive development environment that facilitates the creation and execution of notebooks, code, and data visualizations. I opted for the web version of Jupyter Notebook, which provided the advantage of not requiring local installation on my computer, thereby allowing for seamless access and usage across different devices. Within this JupyterLab environment, I installed all the necessary libraries and dependencies required to process and analyze the dataset effectively.

When it came to developing the Logistic Regression model, I chose to use my personal computer. Given the relatively lightweight nature of the model, my computer's specifications, including an 8 GB CPU, were more than sufficient to handle the computational demands. This setup enabled efficient model development and testing without encountering performance bottlenecks.

3 DETAILED DESCRIPTION OF THE LOGISTIC REGRESSION MODEL DEVELOPMENT

In this section of the thesis, a detailed explanation of the implemented code will be provided, accompanied by relevant screenshots. These visual aids will serve to clearly illustrate the various steps involved in data modification and the overall workflow. By breaking down the code and explaining each component, I aim to offer a comprehensive understanding of the processes and decisions made during the data engineering and model development phases. The screenshots will complement this narrative by providing a visual representation of the code execution, thereby enhancing the clarity and accessibility of the technical details discussed.

3.1 Data Engineering

As part of developing the Logistic Regression model, I undertook a comprehensive data engineering process, which is a critical step in ensuring the creation of an effective and accurate model. Data engineering is essential because, without clean and well-structured data, the reliability and performance of any machine learning model, including Logistic Regression, would be significantly compromised.

The initial step in this process was to download and import all the necessary libraries (Figure 2). These libraries are indispensable tools for data manipulation, visualization, and model development. I utilized several standard libraries, such as Pandas, NumPy, Matplotlib, and Seaborn, to handle and visualize the data efficiently. Pandas and NumPy provided robust data structures and mathematical functions that facilitated the manipulation and cleaning of the dataset. Matplotlib and Seaborn were instrumental in visualizing data patterns and distributions, which are crucial for understanding the underlying characteristics of the dataset.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# pip install scikit-learn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score

# pipeline features
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import FunctionTransformer
```

Figure 2. Necessary libraries for data proceeding

In addition to these foundational libraries, specific machine learning libraries were employed, such as scikit-learn (sklearn), a comprehensive toolset for

building and evaluating machine learning models. Within scikit-learn, modules like `train_test_split` were utilized to partition the data into training and testing sets, ensuring rigorous evaluation of the model before deployment. These tools were pivotal in the successful implementation of the Logistic Regression model, allowing for systematic data preprocessing and reliable model development.

The next step in the data engineering process is reading the data into the environment (Figure 3). The dataset I am working with is in CSV (Comma-Separated Values) format, which is a common and convenient format for storing structured data. As previously mentioned, I obtained this dataset from the open-source platform Kaggle.com. To streamline the workflow and maintain an organized project structure, I uploaded the dataset to the same directory as my Jupyter Notebook. This approach not only simplifies file management but also avoids the need to write lengthy file paths, thereby keeping the code clean and easy to navigate.

```
In [2]: # uploading dataset to juputer notebook
        df = pd.read_csv("bank.csv")
```

Figure 3. Uploading data to the environment

By storing the dataset in the same folder as the Jupyter Notebook, I ensured that the data could be accessed efficiently and without unnecessary complications. This method of organizing files is particularly beneficial when dealing with multiple datasets or when collaboration with others may require clear and straightforward file structures.

When uploading data into Jupyter Notebook, one useful method for an initial examination of the dataset is to use the ***"df.head()"*** function. This command generates a table displaying the first few rows of the dataset, providing an immediate snapshot of the columns and their contents. This preliminary view is crucial for gaining a quick understanding of the dataset's structure and the types of data it contains. For instance, in Figure 4, we present the initial reading of the bank marketing data.

In [3]:

```
# let's read data
df.head()

# I checked from this link https://archive.ics.uci.edu/dataset/222/bank+marketing
# what some of columns mean (their definition and values)

# default --> has credit in default? (binary: "yes","no")
# duration --> last contact duration, in seconds (numeric)
# campaign --> number of contacts performed during this campaign
# and for this client (numeric, includes last contact)
# pdays --> number of days that passed by after the client was last contacted
# from a previous campaign (numeric, -1 means client was not previously contacted)
# previous --> number of contacts performed before this campaign and for this client (numeric)
# poutcome --> outcome of the previous marketing campaign (categorical: "unknown","other","failure","success"
```

Out[3]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	p
0	59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	1042	1	-1	0	
1	56	admin.	married	secondary	no	45	no	no	unknown	5	may	1467	1	-1	0	
2	41	technician	married	secondary	no	1270	yes	no	unknown	5	may	1389	1	-1	0	
3	55	services	married	secondary	no	2476	yes	no	unknown	5	may	579	1	-1	0	
4	54	admin.	married	tertiary	no	184	no	no	unknown	5	may	673	2	-1	0	

Figure 4. Reading data

From this initial inspection, it is evident that the dataset comprises various columns, each representing different attributes or features of the data. Some columns are essential for the analysis, while others may be irrelevant or redundant, making them candidates for removal in the subsequent data cleaning phase. Moreover, this stage of the process is an ideal opportunity to begin formulating a plan for data cleaning and modification. Given that machine learning models, such as Logistic Regression, typically require numeric input, it is imperative to ensure that the dataset is both clean and appropriately formatted. This involves converting categorical variables into numerical representations and filtering out any non-essential data to optimize the model's performance.

The subsequent step in reading the data involves examining the summary statistics, such as the maximum, minimum, mean, and other descriptive metrics (as illustrated in Figure 5). This analysis is crucial for identifying potential outliers and assessing the overall quality of the dataset. By examining these statistical values, we can determine if any data points deviate significantly from the norm, which may indicate the presence of outliers that could skew the model's results. Additionally, these metrics allow us to evaluate the extent of data loss that may have occurred during the data engineering process.

In [4]:
```
# let see how many values we have
# to compare them after the cleaning part
df.describe()
```

Out[4]:

	age	balance	day	duration	campaign	pdays	previous
count	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000
mean	41.231948	1528.538524	15.658036	371.993818	2.508421	51.330407	0.832557
std	11.913369	3225.413326	8.420740	347.128386	2.722077	108.758282	2.292007
min	18.000000	-6847.000000	1.000000	2.000000	1.000000	-1.000000	0.000000
25%	32.000000	122.000000	8.000000	138.000000	1.000000	-1.000000	0.000000
50%	39.000000	550.000000	15.000000	255.000000	2.000000	-1.000000	0.000000
75%	49.000000	1708.000000	22.000000	496.000000	3.000000	20.750000	1.000000
max	95.000000	81204.000000	31.000000	3881.000000	63.000000	854.000000	58.000000

Figure 5. The number of variables

If we observe substantial data loss, it becomes necessary to consider alternative strategies to mitigate this issue. One approach could involve refining the data cleaning methods to minimize data loss. Another potential solution is to augment the dataset through additional data collection methods. For instance, web scraping could be employed—albeit strictly for educational purposes—to gather supplementary data. According to the Cambridge Dictionary, web scraping is defined as the process of extracting information from a website to create a dataset. (Cambridge, 2024) This technique can be particularly useful when relevant data is scarce or incomplete, allowing for the integration of additional sources to enhance the dataset's comprehensiveness. Alternatively, combining multiple datasets could also be considered to enrich the dataset and ensure that the Logistic Regression model is trained on a robust and representative sample.

After completing the initial data analysis, I developed a detailed plan for how I would proceed with data cleaning and preprocessing. This plan is outlined in my Jupyter Notebook, as shown in Figure 6. Creating such a plan within the notebook is highly advantageous for several reasons. First, it allows for a structured approach to data cleaning, enabling me to work on the data in intervals while maintaining a clear understanding of the progress made. This means that even after taking breaks or working on other tasks, I can easily

return to the data cleaning process, knowing exactly where I left off and what steps need to be taken next.

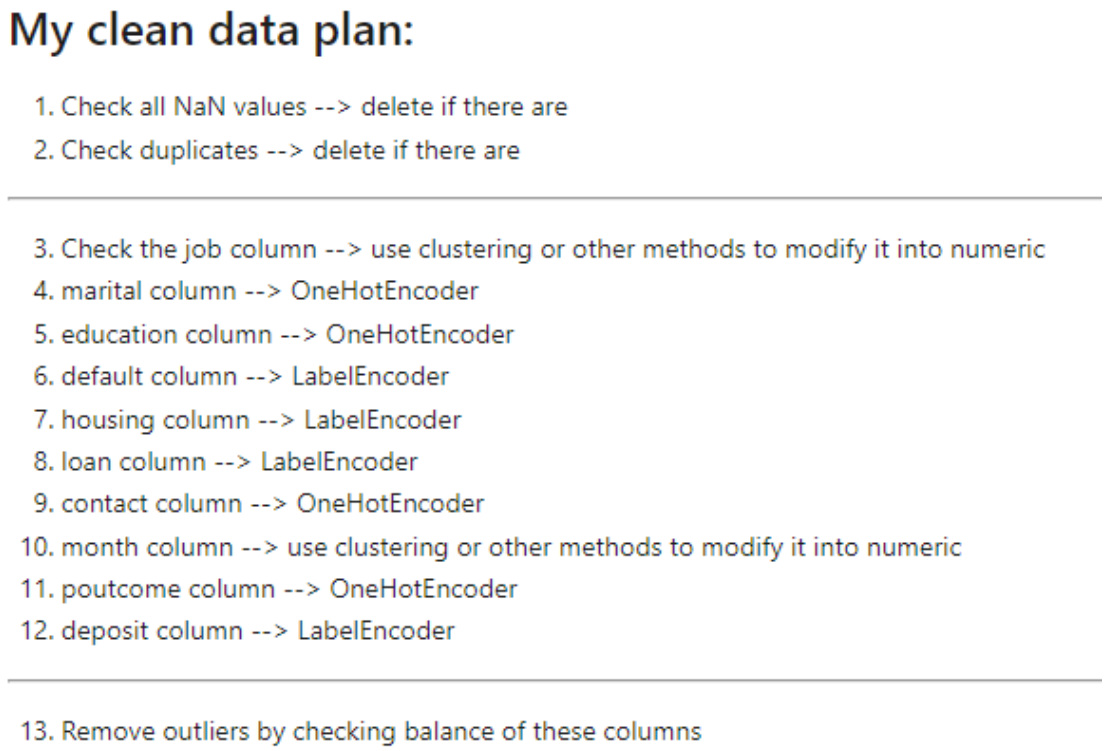


Figure 6. The plan of cleaning data

Additionally, having a predefined data cleaning plan provides a clear roadmap that enhances the overall organization of the project. It ensures that all necessary steps are systematically followed, reducing the likelihood of errors or omissions. Furthermore, this plan serves as a valuable guide for others who may review or replicate the work, offering them a transparent view of the data cleaning process and the rationale behind each step. By outlining the plan in advance, it also helps to clarify the structure of the work for the viewers, making it easier for them to follow the progression from raw data to a cleaned and ready-for-analysis dataset.

The subsequent two steps in the data modification process are standard practices in almost any data preprocessing workflow. The first step involves using the ***"df.isna().sum()"*** function, which is employed to identify any missing values, often referred to as NaN (Not a Number) values, within the dataset. This function provides a count of missing values across each column, allowing us to

quickly assess whether any data is incomplete. If the output in the Jupyter Notebook indicates that there are missing values in any column, it is crucial to address them to ensure data integrity. Common strategies for handling missing values include deleting the affected rows or columns, or imputing the missing values with appropriate substitutes, such as the mean or median of the column.

The second step, as illustrated in Figure 7, involves the use of the ***"df.duplicated().sum()"*** function. This line of code is designed to check for duplicate entries within the dataset. Duplicates can skew the results of a model by over-representing certain data points, so it is essential to identify and remove them to maintain the accuracy of the analysis. In this particular dataset, the output revealed that there were no duplicate entries, eliminating the need to drop any rows. By systematically applying these checks, we ensure that the dataset is both complete and free from redundant data, which are critical steps in preparing the data for effective model training.

```
In [5]:
# 1. Check all NaN values --> delete if there are
df.isna().sum()

# suprisingly, zero :)
```

```
Out[5]:
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
deposit      0
dtype: int64
```

```
In [6]:
# 2. Check duplicates --> delete if there are
# let's check whether we have duplicates
# or one more surprise awaiting us :)
df.duplicated().sum()

# yes, life is full of surprises :)
```

```
Out[6]:
0
```

Figure 7. Check of Nan values and duplicates

To proceed with the development of a classic machine learning model, it is essential to transform all string data into numerical values, as computers process and understand data in numerical form. In this project, I utilized the LabelEncoder from the Scikit-learn library to convert categorical text data into numeric codes, as illustrated in Figure 8. This process ensures that all the textual data is encoded into numerical values, making it suitable for input into the machine learning model.

In [7]:

```
# 3. Check the job column --> use clustering or other methods to modify it into numeric

# type of job (categorical: 'admin.','blue-collar','entrepreneur',
# 'housemaid','management','retired','self-employed','services','student',
# 'technician','unemployed','unknown')

# overwriting the 'job' column with the encoded values

# the package that needed to be installed
# in order to modify data into numeric
from sklearn.preprocessing import LabelEncoder

# initialize LabelEncoder
label_encoder = LabelEncoder()

# fit LabelEncoder and transform 'job' column
df['job'] = label_encoder.fit_transform(df['job'])

# Print the mapping of encoded values to original categories
# I will use these values for tester row and for GUI
print("Encoded values:")
for category, encoded_value in zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)):
    print(f"{category}: {encoded_value}")
```

```
Encoded values:
admin.: 0
blue-collar: 1
entrepreneur: 2
housemaid: 3
management: 4
retired: 5
self-employed: 6
services: 7
student: 8
technician: 9
unemployed: 10
unknown: 11
```

Figure 8. LabelEncinder

For instance, the LabelEncoder transformed the job categories into the following encoded values:

- admin.: 0
- blue-collar: 1
- entrepreneur: 2
- housemaid: 3
- management: 4
- retired: 5
- self-employed: 6
- services: 7

- student: 8
- technician: 9
- unemployed: 10
- unknown: 11

According to the official documentation, LabelEncoder is a tool within the Scikit-learn library that encodes categorical target labels with values ranging between 0 and the number of unique classes minus one (Scikit-learn developers, 2024). This encoding is crucial because it allows the model to process categorical data effectively, treating each category as a distinct numerical value.

In [7]:
```
# 3. Check the job column --> use clustering or other methods to modify it into numeric

# type of job (categorical: 'admin.','blue-collar','entrepreneur',
# 'housemaid','management','retired','self-employed','services','student',
# 'technician','unemployed','unknown')

# overwriting the 'job' column with the encoded values

# the package that needed to be installed
# in order to modify data into numeric
from sklearn.preprocessing import LabelEncoder

# initialize LabelEncoder
label_encoder = LabelEncoder()

# fit LabelEncoder and transform 'job' column
df['job'] = label_encoder.fit_transform(df['job'])

# Print the mapping of encoded values to original categories
# I will use these values for tester row and for GUI
print("Encoded values:")
for category, encoded_value in zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)):
    print(f"{category}: {encoded_value}")
```

```
Encoded values:
admin.: 0
blue-collar: 1
entrepreneur: 2
housemaid: 3
management: 4
retired: 5
self-employed: 6
services: 7
student: 8
technician: 9
unemployed: 10
unknown: 11
```

Figure 8. Modyfing the job column into numeric

In retrospect, the decision to employ this method during my work in data engineering was indeed inappropriate. The categories in question—such as "admin job" and "student"—are not inherently comparable. For instance, an administrative position typically holds a higher hierarchical or professional standing than that of a student. This disparity in categorical significance renders them unsuitable for direct comparison using techniques like LabelEncoder, which assigns arbitrary numerical values to categories without considering their

relative importance. Therefore, it was necessary to utilize the OneHotEncoder, as it treats each category independently, ensuring a more accurate representation of the data. This approach eliminates any unintended bias or misinterpretation that could arise from assigning ordered numerical values to categories with no intrinsic ordinal relationship.

Similar modifications were applied to other categorical columns in the dataset, including the ***marital, education, contact*** and ***poutcome*** columns, ensuring that all relevant text data was properly encoded. These transformations are a fundamental step in preparing the data for machine learning, enabling the model to interpret and analyze the data correctly.

For some columns that are string such as ***default, housing, loan*** and ***deposit*** I used LabelEncoder instead of OneHotEncoder becuase it was able to transform the data to 0 and 1. The process works that if there are only two options for a variables in this column, the LabelEncoder gives them labels and transform to 0 and 1. on the Figure 9 you can see that we can use a list of variables at the same time in order to transform them.

In the process of data preprocessing, particularly for categorical columns containing string data, such as ***default, housing, loan*** and ***deposit*** I opted to use the LabelEncoder instead of the OneHotEncoder. The choice of LabelEncoder was driven by its efficiency in converting categorical data with binary outcomes directly into numerical values, specifically 0 and 1. This approach is particularly advantageous when dealing with variables that have only two possible categories. The LabelEncoder assigns a unique label to each category, transforming the data into a binary format.

As depicted in Figure 9, this method can be applied to multiple variables simultaneously, streamlining the data transformation process. The use of LabelEncoder not only simplifies the encoding process but also optimizes memory usage and computational efficiency, especially when dealing with a large dataset. By converting binary categorical variables to a numerical format, the data becomes more suitable for subsequent machine learning algorithms that require numerical input.

In [15]:

```
# All columns that need the same method can be done in one step
# 6. default column --> LabelEncoder
# 7. housing column --> LabelEncoder
# 8. loan column --> LabelEncoder
# 12. deposit column --> LabelEncoder

# this just converts the value of column to 0 or 1
# factorize in pandas works too, but only one column at a time
from sklearn.preprocessing import LabelEncoder
variables = ['default', 'housing', 'loan', 'deposit']
encoder = LabelEncoder()
df[variables] = df[variables].apply(encoder.fit_transform)
```

Figure 9. Modyfing the string data to numeric

In order to transform the categorical data in the ***month*** column into a numerical format, I employed a straightforward mapping technique. Specifically, I created a dictionary that associates each month with its corresponding numeric representation, where 'jan' is mapped to 1, 'feb' to 2, and so forth, up to 'dec' mapped to 12 (Figure 10). This dictionary was then applied to the ***month*** column using the map function, converting the categorical month names into their respective numeric equivalents.

This approach was selected to avoid the need for creating additional columns, which would have been necessary if using techniques such as one-hot encoding. By directly converting the categorical values into numeric format, this method preserves the chronological order of the months while maintaining the simplicity of the dataset structure. Additionally, this method ensures that the data remains compatible with machine learning models that require numerical input without inflating the dimensionality of the dataset.

In [13]:

```
# 10. month column --> use clustering or other methods to modify it into numeric
# Define a dictionary mapping each month to its numeric representation
month_to_number = {
    'jan': 1,
    'feb': 2,
    'mar': 3,
    'apr': 4,
    'may': 5,
    'jun': 6,
    'jul': 7,
    'aug': 8,
    'sep': 9,
    'oct': 10,
    'nov': 11,
    'dec': 12
}

# Map the 'month' column to its corresponding numeric representation
df['month'] = df['month'].map(month_to_number)

# I used this method in order to avoid creating additional columns
```

Figure 10. Mapping technique for the month column

3.2 Balancing of the data

The subsequent crucial step in data engineering involves the removal of outliers, a process essential for ensuring the accuracy and reliability of the model's results and predictions. Outliers are defined as data points that deviate significantly from the majority of the data and can potentially skew the model's performance if not addressed appropriately. As highlighted by Verma (Verma, 2019), outliers are variables that appear unusual or inconsistent within the context of specific datasets.

Given the importance of maintaining the integrity of the dataset, I initiated the process of outlier removal once the entire dataset was transformed into a numeric format, as shown in Figure 11. This step was followed by balancing the data, which is necessary to ensure that the model is not biased toward any particular class or outcome. By addressing outliers early in the data preprocessing pipeline, I aimed to enhance the robustness and generalizability of the model, ultimately leading to more accurate and reliable predictions.

```
In [17]: # 13. Remove outliers
         df.describe()
```

Out[17]:

	age	job	marital	education	default	balance	housing	loan	contact
count	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000
mean	41.231948	4.487905	1.199337	1.285164	0.015051	1528.538524	0.473123	0.130801	0.489697
std	11.913369	3.225132	0.625552	0.749478	0.121761	3225.413326	0.499299	0.337198	0.818724
min	18.000000	0.000000	0.000000	0.000000	0.000000	-6847.000000	0.000000	0.000000	0.000000
25%	32.000000	1.000000	1.000000	1.000000	0.000000	122.000000	0.000000	0.000000	0.000000
50%	39.000000	4.000000	1.000000	1.000000	0.000000	550.000000	0.000000	0.000000	0.000000
75%	49.000000	7.000000	2.000000	2.000000	0.000000	1708.000000	1.000000	0.000000	1.000000
max	95.000000	11.000000	2.000000	3.000000	1.000000	81204.000000	1.000000	1.000000	2.000000

Figure 11. The numeric dataset

During a course assignment with Tuomas Valtanen, valuable insights were gained into the process of outlier detection and removal using visualization techniques provided by the Matplotlib library (Matplotlib: Visualization with Python). This approach is particularly useful for visual inspecting and graphing each column individually to identify potential outliers. However, given the complexity and the large number of columns in this dataset, I opted for a more efficient and systematic approach to outlier detection—the Z-score method.

The Z-score method standardizes the data and identifies outliers by measuring how many standard deviations a data point is from the mean. Outliers are typically those data points that have a Z-score above or below a certain threshold, commonly set at ±3. As illustrated in Figure 12, the code and output display NaN values where no outliers are detected and numerical values where potential outliers exist. This method allows for a more automated and scalable approach to outlier detection, particularly in datasets with numerous columns.

In this dataset, outliers were detected in columns such as ***age, balance, and previous.*** Consequently, the Matplotlib library was employed to visually inspect these columns and confirm the presence of outliers before their removal. It is crucial to cross-check these results thoroughly to avoid the inadvertent loss of valid data points that may not actually be outliers. This careful approach ensures that the dataset remains robust while minimizing the risk of eliminating data that could be valuable for model training.

In [18]:
```
# I have too many columns to check balannce of every column separetaly
# that is why I will use Z-score and IQR methods
# to check all columns at the same time

# Function to detect outliers using Z-score method
def detect_outliers_zscore(df, threshold=3):
    outliers = pd.DataFrame()
    for col in df.columns:
        z_scores = np.abs((df[col] - df[col].mean()) / df[col].std())
        col_outliers = df[z_scores > threshold][col]
        outliers = pd.concat([outliers, col_outliers], axis=1)
    return outliers

# Detect outliers using Z-score method
outliers_zscore = detect_outliers_zscore(df)
```

In [19]:
```
# Outliers detected using Z-score method
# NaN --> no outliers
# Number --> outlier, that is needed to be checked and removed
outliers_zscore
```

Out[19]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	p
1236	85.0	NaN	NaN	NaN	NaN	12114.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1243	90.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1274	85.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1320	83.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1373	83.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
10817	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	8.0	
10822	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	8.0	
10967	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	17.0	
11007	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12.0	
11054	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	8.0	

1220 rows × 17 columns

Figure 12. Z-score method for removing outliers

As illustrated in Figure 13, a comparison of the "Age" column before and after outlier removal reveals the impact of these anomalous data points on the distribution of the dataset. On the left side of the figure, the presence of outliers is evident, causing the graph to appear skewed and asymmetrical. This distortion can adversely affect the model's performance by introducing bias and reducing the accuracy of predictions.

To address this issue, I applied a filtering method using the query function with the following formula: ***df = df.query("age > age.quantile(0.05) and age < age.quantile(0.75)")***. This approach removes outliers by retaining only those data points that fall within the 5th and 75th percentiles of the "Age" column. By focusing on this interquartile range, the method effectively eliminates extreme values that could potentially skew the data distribution.

On the right side of Figure 13, the effect of this outlier removal is clearly visible, with the "Age" column now exhibiting a more symmetrical and balanced distribution. This improvement not only enhances the visual appeal of the data

but also contributes to the overall integrity of the dataset, making it more suitable for subsequent analysis and model training.

Figure 13. Comparing the balance of data in the "Age" column

The same methodology was applied to the ***balance*** column (Figure 14) to address the presence of outliers that could potentially distort the analysis and predictive modeling. Using the query function, I implemented the following formula: ***df = df.query("balance > balance.quantile(0.001) and balance < balance.quantile(0.85)").*** This approach involves filtering the data to retain only those values that fall within the 0.1th and 85th percentiles of the "balance" column.

By focusing on this specific range, the method effectively removes extreme values that might otherwise introduce bias or noise into the dataset. The decision to adjust the quantile range for the "balance" column, as compared to the "age" column, was based on the observed distribution of the data, which indicated a higher degree of variability and potential for extreme outliers.

This targeted removal of outliers resulted in a more normalized and balanced distribution of the "balance" data, which is crucial for improving the accuracy and reliability of subsequent analyses. By employing this method across

multiple columns, the integrity of the dataset is maintained, ensuring that the data used for model training is both representative and free from distortions caused by outlier values.

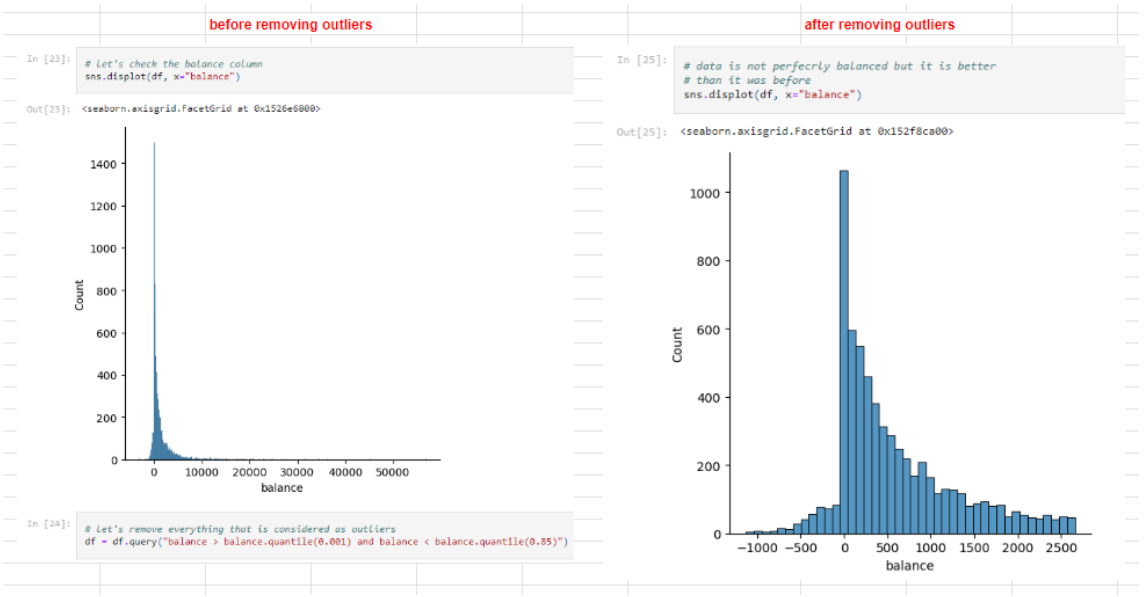


Figure 14. Comparing the balance of data in the "Balance" column

To identify and address outliers in the ***previous*** column (Figure 15), I employed a frequency analysis using the value_counts() function, which allowed me to examine the distribution of values within this column. The initial analysis, as shown on the right side of the figure, revealed a significant concentration of singular occurrences, which were identified as outliers due to their infrequent appearance and potential to skew the dataset.

To remove these outliers, I applied the following filtering method: ***df = df.query("previous < previous.quantile(0.96")).*** The choice of the 96th percentile was determined through an iterative process of trials and adjustments. Different percentile thresholds were experimented with, and it was found that setting the cutoff at the 96th percentile was most effective in isolating and removing only the outliers while preserving the integrity of the majority of the data.

The effectiveness of this approach is evident on the left side of the figure, where the same value counts() function was applied after the outlier removal. The resulting distribution demonstrates a more balanced and even spread of values within the "previous" column, indicating that the dataset is now better suited for

subsequent analysis. This careful calibration of the quantile threshold ensures that the process of outlier removal is both precise and minimally invasive, retaining the essential characteristics of the data while eliminating extreme values that could distort the results.

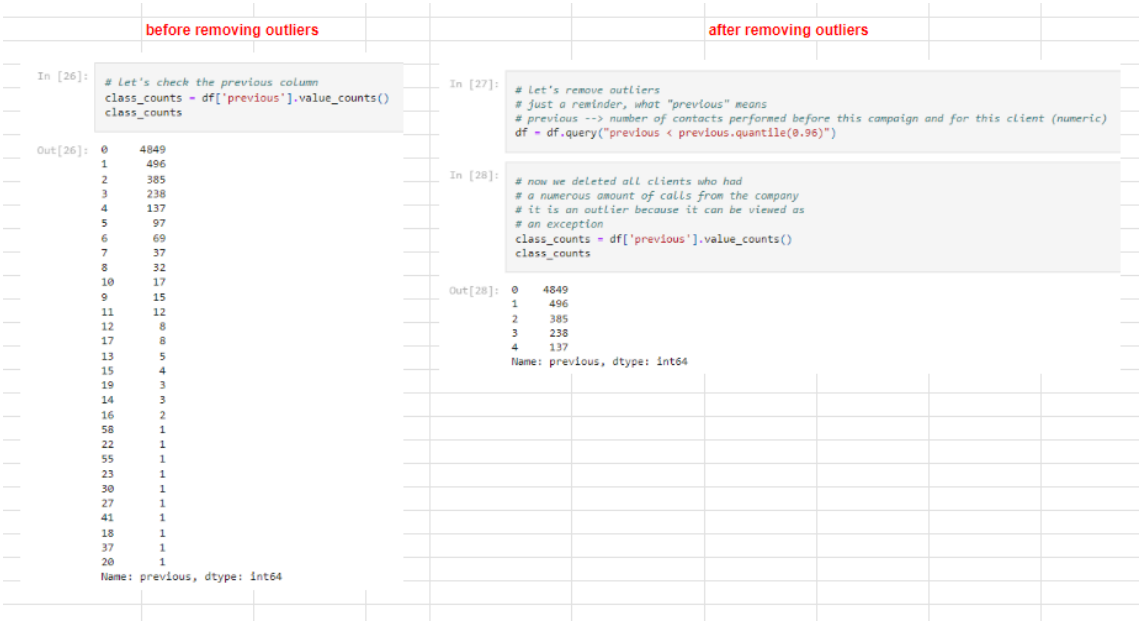


Figure 15. Comparing the balance of data in the "Previous" column

Following the initial removal of outliers using the quantile-based method, I proceeded to further validate the dataset by employing the Interquartile Range (IQR) method. This step was crucial to ensure that any remaining outliers were identified and addressed. The IQR method is a robust statistical technique that detects outliers by measuring the spread of the middle 50% of the data. Specifically, data points that fall below Q1 - 1.5 * IQR or above Q3 + 1.5 * IQR are flagged as potential outliers. (Claster, 2020 & Figure 16)

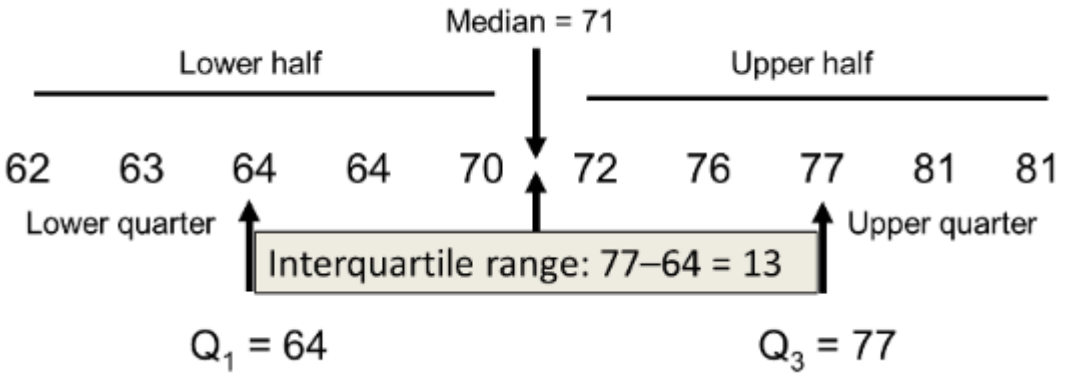


Figure 16. L. Sullivan, W. W. LaMorte. Boston University School of Public Health. InterQuartile Range (IQR).

To automate this process, I developed a custom function, ***detect_outliers_iqr***, which systematically applies the IQR method across all columns in the dataset. The function iterates through each column, calculating the first quartile (Q1) and third quartile (Q3), and subsequently determines the IQR by subtracting Q1 from Q3. It then identifies any data points that fall outside the acceptable range and compiles these outliers into a new dataframe. (Figure 17)

In [29]:
```
# Outliers detected using IQR method
# NaN --> no outliers
# Number --> outlier, that is needed to be checked and removed

# to make a fixed heigh and enable vertical scrolling
# otherwise, there was a bug and the whole table was displayed
from IPython.display import display, HTML

# Function to detect outliers using IQR method
def detect_outliers_iqr(df):
    outliers = pd.DataFrame()
    for col in df.columns:
        q1 = df[col].quantile(0.25)
        q3 = df[col].quantile(0.75)
        iqr = q3 - q1
        col_outliers = df[(df[col] < q1 - 1.5 * iqr) | (df[col] > q3 + 1.5 * iqr)][col]
        outliers = pd.concat([outliers, col_outliers], axis=1)
    return outliers

# Detect outliers using IQR method
outliers_iqr = detect_outliers_iqr(df)

# Display scrollable table
html_outliers_iqr = outliers_iqr.to_html()
display(HTML(f'<div style="height:200px; overflow-y:auto;">{html_outliers_iqr}</div>'))
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous
30	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
114	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
210	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
213	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
216	NaN	NaN	NaN	NaN	1.0	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 17. Removing outliers by implementing the IQR method

The results of this function reveal that, while the majority of the dataset has been effectively cleansed of outliers, the **poutcome** column still contains some outliers that require further examination and potential removal. This finding underscores the importance of applying multiple outlier detection methods to ensure a thorough cleaning process, as different methods may reveal different aspects of the data that require attention.

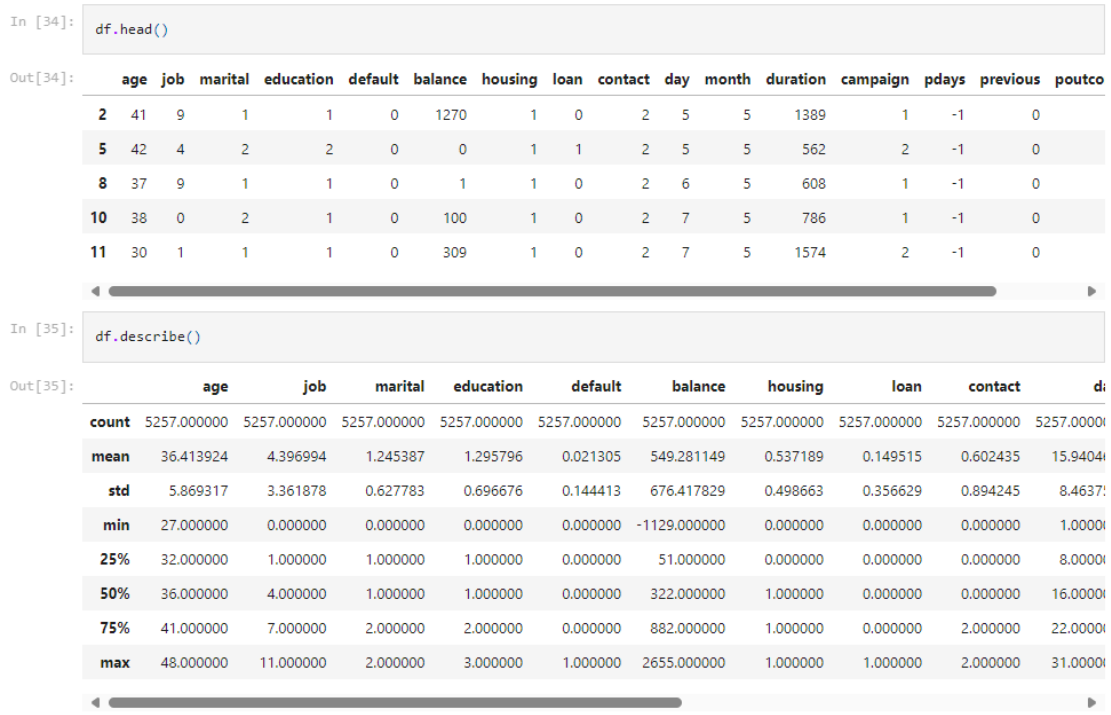
As a result of implementing the IQR method, I was able to identify and remove outliers from the ***poutcome*** column, as demonstrated in Figure 18. This step

was crucial in further refining the dataset and ensuring that the remaining data points are representative of the overall distribution.

```
In [30]:
# remove outliers in poutcome

In [31]:
# let's check outliers in the "poutcome" column
# # poutcome --> outcome of the previous marketing campaign (categorical: "unknown","other","failure","success"
# poutcome --> failure: 0, other: 1, success: 2, unknown: 3
class_counts = df['poutcome'].value_counts()
class_counts

Out[31]: 3    4849
         0     611
         2     408
         1     237
         Name: poutcome, dtype: int64

In [32]:
# remove outliers
df = df.query('poutcome >= poutcome.quantile(q=0.2)')

In [33]:
# now we have lees of
lass_counts = df['poutcome'].value_counts()
class_counts

Out[33]: 3    4849
         0     611
         2     408
         1     237
         Name: poutcome, dtype: int64
```

Figure 18. Removing outlieres from the poutcome column

The removal of these outliers has contributed to a more accurate and balanced dataset, which is essential for the reliability of subsequent analytical processes. Figure 18 visually depicts the ***poutcome*** column before and after the outlier removal, illustrating the impact of this method on the data's distribution. This thorough approach to outlier detection and removal highlights the importance of employing multiple strategies to ensure data quality and integrity.



In [34]:
```
df.head()
```

Out[34]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutco
2	41	9	1	1	0	1270	1	0	2	5	5	1389	1	-1	0	
5	42	4	2	2	0	0	1	1	2	5	5	562	2	-1	0	
8	37	9	1	1	0	1	1	0	2	6	5	608	1	-1	0	
10	38	0	2	1	0	100	1	0	2	7	5	786	1	-1	0	
11	30	1	1	1	0	309	1	0	2	7	5	1574	2	-1	0	

In [35]:
```
df.describe()
```

Out[35]:

	age	job	marital	education	default	balance	housing	loan	contact	da
count	5257.000000	5257.000000	5257.000000	5257.000000	5257.000000	5257.000000	5257.000000	5257.000000	5257.000000	5257.0000
mean	36.413924	4.396994	1.245387	1.295796	0.021305	549.281149	0.537189	0.149515	0.602435	15.9404
std	5.869317	3.361878	0.627783	0.696676	0.144413	676.417829	0.498663	0.356629	0.894245	8.4637
min	27.000000	0.000000	0.000000	0.000000	0.000000	-1129.000000	0.000000	0.000000	0.000000	1.0000
25%	32.000000	1.000000	1.000000	1.000000	0.000000	51.000000	0.000000	0.000000	0.000000	8.0000
50%	36.000000	4.000000	1.000000	1.000000	0.000000	322.000000	1.000000	0.000000	0.000000	16.0000
75%	41.000000	7.000000	2.000000	2.000000	0.000000	882.000000	1.000000	0.000000	2.000000	22.0000
max	48.000000	11.000000	2.000000	3.000000	1.000000	2655.000000	1.000000	1.000000	2.000000	31.0000

Figure 19. The balanced and numeric dataset

Following the completion of the data engineering modifications I have successfully transformed the dataset into a balanced and fully numeric format (Figure 19). This refined dataset is now well-prepared for subsequent stages of analysis and model development.

As a result, the data is now optimally structured, minimizing the risk of bias and maximizing the potential for accurate and reliable model predictions in the next phases of the research.

3.3 Split - train data, multicollinearity and scaling the values

The subsequent step in preparing the data for the development of a logistic regression model involves partitioning the dataset into training and testing subsets. This is a crucial step to ensure that the model can be trained effectively and later evaluated on unseen data to assess its performance. To achieve this, I employed a common technique where the dataset is divided into two components: features (X) and the target variable (y).

Specifically, the feature set (X) includes all columns in the DataFrame except for the target variable, while the target variable (y) comprises only the column that we aim to predict—in this case, the "deposit" column. (Figure 20)

Converting data to X/y and checking multicollinearity by using VIF

In [36]:
```
# converting data to X/y
# a common trick in X/y -split
# X = everything else in the DataFrame minus the target variable
# y = only the target variable
X = df.drop('deposit', axis=1)
y = df['deposit']
```

Figure 20. Converting data to train-split

By defining X as the set of independent variables and y as the dependent variable, this step ensures that the model will be trained to learn the relationship between the features and the target variable.

To assess multicollinearity among the features in the dataset, I employed the Variance Inflation Factor (VIF), a statistical measure that quantifies the extent to which the variance of a regression coefficient is inflated due to multicollinearity among the predictor variables. (Figure 21) High multicollinearity can lead to unreliable estimates of regression coefficients and can confuse the logistic regression model, making it difficult to discern the individual effect of each predictor.

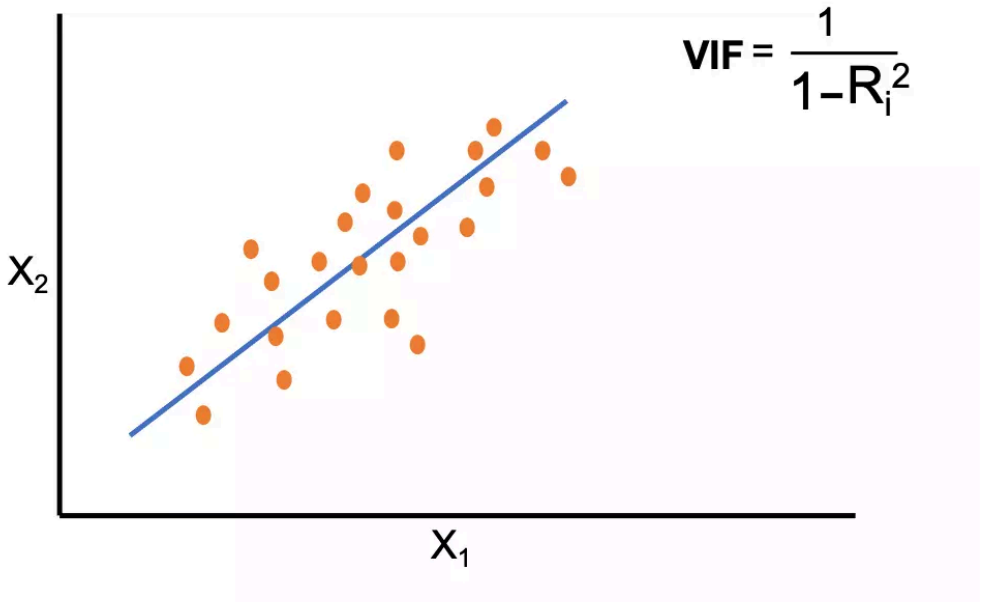


Figure 21. Multicollinearity and Variance inflation factor (VIF) (R. Bedre 2021)

The analysis (Figure 22) revealed that features such as ***default, balance, loan, contact,*** and ***campaign*** are not highly correlated with each other, as evidenced by their relatively low VIF values. However, features like ***age, marital, education, day, month, duration, pdays,*** and ***previous*** exhibited higher levels of multicollinearity.

Most notably, the ***poutcome*** feature displayed an exceptionally high VIF of 91.836809, indicating severe multicollinearity with other predictors in the model. This strong multicollinearity suggests that ***poutcome*** may provide redundant information that overlaps with other variables, potentially confusing the logistic regression model and leading to unstable coefficient estimates. This issue warrants careful consideration, and steps may be required to address it, such as removing or combining highly collinear features.

In [37]:

```
# pip install statsmodels
from statsmodels.stats.outliers_influence import variance_inflation_factor

# VIF dataframe
# VIF = Variance Inflation Factor
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns

# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                          for i in range(len(X.columns))]

# variables with high VIF-value
# can mean multicollinearity (variables providing same linear
# relationships in the data, confusing the logistic regression
vif_data

# "default," "balance," "loan," "contact," "campaign"
# --> are not highly correlated with each other

#  "age," "marital," "education," "day," "month," "duration," "pdays," "previous"
# --> higher levels of multicollinearity.
# A VIF value above 10 is often considered problematic and may require further investigation.

# Notably, the "poutcome" feature has a very high VIF of 91.836809,
# indicating a strong multicollinearity issue with other predictors in the model
```

Figure 22. VIF (Variance Inflation Factor) dataframe

The next crucial step in the data preparation process is to split the dataset into training and testing subsets and then scale the features to ensure uniformity across the data. This step is particularly important for models like logistic regression, which are sensitive to the scale of input features. By standardizing the features, we can enhance the model's performance and convergence during

training. To achieve this, I first imported the necessary module from the ***sklearn*** library. (Figure 23)

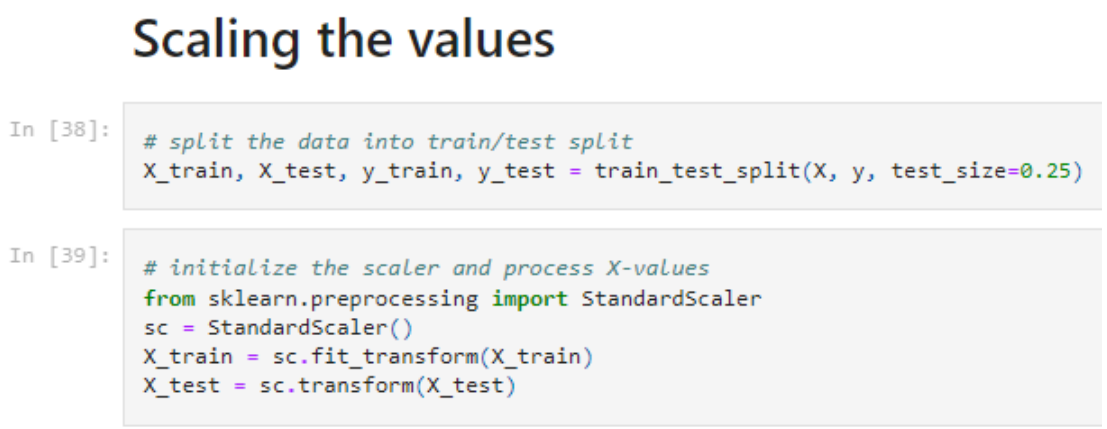


Figure 23. Scaling the values and employing spllit

Before scaling the data, it is essential to split the dataset into training and testing sets. The training set is used to train the model, while the testing set is reserved for evaluating the model's performance on unseen data. This ensures that the model's ability to generalize to new data can be effectively assessed. In figure 23 the train_test_split from the sklearn.model_selection module was used to randomly split the dataset, with 75% of the data allocated for training and 25% for testing.

After splitting the data, the next step involves scaling the features using the StandardScaler from sklearn.preprocessing. Standardization transforms the data so that it has a mean of zero and a standard deviation of one, which helps to ensure that all features contribute equally to the model’s predictions.

The StandardScaler was first fitted on the training data using fit_transform, which both calculates the scaling parameters (mean and standard deviation) and applies the transformation to the training data. The testing data was then transformed using the same parameters, ensuring that both training and testing sets are scaled consistently.

By standardizing the data, we mitigate the impact of features with different units or magnitudes, thereby improving the logistic regression model's ability to learn and make accurate predictions.

3.4 The development of the model

The final step in creating the logistic regression model involves the development and optimization of the model itself. As illustrated in Figure 24, I experimented with various feature combinations and modelling approaches to identify the configuration that yields the best overall accuracy.

Creating the logistic regression and fitting the data

In [40]:

```
# Variant 1:
# logmodel = LogisticRegression()
# logmodel.fit(X_train, y_train)
# Model overall accuracy: 79.62%

# Variant 2:
#logmodel = make_pipeline(StandardScaler(),
                    #    PolynomialFeatures(degree=2, include_bias=False),
                    #    LogisticRegression(solver="sag"))

#logmodel.fit(X_train, y_train)
# Model overall accuracy: 83.12%

# Variant 3:
#Logmodel = make_pipeline(LogisticRegression(solver="newton-cholesky"))
#logmodel.fit(X_train, y_train)
# Model overall accuracy: 81.67%

# Variant 3:
# logmodel = make_pipeline(StandardScaler(), LogisticRegression(solver="newton-cholesky"))
# logmodel.fit(X_train, y_train)
# Model overall accuracy: 81.06%

# Variant 4:
logmodel = make_pipeline(StandardScaler(),
                         PolynomialFeatures(degree=2, include_bias=False),
                         LogisticRegression(solver="newton-cholesky"))
logmodel.fit(X_train, y_train)
# Model overall accuracy: 83.35%
```

Out[40]:

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('polynomialfeatures', PolynomialFeatures(include_bias=False)),
                ('logisticregression',
                 LogisticRegression(solver='newton-cholesky'))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Figure 24. The development of the model

Several variants of the logistic regression model were tested:

1. Variant 1: The initial model was developed using a straightforward logistic regression without additional feature transformations or scaling. This model achieved an overall accuracy of 79.62%.
2. Variant 2: The next variant involved creating a pipeline that included both feature scaling using StandardScaler and polynomial feature transformation (degree 2). The logistic regression was solved using the "sag" solver. This variant improved the model's overall accuracy to 83.12%.

3. Variant 3: In this variant, I applied the "newton-cholesky" solver directly in the logistic regression model, without additional feature scaling or transformation. This approach resulted in an overall accuracy of 81.67%.
4. Variant 4: This variant combined feature scaling using StandardScaler with logistic regression utilizing the "newton-cholesky" solver. The overall accuracy observed with this configuration was 81.06%.
5. Variant 5: Finally, I implemented a pipeline that included both StandardScaler and PolynomialFeatures (degree 2, without bias), followed by logistic regression using the "newton-cholesky" solver. This combination produced the highest overall accuracy at 83.35%.

These variants illustrate the iterative process of model development, where different combinations of scaling, feature engineering, and solvers are tested to optimize performance. The ultimate goal is to refine the model to achieve the best possible accuracy while maintaining robustness and generalizability to unseen data.

Once the optimal model configuration was determined, the next step involved training the model on the training dataset. Model training is a crucial phase where the logistic regression algorithm learns the relationships between the input features and the target variable. (Sibanjan & Mert 2018) During this phase, the model adjusts its parameters to minimize the difference between its predictions and the actual outcomes, thereby improving its accuracy over time.

4 ANALYSIS OF OUTCOMES OF THE LOGISTIC REGRESSION

4.1 Classification error metrics

After developing and training the logistic regression model, the next critical step is to evaluate its performance using classification error metrics. These metrics provide a comprehensive overview of how well the model is performing, particularly in terms of its ability to correctly predict the target variable.

For better comprehension I will provide a visual example from Ted Tigerschiold where he demonstates the concept of a confusion matrix in a simply way. The figure 25 provides a visual representation of the performance of a binary

classification model. It shows the model's predictions (rows) against the actual labels (columns). The matrix includes four key components: True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).

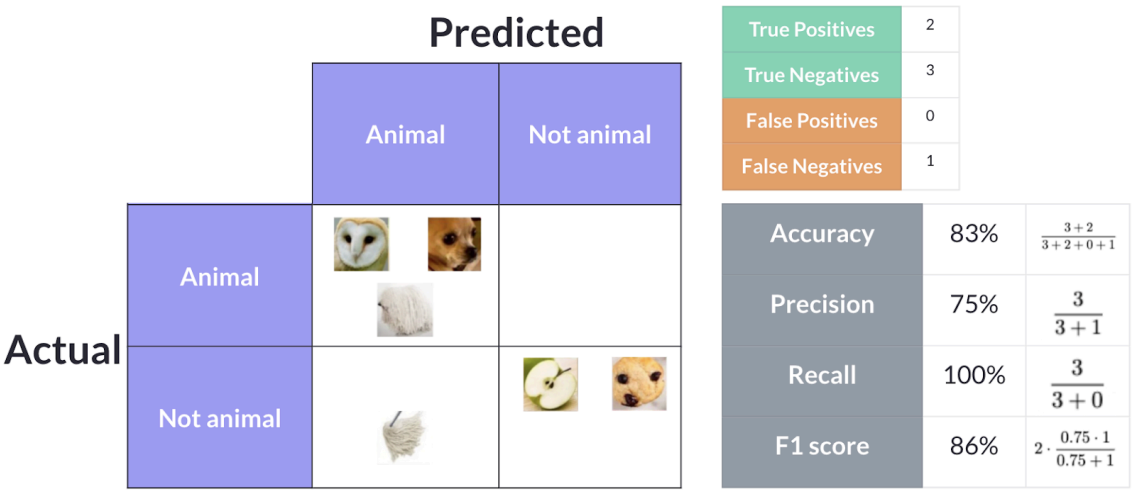


Figure 25. Example of a confusion matrix (Tigerschiold 2022)

To assess the performance of the Logisrtic regressesion model, a classification report was generated using classification_report from the sklearn.metrics module. This report includes key metrics such as precision, recall, and F1-score for each class, along with the overall accuracy of the model. (Figure 26)

In [41]:
```
predictions = logmodel.predict(X_test)
```

In [42]:
```
# print the classification report based on true values and predictions
print(classification_report(y_test, predictions))

# get overall accuracy of the model and print it
acc = accuracy_score(y_test, predictions)
print("\nModel overall accuracy: {:.2f}%".format(acc * 100))

# f1-score is 0.77 for the 1 class due to
# a slightly lower balance between precision
# and recall for class 1 compared to class 0.

# accuracy is 0.82
# I think it could be higher if we had more balanced data

# after several trainings the overal accuraccy decreased
# in the first training period it was 83.35%
```

```
              precision    recall  f1-score   support

           0       0.83      0.86      0.84       768
           1       0.79      0.75      0.77       547

    accuracy                           0.81      1315
   macro avg       0.81      0.81      0.81      1315
weighted avg       0.81      0.81      0.81      1315


Model overall accuracy: 81.44%
```

In [43]:
```
# sns.heatmap(confusion_matrix(y_test, predictions), annot=True)

# True Negative (TN) - False Positive (FP)
# False Negative (FN) - True Positive (TP)
print(confusion_matrix(y_test, predictions))

# It's crucial to assess the balance between
# false positives and false negatives
# In my case, FP is almost doubled compared to FN

# I think, it is becuase unbalanced between positive
# and negative outcome in the "deposit" column in the dataframe
```

```
[[661 107]
 [137 410]]
```

Figure 26. Classification error metrics

Following the initial evaluation of the model's performance using standard classification metrics, I implemented another critical metric, the Area Under the Curve (AUC), to further assess the effectiveness of the predictions. The AUC

score, derived from the Receiver Operating Characteristic (ROC) curve, provides a robust measure of the model's ability to distinguish between the positive and negative classes.

The figure 27 (Jalaj 2018) illustrates four examples of ROC (Receiver Operating Characteristic) curves, each corresponding to different AUC (Area Under the Curve) values. These curves visually represent the performance of a binary classifier as its discrimination threshold is varied, with the AUC score summarizing the overall performance.

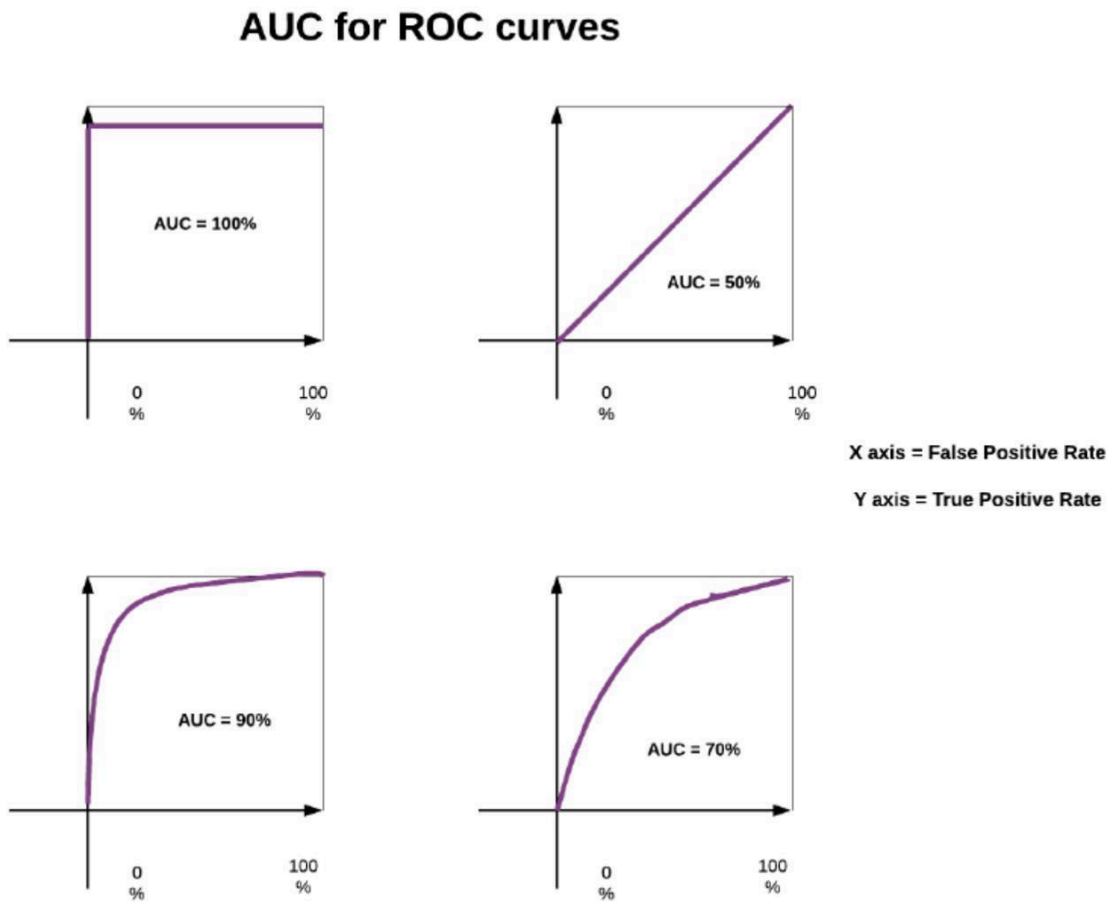


Figure 27. AUC for ROC curves (Jalaj 2018)

- Top Left (AUC = 100%): This curve represents a perfect classifier that correctly distinguishes between the positive and negative classes at all thresholds. The AUC score of 100% (or 1.0) indicates that the model has perfect sensitivity (true positive rate) and specificity (true negative rate), meaning it always makes correct predictions.

- Top Right (AUC = 50%): This curve represents a classifier that performs no better than random guessing. The AUC score of 50% (or 0.5) indicates that the model's ability to discriminate between classes is equivalent to a coin toss. The ROC curve in this scenario is a diagonal line, suggesting that the true positive rate increases at the same rate as the false positive rate.
- Bottom Left (AUC = 90%): This curve represents a high-performing classifier with an AUC of 90% (or 0.9). The curve is close to the top left corner, indicating that the model has a high true positive rate while maintaining a low false positive rate. This suggests that the model is very effective in distinguishing between the two classes.
- Bottom Right (AUC = 70%): This curve represents a classifier with moderate performance, having an AUC of 70% (or 0.7). The curve is above the diagonal line but does not reach the top left corner, indicating that while the model is better than random guessing, there is still room for improvement in its ability to differentiate between the positive and negative classes.

These examples highlight how the ROC curve and AUC score can be used to evaluate the performance of a classification model. A higher AUC score indicates better model performance, with a score closer to 1.0 reflecting a stronger ability to correctly classify instances across various thresholds. This analysis is crucial for understanding the effectiveness of a model, particularly in scenarios where both false positives and false negatives carry significant consequences.

```
In [44]:
# The AUC score is a super sensitive metric
# you often get low scores, even 0.5

# in binary logistic regression, AUC values are often interpreted as follows:
# A binary classifier is useful only when it achieves ROC-AUC score greater than 0.5 and as near to 1 as possible.
# If a classifier yields a score less than 0.5, it simply means that the model is performing worse
# than a random classifier, and therefore is useless.

# In multinomial logistic regression , AUC values are often interpreted as follows:
# 0.5-0.6 (failed)
# 0.6-0.7 (worthless)
# 0.7-0.8 (poor)
# 0.8-0.9 (good)
# > 0.9 (excellent)

# basically 0.5 means you could get the same result with just random guessing
roc_auc_score(y, logmodel.predict_proba(X.values)[:, 1])

# 0.51 is not bad as it could be :)

Out[44]: 0.48640549416010664
```

Figure 28. AUC score

In the analysis, the logistic regression model produced an AUC score of approximately 0.486 (Figure 28). This score, being slightly below 0.5, indicates that the model's predictive performance is worse than random guessing. This result underscores the challenges associated with the current model configuration and suggests that the model has very limited ability to correctly distinguish between the positive and negative classes.

The relatively low AUC score highlights the need for further refinement and optimization of the model. Potential strategies for improvement could include re-examining the feature selection process, addressing any underlying data imbalances, or experimenting with alternative modeling techniques. Despite the modest result, the AUC metric serves as a valuable tool for understanding the limitations of the current model and guiding subsequent efforts to enhance its predictive capabilities.

5 POTENTIAL MARKETING STRATEGIES

5.1 Marketing background of the thesis author

Although my knowledge and experience in machine learning are relatively limited, I bring over three years of proven experience in marketing, specifically within the Business-to-Consumer (B2C) sector. In B2C marketing, the focus is often placed on internet-based strategies, mass customization, relational databases, and customer referrals (Naresh, 2016). This expertise informs the approach I take in this thesis, where I will explore and propose marketing

strategies tailored to B2C businesses, drawing insights from the outcomes of the data engineering and data analysis performed in the previous sections.

This foundational experience in B2C marketing will allow me to align the machine learning insights with practical strategies for reaching and engaging consumers in this category, ensuring the recommendations are both data-driven and grounded in real-world marketing practices.

5.2 Marketing strategies

5.2.1 Visual marketing

Based on the dataset that includes information about customer occupations, we can segment the target audience into different groups and design tailored marketing campaigns for each segment. In B2C marketing, understanding the customer's profession, lifestyle, and preferences allows for highly personalized and effective marketing strategies.

Below are examples of how customer segments can be formed and how specific marketing campaigns can be planned for each.

1. *Blue-Collar Workers*

- Segment Characteristics: Blue-collar workers often have physically demanding jobs. They may value durability, affordability, and functionality in the products they choose.
- Marketing Campaign: Focus on promoting durable, cost-effective products such as workwear, tools, or safety equipment. A direct email campaign featuring discounts on these products, alongside testimonials from satisfied blue-collar customers, would be effective.

In order to develop targeted marketing strategies for the segments detected in the dataset, I asked ChatGPT to generate marketing posters tailored to each segment. Below are the results for the different target groups, including Blue-Collar Workers, Entrepreneurs, and Students. Each poster reflects the unique characteristics and needs of these groups, illustrating how personalized marketing campaigns can enhance engagement and conversion.

Figure 29. Poster 1 which was generated by ChatGPT

Here is a poster designed for blue-collar workers, promoting an affordable loan offer, focusing on their needs for durable, cost-effective work gear.

2. Entrepreneurs

- Segment Characteristics: Entrepreneurs are driven by business success, innovation, and efficiency. They often look for tools that help them scale their businesses and manage their resources effectively.
- Marketing Campaign: Offer content that speaks to their entrepreneurial spirit, such as workshops or webinars on business growth, productivity tools, or software solutions for small businesses. Using social media ads and LinkedIn campaigns would be effective in reaching this segment.

Figure 30. Poster 2 which was generated by ChatGPT

Here is the poster designed for entrepreneurs, promoting a loan offer aimed at fueling business growth

3. Students

- Segment Characteristics: Students are typically tech-savvy, budget-conscious, and focused on learning, socializing, and entertainment.
- Marketing Campaign: Offer discounts on student-related services or products, such as tech gadgets, software for education, or streaming services. A campaign on social media platforms like TikTok, Instagram, or YouTube, coupled with influencer partnerships, would capture their attention.

Figure 31. Poster 3 which was generated by ChatGPT

By segmenting the target audience based on their occupations and other variables from the dataset, marketers can craft highly relevant campaigns that resonate with each group's unique needs and preferences. This approach enhances the efficiency of marketing efforts, driving higher engagement and better conversion rates.

5.2.2 Personalized campaigns

Building on the data we have about the dates when clients were contacted and when they made decisions, we can leverage this information to further personalize marketing campaigns. By analyzing the timing patterns in customer behavior, we can identify the most effective days or periods when customers are more likely to respond positively to a campaign. This data-driven approach allows for:

1. Optimizing Campaign Timing: By identifying specific times (e.g., days of the week, times of day) when customers are most receptive, we can schedule marketing campaigns at optimal times to increase engagement.

Figure 32. Poster 4 which was generated by ChatGPT

2. Seasonal or Event-Based Personalization: If the data reveals trends around specific seasons or events (e.g., holidays, weekends), campaigns can be timed to coincide with these periods when customers are more likely to make purchasing decisions.

Figure 33. Poster 5 which was generated by ChatGPT

3. Behavior-Based Triggers: We can create automated marketing triggers that send personalized messages when customers have historically shown a likelihood to engage, based on past data.

Figure 34. Poster 6 which was generated by ChatGPT

4. Enhancing Retention Strategies: For clients who previously made decisions after multiple contacts, we can create follow-up strategies at precisely timed intervals to nurture their decision-making process.

Figure 35. Poster 7 which was generated by ChatGPT

This data-driven personalization strategy will help ensure that marketing efforts are aligned with customer behavior patterns, ultimately increasing the effectiveness of campaigns and improving conversion rates.

6 DISCUSSION

This thesis aimed to explore the application of machine learning, specifically logistic regression, in optimizing marketing strategies by analyzing customer data. Throughout the project, I have leveraged my experience in marketing, particularly in the B2C sector, to propose actionable strategies informed by the insights derived from data engineering and machine learning analysis. The integration of machine learning with marketing decision-making has the potential to improve the precision and relevance of marketing efforts, which can lead to higher customer engagement and conversion rates.

The logistic regression model developed in this thesis focused on classifying customers' likelihood of subscribing to a product based on historical data. While the initial accuracy of the model was promising, several challenges arose, particularly in terms of handling categorical variables and balancing the dataset. The use of OneHotEncoder for categorical variables provided a more accurate representation of the data, avoiding the pitfalls of LabelEncoder, which would have introduced bias due to the inherent hierarchical differences between categories such as "admin" and "student."

The results of the model, while insightful, also highlighted the limitations of logistic regression when applied to complex customer behavior data. The relatively low AUC score, which indicated that the model performed only slightly better than random guessing, suggests that logistic regression may not always be the most suitable model for this type of analysis. This opens up opportunities for future research to explore alternative models such as decision trees, random forests, or gradient boosting, which might provide better predictive performance in classifying customer decisions.

From a marketing perspective, the segmentation of customers based on occupation and other variables in the dataset allowed for the development of

tailored marketing strategies. By understanding the specific needs and behaviors of segments like blue-collar workers, entrepreneurs, and students, personalized marketing campaigns can be crafted to target these groups more effectively. This kind of data-driven personalization is crucial for improving customer experiences and increasing campaign effectiveness.

Additionally, the analysis of customer contact and decision-making dates demonstrated the importance of timing in marketing. By identifying peak times when customers are most responsive, such as Thursdays between 5 PM and 8 PM, marketing efforts can be better synchronized with customer behavior patterns. This not only enhances engagement but also improves the overall efficiency of marketing campaigns by reaching customers when they are most likely to make purchasing decisions.

7 REFERENCE NOTATION

7.1 BIBLIOGRAPHY

Cambridge University Press & Assessment 2024. Dictionary. Accessed 30 August 2024. https://dictionary.cambridge.org/

Claster, William B., Mathematics and R programming for machine learning : from the ground up. CRC Press 2020. 1st ed. Accessed 5 September 2024. https://luc.finna.fi/lapinamk/Record/luc_electronic_amk.994872132506246?sid=4789452815

D. Aha 1987. UCI Machine Learning Repository. Accessed 23 August 2024. https://archive.ics.uci.edu

D. Sibanjan, C. Umit Mert. Packt Publishing 2018. First edition. Hands-on automated machine learning: a beginner's guide to building automated machine learning systems using AutoML and Python. Accessed 5 September 2024. https://luc.finna.fi/lapinamk/Record/luc_electronic_amk.994713668106246?sid=4782253595

L. Sullivan, W. W. LaMorte. Boston University School of Public Health. InterQuartile Range (IQR). Accessed 5 September 2024. https://sphweb.bumc.bu.edu/otlt/mph-modules/bs/bs704_summarizingdata/bs704_summarizingdata7.html

Naresh K., Uslan C. Relationship marketing re-imagined: marketing's inevitable shift from exchanges to value cocreating relationships. Business Expert Press 2016. First edition. Accessed 14 October 2024. https://luc.finna.fi/lapinamk/Record/luc_electronic_amk.994707783106246?sid=4841172751

Rashmi, A. 2021. Big data, IoT, and machine learning: tools and applications. CRC Press 2021. 1st Edition. Accessed 26 August 2024. https://luc.finna.fi/lapinamk/Record/luc_electronic_amk.994899045406246?sid=4782253595

R. Bedre. 2021. Multicollinearity and variance inflation factor (VIF) in the regression model (with Python code). Accessed 5 September 2024. https://www.reneshbedre.com/blog/variance-inflation-factor.html?utm_content=cmp-true

Sibanjan, D. 2018. Hands-on automated machine learning : a beginner's guide to building automated machine learning systems using AutoML and Python. Packt Publishing 2018. First edition. Accessed 26 August 2024. https://luc.finna.fi/lapinamk/Record/luc_electronic_amk.994713668106246?sid=4782253595

Scikit-learn developers (BSD License). Scikit-learn. Accessed 30 August 2024. https://scikit-learn.org/

The Matplotlib development team. Matplotlib: Visualization with Python. Accessed 4 September 2024. https://matplotlib.org/

T. Tigerschiold. 2022. What is Accuracy, Precision, Recall and F1 Score? Accessed 5 September 2024. https://www.labelf.ai/blog/what-is-accuracy-precision-recall-and-f1-score

Verma, J. P., Abdel-Salam G. Testing statistical assumptions in research. Wiley 2019. 1st edition. Accessed 4 September 2024. https://luc.finna.fi/lapinamk/Record/luc_electronic_amk.994713783706246?sid=4788904388