



MASTER THESIS

# Predicting hotel booking cancellations

WITH AUTOENCODERS AND STATISTICAL  
DIMENSIONALITY REDUCTION

By

**Zerin A. Arif**  
(2736034)

*First supervisor:* prof. dr. Mark Hoogendoorn  
*Second reader:* dr. Rikkert Hindriks

*Submitted in fulfillment of the requirements for the degree of*  
***Master of Science in Business Analytics***  
*at Vrije Universiteit Amsterdam*

September 11, 2025

## Abstract

Predicting hotel booking cancellations is crucial for revenue management, as cancellations can lead to revenue loss and inefficient resource allocation. Traditional machine learning methods have been widely applied to this problem, with feature engineering improving predictive performance. However, feature engineering may also introduce redundancy and noise. Less attention has been given to the role of feature learning and dimensionality reduction techniques. This thesis explores the predictive performance of models trained on features learned with autoencoders, compared to statistical dimensionality reduction techniques, and models trained on the original feature set. The dimensionality reduction techniques applied were principal component analysis (PCA), multiple correspondence analysis (MCA), and autoencoders. Their performance was evaluated with three classifiers: random forest, XGBoost, and a multilayer perceptron (MLP). Across the models evaluated, XGBoost achieved the strongest overall performance ( $F_1 = 0.84$ ) when trained directly on the original features. For the ensemble models, neither PCA nor autoencoder features improved performance. In contrast, the MLP benefited substantially from the autoencoder-derived features, improving from an  $F_1$  score of 0.68 on the original data to 0.78 with an autoencoder bottleneck of 32. Overall, the findings demonstrate that dimensionality reduction is unnecessary for tree-based models, which can already handle sparse, high-dimensional inputs effectively. However, it can make neural networks more competitive. For practical applications, XGBoost trained on the original features remains the most efficient and reliable approach for predicting hotel booking cancellations.

**Keywords**— Hotel booking cancellations, Revenue management, Machine learning, Tabular data, Dimensionality reduction, PCA, Autoencoders, Feature learning

# Contents

<b>1</b>	<b>Introduction</b>	<b>8</b>
<b>2</b>	<b>Literature review</b>	<b>11</b>
2.1	Cancellation behavior in the hospitality industry . . . . .	11
2.1.1	Demand forecasting . . . . .	12
2.1.2	Overbooking . . . . .	12
2.1.3	Dynamic Pricing . . . . .	12
2.2	Previous research on cancellation prediction . . . . .	13
2.3	Statistical approaches for dimensionality reduction . . . . .	15
2.3.1	Numerical data . . . . .	15
2.3.2	Categorical data . . . . .	17
2.4	Autoencoders for dimensionality reduction . . . . .	18
2.5	Concluding remarks . . . . .	20
<b>3</b>	<b>Data</b>	<b>23</b>
3.1	Data description . . . . .	23
3.2	Feature engineering . . . . .	23
3.3	Data exploration . . . . .	26
<b>4</b>	<b>Methods</b>	<b>32</b>
4.1	Dimensionality reduction methods . . . . .	32
4.1.1	Principal Component Analysis . . . . .	32
4.1.2	Multiple Correspondence Analysis . . . . .	34
4.1.3	Autoencoders . . . . .	35
4.2	Model selection . . . . .	38
4.2.1	Random forest . . . . .	38
4.2.2	XGBoost . . . . .	40
4.2.3	Multilayer perceptron . . . . .	42
4.3	Bayesian hyperoptimization . . . . .	43
<b>5</b>	<b>Experimental setup</b>	<b>44</b>
5.1	Data preparation . . . . .	44
5.1.1	Feature preprocessing . . . . .	44
5.1.2	Train-test split . . . . .	45
5.2	Experimental setup . . . . .	45
5.3	Model Training . . . . .	45
5.3.1	Hyperparameter optimization . . . . .	46
5.4	Evaluation . . . . .	47
5.5	Implementation details . . . . .	49

<b>6</b>	<b>Results</b>	<b>50</b>
6.1	Dimensionality reduction results . . . . .	50
6.1.1	Component selection for statistical methods . . . . .	50
6.1.2	Autoencoder architecture . . . . .	52
6.2	Main findings . . . . .	53
6.2.1	Statistical comparison of configurations . . . . .	55
6.3	Best performance for each model . . . . .	56
6.4	Key variables . . . . .	56
6.5	The effect of dimensionality reduction . . . . .	58
6.5.1	Hyperparameter analysis . . . . .	59
6.6	Comparison between AE-32 and AE-16 . . . . .	60
<b>7</b>	<b>Discussion</b>	<b>61</b>
7.1	Autoencoder design and latent space . . . . .	61
7.2	Comparison of the results with prior work . . . . .	62
7.3	Limitations . . . . .	63
7.4	Recommendations for future work . . . . .	63
<b>8</b>	<b>Conclusion</b>	<b>65</b>
	<b>Appendices</b>	<b>66</b>
<b>A</b>	<b>Methods</b>	<b>67</b>
A.1	Principal Component Analysis . . . . .	67
A.2	Multiple Correspondence Analysis . . . . .	69
<b>B</b>	<b>Results</b>	<b>72</b>
B.1	PCA and MCA component summary table . . . . .	72
B.2	McNemar statistical test . . . . .	73
B.3	Confusion matrices of all configurations . . . . .	75
B.4	Hyperparameter in-depth analysis . . . . .	76

# List of Figures

3.1	The ratio of the categorical variables room type, meal plan, and market segment type. . . . .	27
3.2	Distribution of length of stay for each segment. . . . .	28
3.3	Distribution of the lead time. Note: a booking horizon of 0 means last last-minute booking. . . . .	28
3.4	Seasonality in the dataset. . . . .	29
3.5	Average price per night for each room. . . . .	30
3.6	Ratio of canceled vs. not canceled bookings in the dataset for total bookings and for the categories guest type, booking type, and stay type. Red indicates canceled bookings, and green indicates not canceled. . . . .	31
4.1	Illustration of PCA. The original coordinate system is transformed into new orthogonal axes ( $PC_1$ , $PC_2$ ), which are rotated to capture directions of maximum variance in the data. The scattered points represent the distribution of observations, with variance along each principal component indicated by red braces. $PC_1$ captures the largest share of variance, while $PC_2$ captures the smaller variance. . . . .	33
4.2	Illustration of the indicator matrix, which is the same as one-hot encoding. . . . .	34
4.3	Illustration of a multilayer perceptron (MLP) with three input neurons, two hidden layers, and one output neuron. Each neuron computes a weighted sum of the outputs from the previous layer, adds a bias term, and applies a nonlinear activation function as defined in Equation 4.3. . . . .	36
4.4	Autoencoder architecture. The bottleneck layer is called “Code”. Source: <a href="#">Paper</a> . . . . .	37
4.5	Illustration of the random forest algorithm. Each tree is trained on a bootstrap sample and selects a random subset of features at each split. . . . .	39
5.1	Experimental setup of this study. Three approaches are compared: no dimensionality reduction, statistical reduction (PCA + MCA), and deep learning-based reduction (autoencoder). Each representation is evaluated using the same downstream models: RF, XGBoost, and MLP. . . . .	46
6.1	Percentage of explained variance of PCA and MCA. The red bar indicates that 80% cumulative explained variance is achieved. . . . .	51

6.2	Heatmap column coordinates $\in [-1, 1]$ from PCA, displaying the contribution of each feature to each component. An absolute value closer to 1 indicates a stronger relationship with the component, while the sign reflects the direction of the association. . . . .	51
6.3	Validation loss per epoch for each autoencoder. Mean and SD are calculated from five runs per architecture. . . . .	53
6.4	Average validation reconstruction loss by bottleneck size (4, 8, 16, 32) for two-layer networks. Bars show the standard deviation. Performance improves markedly from $4 \rightarrow 8 \rightarrow 16$ , then plateaus between 16 and 32. . . . .	53
6.5	The $F_1$ score and training time for all models and dimensionality reduction configurations. Reported times reflect the hyperparameter tuning of the classifiers only. The additional training time for autoencoders was substantial, whereas PCA/MCA added a negligible training time. . . . .	54
6.6	Pairwise McNemar test with BH correction across all configurations. .	55
6.7	SHAP values for the XGBoost model on the original dataset. Positive values indicate a push toward predicting “canceled”, while negative values push toward “not canceled”. The distance from zero reflects the strength of the impact. Colors represent the feature value (red = high, blue = low). The vertical spread of the points illustrates variability, with wider spreads indicating that the feature has very different effects across samples, and narrower spreads indicating a more consistent effect. . . . .	57
6.8	Confusion matrices illustrating the trade-off between false positives and false negatives for the three models. . . . .	58
6.9	t-SNE projections on the test set of the AE-32 latent space (left) and the AE-16 latent space (right). . . . .	60
A.1	Geometric interpretation of an eigenvector: direction is unchanged, only scaled by $\lambda$ . . . . .	69
B.1	Confusion matrices illustrating the trade-off between false positives and false negatives for the three models. . . . .	75
B.2	Relative hyperparameter importance obtained through Optuna. . . .	76
B.3	Optuna slice plots of the three most influential hyperparameters for each model. Each point represents one of the 50 optimization trials and its corresponding hyperparameter setting. . . . .	76

# List of Tables

2.1	Summary of dimensionality reduction techniques reviewed in literature	21
2.2	Overview of prior studies on cancellation prediction. Note that the study by Morales and Wang (2010) [41] is on cancellation rates and not binary cancellation prediction. Sánchez-Medina and C-Sánchez (2020) [48] and Huang et al. (2013) [35] did not analyze the variables.	22
3.1	Overview of features in the dataset.	24
3.2	Descriptions of categorical variables in the dataset.	25
3.3	Overview of engineered features added to the dataset. These derived variables were constructed to capture booking patterns (e.g., stay duration, spending behavior).	26
3.4	Contingency table room type vs. market segment type. Percentages (in brackets) are normalized by the total number of bookings in the dataset.	27
3.5	Absolute number of canceled and not canceled bookings across multiple categories. Percentages (in parentheses) are normalized by the total number of bookings in each category. Note that guest type and stay type categories sum to 100%, while booking type does not, as it excludes bookings with a lead time between 3 and 45 days.	31
5.1	Hyperparameter tuning details and search spaces for each model. The choice of search ranges is based on prior studies: <sup>1</sup> Probst et al. [45]; <sup>2</sup> Budholiya et al. [18]; <sup>3</sup> Chen and Guestrin [21]; <sup>4</sup> Sommer et al. [51]; <sup>5</sup> Smith [50]. A detailed discussion of these hyperparameters and their role in model performance is provided in Section 4.2 (Model selection).	47
5.2	Confusion matrix	48
6.1	Comparison of autoencoder architectures. Mean and standard deviation (SD) of binary cross-entropy (BCE) reconstruction loss are calculated over five random seeds.	52
6.2	$F_1$ score for all models and dimensionality reduction configurations. Best performance for each DR technique is in <b>bold</b> .	54
6.3	Precision, recall, $F_1$ score, AUC, and accuracy for the best configuration of each model. For random forest and XGBoost, performance was highest without dimensionality reduction. For the MLP, the best performance was obtained using autoencoder features with a bottleneck size of 32. Metrics are reported separately for the two classes (“Canceled” and “Not canceled”).	56

6.4	Optimal hyperparameters per model across dimensionality reduction setups. . . . .	59
B.1	Eigenvalues and explained variance of the principal components in PCA. . . . .	72
B.2	Eigenvalues and explained variance of the principal components in MCA. . . . .	72
B.3	$p$ -values of pairwise McNemar's test with FDR correction. Values $\geq 0.001$ are shown with two decimals; smaller values are shown in scientific notation. Significance: * $p < 0.05$ , ** $p < 0.01$ , *** $p < 0.001$ . Lower triangle only. . . . .	74



# Chapter 1

## Introduction

The concept of hospitality dates back thousands of years, with early evidence found in ancient civilizations such as Mesopotamia and Egypt. Taverns along trade routes provided basic shelter and refreshment for merchants and pilgrims, introducing the earliest examples of organized lodging [32]. As human societies grew and developed, the hospitality industry changed alongside, constantly adapting to new cultures and technologies. While the fundamental idea of offering a place to rest remains unchanged, the modern hospitality sector has grown into a profit-oriented domain.

According to data insights of Statista [33], the US hotel industry is forecasted to reach \$443.07 billion in revenue in 2025 with an annual growth rate of 3.68%. Despite this growth, high booking cancellation rates remain a challenge for the sector. A 2024 report by D-Edge [34] estimates that cancellation rates range from 18% to 42% annually. Major contributors to increased cancellations were the rise of online booking platforms and dynamic pricing, which have made it easier for travelers to reserve multiple accommodations simultaneously and cancel last minute once better deals are found [37]. Beyond leaving rooms unoccupied, cancellations also result in significant revenue losses, reduced operational efficiency, and decreased visibility on Online Travel Agencies (OTA) platforms [55]. Therefore, accurately predicting which bookings are likely to be canceled can reduce uncertainty and increase revenue [6] by enabling more informed decision making, such as sending reminder messages to guests or offering up-selling options.

Due to the limited availability of real hotel data, most research on cancellation prediction relies on data from the aviation industry [8]. Hoteliers who use data-driven approaches to produce accurate revenue forecasts rely on methods that are more extensively researched, such as demand forecasting [54, 61], which will be further discussed in Chapter 2. While these strategies are widely used in revenue management, they often rely on assumptions, and decisions are usually made manually.

As more industries pursue smart data utilization, the hotel industry is also exploring how to make better use of its data [54]. Booking data stored in the Property Management System (PMS) often contains hidden patterns relevant to cancellation prediction. According to Domingos (2012) [23], the most important element in modeling is the features used. However, sometimes the patterns in the raw data are insufficient for learning, and more features have to be derived for an accurate prediction.

As seen in previous studies [6, 7], feature engineering often results in many ad-

ditional features, and while it can significantly enhance model performance, it also introduces redundancy and noise [23]. Studies have shown that dimensionality reduction can improve model performance, reduce overfitting, and enhance training efficiency in classification tasks [5, 19, 23]. This motivates the use of dimensionality reduction in the feature space, such as principal component analysis (PCA) [43], to only retain informative features. Although PCA is a well-known and widely used technique, its linear property limits its ability to capture complex relationships. Moreover, it only works on numeric features and is not suitable for a dataset containing categorical features.

With the rise of deep learning, manual feature engineering is no longer necessary. However, deep neural networks are often considered black-box models, making them less suitable when working with sensitive booking data due to concerns around transparency and accountability. Moreover, hotel booking datasets are tabular and small, which are both underdeveloped domains in the deep learning field that do not outperform traditional methods [49]. Autoencoders offer a middle ground: while they do not fully overcome the transparency characteristics of neural networks, they are well-suited for handling high-dimensional, noisy data [3, 10]. Their architecture compresses input data into a lower-dimensional space through a bottleneck layer, effectively learning a compact representation, though this comes at the expense of less interpretability. This latent representation can then be used for purposes such as data compression, feature extraction, or denoising.

While most existing research applies traditional machine learning techniques such as decision trees, support vector machines, or PCA, the use of dimensionality reduction with deep learning for cancellation prediction remains unexplored. This reveals a gap in the application of more advanced, deep learning-based methods for dimensionality reduction. Therefore, the objective of this research is to determine whether using autoencoders for dimensionality reduction in the feature space can improve the prediction of hotel booking cancellations. This thesis compares the predictive performance of models trained on latent features learned by an autoencoder, models using traditional dimensionality reduction techniques, and models using raw features. The research question becomes: ***How do performance metrics (e.g.,  $F_1$  score) compare when predicting hotel booking cancellations using features learned with autoencoders, compared to traditional dimensionality reduction methods and models without any dimensionality reduction?***

To help answer the research question, the following subquestions are formed:

1. What type of autoencoder architecture (e.g., basic, denoising, variational) yields the best features for cancellation prediction?
2. How does the size of the latent space impact the predictive performance when using autoencoders?
3. Can the learned latent space reveal meaningful patterns related to booking cancellations?

The remainder of this thesis is organized as follows: Chapter 2 reviews the relevant literature on cancellation prediction and dimensionality reduction techniques. Chapter 3 details the dataset and the preprocessing steps undertaken. Chapter 4 describes the methods employed for this thesis. Chapter 5 and Chapter 6 present

the experimental setup and results, respectively. Chapter 7 offers a discussion of the findings and provides recommendations. Finally, Chapter 8 summarizes the main conclusions of the study.

# Chapter 2

## Literature review

In this chapter, the literature is discussed. Section 2.1 discusses key drivers of cancellations and current methods to mitigate them. Section 2.2 reviews prior research on cancellation prediction. Sections 2.3 and 2.4 review dimensionality reduction techniques, including both statistical and deep learning methods. Finally, Section 2.5 concludes with a summary of key insights.

### 2.1 Cancellation behavior in the hospitality industry

Tourism is an extremely dynamic and reactive industry, influenced by global events, economic situations, and traveler preferences. In the hospitality industry, cancellations are a common problem, affecting revenue, occupancy planning, and customer experience [55]. Despite growing access to data, many hoteliers still rely primarily on business rules or personal intuition rather than predictive models [35]. Therefore, understanding the factors leading to booking cancellations is crucial for effective revenue management.

Several factors contribute to cancellations. The paper by Hajibaba et al. (2016) [31] investigates possible responses to large-scale disruptions, such as political instability, natural disasters, or terrorism. In these situations, the preferred response is often to relocate tourists to alternative accommodations farther from the affected area to maintain guest satisfaction and safety.

While such extreme cases are rare, they do highlight the vulnerability of the industry. Day-to-day cancellations are far more common, and although each cancellation is driven by different reasons, they can typically be placed in the categories ‘hotel’, ‘customer’, ‘booking’, or ‘external’ factors. Antonio et al. (2019) [7] provided an overview of factors related to cancellation. They stated that hotel-related factors include the variety of facilities, star category, location, and brand recognition. Customer factors typically include age group, customer type, market segment, distribution channel, gender, and country of origin. Booking factors include price, length of stay, lead time, party size, time of the year, day of the week, events, and cancellation policy. External factors could include recommendations by a third party (e.g., travel agent, company, or family), social reputation, competitors’ prices, special events, and weather. Understanding these influential features is required for developing accurate predictive strategies.

Hoteliers currently use methods such as demand forecasting [54, 61], overbooking predictions [6, 44], or dynamic pricing [2, 27] to produce accurate revenue forecasts and have also been the focus of extensive academic research. In the following paragraphs, a small overview of the three methods is given.

### 2.1.1 Demand forecasting

Demand forecasting methods are used in multiple industries. Not only limited to where booking-based applications are used, but also in the supply chain industry to forecast the number of orders. Demand forecasting mostly consists of traditional forecasting methods that are statistic-based regression methods, such as time series, ARIMA, winter, or Holt regression [61]. Another example is the thesis of Van Leeuwen (2024) [54], where he used cubic smoothing splines in combination with linear programming to model demand. In more recent years, Mediavilla et al. (2022) [40] stated that research has shifted towards machine learning models (e.g., decision trees), either as standalone approaches or in combination with statistical methods.

### 2.1.2 Overbooking

Overbooking and no-show prediction are closely related practices. To mitigate potential revenue losses, hotels often accept more bookings than their actual capacity, anticipating that the made reservations will have no-shows, last-minute cancellations, or modifications. However, this approach requires careful planning to determine the optimal overbooking level. Phumchusri and Maneesophon (2014) [44] used a mathematical approach to determine the optimal number of rooms to overbook when a hotel offers two different room types. However, the challenge with this approach is that hotels typically offer a wide variety of room types (such as suites, double, single, etc.), which makes it difficult to model this mathematically.

Note that overbooking strategies are primarily focused on capacity and rely on historical demand trends, assuming that a certain proportion of bookings will cancel or result in no-shows. In contrast, cancellation prediction seeks to identify which specific bookings are likely to be canceled, relying more on guest characteristics and booking context.

### 2.1.3 Dynamic Pricing

Dynamic pricing is a revenue management strategy that allows hotels to adjust room rates in response to real-time fluctuations in demand and occupancy levels to maximize Revenue per Available Room (RevPAR) [27]. This approach is widely used in industries with perishable inventory and fluctuating demand, such as airlines, sports events [30], and the energy sector [24]. As a result, prices can vary significantly for the same booking details, such as stay period or room type, depending on the timing of the reservation. Hotels leverage a variety of factors to adjust prices, including tangible attributes (e.g., room size, the presence of spas or business facilities), reputational attributes (e.g., star ratings, guest reviews), and contextual factors (e.g., booking time, competitive pricing) [2]. Research has shown that booking time plays a crucial role in price adjustments, helping hotels distinguish between leisure and business travelers and set prices accordingly.

## 2.2 Previous research on cancellation prediction

Discussions with various hotels revealed that, due to limited resources, cancellation prediction models or other data science techniques are often not implemented. Smaller hotels, in particular, may lack the means to maintain high-quality data or the expertise to develop and manage predictive models. As a result, managers often rely on intuition rather than data when making overbooking or pricing decisions. Nevertheless, some researchers have explored how far machine learning can go in forecasting cancellations, mostly using traditional machine learning techniques.

In the study by Antonio et al. (2019) [7], PMS data with additional external data sources were used for cancellation prediction on eight different hotels. A total of 37 variables were used, coming from a PMS, weather forecasts, national and local holidays, and social reputation from online views. The final dataset had 12 features directly usable from the data sources and 25 engineered features. The rationale behind including external data sources is the idea that these features might positively contribute to the accuracy [23]. Four XGBoost models, each with different input features, were created to assess performance. Models 1 and 2 were trained solely on PMS data, but on different time periods: from January 2016 to November 2017 and from August 2016 to November 2017, respectively, to determine the model performance on a reduced number of observations. Model 3 incorporated all available data sources (PMS, weather, holidays, and social reputation) over the full time period, and Model 4 was specifically trained and tailored for two individual hotels and is therefore not considered in this thesis. The results showed that Model 2, which only used PMS data, achieved an average accuracy of 0.8268, slightly outperforming Model 3 with an average accuracy of 0.8256, which did include external sources. The precision was much lower than the accuracy, with an average of 0.45 across all models. This suggests that incorporating external data is not necessarily beneficial and that high-quality feature engineering using PMS data alone may be more valuable. Furthermore, in the feature importance analysis, it was found that only 13 to 15 features were actively used by the models, depending on the hotel. Notably, all of these features originated from the PMS system, while external data sources were ignored. Although the importance scores varied across models, features such as lead time, country, deposit (yes/no), week stay (yes/no), distribution channel, and booking changes (integer) were generally deemed most important.

Interestingly, earlier work by the same authors had already demonstrated that high accuracy could be achieved using PMS data alone, without incorporating any external sources. The study by Antonio et al. (2017) [6] used booking data from four hotels located in the region of the Algarve, Portugal. The data were extracted from their PMS and consisted of basic variables such as arrival date, room type, guest characteristics, and length of stay. The researchers conducted minimal feature engineering and used a total of 32 features. The authors used a boosted decision tree, decision forest, decision jungle, locally deep support vector machine, and a neural network to predict cancellations. Unfortunately, the paper does not further specify the models, e.g., the architecture of the neural network. The boosted decision tree and Decision Forest performed best with accuracies above 0.90 for each hotel. The models had larger variability in the  $F_1$  score, ranging between 0.639 and 0.927. The

results showed that features extracted from PMS are proficient at predicting cancellations with high accuracy. Interestingly, not every feature had the same order of importance or contribution to the cancellation prediction for each hotel. For example, the nationality of guests was important across all hotels, but its importance varied, and the feature *required parking space* was only important for hotels 1 and 4. In general, hotel location, services, facilities, nationality of guests, markets, and distribution channels were deemed important.

Building on this idea, Sánchez-Medina and C-Sánchez (2020) [48] aimed to create a more interpretable model on booking cancellations by using only 13 commonly available PMS features (e.g., number of adults, advance payment (yes/no), number of nights, source of booking, check-in date, average price). The advantage of using a reduced number of features is the possibility to understand why and what type of customers cancel. Four models were compared: C5.0, random forest, support vector machine (SVM), and an artificial neural network (ANN) that was optimized using a genetic algorithm. The results show that the ANN performed best, followed by Random Forest and C5.0, which had similar performance, while the SVM performed the worst. Their respective accuracies were 0.980, 0.804, 0.790, and 0.753. The same order was observed for the  $F_1$  score and precision, whereas recall and AUC showed greater variation between C5.0 and Random Forest. Unfortunately, no analysis of feature importance or model details was provided.

Together, these studies suggest that well-structured PMS data, even without external variables, can provide sufficient predictive power for hotel cancellation forecasting.

In contrast to the studies discussed above, which focus on binary cancellation prediction using PMS data, the work by Morales and Wang (2010) [41] takes a different approach by forecasting cancellation rates throughout the booking horizon  $t$ . 14 variables, of which 11 are nominal (e.g., channel, company, day, market, month, rate code, refundable, room type), and three numeric features (length of stay, time booking, and price) are used. The models evaluated include average cancellation rate (AVG), seasonally averaged rate (SAVG), logistic regression (LR), decision tree (DT), minimum squared expected error tree, random forest (RF), SVM, and kernel logistic regression. Since the objective was to forecast the cancellation rate (i.e., continuous target), model performance was assessed using Total Absolute Error (TAE). The results showed that the best-performing model and features vary depending on the booking horizon  $t$  (i.e., time-dependent). Overall, SVM, kernel LR, and RF achieved the lowest TAE values (719.9, 733.9, and 723.3, respectively), while SAVG performed the worst with a TAE of 937.6.

Beyond the hospitality sector, cancellation prediction has also been explored outside the hotel industry. For instance, Huang et al. (2013) [35] used a feedforward neural network trained with backpropagation (BPN) and general regression neural networks (GRNN) to study reservation cancellations in the restaurant sector. Reservation data from 1,400 customers was collected from a Western restaurant chain in Taiwan in 2011. Of these reservations, 251 were canceled (17.93%). The dataset included 12 features, grouped into temporal features (e.g., month), guest characteristics (e.g., age, income), and historical behavior (cancellation record and



cumulative number of cancellations). The BPN achieved higher AUC (74.86%) and sensitivity (80.87%), while the GRNN performed better on specificity (87.14%).

In summary, a variety of machine learning models, feature sets, and application domains have been explored by previous research on cancellation prediction. While some studies incorporated externally sourced features, evidence shows that relying only on variables available from internal systems like Property Management Systems (PMS) can be equally effective, suggesting that the quality and relevance of input features may be more important than data volume or model complexity alone. Furthermore, studies conducted in related fields (e.g., restaurants) demonstrate how applicable cancellation prediction methods are to a variety of industries. Across hotel and restaurant datasets, models such as decision trees, random forests, support vector machines, and neural networks have all shown promise. These studies demonstrate the importance of data quality, feature selection, and contextual understanding in building effective and practical predictive models for reservation cancellations.

While feature engineering can improve accuracy [23], incorporating multiple sources of information or performing extensive feature engineering can substantially increase the number of variables, leading to high-dimensional datasets. This creates additional modeling challenges that can be addressed through dimensionality reduction techniques. The next section discusses commonly used statistical methods for reducing dimensionality while preserving the most relevant information.

## 2.3 Statistical approaches for dimensionality reduction

One of the main challenges in high-dimensional datasets is the **curse of dimensionality**, a term introduced by Bellman in 1961 [11], which describes how generalization in models becomes exponentially harder as the dimensionality of the input space (i.e., the number of variables used in prediction) increases. In high dimensions, data points become sparse, distance measures lose their meaning, and the amount of data required to train a model effectively grows exponentially. As a result, many algorithms that perform well in low-dimensional spaces struggle or become computationally infeasible in higher dimensions. To reduce this effect, dimensionality reduction techniques can be used to transform the dataset into a lower-dimensional representation that seeks to preserve as much relevant information as possible. The following sections review some commonly used statistical methods for dimensionality reduction on numerical and categorical data.

### 2.3.1 Numerical data

Common approaches for dimension reduction on numerical data include linear techniques such as Principal Component Analysis (PCA), as well as nonlinear extensions like Kernel PCA (KPCA). A comprehensive review of dimensionality reduction techniques and the mathematical principles underlying them is provided in the study by Sorzano et al. (2014) [52]. Although their citation analysis covered studies from 2003 to 2012, the survey remains relevant since techniques like PCA, KPCA, and



manifold learning continue to be applied in modern applications. Their analysis revealed a clear increase in the popularity of manifold learning approaches, such as ISOMAP, Locally Linear Embedding, and Laplacian Eigenmaps, due to their ability to model nonlinear relationships and adapt to local data structure. This growth has partially come at the expense of earlier nonlinear PCA variants and Self-Organizing Maps. Nevertheless, component-based methods, particularly PCA and its variants (e.g., KPCA), remain among the most widely applied techniques because of their simplicity, interpretability, and computational efficiency.

While Sorzano et al. (2014) [52] summarizes trends in method usage, Cao et al. (2003) [19] empirically compared the performance of PCA, ICA, and KPCA across three different time series benchmark datasets (sunspot time series, Santa Fe, financial time series) using an SVM regressor. For each experiment, four SVM models were trained: one using the original data without dimensionality reduction, and three using features extracted via PCA, ICA, and KPCA. Model performance was evaluated using the normalized mean square error (NMSE). Across all three experiments, SVM models trained on features extracted from dimensionality reduction techniques outperformed those trained on the original input data, suggesting that dimensionality reduction increases the model’s ability to capture complex patterns in the data. In particular, KPCA and ICA consistently performed better than PCA, likely due to their ability to extract nonlinear features, in contrast to the linear transformations of PCA.

As the field progressed, researchers began comparing a broader range of dimensionality reduction techniques. The study by van der Maaten et al. (2007) [53] compared several nonlinear dimensionality reduction techniques, including manifold learning and multilayer autoencoders, with PCA. They discussed 12 techniques: (1) Multidimensional scaling, (2) Isomap, (3) Maximum Variance Unfolding, (4) Kernel PCA, (5) Diffusion maps, (6) Multilayer autoencoders, (7) Locally Linear Embedding, (8) Laplacian Eigenmaps, (9) Hessian LLE, (10) Local Tangent Space Analysis, (11) Locally Linear Coordination, and (12) Manifold charting. The methods were compared by evaluating the generalization errors in classification tasks, and experiments were conducted on five real-world standard, publicly available datasets: MNIST, COIL20, NiSIS, ORL, and HIVA. Unfortunately, these datasets were not traditional tabular data, but rather image datasets that have been converted into numerical format using pixel values, except for the HIVA dataset, which is a drug discovery dataset consisting of 3,845 samples with 1,617 features and two target classes. The results showed that PCA outperformed all nonlinear techniques on three of the five datasets, suggesting that nonlinear approaches have not yet consistently surpassed traditional PCA for dimensionality reduction. Kernel PCA and autoencoders showed strong performance across most datasets, leading the authors to anticipate further development of these methods as a promising balance between performance and computational feasibility. However, as this study was conducted in 2007, the evaluation of autoencoders reflects the capabilities of that period, when deep learning architectures, optimizers, loss functions, and regularization techniques were far less advanced than today. Thus, the reported performance of autoencoders in this study likely underestimates their potential with modern implementations.

A more recent analysis by Chang (2025) [20] emphasizes that the most suitable dimensionality reduction technique depends on dataset characteristics, the need for interpretability, computational constraints, and the specific objectives of the analysis. Despite the limitations in capturing non-linear structures, PCA remains a strong choice for tabular data due to the simplicity, interpretability, and computational efficiency. Earlier work by Cao et al. (2003) [19] and Van der Maaten et al. (2007) [53] provided empirical evidence for this perspective, by demonstrating that PCA often performed comparably to, and in some cases outperformed, more complex nonlinear dimensionality reduction techniques. Given the structure of the dataset in this thesis and the objective of developing transparent and scalable models, PCA is therefore selected as the preferred statistical dimensionality reduction technique.

### 2.3.2 Categorical data

In addition to numerical features, many tabular datasets also contain categorical variables that cannot be directly processed by techniques like PCA. A common approach to reduce the dimensionality of categorical data is Multiple Correspondence Analysis (MCA), an extension of Correspondence Analysis (CA), which itself is used to analyze relationships between two categorical variables [1]. Similar to PCA, MCA identifies latent dimensions that aim to explain as much variability in the data as possible, but it does so using a chi-square metric and is based on an indicator matrix created from one-hot encoding [28]. While MCA is beneficial for its interpretability, it can be sensitive to rare categories and noise.

The study by Bera et al. (2023) [12] presented a comprehensive evaluation of 13 categorical dimensionality reduction techniques, including one-hot encoding, feature hashing, Latent Semantic Analysis (LSA), Latent Dirichlet Allocation (LDA), PCA, MCA, and variational autoencoders. These methods were assessed on seven benchmark clustering datasets, including text corpora (e.g., NIPS papers, Enron Emails, NYTimes articles), Gisette handwriting dataset, and a biological dataset of mouse brain cells. These datasets had extremely high dimensionality, ranging from 5,000 to 1.3 million features, and different levels of sparsity (0.07% to 30%). The findings showed that MCA offers a trade-off between efficiency and interpretability and is substantially faster than the majority of the other techniques, particularly compared to one-hot encoding. Although more complex methods, such as variational autoencoders, achieved higher accuracy, they tend to be most successful in settings with extremely high dimensions. In contrast, booking datasets, such as those discussed in Section 2.2, have a considerably lower dimensionality.

Another technique for dimensionality reduction is Factor Analysis of Mixed Data (FAMD), which combines both PCA and MCA into a single analysis. It applies PCA to the numerical features and MCA to the categorical features simultaneously, ensuring that both types contribute equally to the resulting components. The study by Visbal-Cadavid et al. (2020) [56] applied FAMD in an educational context where the main objective is to derive clusters based on the resulting principal components. The study focused on the interpretation of the clusters by examining how variables contributed to their formation. For example, the authors analyzed which variables

were most important for each cluster and related these to university performance, noting that institutions with greater resource availability tend to rank higher. Although this provides qualitative insights, it limits the ability to objectively assess the effectiveness of the dimensionality reduction or clustering.

While the studies above used purely statistical methods, Guo and Berkhahn (2016) [29] proposed using embeddings for categorical variables. They argued that the continuous nature of neural networks makes them poorly suited for categorical inputs, particularly when categories are represented as integers. To address this, the authors introduced learning vector representations (i.e, embeddings) of categorical variables directly within the neural network. Rather than relying on one-hot or integer encodings, each category was mapped to a trainable vector of real numbers optimized during supervised training, allowing categories with similar effects on the outcome to be closer together in the embedding space. Their method was applied to the Rossmann Store Sales dataset from Kaggle, which includes seven categorical features with a total number of 1,017,210 records. By mapping categories into a continuous embedding space, similar values are placed closer together, capturing latent relationships between them. This approach not only reduces memory usage and training time compared to one-hot encoding but also improves generalization, especially in sparse datasets or when statistical assumptions are weak. Moreover, when these learned embeddings are used as input features for other models, such as k-nearest neighbors, random forests, and gradient boosted trees, they significantly improve predictive performance. For instance, mean absolute percentage error (MAPE) for random forests improved from 0.167 to 0.089, and for gradient boosted trees from 0.122 to 0.071, while neural networks maintained a MAPE of 0.070 with and without embeddings.

Each of these approaches have their own strengths. Entity embeddings are particularly powerful for large-scale datasets with many high-cardinality features, while FAMD is well-suited for mixed-type data where both numerical and categorical features are prominent. However, given the moderate dimensionality and limited cardinality of the current dataset, MCA offers a practical and interpretable solution for categorical dimensionality reduction in this research.

## 2.4 Autoencoders for dimensionality reduction

While Section 2.3 reviewed statistical methods for dimensionality reduction, such as PCA and MCA, deep learning-based approaches are increasingly explored. Although alternatives such as Restricted Boltzmann Machines [58] or specialized architectures like EVNet [62] have been proposed, autoencoders remain the dominant deep learning approach for dimensionality reduction. They are frequently included in studies, such as Bera et al. (2023) [12] and Van der Maaten et al. (2007) [53], which were already discussed in the previous section, as comparison against other methods. Therefore, this thesis will focus on autoencoders as a deep learning-based technique for dimensionality reduction.

Autoencoders have gained significant attention due to their ability to learn non-linear, hierarchical representations directly from the data [3, 10]. Unlike linear techniques such as PCA, autoencoders use neural networks designed to learn efficient, compressed representations of input data by minimizing reconstruction error.

This makes them particularly useful for high-dimensional datasets in which patterns may not be linearly separable. This ability has led to their growing use, particularly in domains involving unstructured data such as computer vision and natural language processing [10, 59]. However, it remains uncertain whether this success translates to other data types, specifically tabular data, since deep neural networks have struggled to consistently outperform traditional machine learning techniques for prediction on such structured data [49]. The remainder of this section reviews some studies that use autoencoders for dimensionality reduction with tabular data.

The study by Bank et al. (2021) [10] provides a review of different architectures and applications of autoencoders. A notable experiment was the standard (vanilla) autoencoder used to generate compressed representations, which was subsequently used as input to a clustering algorithm (e.g., K-means). Since clustering algorithms often suffer from the curse of dimensionality, the low-dimensional latent representation produced by autoencoders is preferred. As noted in the paper, a limitation of using vanilla autoencoders for clustering is that “the embeddings are trained solely for reconstruction and not for the clustering application” [10], meaning the latent representations are not explicitly optimized to enhance cluster separation.

While Bank et al. (2021) [10] provides a broad overview of autoencoder architectures and their potential for unsupervised learning, the work by Alkhayrat et al. (2020) [5] compared the effectiveness of two dimensionality reduction approaches: PCA and autoencoders, to improve customer segmentation on an extensive telecom customer dataset. Although PCA was included as a baseline, the main contribution lay in applying autoencoders for dimensionality reduction. The dataset consisted of 220 behavioral, service, and demographic features from 100,000 customers. With such a large feature space, there is a greater chance that columns will be correlated to each other and therefore be redundant. To address this curse of dimensionality, the authors compared the effectiveness of two dimensionality reduction approaches before applying k-means clustering for customer segmentation. PCA was tested with 10, 20, 30, and 50 components, and it was found that 20 components explain 90% of the variance. The authors implemented a symmetric encoder-decoder autoencoder, with mean squared error as the loss function, Adam optimizer with learning rate 0.001, and ReLU activations throughout. Several autoencoder architectures were tested with bottleneck dimensions of 20, 30, and 50. After obtaining a reduced space, the authors conducted k-means clustering with varying numbers of clusters and compared the quality using the silhouette score. Results showed that both PCA and autoencoders improved clustering quality compared to the original dataset, with the autoencoder achieving the best silhouette score of 0.682 (3 clusters) compared to 0.581 for the original data and 0.476 for PCA. The comparison with PCA illustrates how autoencoders can outperform traditional techniques.

Apart from the use of autoencoders for unsupervised learning in clustering, their utility can also be tested in a supervised classification setting. The study by Volovăţ et al. (2024) [57] forecasted tumor recurrence following Gamma Knife radiosurgery for patients with brain metastases with autoencoder-derived features. The dataset consisted of 77 patients and 13 variables covering socio-demographic, clinical, treatment, and radiosurgery factors, with the target variable indicating either tumor

regression (cured) or tumor progression (not cured). The dataset was notably small and imbalanced, with only six cases in the not-cured class, and to address this, the authors applied SMOTE to generate a more balanced dataset. Two types of autoencoders were designed: one without compression (meaning the bottleneck layer retained the same number of neurons as the input), and one with compression (reducing the latent space to half the number of features). Both architectures had symmetric encoder-decoder structures, batch normalization, and LeakyReLU activations. These learned features were then used as input to various classifiers, including Logistic Regression, SVM, KNN, Random Forest, Extra Trees, and XGBoost. Results showed that compressed autoencoder features generally improved classification performance compared to both the original data and non-compressed representations, particularly for SVM and Logistic Regression, where the accuracies increased from 0.85 to 0.96 and from 0.91 to 0.94, respectively. XGBoost performed consistently well across all feature sets with an accuracy of 0.94, while Extra Trees showed overfitting with both accuracy and  $F1$  score being 1. The findings suggest that models trained on compressed autoencoder features generally outperform the original feature set and the non-compressed latent representations. However, the small sample size and number of variables are major limitations and should be considered when interpreting these findings.

In conclusion, these studies demonstrate that learning compressed representations that retain essential information (i.e., autoencoders for dimensionality reduction) can enhance downstream performance across various tasks and domains.

## 2.5 Concluding remarks

This chapter reviewed prior research on cancellation prediction and dimensionality reduction techniques. Ensemble methods, particularly XGBoost, consistently demonstrate high performance in cancellation prediction tasks and are valuable for uncovering the drivers behind cancellations [41] due to their interpretability. Moreover, recent studies by Shwartz-Ziv and Armon (2021) [49] and Borisov et al. (2024) [14] showed that ensemble methods are often preferred over deep neural networks for tabular data, and often outperform them in terms of generalization and robustness.

Traditional dimensionality reduction techniques such as PCA and MCA are widely used due to their simplicity, transparency, and computational efficiency. However, their linear nature limits their ability to capture complex, nonlinear patterns in the data. Autoencoders address this limitation by their ability to learn nonlinear representations, and previous studies have shown that their latent spaces can improve performance in both clustering [5, 10] and classification [57] tasks. However, autoencoders introduce additional model complexity, reduce interpretability, and often require more computational resources. Their benefits are most evident in high-dimensional or highly nonlinear datasets.

In summary, while linear techniques like PCA and MCA offer interpretability and speed, autoencoders present a promising alternative for capturing more complex structures in the data. Given the moderate dimensionality and mixed feature types of the booking dataset used in this research, directly comparing these approaches is valuable to assess whether the added complexity of nonlinear methods translates into performance improvements.

Technique	Category	Strengths	Limitations
PCA	Statistical	Simple implementation, fast, interpretable, useful for visualization	Only numeric data, assumes linear relationships
MCA	Statistical	Suitable for categorical data, interpretable, useful in survey or questionnaire analysis	Sensitive to rare or sparse categories
Autoencoder	Deep learning	Captures complex and non-linear patterns, suitable for high-dimensional data, flexible architectures (deep, sparse, variational, etc.)	Requires tuning, less interpretable, computationally intensive, performance varies with dataset size and noise

Table 2.1: Summary of dimensionality reduction techniques reviewed in literature

Study	Data Description	Features	Models Used	Key Findings
<i>Task: binary cancellation prediction</i>				
Antonio et al. (2017)[6]	PMS data from four hotels in Algarve, Portugal	32 (demographics, booking details)	Boosted Decision Tree, Decision Forest, Decision Jungle, LD-SVM, NN	<b>Performance:</b> Boosted DT and Decision Forest performed best. <b>Feature insights:</b> varying importance across hotels.
Antonio et al. (2019)[7]	PMS data from eight Portuguese hotels + external data	37 (booking details, engineered booking features, weather, holidays, social reputation)	XGBoost (4 variants)	<b>Performance:</b> External data didn't improve performance; PMS-only model with engineered features performed best. <b>Feature insights:</b> top features were lead time, country, deposit (yes/no), week stay (yes/no), distribution channel, and booking changes (int).
Sanchez-Medina et al. (2020)[48]	PMS data from a hotel in Gran Canaria	13 (demographics, payment and booking details)	RF, C5.0, SVM, ANN (optimized with GA)	<b>Performance:</b> ANN achieved highest accuracy, followed by RF and C5.0, while SVM performed the worst. <b>Feature insights:</b> N/A.
Huang et al. (2013)[35]	Restaurant reservations from Taiwan	12 (demographics, reservation details)	BPN, GRNN	<b>Performance:</b> BPN outperformed GRNN on AUC and sensitivity; GRNN better on specificity. <b>Feature insights:</b> N/A.
<i>Task: time-specific cancellation rates</i>				
Romero Morales & Wang (2010)[41]	personal name records (PNR) data	14 (only booking details)	AVG, SAVG, LR, DT, minimum squared error tree, RF, SVM, kernel LR	<b>Performance:</b> Best-performing model varied over time $t$ . Overall, SVM and RF showed strong performance. <b>Feature insights:</b> relative importance of variables is time-dependent.

Table 2.2: Overview of prior studies on cancellation prediction. Note that the study by Morales and Wang (2010) [41] is on cancellation rates and not binary cancellation prediction. Sánchez-Medina and C-Sánchez (2020) [48] and Huang et al. (2013) [35] did not analyze the variables.



# Chapter 3

## Data

This chapter provides an overview of the dataset used in this thesis. The dataset is obtained from a Kaggle competition that has been active for over two years and was initiated in response to the growing number of hotel reservations being canceled or that result in no-shows. The objective is to develop a predictive model that can forecast whether a guest will cancel their reservation. Section 3.1 describes the dataset at a high level, Section 3.2 outlines the feature engineering steps, and Section 3.3 presents the exploratory data analysis.

### 3.1 Data description

This thesis utilizes booking records from a hotel’s Property Management System (PMS) found on Kaggle<sup>1</sup>, which is inspired by the data presented in a study on hotel booking demand by Antonio et al. (2019) [8]. The Kaggle dataset differs in several aspects: it does not include all variables from the original study. Additionally, while Antonio et al. (2019) [8] stated that their data originated from two Portuguese hotels, the Kaggle dataset provides no information about the specific hotel or its location. Furthermore, the study mentions that the data ranges from 2015 to 2017, while the Kaggle dataset covers the period from 2017 to 2018. Therefore, the exact relationship between the two datasets cannot be verified.

The dataset consists of 36,275 unique bookings and contains 19 variables covering the period from July 1, 2017, to December 31, 2018. The dataset was already cleaned, and no missing values or outliers were found in the data. Table 3.1 presents the features with explanations, and Table 3.2 describes the categorical variables.

32

### 3.2 Feature engineering

To enrich the dataset, several new features are engineered based on booking and stay characteristics. Prior work by Antonio et al. (2017) [6], Antonio et al. (2019) [7], Morales and Wang (2010) [41], and Van Leeuwen (2024) [54] provides the inspiration for these features, while also giving evidence of their predictive value.

From the three original arrival date columns, the exact *arrival date* in YYYY-MM-DD format and the corresponding day of the week are computed. Van Leeuwen

---

<sup>1</sup><https://www.kaggle.com/datasets/ahsan81/hotel-reservations-classification-dataset/data>



Feature	Type	Description
BookingID	String	Unique identifier of each booking
Number of adults	Numeric	Number of guests
Number of children	Numeric	Number of children
Number of weekend nights	Numeric	Amount of weekend nights the guest booked to stay at the hotel
Number of week nights	Numeric	Amount of week nights (Monday to Friday) the guest booked to stay at the hotel
Type of meal plan	Category	Type of meal plan booked by the guest
Required car parking space	Boolean	Does the customer require a car parking space?
Room type reserved	Category	Type of room reserved by the guest
Lead time	Numeric	Number of days between the date of booking and the arrival date
Year of arrival	Numeric	Year of arrival date
Month of arrival	Numeric	Month of arrival date
Date of arrival	Numeric	Date of arrival date
Market segment type	Category	Market segment designation
Repeated guest	Boolean	Is the customer a repeated guest?
Number of previous cancellations	Numeric	Number of previous bookings that were canceled by the guest prior to the current booking
Number of previous bookings not canceled	Numeric	Number of previous bookings not canceled by the guest prior to the current booking
Average price per room	Numeric	Average price per day of the reservation; prices of the rooms are dynamic. (in euros)
Number of special requests	Numeric	Total number of special requests made by the customer (e.g., high floor, view from the room, etc.)
Booking status (target)	Boolean	Flag indicating if the booking was canceled (1) or not (0)

Table 3.1: Overview of features in the dataset.

Categorical variable	Descriptions
Meal plan	Not Selected – No meal plan selected Meal Plan 1 – Breakfast Meal Plan 2 – Half board (breakfast and one other meal) Meal Plan 3 – Full board (breakfast, lunch, and dinner)
Market segment type	Aviation – Person in the aviation industry (e.g., pilot or air hostess) Complementary – Complimentary stay Offline – Booking done in person (private booking) Online – Booking done online (private booking) Corporate – Booking via a corporate forum or platform
Room type reserved	Values range from 1 to 7, encoded by INN Hotels to refer to room categories (e.g., suite, deluxe, economy)

Table 3.2: Descriptions of categorical variables in the dataset.

(2024) [54] showed that guest type often varies by day of arrival, with business guests typically arriving and departing during weekdays (Monday–Thursday) and leisure guests arriving on weekends (Friday–Sunday). Similarly, Antonio et al. (2017) [6] and Morales and Wang (2010) [41] found that temporal variables such as month and day of the week, while sometimes modest in individual importance, can become more predictive when combined with other variables.

Additional variables are constructed to capture stay characteristics, such as *length of stay*, *total number of guests*, and two binary indicators: *includes weekend* and *travel with kids*. Van Leeuwen (2024) [54] reported that weekend stays and the presence of children are associated with different amenity usage patterns and booking behaviors, while Antonio et al. (2019) [7] found that the number of children was a consistently useful predictor across models.

Bookings were categorized into *one night stay*, *short stay*, and *long stay* following the thresholds used by Van Leeuwen (2024) [54], corresponding to stays of 1 night, 2–3 nights, and more than 3 nights, respectively. Likewise, from booking lead time, two categorical indicators are created to flag *last-minute bookers* (less than 3 days in advance) and *early bird bookers* (more than 45 days in advance). Bookings made between these timeframes are considered “regular” and are not assigned a specific category. Van Leeuwen (2024) [54] notes that the categorization can provide the model with additional context useful for prediction.

Two revenue-related features, *total spend* and *spend per guest*, are added to capture booking value. Van Leeuwen (2024) [54] noted that spending patterns could be used to distinguish between guest segments, which in turn may influence cancellation behavior. Finally, an interaction term between lead time and stay length is introduced to capture potential nonlinear relationships between the timing of a booking and the duration of stay.

The engineered features and their derivations are summarized in Table 3.3, and the enriched dataset is then used for further analysis.

Feature	Type	Description
Day of the week	String	Name of the weekday corresponding to the arrival date
Length of stay	Numeric	Total number of nights stayed (weekend + week nights)
Number of guests	Numeric	Total number of guests (adults + children)
Includes weekend	Binary	Indicator of whether the stay includes a weekend night
Room night stay	Binary	Bookings for single-night stays
Short stay	Binary	Bookings for short stays (2–3 nights)
Long stay	Binary	Bookings for long stays ( $> 3$ nights)
Last-minute booker	Binary	Bookings made $< 3$ days in advance
Early-bird booker	Binary	Bookings made $> 45$ days in advance
Travel with kids	Binary	Bookings with at least one child
Total spend	Numeric	Total estimated revenue per booking (nights $\times$ average room price)
Spend per guest	Numeric	Estimated cost per guest for the full stay (total spend / number of guests)
Lead time per night	Numeric	Interaction term (lead time $\times$ number of nights)

Table 3.3: Overview of engineered features added to the dataset. These derived variables were constructed to capture booking patterns (e.g., stay duration, spending behavior).

### 3.3 Data exploration

After the feature engineering, the dataset consists of 32 variables. An exploratory data analysis (EDA) is performed to gain insights into the distribution of variables, identify patterns, and understand characteristics that may influence booking cancellations.

The dataset includes at least one booking for each day within the date range. However, 37 entries contain an invalid date (February 29, 2018), which does not exist. In addition, 78 entries have zero guests, with the average room price also being equal to zero. A further 139 entries have no adults in the booking but do include children. Given that most hotels do not allow minors to rent rooms without an adult and that the number of guests should not be zero, these entries, along with those containing invalid dates, are assumed to be errors and are removed from the dataset. Theoretically, they could represent add-ons to another booking containing adults, as noted in the study that provided this dataset [8]. However, the dataset does not provide a way to verify this, as each booking has a unique identifier and there is no mechanism to link records. Therefore, it is unclear whether these are supplementary bookings or data entry errors. The final dataframe consists of 36,021 rows.

**Categorical variables.** The three categorical variables are shown in Figure 3.1, and additionally, the categorical variables room type and market segment type are summarized in Table 3.4. For market segment type, the majority of bookings are

made directly by guests, either online or offline, accounting for more than 93% of all reservations. The remaining segments (Corporate, Complementary, and Aviation) together make up less than 10%. Room type 1 is by far the most booked, followed by Room Type 4, Room Type 6, and Room Type 2. Room types 3, 5, and 7 make up slightly more than 1% of all bookings. Particularly interesting is Room Type 3, which is only booked seven times. The second least common room type, Type 7, is mostly booked for complementary guests. The aviation category is booked exclusively in Room Types 1 and 4. Meal Plan 1, which includes only breakfast, is the most popular option. Meal Plan 3 is chosen in less than 1% of the bookings.

	Market segment type					Total
	Aviation	Complementary	Corporate	Offline	Online	
Room Type 1	60 (<1)	236 (<1)	1828 (5.07)	9,738 (27.03)	16,176 (44.91)	28,038 (77.84)
Room Type 2	0	16 (<0.1)	2 (<0.01)	57 (<1)	482 (1.34)	557 (1.54)
Room Type 3	0	2 (<0.01)	1 (<0.01)	2 (<0.01)	2 (<0.01)	7 (<0.1)
Room Type 4	65 (<1)	49 (<1)	99 (<1)	612 (1.70)	5,215 (14.48)	6,040 (16.78)
Room Type 5	0	17 (<0.1)	73 (<1)	80 (<1)	93 (<1)	263 (<1)
Room Type 6	0	14 (<0.1)	3 (<0.01)	23 (<0.1)	920 (2.55)	960 (2.67)
Room Type 7	0	39 (<1)	5 (<0.1)	5 (<0.1)	107 (<1)	156 (<1)
Total	125 (<1)	373 (1.04)	2011 (5.58)	10,517 (29.20)	22,995 (63.84)	36,021 (100)

Table 3.4: Contingency table room type vs. market segment type. Percentages (in brackets) are normalized by the total number of bookings in the dataset.

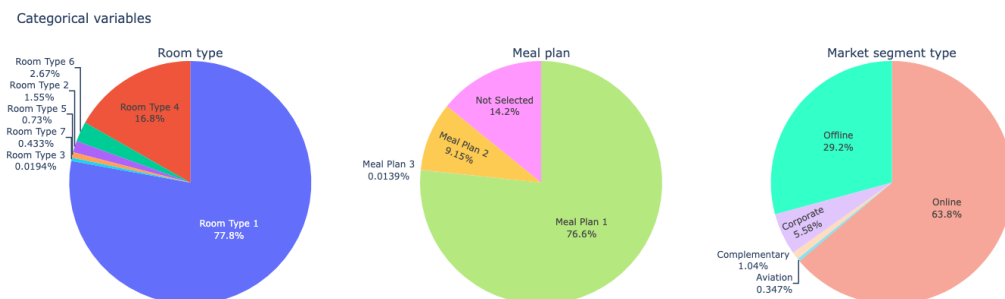


Figure 3.1: The ratio of the categorical variables room type, meal plan, and market segment type.

**Length of stay.** Figure 3.2 shows the distribution of length of stay per market segment, with clear differences observed between segments. For example, the corporate segment, which is likely to consist of business travelers, tends to have shorter stays and low variability. The complementary segment also has relatively short stays, possibly indicating promotional or influencer stays. Aviation-related bookings tend to span only a few days with few outliers, which is consistent with the short layover durations typical for pilots and flight attendants. The two most traditional booking channels, online and offline bookings, which are private bookings, show different patterns as well. Offline bookings tend to have a shorter average length of stay and

less variability compared to online bookings. Furthermore, online bookings have many more outliers than offline bookings.

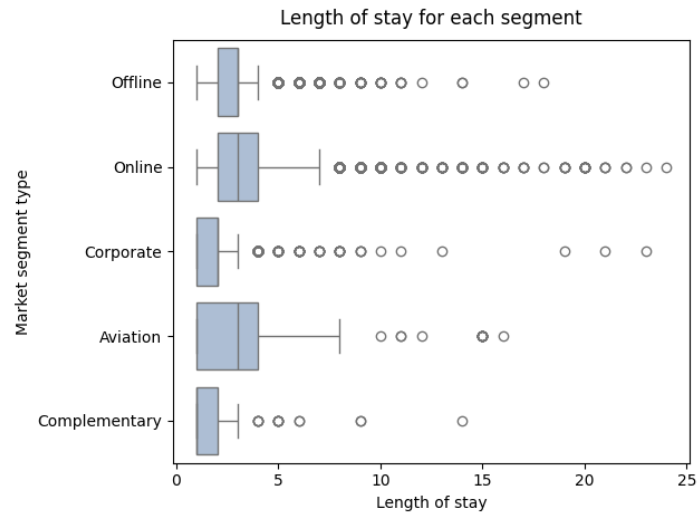


Figure 3.2: Distribution of length of stay for each segment.

**Lead time.** Figure 3.3 shows the distribution of lead time, defined as the number of days between the booking date and the arrival date. The histogram reveals a right-skewed distribution, indicating that the majority of bookings are made close to the arrival date. Approximately 66.9% of all bookings are made within 100 days of arrival, and there is a particularly steep increase in bookings within the final 50 days. The last bin, representing same-day or short-term bookings (up to 15 days), contains the highest count with over 8,000 bookings. On the other hand, bookings made more than 200 days in advance are rare. This booking behavior is important when assessing cancellation risks and designing overbooking strategies.

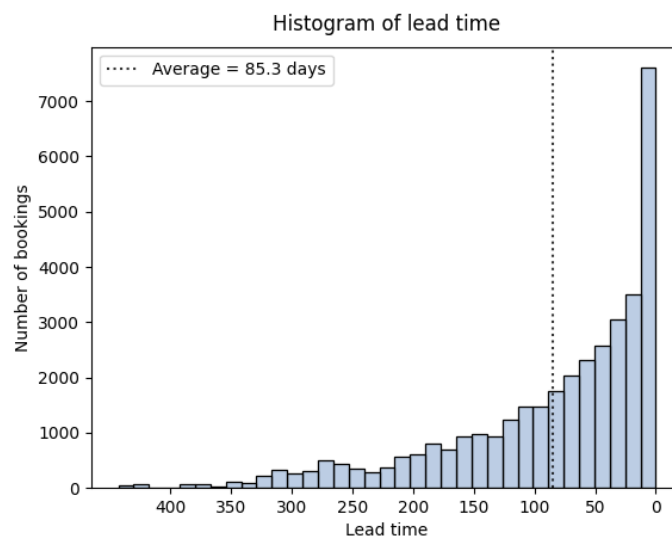
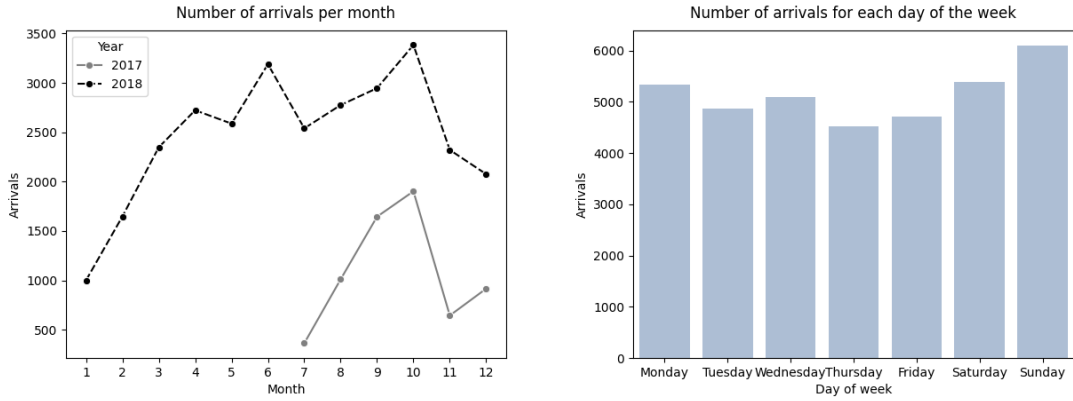


Figure 3.3: Distribution of the lead time. Note: a booking horizon of 0 means last last-minute booking.

**Seasonality.** The dataset covers approximately 1.5 years of data. Figure 3.4 shows the number of bookings per month and day of the week, revealing strong seasonality. Reservations are relatively low in January, but they steadily increase until April. Bookings peak in June before dropping in July again. For both 2017 and 2018, the bookings increase from July to October, indicating a clear summer trend, before decreasing afterwards. Interestingly, the number of bookings in 2017 is consistently lower than in 2018, which may be due to increased demand.

The day-of-week seasonality in Figure 3.4b remains constant over the weekdays. Sunday has a slight increase, while Thursday has the fewest arrivals.



(a) Number of arrivals per month

(b) Number of arrivals per day of the week

Figure 3.4: Seasonality in the dataset.

**Average spending.** Figure 3.5 presents the distribution of the average nightly room price per room type. The pricing appears to follow dynamic pricing strategies, with wide ranges and outliers in most room types. Room Type 1, the most booked room type, as shown in Table 3.4, contains a notable outlier exceeding €500 per night. In contrast, Room Type 7 has the lowest first quartile price of €0, indicating that 25% of its bookings are made at no cost. Despite this, it has a relatively high median price, which may suggest the use of promotional offers or price variability across different booking periods. This is consistent with Table 3.4, which shows that Room Type 7 is more frequently booked by the Complementary segment. A quick data examination reveals that the Complementary segment can have associated costs, possibly due to a meal plan or making specific requests. Overall, the average price tends to increase with the room types, except for Types 3 and 7, implying an ordering in room type. Room Type 3 does not follow the trend, but it is selected only seven times in the dataset and is therefore not representative, whereas Type 7 is often complementary, explaining the deviation. This could provide additional booking context, for example, Room Type 4 may show lower cancellation rates due to the link with the aviation industry, and for Room Type 7, due to complementary offers or higher costs.

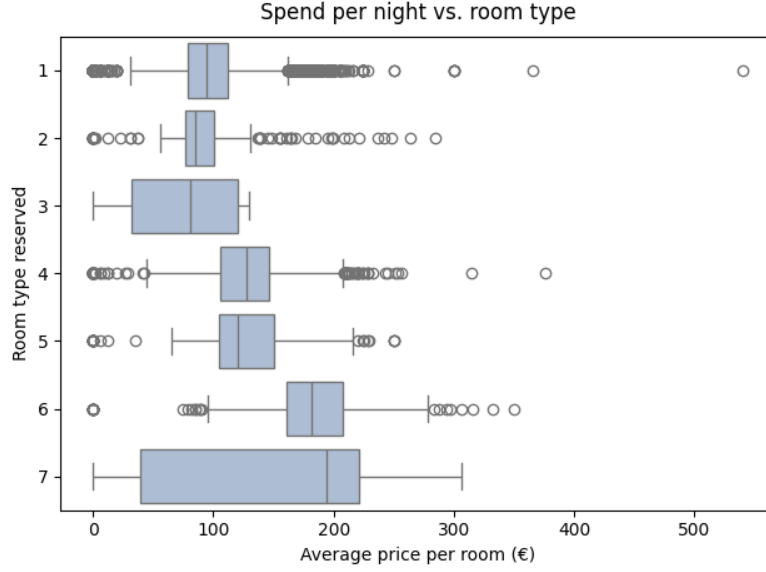


Figure 3.5: Average price per night for each room.

**Booking status.** The target variable in this study is the *booking status*, which indicates whether a reservation is canceled or not. Table 3.5 displays the distribution of canceled and non-canceled bookings across three categories: guest type, booking type, and stay type. Guest type distinguishes between new and repeated guests. Booking type includes early bird bookers (those who reserve more than 45 days in advance) and last-minute bookers (who book within 3 days of arrival). Stay type is categorized based on the duration of the booking: one-night stays (1 night), short stays (2–3 nights), and long stays (more than 3 nights). The formal definitions are also provided in Table 3.3. Overall, the dataset consists of 24,189 total bookings that are not canceled and 11,832 bookings that are canceled. Nearly all bookings are made by new guests, with the total number of repeated guests being less than 1000. A guest is marked as repeated if the booking was linked to an existing profile created prior to the booking date [8]. Most bookings are made more than 45 days in advance. This is consistent with what was observed in Figure 3.3. Regarding the stay type, short stays are the most common in the dataset.

Figure 3.6 shows the distribution of canceled and non-canceled bookings across guest type, booking type, and stay type. Approximately two-thirds of bookings are not canceled, indicating a moderately imbalanced dataset, though not extreme. Across the categories, the ratio of canceled to non-canceled bookings is relatively similar, indicating that these features may not strongly influence cancellation behavior. One clear exception is repeated guests, who rarely cancel. Although the overall number of repeated guests is low, it is intuitively plausible that returning customers are more loyal to their bookings.

Category	Subcategory	Not canceled	Canceled
Total		24,189 (67.2%)	11,832 (32.8%)
Guest type	Repeated guest	907 (2.5%)	15 (< 0.1%)
	New guest	23,282 (64.6%)	11,817 (32.8%)
Booking type	Last minute booker	2,036 (5.7%)	930 (2.6%)
	Early bird booker	13,438 (37.3%)	6,585 (18.3%)
Stay type	One night	4,385 (12.2%)	2,148 (6.0%)
	Short stay	12,269 (34.1%)	6,047 (16.8%)
	Long stay	7,370 (20.5%)	3,548 (9.9%)

Table 3.5: Absolute number of canceled and not canceled bookings across multiple categories. Percentages (in parentheses) are normalized by the total number of bookings in each category. Note that guest type and stay type categories sum to 100%, while booking type does not, as it excludes bookings with a lead time between 3 and 45 days.

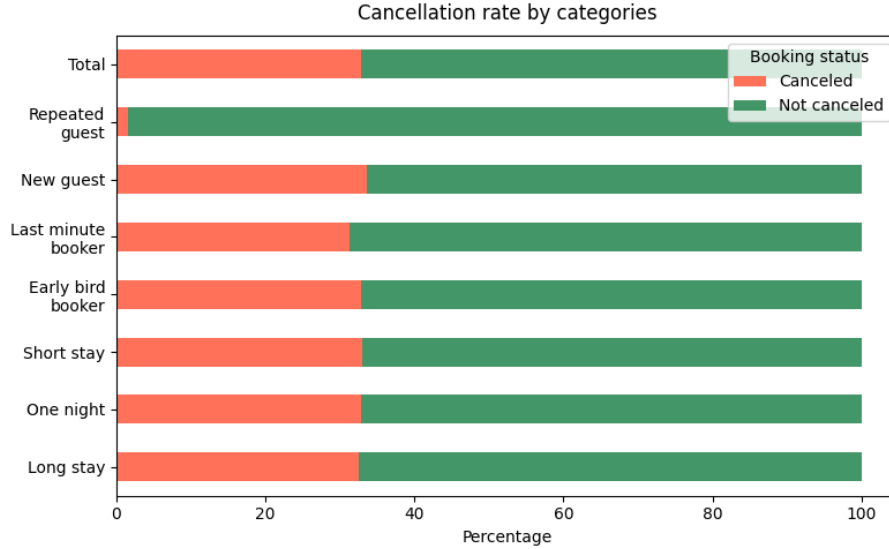


Figure 3.6: Ratio of canceled vs. not canceled bookings in the dataset for total bookings and for the categories guest type, booking type, and stay type. Red indicates canceled bookings, and green indicates not canceled.

The cancellation policy (e.g., penalties) is unknown, as neither the original paper [8] nor the Kaggle competition provides details. However, prior research suggests that cancellation policies can substantially impact both revenue and customer behavior [37]. Hotels often offer two rate types: a free cancellation rate, which provides the guest flexibility at a higher price, and a non-refundable rate, which offers a lower price in exchange for a booking commitment. Both cancellation options can be beneficial; free cancellation increases profit through upselling, while non-refundable rates reduce last-minute cancellations and help secure revenue through penalties.



# Chapter 4

## Methods

In this chapter, the different methods used in this thesis are discussed. Section 4.1 covers the dimensionality reduction techniques: Principal Component Analysis, Multiple Correspondence Analysis, and autoencoders. Section 4.2 discusses the models: random forest, XGBoost, and a multilayer perceptron. Furthermore, Bayesian optimization for hyperparameter optimization is briefly introduced in Section 4.3.

### 4.1 Dimensionality reduction methods

This thesis compares statistical and deep learning approaches to dimensionality reduction. Two statistical techniques are applied: Principal Component Analysis (PCA) (Section 4.1.1) and Multiple Correspondence Analysis (MCA) (Section 4.1.2), which reduce numerical and categorical feature spaces, respectively. In addition, autoencoders are employed as a deep learning method to reduce both numerical and categorical feature spaces (Section 4.1.3).

#### 4.1.1 Principal Component Analysis

Principal Component Analysis (PCA) is one of the most commonly used techniques for dimensionality reduction and was introduced by Karl Pearson in 1901 [43]. The idea behind PCA is that features with higher variance are assumed to carry more information, while low-variance features may reflect noise or redundancy [47]. PCA transforms a set of potentially correlated variables into a new set of uncorrelated variables called principal components (PCs), through linear combinations of the original variables. These components are ordered in such a way that the first few retain most of the variation present in the original variables. For example, if a dataset consists of  $n$  variables, PCA will produce  $n$  principal components, with the first capturing the largest possible variance, the second capturing the next largest (while being orthogonal to the first), and so on. This enables dimensionality reduction while preserving the dataset's most significant patterns.

Within this thesis, PCA is applied as a dimensionality reduction technique for continuous variables. Because it relies on variance as Euclidean distance, it is only suitable for continuous variables, and applying it to categorical variables (even when numerically encoded) may lead to misleading results. Its ability to capture the majority of variance in fewer components helps reduce noise and potential multicollinearity in the input space. As supported by earlier studies reviewed in Sec-

tion 2.3, it serves as a strong baseline for comparison with more complex non-linear methods such as autoencoders. Below a short explanation of PCA is given.

PCA begins by standardizing the variables  $Z = \frac{X - \mu}{\sigma}$  so that each feature contributes equally to the analysis, independent of its original scale (e.g., height in cm, salary in euros). From this standardized data, a covariance matrix  $\mathbf{C}$  is computed to capture how variables relate to one another. When two features are highly correlated, they may introduce redundancy by providing similar information.

PCA then finds new orthogonal directions in the data, called principal components, that maximize variance. In practice, this means that PCA is seeking directions in which the data points show the greatest spread, under the assumption that directions with high variance capture the underlying structure of the data. Mathematically, these directions correspond to the eigenvectors of the covariance matrix, and the amount of variance captured along each direction is given by the associated eigenvalue. This corresponds to solving the eigenvalue equation:

$$\mathbf{C}\mathbf{v} = \lambda\mathbf{v} \quad (4.1)$$

where  $\mathbf{v}$  is a principal component and  $\lambda$  is its explained variance (i.e., the eigenvalue).

The eigenvectors are ranked by their eigenvalues, with the first principal component capturing the most variance, the second capturing the next largest amount of variance (while being orthogonal to the first), and so forth. By selecting the top  $p$  components (which are often those explaining a chosen percentage of the total variance), the original dataset is projected into a lower-dimensional space, where  $p < n$ . For additional mathematical details, see Appendix A.1.

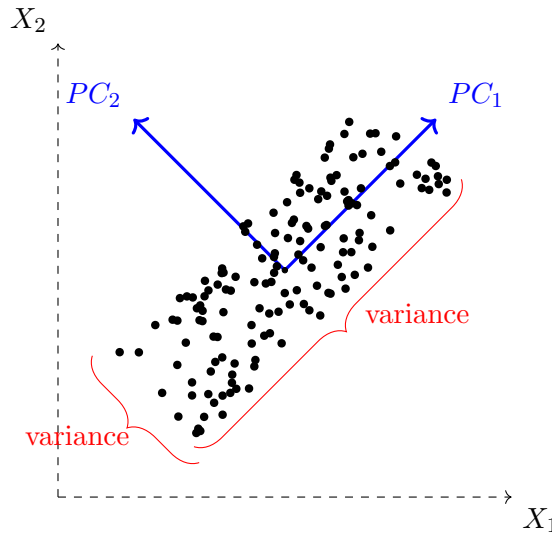
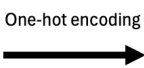


Figure 4.1: Illustration of PCA. The original coordinate system is transformed into new orthogonal axes ( $PC_1$ ,  $PC_2$ ), which are rotated to capture directions of maximum variance in the data. The scattered points represent the distribution of observations, with variance along each principal component indicated by red braces.  $PC_1$  captures the largest share of variance, while  $PC_2$  captures the smaller variance.

### 4.1.2 Multiple Correspondence Analysis

Multiple Correspondence Analysis (MCA) is a dimensionality reduction technique specifically designed for categorical data. It generalizes Correspondence Analysis (CA) to handle multiple categorical variables simultaneously, uncovering patterns and associations between them. Conceptually, MCA serves a similar role for categorical data as PCA does for continuous data, by producing orthogonal components with associated eigenvalues and percentages of explained variance [17]. As discussed in Section 2.3, MCA has proven effective in revealing latent structures in categorical data, making it the preferred dimensionality reduction method for categorical features in this thesis.

Before applying MCA, the categorical data is transformed into a numerical format suitable for matrix operations. Each category level is converted into a binary indicator variable through one-hot encoding, resulting in an indicator matrix (or complete disjunctive table) where each column corresponds to a category level and each row to an observation.



Index	Market segment type
1	Online
2	Complementary
3	Online
4	Aviation
5	Corporate

Index	Aviation	Complementary	Corporate	Offline	Online
1	0	0	0	0	1
2	0	1	0	0	0
3	0	0	0	0	1
4	1	0	0	0	0
5	0	0	1	0	0

Figure 4.2: Illustration of the indicator matrix, which is the same as one-hot encoding.

This indicator matrix is then converted into a correspondence matrix by dividing each entry by the total counts, producing relative frequencies. The row and column sums of this matrix give the marginal distributions, which are used to center and normalize the data. This step removes the influence of expected frequencies under the assumption of independence and ensures equal contribution of all categories, preventing domination by those with large marginal frequencies. From the centered and normalized data, MCA identifies principal components by applying Singular Value Decomposition (SVD):

$$\mathbf{S} = \mathbf{P}\mathbf{\Delta}\mathbf{Q}^T \quad (4.2)$$

Here,  $\mathbf{\Delta}$  is a diagonal matrix containing the singular values  $\delta$ , and the squared singular values  $\delta^2$  correspond to eigenvalues that indicate the proportion of inertia (analogous to variance in PCA) explained by each component. The SVD also provides coordinates for both observations and categories in the reduced component space. These coordinates allow for the interpretation of relationships between observations and categorical variables in fewer dimensions.

MCA uses the  $\chi^2$  distance to measure dissimilarities between observations and between categories. This metric reflects the extent to which an observation or category deviates from the average profile, taking into account the frequency distribution of the data. By incorporating both observed and expected frequencies, the  $\chi^2$  distance is sensitive to associations involving both rare and common categories, making MCA particularly effective at uncovering subtle relationships in categorical data.

## Component selection

For PCA and MCA, a critical step is deciding how many components to retain. According to the study by Brown (2009) [16], researchers generally rely on five methods for component selection in PCA. Two approaches are Kaiser’s stopping rule, which retains only components with eigenvalues greater than 1, and the cumulative percentage of explained variance, where a threshold (e.g., 80-95%; with  $n$  features,  $n$  principal components account for 100% of the variance) is chosen to determine how many components to retain. However, as Brown notes, there is no universally accepted cut-off point, and the choice often depends on the context and objectives of the analysis.

MCA differs from PCA in that it rarely produces a small set of dominant components, since categorical variables are transformed into a sparse indicator matrix. As a result, the total variance is spread thinly across many dummy variables, meaning that each component explains only a small fraction of the variance. By default, MCA eigenvalues are  $\leq 1$  (since they are derived from normalized  $\chi^2$  distances), further limiting the variance explained by individual components. Intuitively, this is expected, since categorical features do not provide strong continuous directions of variation in the same way as numerical features [1]. In practice, researchers often apply the same selection criteria as for PCA (e.g., explained variance thresholds or scree plots) [17].

### 4.1.3 Autoencoders

Autoencoders are a type of neural network specifically designed for unsupervised representation learning and are often used for dimensionality reduction. Before discussing autoencoders in detail, a short introduction to the neural network architecture is provided.

**Introduction to the neural network architecture.** An artificial neural network (ANN) is a machine learning model inspired by the structure of the human brain. The most common ANN is the multilayer perceptron (MLP), which was designed by Rosenblatt in 1958 [46]. An ANN typically consists of an input layer that receives raw features, followed by one or more hidden layers that perform feature transformations, and an output layer. In each neuron, the inputs are combined through a weighted sum and then transformed using a non-linear activation function, after which the result is passed to the next layer. Activation functions allow the network to capture non-linear patterns. They can differ per layer, but typically hidden layers use a ReLU (Rectified Linear Unit) function  $A(x) = \max(0, x)$  due to its benefits in avoiding vanishing gradients and encouraging sparse activations. The depth (i.e., number of layers) and width (i.e., number of neurons per layer) determine the model’s capacity. Deeper networks can learn complex patterns but are also more prone to overfitting. Formally, a neural network layer can be written as:

$$\mathbf{h}^{(l)} = \phi(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}) \quad (4.3)$$

where  $\mathbf{h}^{(l)}$  is the output of layer  $l$  with  $\mathbf{h}^{(0)}$  being the input vector  $\mathbf{x} \in \mathbb{R}$ ,  $\mathbf{W}$  and

$\mathbf{b}$  are trainable weights and biases, and  $\phi(\cdot)$  is a nonlinear activation function (e.g., ReLU, tanh).

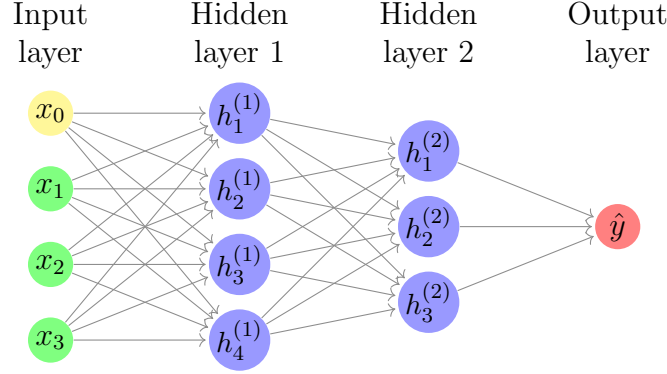


Figure 4.3: Illustration of a multilayer perceptron (MLP) with three input neurons, two hidden layers, and one output neuron. Each neuron computes a weighted sum of the outputs from the previous layer, adds a bias term, and applies a nonlinear activation function as defined in Equation 4.3.

The role of the output layer depends on the task at hand: in supervised learning, it produces a classification or regression output, while in autoencoders, it generates a reconstruction of the original input. Weights  $\mathbf{W}$  and biases  $\mathbf{b}$  are trained using backpropagation, which computes the gradients of a loss function  $\mathcal{L}$  numerically using the chain rule, with respect to each parameter. Then, at each iteration  $t$ , the weights are updated according to:

$$\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}_t} \quad \mathbf{b}_{t+1} \leftarrow \mathbf{b}_t - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{b}_t} \quad (4.4)$$

where  $\eta$  is the learning rate controlling the step size. If  $\eta$  is too high, training may diverge or oscillate, while if too low, training may converge slowly or get stuck in a suboptimal minimum.

**Autoencoders.** Building on the general neural network framework, an autoencoder is a specific type of neural network trained not to predict labels, but to reconstruct the input. Their architecture consists of two main parts: an encoder, which converts the input data into a lower-dimensional latent space (also called a bottleneck), and a decoder, which reconstructs the original input from this compressed representation. The bottleneck thus provides a way to perform dimensionality reduction. Unlike PCA and MCA, autoencoders can capture more complex patterns from the data due to the use of the non-linear activation functions. The architecture of an autoencoder is shown in Figure 4.4.

Following Bank et al. (2021) [10], the learning process of an autoencoder can be expressed as learning two functions, the encoder  $A : \mathbb{R}^n \rightarrow \mathbb{R}^p$  and the decoder  $B : \mathbb{R}^p \rightarrow \mathbb{R}^n$ , such that the reconstruction error is minimized:

$$\arg \min_{A, B} \mathbb{E}_{\mathbf{X}} [\Delta(\mathbf{X}, (B \circ A)(\mathbf{X}))] \quad (4.5)$$

where  $\mathbf{X} \in \mathbb{R}^{m \times n}$  represents the input data with  $m$  observations and  $n$  features and  $\Delta$  is the reconstruction loss function measuring the difference between the original

input and the reconstructed output of the decoder. Common choices for the loss function include the mean squared error (MSE) for continuous data and binary cross-entropy (BCE) for data scaled between 0 and 1. The latter loss function aligns well with binary inputs, by treating each value as the probability of an “on” or “off” state [22]. In combination with the BCE loss, the decoder typically uses the sigmoid activation function  $\sigma = \frac{1}{1+e^{-x}}$  to bound the output to  $[0, 1]$ . Since the features in this dataset vary in scale, they are standardized to the range  $[0, 1]$ , and therefore the model is trained to minimize the binary cross-entropy loss:

$$\mathcal{L} = -\frac{1}{m} \sum_{i=1}^m [x_i \log \hat{x}_i + (1 - x_i) \log(1 - \hat{x}_i)] \quad (4.6)$$

with  $m$  being the number of observations,  $x_i$  is the input value, and  $\hat{x}_i$  is the reconstruction of the  $i^{th}$  observation.

Theoretically, any type of autoencoder (e.g., sparse, denoising, or variational autoencoder) can be used for dimensionality reduction since the reduction takes place in the bottleneck layer, where the latent space has fewer dimensions than the original input [10]. However, depending on the use case, some variants may be more suitable. For instance, denoising autoencoders are trained by corrupting the input with random noise and learn to reconstruct the original clean input. This is particularly useful for image data or signal denoising tasks, where the autoencoder can remove noise from unseen corrupted input samples. Another example are variational autoencoders, which is a probabilistic generative model that learns the underlying distribution to create entirely new samples that resemble the training data. However, in most studies applying autoencoders purely for dimensionality reduction [5, 53, 57], the specific architecture is often not explicitly stated (see Section 2.4). Given the relatively small size and moderate complexity of the dataset in this thesis, a standard (“vanilla”) autoencoder is deemed sufficient for extracting compressed representations. Therefore, the remainder of this thesis focuses exclusively on the vanilla autoencoder. The following subsections describe each of the autoencoder components in more detail.

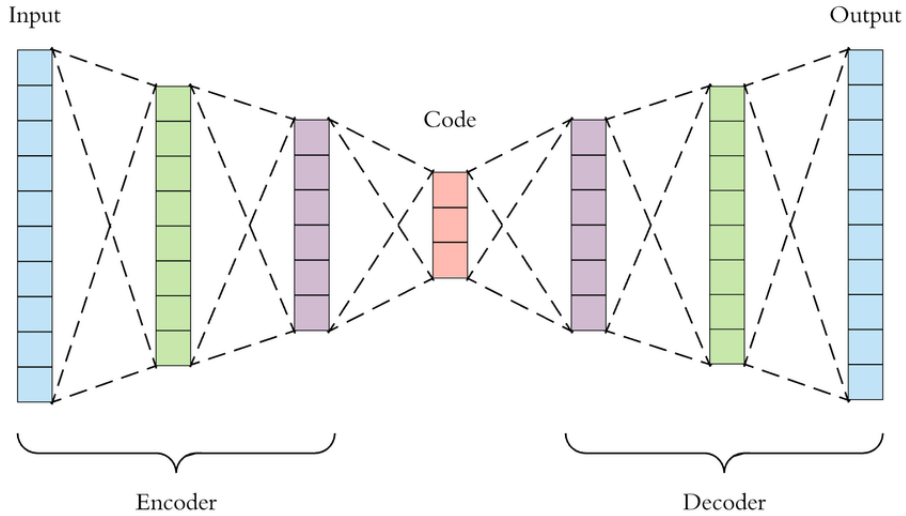


Figure 4.4: Autoencoder architecture. The bottleneck layer is called “Code”. Source: [Paper](#).

**Encoder.** The encoder is designed to transform the input  $\mathbf{x} \in \mathbb{R}^n$  to a latent representation  $\mathbf{z} \in \mathbb{R}^p$ , where typically  $p \ll n$ . As illustrated in Figure 4.4, the input passes through one or more hidden layers (green and purple layers) that apply affine transformations followed by non-linear activation functions (e.g., ReLU or tanh). These transformations capture the underlying structure in the data while reducing the dimensionality. For structured tabular data, the encoder can operate directly on numerical features and on transformed categorical variables (e.g., one-hot encodings). Note that when only linear activation functions are used, the encoder behaves similarly to PCA [10, 53].

**Latent space (Bottleneck).** The bottleneck layer or latent space  $\mathbf{z} = A(\mathbf{X})$ , contains the most compact representation of the input data. By constraining this layer to have a lower dimensionality than the input, the autoencoder enforces an “information bottleneck”, encouraging the network to retain only the most relevant patterns while discarding noise and redundancy. The size of this layer directly influences the balance between compression and reconstruction quality. Beyond dimensionality reduction, the latent representation can also be used for tasks such as anomaly detection and clustering [10].

**Decoder.** The decoder takes the latent representation  $\mathbf{z}$  and attempts to reconstruct the original input  $\hat{\mathbf{X}} = B(\mathbf{z})$ . The decoder architecture typically mirrors that of the encoder, but in reverse, gradually expanding the dimensions until they match the original input size. The reconstructed output  $\hat{\mathbf{X}}$  is then compared to the original  $\mathbf{X}$  using the chosen reconstruction loss function, and the resulting error is propagated backward through the network. As in other neural networks, the weights of both the encoder and decoder are updated jointly via gradient descent to minimize this loss, enabling the model to learn a latent representation that captures the most informative features.

## 4.2 Model selection

The performance of the dimensionality reduction techniques is tested on three different models. Two ensemble models, random forest and XGBoost, are chosen due to their high performance in cancellation prediction [7, 48] and strong performance against deep models [14, 49]. To ensure a fair comparison of dimensionality reduction techniques, in addition to ensemble models, a multilayer perceptron (MLP) or feedforward neural network is included.

### 4.2.1 Random forest

Random forest (RF) [15] is an ensemble learning method that combines multiple decision trees to perform classification or regression tasks. Each individual decision tree consists of a sequence of decision rules that recursively split the feature space based on threshold conditions, resulting in interpretable tree structures. However, single decision trees are prone to overfitting, especially when they grow deep and memorize training data [15]. Random forests try to avoid this by averaging the predictions of multiple (deep) decision trees, each trained on different bootstrap



samples of the training set, to reduce variance and improve generalization. They do this by introducing two types of bagging:

### 1. Random sampling with replacement

Given a training set  $\mathcal{X} = x_1, \dots, x_m$  with corresponding labels  $\mathcal{Y} = y_1, \dots, y_m$ , multiple bootstrap samples  $B$  are drawn with replacement. Each sample  $X_b, Y_b$  is used to train a separate tree  $f_b$ . Since each tree is trained on a different subset of the data, this introduces variability across the ensemble.

### 2. Feature bagging

At each split of the tree, a random subset of features is considered rather than evaluating all possible features. This encourages diversity among the trees and reduces correlation between them.

The final prediction of the ensemble is typically obtained through majority voting for classification tasks, or averaging for regression tasks. An illustration of the random forest algorithm is shown in Figure 4.5.

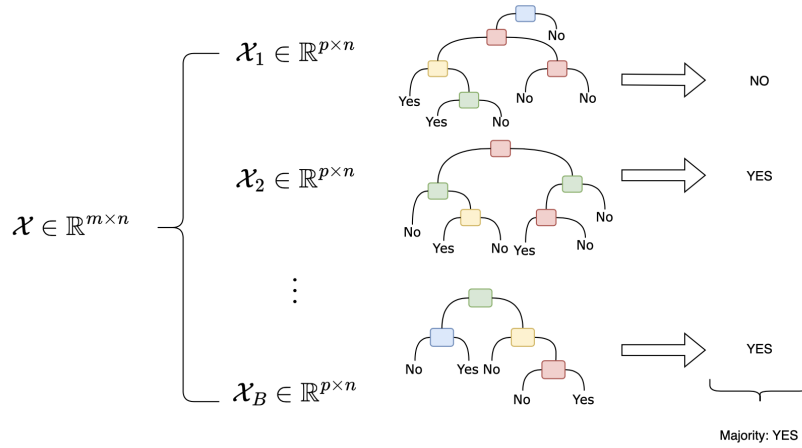


Figure 4.5: Illustration of the random forest algorithm. Each tree is trained on a bootstrap sample and selects a random subset of features at each split.

Random forests are particularly well-suited for structured tabular data, are robust to outliers and noise, and require minimal preprocessing. Their performance can be fine-tuned through several hyperparameters, which will be discussed in the next section.

## Parameters

Although the default parameters of RF often perform reasonably well, it is worthwhile to tune certain parameters to improve performance [45].

The number of trees (`n_estimators`) determines how many trees are grown in the forest. Increasing the number of trees generally leads to better performance by reducing variance. However, for larger datasets, the performance tends to converge, and a relatively low number of trees (around 128) may already be sufficient [42].

The number of features considered at each split (`max_features`) influences the correlation between trees. Using fewer features reduces the correlation, which can



improve overall stability. Moreover, the likelihood of the model repeatedly picking the same variables with a stronger relationship to the response variable reduces, allowing weaker predictors that may carry information for specific subgroups. On the contrary, choosing weaker variables could lead to less informative splits, lowering the average performance of the trees. The default value is  $\sqrt{n}$ , where  $n$  is the total number of features.

The three parameters (`max_depth`), (`min_samples_split`), and (`min_samples_leaf`) are often tuned alongside each other. The `max_depth` determines the maximum depth permitted for each decision tree. By default, trees grow until all leaves are pure, no further splits are possible because too few samples remain (set by parameter `min_samples_split`), or no variables are left to split on. Limiting depth lowers model complexity but may also cause underfitting. On the other hand, overly deep trees risk memorizing the training data and thus overfit. Increasing `min_samples_split` (by default set to 2) forces the tree to have more samples before forming a new branch, which generally results in shallower trees and helps reduce overfitting. Note that both parameters reduce tree depth, but in different ways. The minimum number of samples required to form a leaf node is set by `min_samples_leaf`. A higher number can reduce overfitting by preventing the creation of overly small, potentially noisy leaves, leading to simpler and more generalizable trees. By default, this value is 1, meaning that leaves can be formed from single samples.

## 4.2.2 XGBoost

eXtreme Gradient Boosting (XGBoost) [21] is an ensemble technique based on gradient boosting. Unlike random forest, which builds trees independently and combines the predictions through bagging, gradient boosting builds trees sequentially, with each tree attempting to correct the prediction errors of the ensemble built so far.

Given a training set  $\mathcal{X} = x_1, \dots, x_m$  with corresponding labels  $\mathcal{Y} = y_1, \dots, y_m$ , XGBoost starts from an initial prediction  $\hat{y}^{(0)}$  and adds a new decision tree  $f_t$  at each iteration  $t$  to minimize a given loss function. The model update can be expressed as:

$$\hat{y}^{(t)} = \hat{y}^{(t-1)} + f_t(x) \quad (4.7)$$

where  $\hat{y}^{(t-1)}$  is the prediction from the previous iteration and  $f_t(x)$  is the output of the  $t^{\text{th}}$  tree. The final prediction is given by:

$$\hat{y} = \sum_K f_k(\mathbf{x}), \quad f_k \in \mathcal{F} \quad (4.8)$$

where  $\mathcal{F} = \{f(\mathbf{x}) = w_{q(\mathbf{x})}\}$  is the space of the trees,  $q(\mathbf{x})$  maps the input  $\mathbf{x}$  to a leaf index, and  $w_{q(\mathbf{x})}$  is the weight associated with that leaf.  $K$  is the total number of trees in the ensemble. The model parameters are learned by minimizing a regularized objective function:

$$\mathcal{L}^{(t)} = \sum_{i=1}^m l(y_i, \hat{y}_i^{(t)}) + \sum_{t=1}^T \Omega(f_t) \quad (4.9)$$

with  $\Omega(f) = \gamma T + \frac{1}{2} \sum_T \|w\|^2$

where  $l$  is a differentiable loss function (e.g., binary cross-entropy for classification, see Section 4.1.3 for a discussion on loss functions) and  $\Omega(f)$  is a regularization term which penalizes model complexity.  $T$  is the number of leaves in the tree,  $w_j$  the weight of leaf  $j$ ,  $\gamma$  controlling the penalty for adding more leaves, and  $\lambda$  controlling L2 regularization on leaf weights.

As with neural networks (see Section 4.1.3), XGBoost optimizes the loss by calculating the errors between the predicted and actual values, and each new tree attempts to correct the errors made by the previous trees. It uses gradient-based optimization with a second-order approximation via the Newton–Raphson method, making use of both the first derivative (gradient) and second derivative (Hessian) of the loss function for each training instance. This allows for more accurate and efficient updates compared to first-order methods.

While Random Forest aims to reduce variance through randomness and averaging, XGBoost usually achieves better performance by reducing both bias and variance through sequentially correcting errors and directly optimizing a loss function.

## Parameters

XGBoost has many parameters to tune, some of which overlap with those in random forest. According to Budholiya et al. (2022) [18], Chen and Guestrin (2016) [21], and Sommer et al. (2019) [51] the following parameters are most often tuned:

The number of trees (**n\_estimators**) represents the number of boosting iterations. More trees increase the risk of overfitting, while too few may lead to underfitting. A high number of trees combined with a low learning rate typically gives the best results.

The learning rate (**learning\_rate**) or shrinkage controls the step size  $\eta$  for each boosting step. After each iteration, new trees are added to correct the errors of the previous ones, and the learning rate shrinks the contribution of each new tree by a factor  $\eta$  to avoid overfitting. Lower values (e.g., 0.01-0.1) slow down learning but generally improve generalization, while larger values (e.g., 0.2 or more) speed up learning but may lead to suboptimal convergence.

The parameters **max\_depth**, **subsample**, **colsample\_bytree**, and **min\_child\_weight** are similar to those in random forest. The maximum tree depth typically ranges from 3 to 10. For subsampling, all data is used for every tree by default, but values ranging from 0.5 to 0.8 are common to help generalization by introducing randomness and preventing overfitting. Column subsampling (**colsample\_bytree**), similar to feature bagging in random forest, with typical values ranging from 0.3 to 0.8. The minimum child weight (**min\_child\_weight**) represents the minimum number of samples required in a node to split.

The regularization parameters (**lambda**, **alpha**) refer to L2 and L1 penalties, respectively, applied to the leaf weights ( $w_{q(x)}$ ) of the trees. **lambda** (default 1)

reduces the influence of large weights to improve generalization, and **alpha** (default 0) encourages sparsity by shrinking some weights to zero, effectively performing feature selection. Non-zero values for **alpha** are useful for high-dimensional data when only a subset of features is expected to be important.

### 4.2.3 Multilayer perceptron

A multilayer perceptron (MLP) is a feedforward neural network architecture commonly used for supervised learning tasks. Its structure follows the general neural network architecture described in Section 4.1.3 and Figure 4.3 illustrates a typical MLP architecture.

The MLP is applied here as a supervised model for binary classification. The output layer uses the sigmoid activation function to produce probabilities in  $[0, 1]$ . As previously described for autoencoders in Section 4.1.3, training proceeds by back-propagation, which computes the gradients of  $\mathcal{L}$  with respect to each parameter. In contrast to autoencoders, where the objective is to reconstruct the input, the MLP is trained to classify the inputs correctly. Nonetheless, the binary cross-entropy loss from Equation 4.6 can still be used, by reflecting the difference between predicted outcome and true class label.

#### Parameters

The architecture and training setup (activation function, loss function, etc.) have been described in the previous section (Section 4.1.3). Here, the focus is on hyperparameters that can be tuned to improve performance.

The MLP architecture is defined by the number of layers (**n\_layers**) and the number of neurons in each layer (**n\_units**). For simpler tasks, typically 1 to 3 hidden layers are used, and for more complex tasks, deeper models are often required. There is no clear rule of thumb for choosing the architecture, and performance depends on the task at hand. Both can be tuned alongside regularization parameters to ensure the model still generalizes.

Neural networks minimize the loss function by updating the weights through gradient descent with an **optimizer**. Common choices include Stochastic Gradient Descent (SGD), Adam, and RMSprop. Adam is widely used because it adapts learning rates per parameter and often converges quickly with default settings. The choice of optimizer often introduces sub-hyperparameters, such as a learning rate (discussed below), momentum (for SGD, typically around 0.9), and decay rates  $\beta_1$ ,  $\beta_2$  for Adam (usually left at their defaults, 0.9 and 0.999).

The **learning\_rate** is arguably the most important hyperparameter for tuning and controls the step size in gradient descent. For Adam, a common default is  $1e-3$ , but values ranging from  $1e-4$  to  $1e-2$  are often explored. As noted by Smith (2018) [50], for a shallow 3-layer network, a learning rate of 0.01 is already considered large. For SGD, typical values range from  $1e-2$  to 0.1.

Regularization hyperparameters include **dropout**, **L2 weight decay**, and **early stopping**. Dropout randomly disables a fraction of neurons during training, with common values in the range of 0.2 and 0.5 for hidden layers. L2 weight decay penalizes large weights and encourages simpler models. It often interacts with the learning rate, where higher learning rates often require stronger regularization. Shallow architectures benefit from more aggressive regularization, with recommended values

in the range of  $1e-3$  to  $1e-5$  [50]. Another method to combat overfitting is early stopping, which refers to training the model until a certain stopping criterion is met. For example, the model performance can be monitored by splitting the dataset into a training and validation set, and when the error on the validation set starts to increase, training can stop, ensuring model generalization.

### 4.3 Bayesian hyperoptimization

Bayesian optimization is a global optimization technique suited for optimizing expensive and black-box models [26]. It has become increasingly popular for hyperparameter tuning in machine learning, especially for deep neural networks, where evaluating a single configuration is expensive. Unlike traditional search strategies such as manual search, random search, or grid search, which evaluate each hyperparameter independently, Bayesian optimization models the relationship between hyperparameter configurations and performance using a probabilistic model. Formally, the goal is to solve:

$$\max_{\boldsymbol{\lambda} \in \mathcal{H}} f(\boldsymbol{\lambda}), \quad (4.10)$$

where  $f(\boldsymbol{\lambda})$  is the objective function,  $\boldsymbol{\lambda}$  denotes a set of hyperparameters, and  $\mathcal{H}$  is the hyperparameter search space defined by specifying ranges of values for each hyperparameter (e.g., learning rate  $\in [10^{-5}, 10^{-1}]$ , number of layers  $\in [1, 2, 3]$ ). In this thesis,  $f(\boldsymbol{\lambda})$  corresponds to the validation AUC, which is preferred over accuracy due to the class imbalance in the dataset (more details are provided in Section 5.4).

This thesis makes use of Optuna [4], which applies the Tree-structured Parzen Estimator (TPE) to guide the search and is scalable for high-dimensional and mixed discrete-continuous search spaces [13]. At each iteration, the algorithm suggests a new hyperparameter configuration  $\boldsymbol{\lambda}_{t+1}$  to evaluate, updates its model with the result, and continues until a stopping criterion is reached (such as a maximum number of trials or limited improvement). This approach allows efficient exploration of the hyperparameter space while keeping computational cost manageable.

# Chapter 5

## Experimental setup

This chapter discusses the experimental design and setup, building on the methods described in the previous chapter. In Section 5.1, a brief recap of the data preparation is provided together with additional processing steps. Section 5.2 discusses the setup of three dimensionality techniques. Section 5.3 outlines the model training and marks the hyperparameters that are tuned for each model. Finally, in Section 5.4, the evaluation metrics used are discussed.

### 5.1 Data preparation

The preprocessing steps have been extensively discussed in Section 3.3. In summary, 254 entries were removed due to invalid dates, bookings with zero guests, or reservations consisting only of children, resulting in a final dataset of 36,021 observations. The dataset contains 32 variables, as listed in Tables 3.1 and 3.3. Two variables, *booking ID* and *arrival date*, were dropped as they carried no predictive information. However, the day of the week was derived by combining the arrival year, month, and date.

#### 5.1.1 Feature preprocessing

Features were grouped based on their type and preprocessing requirements:

- **Numerical features:** continuous variables such as *average price per room* and *number of special requests* were normalized to the  $[0, 1]$  range using a MinMax scaler. Normalization was applied for methods sensitive to feature scaling (i.e., PCA, MCA, autoencoders, and MLP). For the tree-based models (RF, XGBoost), unscaled numerical features were used. The scaler was always fitted on the training set only and subsequently applied to the validation and test sets to prevent data leakage.
- **Binary category features:** variables that were already encoded as binary indicators, such as *required car parking space*, *repeated guest*, *includes weekend*, *short stay*, *long stay*, *last minute booker*, *early bird booker*, and *travel with kids* were retained without modification.
- **Nominal categorical features:** variables with multiple categories (*type of meal plan*, *room type reserved*, *arrival year*, *arrival month*, *market segment type*, *day of week arrival*) were transformed using one-hot encoding.

This preprocessing step resulted in a total of 61 features: 46 one-hot encoded categorical features, 14 numerical features, and one target feature.

### 5.1.2 Train-test split

The dataset was split into training and testing sets using an 80/20 ratio. As previously shown in Figure 3.6 and Table 3.5, the target variable *booking status* has a moderate imbalance, with approximately 66% of the bookings not canceled while 34% are canceled. Therefore, a stratified split on the target variable was applied to ensure the models are trained and tested with enough samples of both classes and support generalization. The use of a synthetic data generation method like SMOTE was not used, since for strong classifiers (e.g., XGBoost), SMOTE shows very little improvement [25].

During autoencoder training, an additional 80/20 stratified validation split was created from the training set. This validation data was used for early stopping and to monitor overfitting during training. To ensure a fair comparison, the same train-test split was used across all models and dimensionality reduction strategies. Importantly, all dimensionality reduction techniques were fitted solely on the training set, and the resulting transformations were applied to the validation and test sets to prevent data leakage.

## 5.2 Experimental setup

In this thesis, dimensionality reduction was treated as a preprocessing step that transforms the input space into new feature representations, which were later used to train downstream classifiers. In summary, the effectiveness of dimensionality reduction methods was examined by three different approaches:

1. **No dimensionality reduction:** The entire dataset, consisting of 60 features (46 one-hot encoded categorical and 14 numerical), was directly used to train the models.
2. **Statistical dimensionality reduction:** The dataset features were divided into numerical and categorical. PCA was applied to the numerical features, while MCA was applied to the categorical ones.
3. **Deep learning based dimensionality reduction:** An autoencoder was trained on the full feature set (i.e., 60 features) to compress it into a lower-dimensional representation through a bottleneck layer. The size of the latent space was tested with different dimensionalities of 4, 8, 16, and 32 to examine the effect of dimensionality reduction on information retention.

In the next chapter, Section 6.1.1 discusses the component selection, and Section 6.1.2 discusses the architecture of autoencoders.

## 5.3 Model Training

Once the feature representations were obtained, they were used as input for downstream classifiers. To ensure comparability across methods, the same downstream

classifiers, random forest, XGBoost, and an MLP neural network, were used. A baseline model was also included by classifying all instances as belonging to the majority class (i.e., not canceled). The structure of the experimental design is illustrated in Figure 5.1.

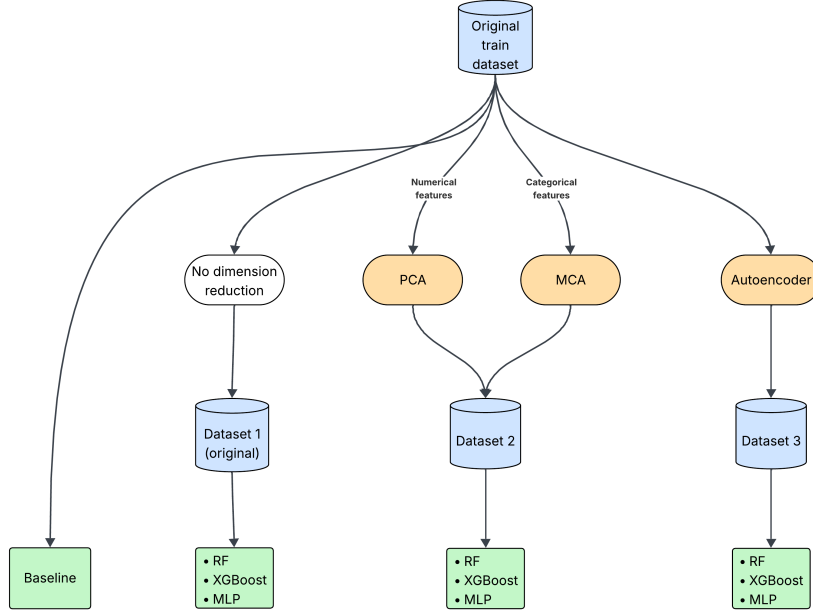


Figure 5.1: Experimental setup of this study. Three approaches are compared: no dimensionality reduction, statistical reduction (PCA + MCA), and deep learning-based reduction (autoencoder). Each representation is evaluated using the same downstream models: RF, XGBoost, and MLP.

### 5.3.1 Hyperparameter optimization

The models were optimized through hyperparameter optimization using Bayesian optimization from the `Optuna` package [4]. To improve performance, a stratified 5-fold cross-validation on the training set was used to validate performance while maintaining class balance. Optimization was performed with respect to the Area Under the ROC Curve (AUC), which is a metric that balances the trade-off between the true positive rate and the false positive rate. A detailed description of all evaluation metrics is provided in Section 5.4. The number of trials was set to 50, meaning that 50 different combinations of the parameter values are evaluated.

During MLP optimization, in addition to dropout and L2 regularization, early stopping was included. Early stopping helps to reduce overfitting and simultaneously reduces training time. It was implemented using the built-in `Keras` function that monitors the validation AUC. If after five consecutive epochs the validation AUC did not improve by at least  $1e - 4$ , training was stopped. A tolerance of five was chosen to permit slight variations in AUC while still avoiding overfitting, while the delta threshold ensured that only meaningful improvements were considered.



Model	Tuned parameters	Search space
Random forest <sup>1</sup>	n_estimators	int [100, 1000]
	max_depth	int [5, 20]
	min_samples_split	int [2, 20]
	min_samples_leaf	int [2, 10]
	max_features	cat [sqrt, log, 0.2, 0.5, 0.8]
XGBoost <sup>2,3,4</sup>	n_estimators	int [200, 1000]
	max_depth	int [3, 10]
	learning_rate	float [0.01, 0.3] (log)
	subsample	float [0.5, 0.9]
	colsample_bytree	float [0.3, 0.8]
	min_child_weight	int [1, 10]
	alpha	float [0.0, 1.0]
	lambda	float [1.0, 5.0]
Multilayer perceptron <sup>5</sup>	n_layers	int [1, 3]
	n_units	cat [8, 16, 32, 64, 128]
	optimizer	cat [adam, sgd, rmsprop]
	learning_rate	float [0.0001, 0.1] (log)
	dropout_rate	float [0.0, 0.6]
	L2_reg	float [1e-6, 0.01]

Table 5.1: Hyperparameter tuning details and search spaces for each model. The choice of search ranges is based on prior studies: <sup>1</sup> Probst et al. [45]; <sup>2</sup> Budholiya et al. [18]; <sup>3</sup> Chen and Guestrin [21]; <sup>4</sup> Sommer et al. [51]; <sup>5</sup> Smith [50]. A detailed discussion of these hyperparameters and their role in model performance is provided in Section 4.2 (Model selection).

## 5.4 Evaluation

To evaluate the performance of classification models, an evaluation or objective function is necessary [23]. In the context of cancellation prediction, the task is a binary classification problem where the model predicts either a 0 (not canceled) or a 1 (canceled). Each prediction falls into one of four categories:

- True Positive (TP): correctly predicted canceled booking,
- True Negative (TN): correctly predicted non-canceled booking,
- False Positive (FP): incorrectly predicted canceled booking (actually not canceled),
- False Negative (FN): incorrectly predicted non-canceled booking (actually canceled).

These four outcomes are useful for the evaluation functions. A confusion matrix 5.2 is used to summarize the four outcomes:



		Predicted	
		Not canceled	Canceled
Actual	Not canceled	TN	FP
	Canceled	FN	TP

Table 5.2: Confusion matrix

Common metrics used in cancellation prediction literature include accuracy, precision, recall, the  $F_1$  score, and the AUC [6, 7, 48]. They are explained below.

**Accuracy.** The most common evaluation metric for classification is the accuracy, which measures the proportion of correct predictions over the total number of observations.

$$accuracy = \frac{\text{correct classifications}}{\text{total classifications}} = \frac{TP + TN}{TP + TN + FP + FN}$$

**Recall.** Recall (also known as sensitivity or true positive rate) measures the proportion of actual canceled bookings that were correctly identified. High recall indicates that most cancellations are correctly detected, which is important in avoiding revenue loss from unanticipated no-shows.

$$recall = \frac{\text{correctly classified actual positives}}{\text{all actual positives}} = \frac{TP}{TP + FN}$$

**Precision.** Precision measures the proportion of bookings predicted as canceled that actually were canceled. Precision reflects how confident the model is that a predicted cancellation truly represents a canceled booking.

$$precision = \frac{\text{correctly classified actual positives}}{\text{everything classified as positives}} = \frac{TP}{TP + FP}$$

**$F_1$  score.** The  $F_1$  score is the harmonic mean of precision and recall, providing a balanced measure that accounts for both false positives and false negatives. Therefore, it is more suitable for an imbalanced dataset, where the accuracy may be misleading in measuring the models' performance.

$$F_1 \text{ score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}}$$

**AUC-ROC.** The Area Under the Receiver Operating Characteristic Curve (AUC-ROC) evaluates the ability of a model to distinguish between two classes using the probability estimates produced by the model. Instead of making predictions with a fixed threshold, the ROC curve is generated by varying the classification threshold over the full range of predicted possibilities. For each threshold, the true positive rate (TPR, or recall) and false positive rate (FPR, where  $FPR = \frac{FP}{FP+TN}$ ) are computed and plotted. The AUC represents the area under this curve: a value of 0.5 indicates random guessing, while 1.0 corresponds to a perfect classifier. Because it summarizes performance across all possible probability thresholds, it effectively

ranks instances by their likelihood of belonging to the positive class. Therefore, AUC represents the ability of the model to assign higher predicted probabilities to positive outcomes (class 1) than to negative outcomes (class 0). It is especially useful in imbalanced datasets, though it can be overly optimistic if true negatives dominate.

As discussed previously, non-canceled bookings dominate in the dataset, and although accuracy is widely used, solely relying on accuracy is not desired in imbalanced datasets. In such cases, a model could achieve high accuracy by predominantly predicting the majority class (i.e., “not canceled”), leading to biased results. Moreover, there might be different costs for misclassifications. A false positive (predicting a cancellation that does not occur) could lead to overbooking, harming the hotel’s credibility, while a false negative (failing to predict a cancellation) might prevent proactive retention strategies. Additionally, Antonio et al. (2017) [6] highlighted the importance of minimizing false positives because they can cause unnecessary costs, since the hotel might invest effort or money (e.g., offering discounts) to retain bookings that would not have been canceled anyway. Thus, both errors are undesirable, but false positives can be particularly costly in operational contexts [6, 7]. Nonetheless, no specific cost weights are incorporated in the training procedure to adjust for different misclassification costs. Incorporating such weights requires input from domain experts or stakeholders to correctly quantify the operational costs associated with false positives and false negatives. Since this thesis does not involve an actual case study, the emphasis is instead placed on analyzing model performance through relevant metrics (e.g., precision and recall) and discussing their implications based on general domain knowledge.

To balance these considerations, the AUC is used for hyperparameter optimization, since it assesses model performance across all classification thresholds. The final evaluation is then performed with the  $F_1$  score, to provide a trade-off between precision and recall by penalizing false positives and false negatives equally.

## 5.5 Implementation details

All experiments were conducted on a MacBook Pro with the following hardware specifications:

- Processor: 2.3 GHz 8-Core Intel Core i9
- Graphics: AMD Radeon Pro 5500M (8 GB) and Intel UHD Graphics 630 (1536 MB)
- Memory: 32 GB 2667 MHz DDR4

The models were implemented in Python using TensorFlow/Keras and scikit-learn, and optimization was performed with Optuna.

# Chapter 6

## Results

In this chapter, the results of the downstream classifiers trained on different feature representations are presented. First, Section 6.1 discusses the obtained feature representations of statistical and autoencoder dimensionality reduction. Section 6.2 provides the main results of all the models and dimensionality reduction techniques. Section 6.3 will discuss the best results in more detail, by providing multiple evaluation metrics. Section 6.4 details the important variables for cancellation prediction using SHAP values. Section 6.5 investigates the effect of dimensionality reduction. Last, Section 6.6 compares the latent space of the best two autoencoder architectures. Note that all results are given on the test set, unless stated otherwise.

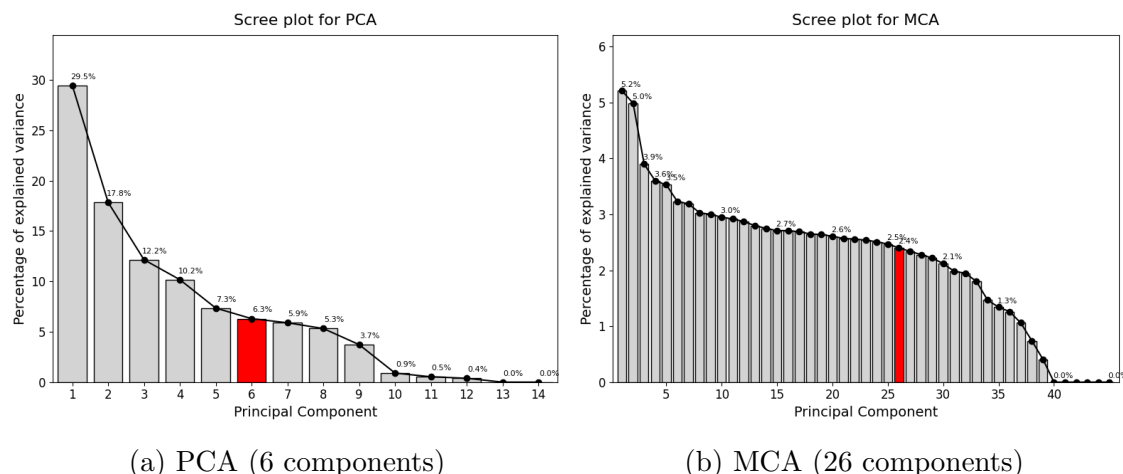
### 6.1 Dimensionality reduction results

This section reports the results of the dimensionality reduction analyses. For the statistical methods (PCA/MCA), the number of retained components is determined based on explained variance, while for autoencoders, different bottleneck sizes are compared to assess reconstruction quality. Section 6.1.1 first presents the component selection for PCA/MCA, and Section 6.1.2 presents the evaluation of bottleneck sizes for autoencoders.

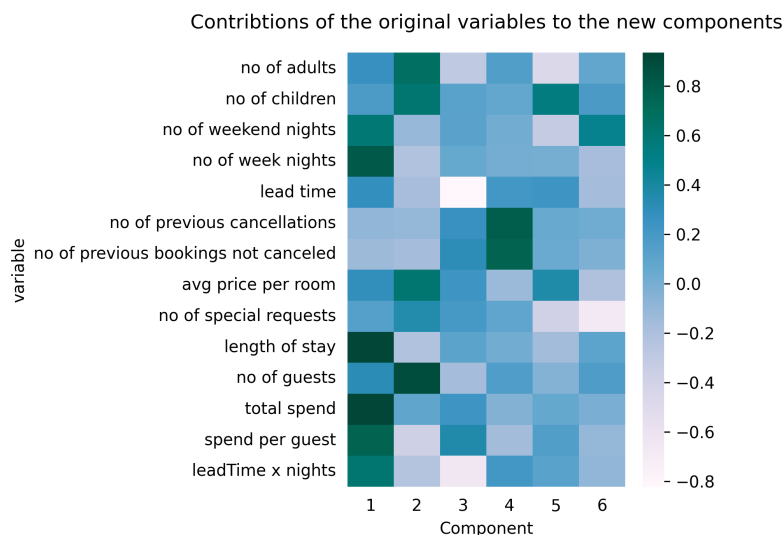
#### 6.1.1 Component selection for statistical methods

For statistical dimensionality reduction, numerical and categorical features were treated separately. PCA was applied to the numerical features, while MCA was applied to the categorical ones.

Following the cumulative percentage of explained variance criterion introduced in Section 4.1, an 80% threshold was adopted. This approach is consistent with Alkhayrat et al. (2020) [5], who also determined the number of dimensions by examining the variance drop-off point in scree plots. The scree plots for both PCA and MCA (Figures 6.1a and 6.1b) confirm that beyond the 80% threshold, little additional information is gained. Especially in MCA, the first component explains just over 5%, and the curve quickly levels off, suggesting that many categorical variables contribute little incremental variance. This aligns with Section 4.1, where it was noted that MCA often does not produce a small set of dominant components. Using the 80% threshold, 26 components were retained, and while this reduction is modest, it still lowered the dimensionality from 46 to 26. For PCA, this resulted in retaining



Because the explained variance in MCA is spread across many components without an apparent dominant component, more detailed insights were obtained from PCA. **Figure 6.2** displays the contribution of the original features to the principal components. Since the data was standardized, values fall within  $[-1, 1]$ .



The first component primarily describes variables related to length of stay (e.g., number of week nights and weekend nights, total spend), while the second compo-

ment captures guest characteristics such as the number of adults and children. It further illustrates redundancy among variables in the dataset. For instance, the variables “no of previous cancellations” and “no of previous bookings not canceled” both have a high contribution to PC4, suggesting that they capture similar information.

### 6.1.2 Autoencoder architecture

The autoencoder architectures used in this thesis were inspired by prior work on dimensionality reduction for tabular data [5, 57]. The reconstruction layer used a sigmoid activation function with binary cross-entropy loss, since the inputs were scaled to a  $[0, 1]$  range (see Chapter 4 for further motivation). Following Alkhayrat et al. (2020) [5] and Volovăț et al. (2024) [57], ReLU activation functions are applied between hidden layers, and training was performed with the Adam optimizer at a learning rate of  $1e - 3$ . Given the moderate dataset size, a batch size of 32 was chosen to avoid overly large batches while maintaining stable gradients. Prior work also suggests that smaller batch sizes can improve generalization [36]. A maximum of 100 epochs was specified; however, training duration was controlled through early stopping on the validation loss (patience of 10, minimum delta of  $1e - 5$ ), ensuring that the autoencoder did not overfit.

Seven different autoencoder architectures were tested, varying in both depth and bottleneck size. Since the dataset was relatively small and not highly complex, shallow networks were chosen. To account for stochasticity during training (e.g., random initialization of weights), each architecture was trained five times with different random seeds. For each architecture, reconstruction loss was summarized as the mean and standard deviation across these runs (Table 6.1), along with the number of parameters and bottleneck size. Figure 6.3 displays the validation loss curves for all five runs of each autoencoder, demonstrating stable convergence around 30 epochs for most models. For bottleneck sizes 16 and 32, convergence occurred even earlier, within 10 epochs.

Architecture	Bottleneck	Params	Loss (mean $\pm$ SD)	
			Train	Validation
60-40-32-40-60	32	5,732	$0.0610 \pm 0.0002$	$0.0613 \pm 0.0002$
60-40-20-16-20-40-60	16	6,736	$0.0617 \pm 0.0001$	$0.0623 \pm 0.0002$
60-40-16-40-60	16	4,116	$0.0623 \pm 0.0003$	$0.0630 \pm 0.0005$
60-40-20-8-20-40-60	8	6,248	$0.0674 \pm 0.0005$	$0.0683 \pm 0.0006$
60-40-8-40-60	8	3,308	$0.1051 \pm 0.0034$	$0.1054 \pm 0.0036$
60-40-20-4-20-40-60	4	6,004	$0.1162 \pm 0.0029$	$0.1179 \pm 0.0029$
60-40-4-40-60	4	2,904	$0.1936 \pm 0.0010$	$0.1933 \pm 0.0013$

Table 6.1: Comparison of autoencoder architectures. Mean and standard deviation (SD) of binary cross-entropy (BCE) reconstruction loss are calculated over five random seeds.

Table 6.1 and Figure 6.3 show that architectures  $[60 - 40 - 32 - 40 - 60]$ ,  $[60 - 40 - 16 - 40 - 60]$ , and  $[60 - 40 - 20 - 16 - 20 - 40 - 60]$  achieved the

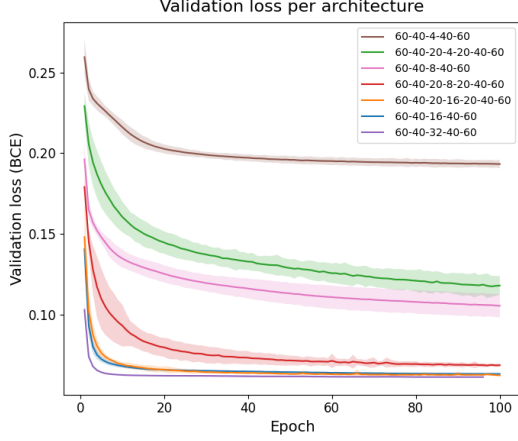


Figure 6.3: Validation loss per epoch for each autoencoder. Mean and SD are calculated from five runs per architecture.

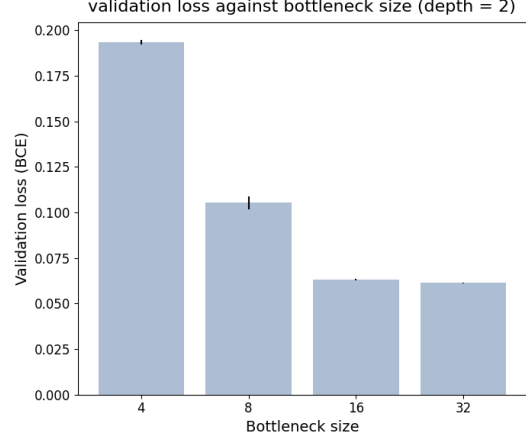


Figure 6.4: Average validation reconstruction loss by bottleneck size (4, 8, 16, 32) for two-layer networks. Bars show the standard deviation. Performance improves markedly from  $4 \rightarrow 8 \rightarrow 16$ , then plateaus between 16 and 32.

lowest reconstruction losses, while models with bottleneck size 4 performed substantially worse, even when depth was increased. Architectures with eight nodes in the bottleneck layer also yielded higher losses, though performance improved when an additional encoder layer was added. Interestingly, the two architectures with bottleneck size 16 performed almost identically, indicating that additional depth does not provide benefits at this dimensionality. In general, simpler models are preferred, as they require fewer parameters and are therefore more computationally efficient (i.e., lower training time) [23].

The barplot in Figure 6.4 shows the trade-off between compression and reconstruction error for architectures with two encoder layers. Models with bottleneck sizes of 4 and 8 performed noticeably worse than those with 16 and 32. Beyond 16, the improvement flattened, indicating diminishing returns.

Overall, shrinking the latent space too aggressively (e.g., to 4 nodes) resulted in substantial information loss and poor reconstruction, while very large bottleneck sizes diminished the benefits of dimensionality reduction. The results suggest that a bottleneck of size 16 provides the best balance between compression and accuracy, with 32 included as a slightly larger latent space to test whether additional capacity improves downstream performance. Therefore, two architectures  $[60-40-i-40-60]$  with  $i = 16, 32$  were chosen for further modeling, and are hereafter denoted as AE-16 and AE-32.

## 6.2 Main findings

The  $F_1$  scores across the three models and four dimensionality reduction (DR) configurations are presented in Table 6.2. As a baseline, classifying all instances to the majority class (i.e., not canceled) achieved an  $F_1$  score of 0.0 and accuracy of 0.671. XGBoost consistently achieved the highest  $F_1$  score for each DR technique, confirming its strong performance for tabular data. Random forest also performed

competitively, although slightly below XGBoost, except when no DR was used. However, note that this difference was only 0.002. For both RF and XGBoost, performance declined in the order: original feature set, statistical DR, AE-32, and AE-16. The MLP initially performed considerably worse when trained directly on the raw feature set; however, its performance improved substantially when trained on features derived from autoencoders. These findings align with the expectation that neural networks are better suited to dense, numerical inputs rather than sparse, one-hot encoded data, as noted in Section 4.2 by Guo and Berkhahn (2016) [29].

	Random forest	XGBoost	MLP
No DR	<b>0.84622</b>	0.84479	0.68402
Statistical	0.81253	<b>0.81593</b>	0.69866
AE-32	0.79510	<b>0.80510</b>	0.77704
AE-16	0.78770	<b>0.79586</b>	0.74747

Table 6.2:  $F_1$  score for all models and dimensionality reduction configurations. Best performance for each DR technique is in **bold**.

Figure 6.5 further compares training time and predictive performance. Here, training time refers to the full hyperparameter optimization process (Section 6.5.1), i.e., the cumulative cost of finding the best model. This reflects the trade-off between performance and model development efficiency. Ideally, a low training time, combined with a high  $F_1$  score, is desired. XGBoost achieved both high  $F_1$  scores and the lowest training time, making it the most efficient model compared to the RF and MLP. Although RF is also an ensemble method, it had considerably higher training times, except when trained on the original feature set. For MLP, DR not only improved predictive performance but also reduced training time, particularly with autoencoder-derived feature representations.

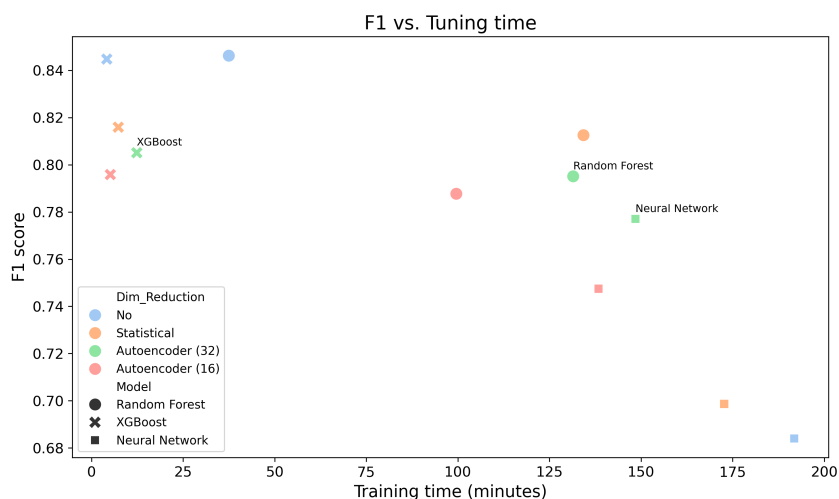


Figure 6.5: The  $F_1$  score and training time for all models and dimensionality reduction configurations. Reported times reflect the hyperparameter tuning of the classifiers only. The additional training time for autoencoders was substantial, whereas PCA/MCA added a negligible training time.



Taken together, these results suggest that DR is not necessary for tree-based methods, which already perform well on the original feature set. However, autoencoder representations provide clear value for neural networks, substantially improving both their  $F_1$  score and efficiency. The overall best performing model remains XGBoost without DR, which is consistent with prior findings that boosting methods often outperform deep learning on tabular data [49].

### 6.2.1 Statistical comparison of configurations

Following Volovăț et al. (2024) [57], statistical testing was conducted to assess whether differences in model performance were significant ( $p < 0.05$ ). Pairwise McNemar’s tests were applied across all model–configuration combinations, and the results are shown in Figure 6.6.

McNemar’s test is specifically designed for paired nominal data (e.g., binary outcomes) and is often used to compare multiple classifiers on the same dataset [60]. It evaluates whether two classifiers significantly differ in their error rates. However, since twelve models were compared,  $\binom{12}{2} = 66$  pairwise comparisons were performed, and conducting multiple tests increases the risk of Type I errors (i.e., false positives). To mitigate this, the Benjamini-Hochberg (BH) correction was applied, which controls the false discovery rate and is often preferred to more conservative approaches such as Bonferroni [39].

Most pairwise comparisons showed significant differences, with nine exceptions. Performance for the MLP trained with statistical features did not differ significantly from the MLP trained on the original dataset. For RF models, the AE-32 and AE-16 features produced similar results. Similarly, XGBoost with AE-32 features did not show a statistically significant difference from the statistical feature set. Within the ensemble models, RF and XGBoost did not differ significantly when trained on the original dataset, nor when trained with statistical DR, AE-16, or AE-32 features, indicating that the performance of the two models is comparable. The full table of  $p$ -values of all pairwise comparisons is provided in Appendix B.3.

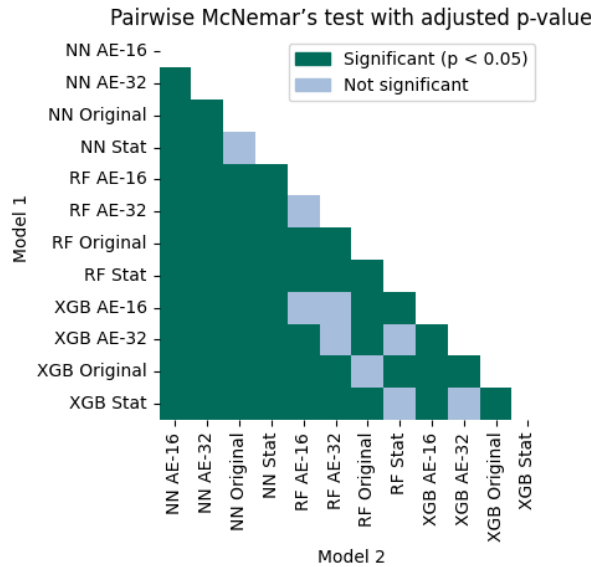


Figure 6.6: Pairwise McNemar test with BH correction across all configurations.



### 6.3 Best performance for each model

The precision, recall,  $F_1$  score, AUC, and accuracy for the best configuration of each model are presented in Table 6.3. The ensemble models (RF and XGBoost) achieved the highest overall accuracy ( $\approx 0.90$ ) and very similar  $F_1$  scores when trained without DR. The MLP performed less strongly overall, but benefited substantially from the autoencoder features (AE-32). Across all models, the “not canceled” class consistently achieved higher scores, reflecting the class imbalance in the data. Additionally, the models performed better than the baseline model, which achieved an accuracy of 0.671.

Model	DR	Class	Precision	Recall	$F_1$ score	AUC	Accuracy
Random forest	None	Not canceled	0.9139	0.9434	0.9284	0.9555	0.9023
		Canceled	0.8761	0.8183	0.8462		
XGBoost	None	Not canceled	0.9117	0.9454	0.9283	0.8791	0.9019
		Canceled	0.8793	0.8128	0.8448		
MLP	AE-32	Not canceled	0.8777	0.9212	0.8990	0.8294	0.8609
		Canceled	0.8209	0.7376	0.7770		

Table 6.3: Precision, recall,  $F_1$  score, AUC, and accuracy for the best configuration of each model. For random forest and XGBoost, performance was highest without dimensionality reduction. For the MLP, the best performance was obtained using autoencoder features with a bottleneck size of 32. Metrics are reported separately for the two classes (“Canceled” and “Not canceled”).

An AUC of 0.95 for random forest indicates that it better separates positive and negative classes than the other models. The  $F_1$  score provided more insights into the two classes. For the “Canceled” class, recall was consistently lower (0.74–0.82) than precision (0.82–0.88), indicating that the models tend to miss cancellations (false negatives), which reduced  $F_1$ . The discrepancy between high AUC and lower  $F_1$  shows that, while the models rank bookings well, the default decision threshold (0.5) is suboptimal. Adjusting the threshold (e.g., using cost-sensitive criteria) could improve cancellation prediction without affecting AUC.

In terms of error trade-offs, precision exceeded recall for the “Canceled” class, indicating that the models were more accurately identifying cancellations (fewer false positives) but at the expense of missing true cancellations (more false negatives). Random forest attained slightly higher recall, producing fewer false negatives, whereas XGBoost achieved higher precision, creating fewer false positives. Since false positives are operationally more costly for hotels (see Section 5.4), XGBoost is slightly more favorable, despite the small numerical difference between the two models.

### 6.4 Key variables

The most important variables for the best model (i.e., XGBoost without DR) were examined using the SHAP values in Figure 6.7, which shows how each feature affects the model’s predictions. Marcilio and Eler (2020) [38] describe SHAP values as a particularly effective tool for interpreting model decisions, since they assign a

contribution value to each feature for every data point, which can then be combined to calculate the feature importance.

The features were ranked by their overall contribution across all predictions, meaning that the number of special requests, lead time, and the average price per room are the top three drivers of cancellations. The SHAP value for the number of special requests indicated that guests with more requests were less likely to cancel. Conversely, few or no special requests pushed the prediction towards cancellation. The lead time suggested that long lead times increased the likelihood of canceling, while short lead times reduced it; however, there was some variability in short lead times. Average price per room followed a similar trend; higher room prices were associated with an increased likelihood of cancellation.

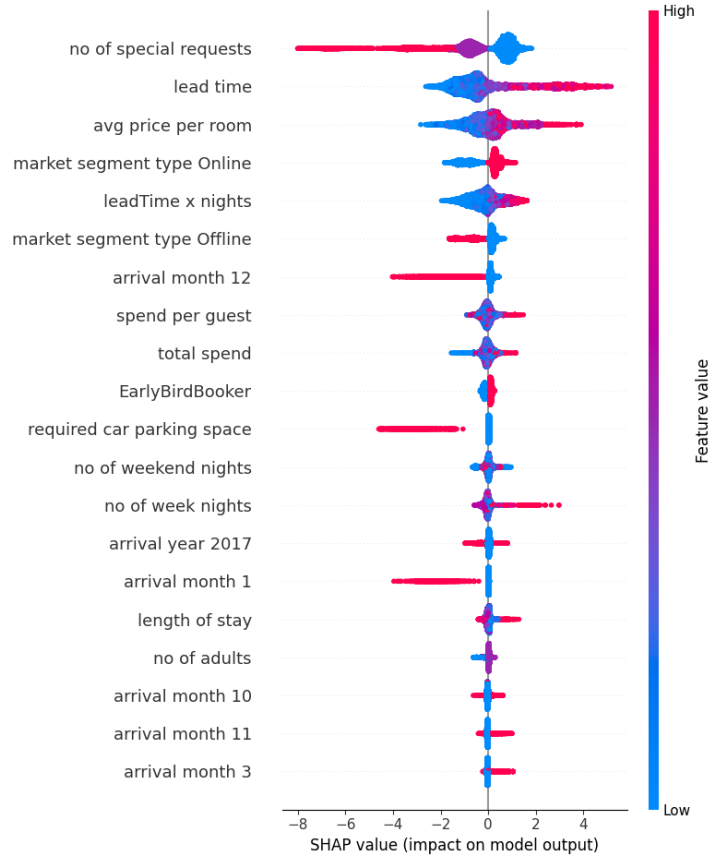


Figure 6.7: SHAP values for the XGBoost model on the original dataset. Positive values indicate a push toward predicting “canceled”, while negative values push toward “not canceled”. The distance from zero reflects the strength of the impact. Colors represent the feature value (red = high, blue = low). The vertical spread of the points illustrates variability, with wider spreads indicating that the feature has very different effects across samples, and narrower spreads indicating a more consistent effect.

Several other noteworthy observations were made. For example, the feature “required car parking space”, exhibited that bookings that required a parking space were less likely to be canceled. Seasonal effects were also apparent, with certain months (e.g., December and January) showing a higher tendency toward not canceling. Finally, when comparing the market segments, online bookings were more

strongly associated with cancellations, whereas offline bookings tended to be more retained.

## 6.5 The effect of dimensionality reduction

Figure 6.8 presents the confusion matrices for the three models trained without dimensionality reduction and with AE-32, respectively. These results are consistent with the metrics in Table 6.3. The other confusion matrices of the statistical DR and AE-16 are provided in Appendix B.3.

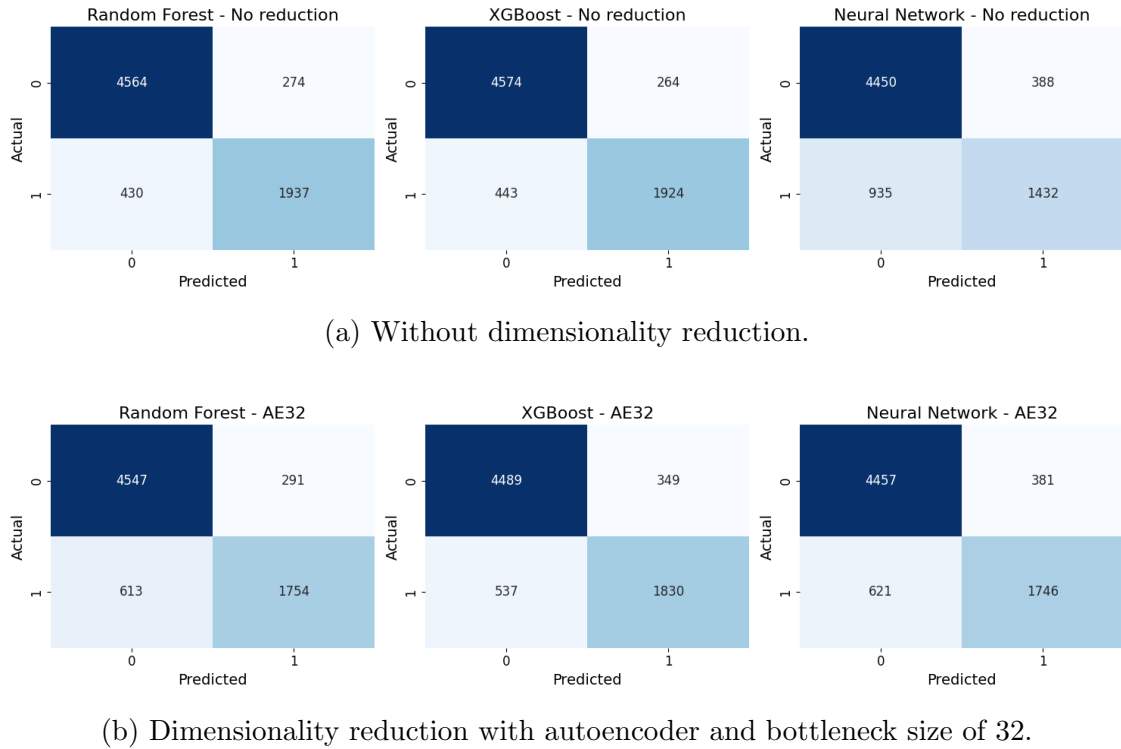


Figure 6.8: Confusion matrices illustrating the trade-off between false positives and false negatives for the three models.

The MLP benefited significantly from the autoencoder features, with performance improving substantially compared to training on the original data with one-hot encoded features, which is consistent with the observations in Section 6.2. However, both confusion matrices still show more false positives and false negatives compared to the ensemble models.

In the comparison of ensemble methods, the features derived from the autoencoder further highlight the advantage of XGBoost. While XGBoost produced somewhat more false positives than random forest, it achieved a higher number of true positives. Moreover, the gap between false positives and false negatives is smaller in XGBoost (188) than in random forest (394) and MLP (240), indicating a more balanced error distribution.

### 6.5.1 Hyperparameter analysis

The optimal hyperparameter settings for each model across the four DR techniques are reported in Table 6.4. For RF, the number of estimators steadily increased when DR was applied, while the number of features considered at each split decreased. This suggests that compressed features are less diverse, but the trees may have more difficulty with dense numeric features.

For XGBoost, the opposite trend was observed: fewer estimators were required when DR is applied, with the learning rate remaining stable across setups (0.02-0.04). Both alpha and lambda increased with DR-derived features, particularly for autoencoders (e.g.,  $\lambda = 4.44$  with AE-32 vs.  $\lambda \approx 1$  with no DR), suggesting that autoencoder features may introduce stronger feature interactions that require more regularization.

The architecture of the MLP varied considerably depending on the DR setup. Without DR, a single hidden layer with 128 units performed best, while statistical DR required a three-layer architecture. With autoencoder features, the network used two layers with fewer units, suggesting that autoencoders compress information into more compact representations, thereby reducing the need for deeper networks. Dropout rates are often close to zero, while L2 regularization increased substantially under AE-16, indicating that larger weights were penalized more. This penalization can also be caused by the use of stochastic gradient descent as an optimizer, which has the problem of exploding gradients with large weights.

Model	Parameters	Dimensionality reduction technique			
		None	Statistical	AE-32	AE-16
Random forest	n_estimators	265	399	735	983
	max_depth	20	16	19	17
	min_samples_split	4	4	6	2
	min_samples_leaf	2	3	3	3
	max_features	0.8	0.5	0.2	sqrt
XGBoost	n_estimators	961	960	719	513
	max_depth	9	8	8	10
	learning_rate	0.0246	0.0476	0.0294	0.0297
	subsample	0.8820	0.8696	0.8806	0.8345
	colsample_bytree	0.5859	0.7228	0.7965	0.7484
	min_child_weight	1	5	1	2
	alpha	0.0023	0.5074	0.6393	0.5674
	lambda	1.0023	1.9202	4.4430	2.7708
Multilayer perceptron	n_layers	1	3	2	2
	n_units	128	128, 64, 16	64, 32	64, 64
	optimizer	adam	sgd	adam	sgd
	learning_rate	0.00045	0.0275	0.0023	0.00092
	dropout_rate	0.1507	0.0854	0.0512	0.1351
	L2_reg	5.91e-06	5.17e-06	2.03e-06	3.77e-04

Table 6.4: Optimal hyperparameters per model across dimensionality reduction setups.

## 6.6 Comparison between AE-32 and AE-16

Predictive performance for AE-16 and AE-32 was very similar across all models (Table 6.2), suggesting that once the latent space was sufficiently large to capture the main structure of the data, further increasing its size provided limited additional benefit. Among the downstream classifiers, only the MLP benefited from DR through autoencoders, while the ensemble models performed slightly worse, although the differences were minor.

To further examine the learned representations, Figure 6.9 shows a two-dimensional projection of the AE-32 and AE-16 latent spaces using t-SNE (t-distributed Stochastic Neighbor Embedding). t-SNE is a nonlinear dimensionality reduction method commonly used for visualization, aiming to preserve local neighborhoods, i.e., points that are close in the original high-dimensional latent space remain close in the 2D projection.

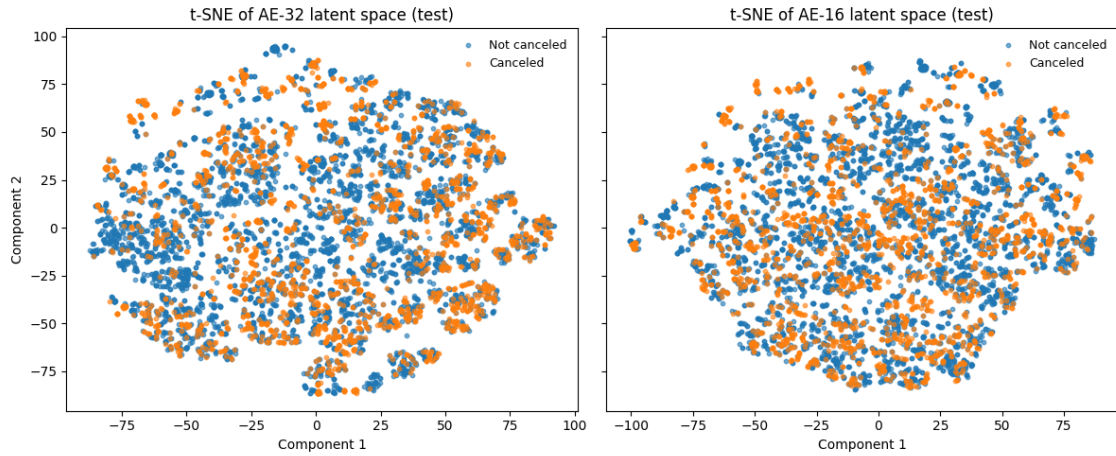


Figure 6.9: t-SNE projections on the test set of the AE-32 latent space (left) and the AE-16 latent space (right).

Both plots show that the two classes were largely mixed, indicating that the latent space did not linearly separate canceled from non-canceled bookings. Some areas were dominated by a single class, suggesting that the autoencoder successfully grouped similar bookings and captured latent patterns. However, a classifier was still required to learn the decision boundaries. This outcome is expected, since cancellations depend on multiple interacting factors (Figure 6.7) that cannot be isolated by compression alone.

When comparing AE-32 with AE-16, no apparent differences in cluster structure were observed. Both latent spaces captured important patterns, but neither produced a representation with substantially better class separation. This reinforces the earlier finding that once the latent space is sufficiently large (in this case, 16 dimensions), further increasing dimensionality provides limited additional benefit in predictive performance.

# Chapter 7

## Discussion

This thesis researched the question: *How do performance metrics (e.g.,  $F_1$  score) compare when predicting hotel booking cancellations using features learned with autoencoders, compared to traditional dimensionality reduction methods and models without any dimensionality reduction?* The objective was to test three dimensionality reduction techniques (i) no dimensionality reduction, (ii) statistical dimensionality reduction through PCA and MCA, and (iii) deep learning based dimensionality reduction through autoencoders, and evaluate them with three downstream classifiers: random forest, XGBoost, and multilayer perceptron.

This discussion first addresses the research question by answering the three sub-questions in Section 7.1. Afterwards, in Section 7.2, the results are compared to the related literature. Finally, the limitations of this research and recommendations for future work are provided in Sections 7.3 and 7.4, respectively.

### 7.1 Autoencoder design and latent space

#### Autoencoder design (RQ1 and RQ2)

*What type of autoencoder architecture (e.g., basic, denoising, variational) yields the best features for cancellation prediction?*

Prior work has revealed that for structured tabular data, vanilla autoencoders have generally shown stronger performance than specialized variants such as denoising or sparse autoencoders [5, 10, 57]. Therefore, this thesis used shallow vanilla autoencoders for dimensionality reduction.

*How does the size of the latent space impact the predictive performance when using autoencoders?*

The findings show that the bottleneck sizes of 16 and 32 yielded the most effective feature representations for this dataset. Very small bottlenecks (8 or 4 units) resulted in substantially higher reconstruction loss (Table 6.1), indicating that too much information was discarded. As a result, these representations were excluded for downstream classification, since poor reconstructions would likely translate into weak predictive performance, and combined with the high training times, further evaluation was not considered worthwhile. In contrast, both AE-16 and AE-32 provided stable and comparable results, suggesting that once the latent space is sufficiently large to capture the main structure of the data, further increasing the

dimensionality offers little additional benefit (i.e., diminishing returns). The limited complexity of the dataset likely explains the finding that relatively shallow autoencoders were sufficient. With only 60 input features, most of which are categorical variables transformed into one-hot encodings, deep architectures are unnecessary.

### The latent space (RQ3)

*Can the learned latent space reveal meaningful patterns related to booking cancellations?*

The latent space itself also provides insights. The t-SNE projections of AE-16 and AE-32 in [Figure 6.9](#) showed that the two outcome classes (canceled vs. not canceled) were largely overlapping, with only some regions where one class was more prevalent. This indicates that while the autoencoder learns compact and meaningful representations, they do not linearly separate cancellation outcomes. Therefore, a downstream classifier is necessary to exploit the nonlinear patterns embedded in the latent features. This outcome is expected, as the SHAP analysis revealed that cancellations depend on multiple interacting factors such as lead time, room price, booking channel, and special requests. Since no single factor dominates, the latent space alone cannot cluster canceled and non-canceled bookings into clearly distinct groups. Instead, the autoencoder serves to denoise and compress the input, producing features that are more suitable for neural networks but add little to no benefit to ensemble methods, which are already robust to sparse and high-dimensional inputs. This is supported by the results of statistical DR, which also reduced the input space to 32 features, and performance did not differ significantly from the original features for the ensemble models. Therefore, the main added value of AEs was not in the interpretability of the latent space, but in producing denser inputs that neural networks could exploit.

## 7.2 Comparison of the results with prior work

The results of this thesis show that XGBoost achieved the strongest overall performance, with an accuracy of 0.90,  $F_1$  score of 0.84, precision of 0.88, and AUC of 0.88. These findings can be compared to prior work on hotel booking cancellation prediction by Antonio et al. (2017) [6], Antonio et al. (2019) [7], and Sánchez-Medina and C-Sánchez (2020) [48].

Compared to Antonio et al. (2019) [7], the models in this thesis achieved higher accuracy and precision. Their study reported an average test accuracy of 0.827 and an AUC of 0.78, with precision scores below 0.77. A consistent trend in their results was that performance declined from training to validation to test, particularly in terms of precision, suggesting possible overfitting. Antonio et al. (2017) [6] reported similar accuracies around 0.90. Their reported precision and AUC values were also comparable, though recall and  $F_1$  scores varied more widely across hotels. However, it should be noted that both studies used data from multiple hotels, which may have introduced more diversity in the dataset. Finally, Sánchez-Medina and C-Sánchez (2020) [48] also confirmed the strong performance of ensemble models, but in direct comparison, the tuned XGBoost and Random Forest in this thesis achieved higher accuracy, precision, recall,  $F_1$ , and AUC.



When comparing feature importance with previous studies, several similarities in predictors of cancellation can be observed. Both this thesis and prior work highlight lead time as one of the strongest predictors of cancellation [6, 7]. In addition to lead time, this thesis found that special requests and parking availability reduce the likelihood of cancellation, which is consistent with Antonio et al. (2017) [6]. Other overlapping findings include temporal features, where specific months or holiday periods decrease the likelihood of cancellation, which is consistent with the results of Sánchez-Medina and C-Sánchez (2020) [48]. Furthermore, this thesis found that higher room prices are associated with an increased likelihood of cancellation, aligning with Antonio et al. (2017) [6], who identified room type as an influential factor. Finally, differences between market segments observed here reflect the importance of market segment type reported by Antonio et al. (2019) [7], though the specific segments were not explicitly specified.

Statistical dimensionality reduction methods remain a competitive baseline, in accordance with comparative reviews on dimensionality reduction methods [20, 52, 53]. However, as suggested in the literature on deep learning for tabular data [49], neural networks benefited the most from autoencoder-derived features, whereas ensembles did not. Taken together, these findings emphasize that the effectiveness of dimensionality reduction relies not solely on the method itself, but also on the interaction with the downstream classifier and the complexity of the dataset.

### 7.3 Limitations

Several limitations of this thesis should be acknowledged. First, the dataset consisted of slightly over 36,000 records, which is relatively small by deep learning standards. The feature space was also modest in size (60 features, mostly categorical), which may limit the benefits of dimensionality reduction compared to studies with richer or more complex feature spaces such as Alkhayrat et al. (2020) [5]. This likely contributed to the strong performance of ensemble methods without dimensionality reduction, since neural networks typically require larger and more complex datasets to show their advantages. Second, the analysis was conducted on a single dataset, with the characteristics of the hotel remaining unknown. Prior work [7, 8] has shown that cancellation drivers and model performance can vary substantially between hotels. As a result, the present findings may be less generalizable; thus, replication across multiple hotels would be required to confirm broader applicability. Finally, no cost weighting was applied to false positives and false negatives due to the lack of stakeholders, as previously discussed in Section 5.4. Therefore, the models penalized both types of misclassifications equally, despite their different operational consequences. This may bias results, as shown in Section 6.3. Incorporating cost-sensitive evaluation would likely provide more representative performance estimates.

### 7.4 Recommendations for future work

The findings of this thesis suggest new opportunities for further research in cancellation prediction and dimensionality reduction. First, exploring cost-weighting approaches, such as threshold tuning or weighted loss functions, would be valuable,



as they may better reflect the costs of misclassifications in hotel operations. Second, validating the findings on larger and more diverse datasets would allow assessing the generalizability of the models across different hotels. Third, this thesis evaluated feature representations extracted from autoencoders with bottlenecks of size 16 and 32 on downstream models. Although bottlenecks of 8 and 4 were tested initially, their reconstruction quality was poor, and due to high training times, it was impractical to include them. Further exploration of different compression levels could provide insights into the trade-off between dimensionality reduction and predictive performance. Finally, future work could explore specialized neural network architectures designed for tabular data, such as TabNet [9], to determine whether they provide richer feature representations and outperform the approaches studied in this thesis.

# Chapter 8

## Conclusion

This thesis examined how predictive performance differs when hotel booking cancellations are modeled using features learned with autoencoders, compared to statistical dimensionality reduction techniques and models trained without dimensionality reduction. The feature sets were evaluated using three models: random forest, XGBoost, and a multilayer perceptron.

The results demonstrate that dimensionality reduction was not necessary for ensemble models. Both random forest and XGBoost performed best on the original feature set, with only minor differences when PCA or autoencoder features were used. XGBoost in particular achieved high predictive performance while also requiring the lowest training time, making it the preferred approach for this type of structured tabular data. For hoteliers, this implies that investing in complex feature learning pipelines is not worthwhile, as a well-tuned XGBoost already achieves strong and explainable performance. In contrast, autoencoder-derived features were very beneficial to neural networks. Compared to the original data, which contained sparse one-hot encoded features, training on compressed, dense representations improved both predictive performance and efficiency. However, despite these improvements, the neural network’s performance still did not surpass that of the ensemble models. This demonstrates that the added value of autoencoders depends strongly on the task and the downstream model: they can make neural networks viable, but do not improve methods that already handle sparse, high-dimensional input effectively.

Overall, the findings suggest that autoencoders are not a universal solution for tabular data, particularly for cancellation prediction. While they provide meaningful improvements for neural networks, autoencoders do not outperform simpler baselines such as PCA for ensemble models, and the differences are small, making dimensionality reduction generally unnecessary. For the dataset analyzed here, XGBoost trained directly on the original feature set remains the most effective and efficient solution.

Future work should further validate these findings on larger and more diverse datasets, where the benefits of representation learning may become more pronounced. Systematic testing of smaller bottlenecks (e.g., 8 or 4) could also clarify the trade-off between compression and predictive performance.

# Appendices

# Appendix A

## Methods

### A.1 Principal Component Analysis

Consider dataset  $\mathbf{X}$  with  $m$  instances (rows) and  $n$  variables (columns), i.e., a matrix of size  $m \times n$ . PCA aims to reduce the number of variables while preserving as much of the data's variability (i.e., information) as possible. The main steps are as follows:

**Step 1: Standardizing the variables**

Since the variables may be measured on different scales (e.g., height in cm vs. salary in euros), each variable must be standardized so that each variable contributes equally to the analysis:

$$Z_{ij} = \frac{X_{ij} - \mu_j}{\sigma_j} \quad (\text{A.1})$$

where  $X_{ij}$  denotes the value of feature  $j$  for instance  $i$  ( $j = 1, \dots, n$ ,  $i = 1, \dots, m$ ), and  $\mu_j$  and  $\sigma_j$  are the mean and standard deviation of feature  $j$ , respectively. The resulting standardized matrix  $\mathbf{Z} \in \mathbb{R}^{m \times n}$  is used for subsequent steps.

**Step 2: Computing the covariance matrix**

To understand how variables relate to one another, a covariance matrix  $\mathbf{C} \in \mathbb{R}^{n \times n}$  is computed from the standardized data matrix. This matrix captures the pairwise relationships between variables. When two features are highly correlated, they introduce redundancy by adding the same information twice. If the covariance is positive, two variables increase or decrease together, meaning they are correlated. When the covariance is negative, one variable increases when the other decreases, meaning they are inversely correlated.

The covariance between features  $j$  and  $k$  is defined as:

$$\text{Cov}(x_j, x_k) = \frac{1}{m} \sum_{i=1}^m (X_{ij} - \mu_j)(X_{ik} - \mu_k) \quad (\text{A.2})$$

Where  $X_{ij}$  is the value of feature  $j$  for instance  $i$ ,  $\mu_j$  is the mean of feature  $j$ , and  $m$  is the number of instances (rows). This results in a symmetric covariance matrix:

$$\begin{bmatrix} \text{Cov}(x_1, x_1) & \text{Cov}(x_1, x_2) & \dots & \text{Cov}(x_1, x_n) \\ \text{Cov}(x_2, x_1) & \text{Cov}(x_2, x_2) & \dots & \text{Cov}(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}(x_n, x_1) & \text{Cov}(x_n, x_2) & \dots & \text{Cov}(x_n, x_n) \end{bmatrix}$$

Note that  $\text{Cov}(x_j, x_k) = \text{Cov}(x_k, x_j)$ , making the matrix symmetric. The diagonal entries correspond to the variance of each feature:  $\text{Cov}(x_j, x_j) = \text{Var}(x_j)$ .

**Step 3: Find the principal components**

The goal of PCA is to find new orthogonal directions in feature space (called principal components) that capture the maximum variance in the data. These directions are derived from the eigenvectors of the covariance matrix  $\mathbf{C} \in \mathbb{R}^{n \times n}$ .

To compute them, the characteristic equation to obtain the eigenvalues  $\lambda$  is solved:

$$\det(\mathbf{C} - \lambda \mathbf{I}) = 0 \tag{A.3}$$

Each solution  $\lambda$  is then substituted into the eigenvalue equation to find the corresponding eigenvector  $\mathbf{v}$  by satisfying:

$$\mathbf{C}\mathbf{v} = \lambda\mathbf{v} \tag{A.4}$$

where:

- $\mathbf{v} \in \mathbb{R}^n$  is an eigenvector (i.e., the direction of a principal component)
- $\lambda \in \mathbb{R}$  is the corresponding eigenvalue (i.e., the amount of variance captured along  $\mathbf{v}$ )

Each eigenvector  $\mathbf{v}$  defines a new axis in the transformed feature space, and the associated eigenvalue  $\lambda$  quantifies how much of the total variance is explained in that direction. Since the covariance matrix is symmetric and positive semi-definite, all eigenvalues are real and non-negative, and the eigenvectors form an orthonormal basis.

A property of this transformation is that multiplying  $\mathbf{C}$  by an eigenvector  $\mathbf{v}$  results in a scaled version of  $\mathbf{v}$ , without changing its direction. This is what makes eigenvectors suitable as the new axes in PCA. The image below illustrates this concept visually:

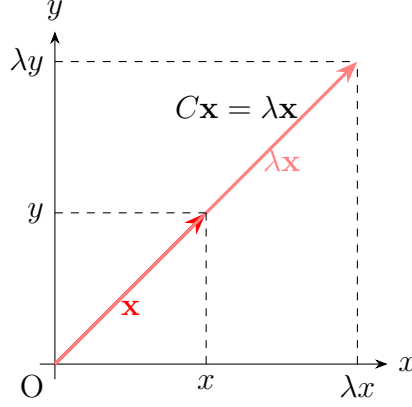


Figure A.1: Geometric interpretation of an eigenvector: direction is unchanged, only scaled by  $\lambda$ .

**Step 4: Picking top directions**

The eigenvectors are sorted in descending order of their corresponding eigenvalues. The first principal component corresponds to the eigenvector with the largest eigenvalue and captures the maximum variance in the data; the second component captures the next highest amount of variance (orthogonal to the first), and so on.

By selecting the top  $p$  principal components (e.g., those explaining 80% of the total variance), the original dataset is projected onto a lower-dimensional space. This results in a new dataset of size  $m \times p$ , where  $p < n$ .

## A.2 Multiple Correspondence Analysis

Suppose we have a dataset with  $m$  observations and  $K$  categorical variables. Let each variable  $k$  have  $J_k$  distinct levels (categories), with  $J = \sum_{k=1}^K J_k$  being the total number of category levels across all variables. The resulting indicator matrix  $\mathbf{X} \in \mathbb{R}^{m \times J}$  contains binary entries, where  $X_{ij} = 1$  if observation  $i$  belongs to category  $j$ , and 0 otherwise. The steps of MCA can be outlined as follows:

**Step 1: Compute the indicator matrix**

Each categorical variable is transformed via one-hot encoding, producing an indicator matrix  $\mathbf{X} \in \mathbb{R}^{m \times J}$ . Each row corresponds to an observation, and each column to a unique category level. The entries of  $\mathbf{X}$  are binary, such that  $X_{ij} = 1$  if observation  $i$  belongs to category level  $j$ , and 0 otherwise.

**Step 2: Compute the correspondence matrix**

The indicator matrix is converted into a correspondence matrix by normalizing each element by the total number of all entries. Since each observation selects one level per categorical variable, the total number of ones in  $\mathbf{X}$  is  $N = \sum_{i=1}^m \sum_{j=1}^J X_{ij}$ . A normalized matrix  $\mathbf{Z}$  is made by converting the absolute counts into relative frequencies:

$$\mathbf{Z} = \frac{1}{N} \mathbf{X} \quad (\text{A.5})$$

**Step 3: Compute row and column marginals**

The marginal distributions (i.e., relative frequencies) of the rows and columns of the correspondence matrix are computed as:

$$\mathbf{r} = \mathbf{Z}\mathbf{1} \quad \text{and} \quad \mathbf{c} = \mathbf{Z}^T\mathbf{1} \quad (\text{A.6})$$

Where  $\mathbf{r} \in \mathbb{R}^{m \times 1}$  contains the row sums (marginal frequencies of observations) and  $\mathbf{c} \in \mathbb{R}^{J \times 1}$  contains the column sums (marginal frequencies of category levels)

Let  $D_r = \text{diag}(\mathbf{r})$  and  $D_c = \text{diag}(\mathbf{c})$  denote the diagonal matrices formed from the row and column marginals, respectively. These will be used for normalization in the next step.

**Step 4: Centering and normalization**

The influence of expected frequencies under independence are removed by centering the correspondence matrix  $\mathbf{Z}$ .

$$\mathbf{Z} - \mathbf{r}\mathbf{c}^T \quad (\text{A.7})$$

Here,  $\mathbf{r}\mathbf{c}^T$  represents the matrix of expected frequencies under the assumption that rows and columns are independent. Next, the centered matrix is scaled by the inverse square roots of the row and column marginal frequencies to normalize the contribution of each cell. This yields the standardized residual matrix:

$$\mathbf{S} = D_r^{-1/2}(\mathbf{Z} - \mathbf{r}\mathbf{c}^T)D_c^{-1/2} \quad (\text{A.8})$$

This transformation ensures that all categories and observations are weighted equally, preventing categories with large marginal frequencies from dominating the analysis.

**Step 5: Singular Value Decomposition (SVD)**

The principal components found by decomposing the standardized residual matrix  $\mathbf{S}$  using singular value decomposition (SVD):

$$\mathbf{S} = \mathbf{P}\mathbf{\Delta}\mathbf{Q}^T \quad (\text{A.9})$$

where,  $\mathbf{P}$  contains the left singular vectors (associated with the rows),  $\mathbf{Q}$  contains the right singular vectors (associated with the columns), and  $\mathbf{\Delta}$  is a diagonal matrix of singular values  $\delta_1, \delta_2, \dots$

The squared singular values,  $\delta_i^2$ , represent the eigenvalues of the matrix  $\mathbf{S}^T\mathbf{S}$ , and indicate the proportion of inertia (similar to variance in PCA) explained by each principal component.

**Step 6: Compute coordinates**

The coordinates of the rows and columns in the new reduced space are computed using the singular vectors and singular values obtained from the SVD.

These coordinates define the projection of observations and categories onto the principal components.

$$F = D_r^{-1/2} P \Delta \quad \text{and} \quad G = D_c^{-1/2} Q \Delta \quad (\text{A.10})$$

Where  $F$  contains the coordinates of the  $m$  observations (rows),  $G$  contains the coordinates of the  $J$  category levels (columns),  $P$  and  $Q$  are the left and right singular vectors from the decomposition of  $\mathbf{S}$ , and  $\Delta$  is the diagonal matrix of singular values.

These coordinates are used to interpret the relationships between observations and categorical variables in the lower-dimensional space.

**Step 7: Distance measure**

MCA uses the  $\chi^2$  distance to quantify dissimilarities between observations (rows) and between categories (columns). This distance metric reflects how much an observation or category deviates from the average profile, taking into account the frequency distribution of the data.

In the reduced component space, the squared distances of observations and categories from the centroid can be approximated by:

$$d_r = \text{diag}\{FF^T\} \quad \text{and} \quad d_c = \text{diag}\{GG^T\} \quad (\text{A.11})$$

where  $F$  and  $G$  are the coordinate matrices of the observations and category levels, respectively.

The  $\chi^2$  distance incorporates both observed and expected frequencies, making it sensitive to associations between rare and common categories. As a result, MCA is particularly well-suited to uncovering subtle relationships in categorical data.



# Appendix B

## Results

### B.1 PCA and MCA component summary table

component	eigenvalue	% of variance	% of variance (cumulative)
1	4.125	29.46	29.46%
2	2.497	17.84	47.30%
3	1.701	12.15	59.45%
4	1.424	10.17	69.62%
5	1.029	7.35	76.97%
6	0.881	6.3	83.27%
7	0.824	5.89	89.16%
8	0.745	5.32	94.48%
9	0.521	3.72	98.20%
10	0.126	0.9	99.10%
11	0.074	0.53	99.62%
12	0.053	0.38	100.00%
13	0.000	0.0	100.00%
14	0.000	0.0	100.00%

Table B.1: Eigenvalues and explained variance of the principal components in PCA.

Table B.2: Eigenvalues and explained variance of the principal components in MCA.

Component	Eigenvalue	% of variance	% of variance (cumulative)
1	0.236	5.21%	5.21%
2	0.226	4.99%	10.19%
3	0.177	3.90%	14.09%
4	0.163	3.60%	17.69%
5	0.160	3.53%	21.23%
6	0.146	3.23%	24.46%
7	0.144	3.19%	27.65%
8	0.137	3.03%	30.68%

*Continued on next page*

Table B.2 – *continued from previous page*

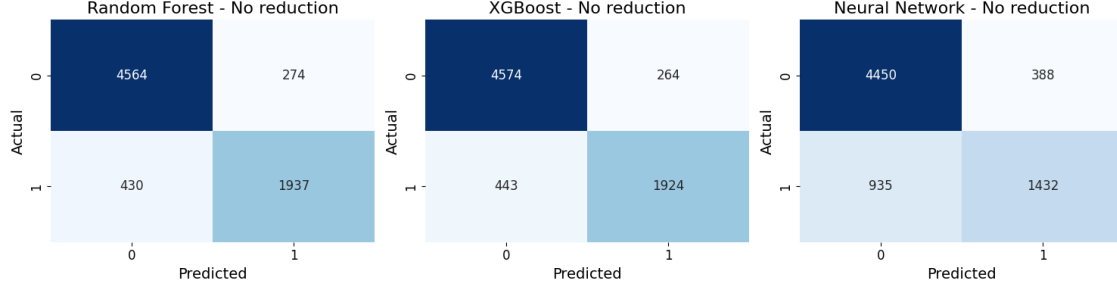
Component	Eigenvalue	% of variance	% of variance (cumulative)
9	0.136	3.00%	33.68%
10	0.134	2.95%	36.63%
11	0.132	2.92%	39.55%
12	0.130	2.87%	42.42%
13	0.127	2.80%	45.21%
14	0.124	2.75%	47.96%
15	0.123	2.71%	50.68%
16	0.123	2.71%	53.38%
17	0.122	2.69%	56.08%
18	0.120	2.65%	58.73%
19	0.120	2.64%	61.37%
20	0.118	2.61%	63.97%
21	0.116	2.57%	66.54%
22	0.116	2.56%	69.10%
23	0.115	2.54%	71.65%
24	0.114	2.51%	74.16%
25	0.112	2.47%	76.63%
26	0.109	2.40%	79.03%
27	0.106	2.34%	81.36%
28	0.103	2.28%	83.64%
29	0.100	2.22%	85.86%
30	0.096	2.12%	87.98%
31	0.090	1.98%	89.97%
32	0.088	1.95%	91.92%
33	0.081	1.80%	93.72%
34	0.066	1.47%	95.18%
35	0.061	1.34%	96.52%
36	0.057	1.26%	97.78%
37	0.048	1.06%	98.85%
38	0.034	0.74%	99.59%
39	0.019	0.41%	100.00%
40	0.000	0.00%	100.00%
41	0.000	0.00%	100.00%
42	0.000	0.00%	100.00%
43	0.000	0.00%	100.00%
44	0.000	0.00%	100.00%
45	0.000	0.00%	100.00%

## B.2 McNemar statistical test

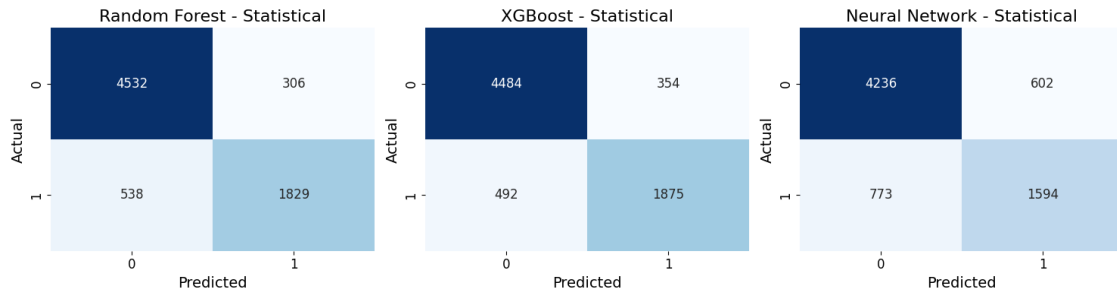
	MLP				RF				XGB			
	AE-16	AE-32	PCA	None	AE-16	AE-32	PCA	None	AE-16	AE-32	PCA	None
MLP (AE-16)	1.00											
MLP (AE-32)	9.8e-05***	1.00										
MLP (PCA)	2.1e-20***	1.9e-36***	1.00									
MLP (None)	1.5e-14***	1.5e-26***	0.12	1.00								
RF (AE-16)	6.8e-11***	0.01**	5.7e-42***	3.2e-34***	1.00	1.00						
RF (AE-32)	1.4e-13***	1.7e-04***	2.1e-47***	9.2e-40***	0.10	0.01**	1.00					
RF (PCA)	1.2e-21***	1.1e-09***	2.6e-67***	1.5e-54***	1.0e-04***	1.6e-18***	2.2e-11***	1.00				
RF (None)	9.5e-51***	1.4e-32***	1.4e-108***	1.5e-90***	9.7e-23***	0.76	3.0e-04***	1.1e-21***	1.00			
XGB (AE-16)	6.8e-11***	0.01**	9.9e-43***	2.0e-34***	0.02*	0.28	0.06	3.1e-16***	0.04*	1.00		
XGB (AE-32)	7.1e-16***	4.4e-06***	1.7e-51***	3.5e-42***	1.2e-04***	0.01**	0.96	9.7e-11***	2.3e-04***	0.06	1.00	
XGB (PCA)	3.5e-19***	8.2e-09***	6.1e-60***	3.9e-50***	8.3e-22***	8.8e-18***	7.2e-11***	0.90	6.9e-21***	1.0e-15***	9.3e-11***	1.00
XGB (None)	6.9e-49***	1.4e-31***	2.4e-105***	2.1e-89***								

Table B.3:  $p$ -values of pairwise McNemar's test with FDR correction. Values  $\geq 0.001$  are shown with two decimals; smaller values are shown in scientific notation. Significance: \*  $p < 0.05$ , \*\*  $p < 0.01$ , \*\*\*  $p < 0.001$ . Lower triangle only.

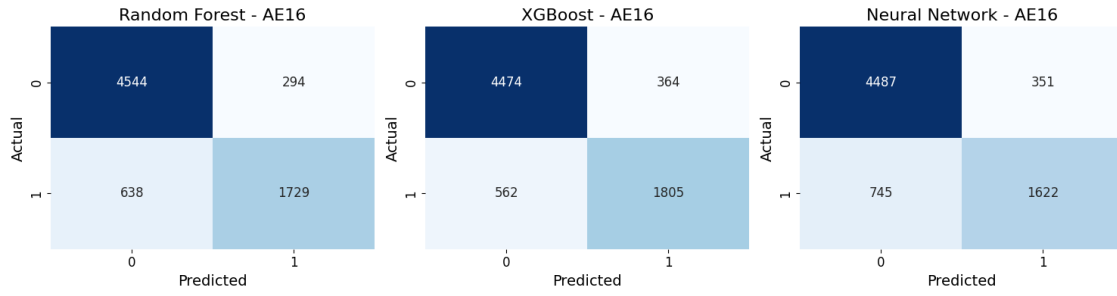
### B.3 Confusion matrices of all configurations



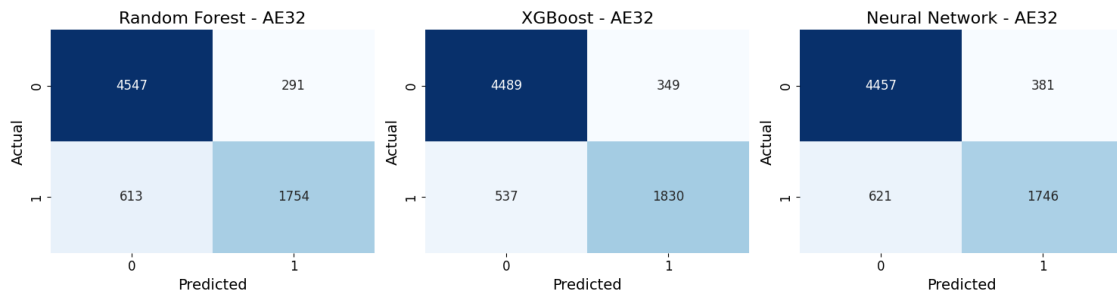
(a) Without dimensionality reduction.



(b) Statistical dimensionality reduction.



(c) Dimensionality reduction with autoencoder and bottleneck size of 16.



(d) Dimensionality reduction with autoencoder and bottleneck size of 32.

Figure B.1: Confusion matrices illustrating the trade-off between false positives and false negatives for the three models.

## B.4 Hyperparameter in-depth analysis

To illustrate the optimization process in more detail, Figures B.2 and B.3 present the hyperparameter importance and slice plots provided by Optuna for two representative cases: XGBoost without DR and the MLP with AE-32. For XGBoost, `max_depth` emerges as the dominant hyperparameter, with shallow trees performing poorly and deeper trees stabilizing quickly (Figure B.3a). For `colsample_bytree`, which controls the number of features considered in each tree, the objective values remain stable across the tested range, indicating that model performance does not systematically improve or worsen with higher values. This is consistent with the low importance score.

For the MLP with AE-32, Figure B.2b shows that dropout rate, learning rate, and the number of hidden units are the most influential parameters. Upon investigating Figure B.3b, performance is best with very low dropout rates, while increasing dropout quickly reduces the objective.

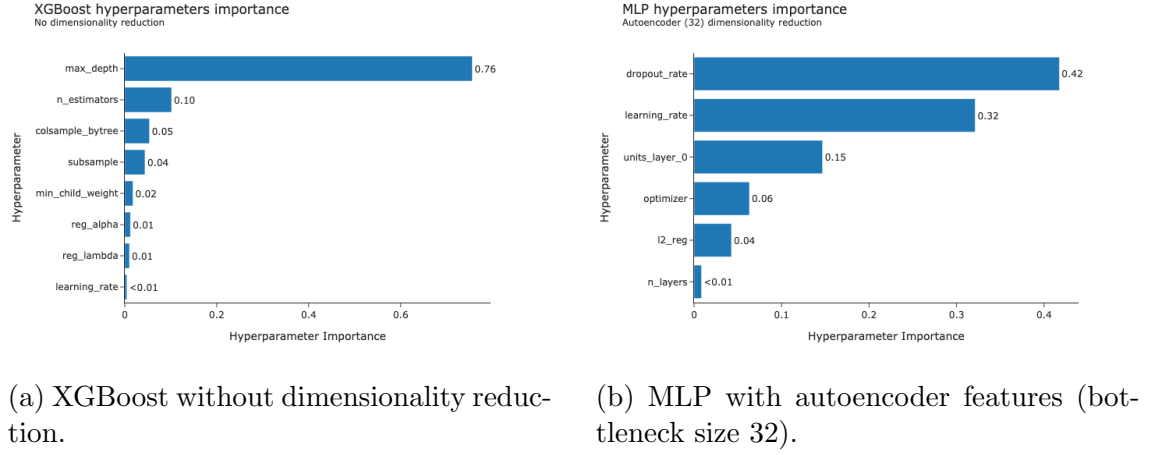


Figure B.2: Relative hyperparameter importance obtained through Optuna.

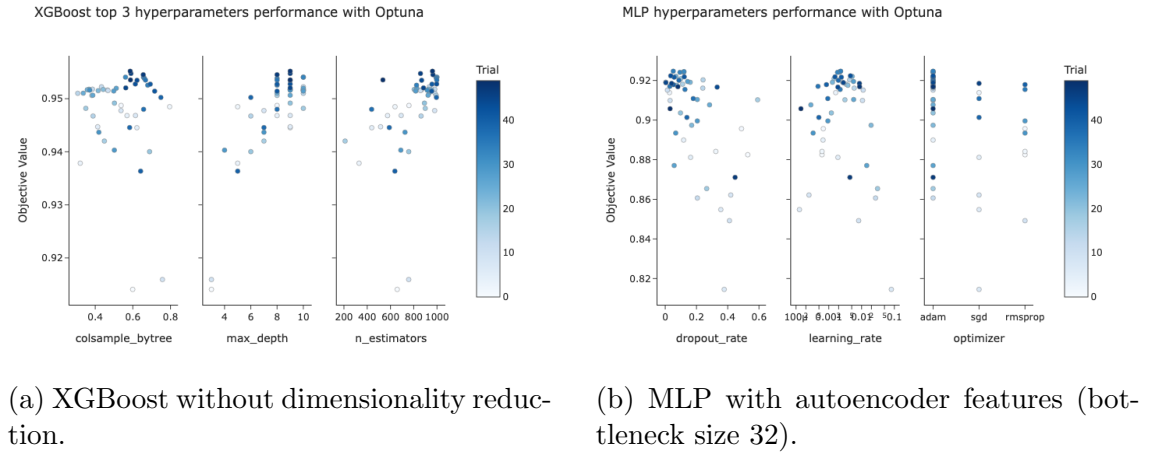


Figure B.3: Optuna slice plots of the three most influential hyperparameters for each model. Each point represents one of the 50 optimization trials and its corresponding hyperparameter setting.

# Bibliography

- [1] H. Abdi and D. Valentin. “Multiple Correspondence Analysis”. In: *Encyclopedia of Measurement and Statistics* (2007).
- [2] G. Abrate and G. Viglia. “Strategic and Tactical Price Decisions in Hotel Revenue Management”. In: *Tourism Management* 55 (Aug. 2016), pp. 123–132. ISSN: 02615177. DOI: [10.1016/j.tourman.2016.02.006](https://doi.org/10.1016/j.tourman.2016.02.006). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0261517716300188>.
- [3] S. F. Ahmed et al. “Deep Learning Modelling Techniques: Current Progress, Applications, Advantages, and Challenges”. In: *Artificial Intelligence Review* 56.11 (Nov. 2023), pp. 13521–13617. ISSN: 0269-2821, 1573-7462. DOI: [10.1007/s10462-023-10466-8](https://doi.org/10.1007/s10462-023-10466-8). URL: <https://link.springer.com/10.1007/s10462-023-10466-8>.
- [4] T. Akiba et al. *Optuna: A Next-generation Hyperparameter Optimization Framework*. July 25, 2019. DOI: [10.48550/arXiv.1907.10902](https://doi.org/10.48550/arXiv.1907.10902). arXiv: [1907.10902 \[cs\]](https://arxiv.org/abs/1907.10902). URL: <http://arxiv.org/abs/1907.10902>. Pre-published.
- [5] M. Alkhayrat et al. “A Comparative Dimensionality Reduction Study in Telecom Customer Segmentation Using Deep Learning and PCA”. In: *Journal of Big Data* 7.1 (Dec. 2020), p. 9. ISSN: 2196-1115. DOI: [10.1186/s40537-020-0286-0](https://doi.org/10.1186/s40537-020-0286-0). URL: <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-020-0286-0>.
- [6] N. Antonio et al. “Predicting Hotel Booking Cancellations to Decrease Uncertainty and Increase Revenue”. In: *Tourism & Management Studies* 13.2 (Apr. 30, 2017), pp. 25–39. ISSN: 21828458, 21828466. DOI: [10.18089/tms.2017.13203](https://doi.org/10.18089/tms.2017.13203). URL: [http://www.tmstudies.net/index.php/ectms/article/view/1000/pdf\\_51](http://www.tmstudies.net/index.php/ectms/article/view/1000/pdf_51).
- [7] N. Antonio et al. “Big Data in Hotel Revenue Management: Exploring Cancellation Drivers to Gain Insights Into Booking Cancellation Behavior”. In: *Cornell Hospitality Quarterly* 60.4 (Nov. 2019), pp. 298–319. ISSN: 1938-9655, 1938-9663. DOI: [10.1177/1938965519851466](https://doi.org/10.1177/1938965519851466). URL: <https://journals.sagepub.com/doi/10.1177/1938965519851466>.
- [8] N. Antonio et al. “Hotel Booking Demand Datasets”. In: *Data in Brief* 22 (Feb. 2019), pp. 41–49. ISSN: 23523409. DOI: [10.1016/j.dib.2018.11.126](https://doi.org/10.1016/j.dib.2018.11.126). URL: <https://linkinghub.elsevier.com/retrieve/pii/S2352340918315191>.
- [9] S. O. Arik and T. Pfister. *TabNet: Attentive Interpretable Tabular Learning*. Dec. 9, 2020. DOI: [10.48550/arXiv.1908.07442](https://doi.org/10.48550/arXiv.1908.07442). arXiv: [1908.07442 \[cs\]](https://arxiv.org/abs/1908.07442). URL: <http://arxiv.org/abs/1908.07442>. Pre-published.

- [10] D. Bank et al. *Autoencoders*. Apr. 3, 2021. DOI: [10.48550/arXiv.2003.05991](https://doi.org/10.48550/arXiv.2003.05991). arXiv: [2003.05991 \[cs\]](https://arxiv.org/abs/2003.05991). URL: <http://arxiv.org/abs/2003.05991>. Pre-published.
- [11] R. Bellman. *Adaptive Control Processes*. Princeton University Press, 1961. ISBN: 978-0-691-07901-1. JSTOR: [j.ctt183ph6v](https://www.jstor.org/stable/j.ctt183ph6v). URL: <http://www.jstor.org/stable/j.ctt183ph6v>.
- [12] D. Bera et al. “Dimensionality Reduction for Categorical Data”. In: *IEEE Transactions on Knowledge and Data Engineering* 35.4 (Apr. 1, 2023), pp. 3658–3671. ISSN: 1041-4347, 1558-2191, 2326-3865. DOI: [10.1109/TKDE.2021.3132373](https://doi.org/10.1109/TKDE.2021.3132373). arXiv: [2112.00362 \[cs\]](https://arxiv.org/abs/2112.00362). URL: <http://arxiv.org/abs/2112.00362>.
- [13] J. Bergstra et al. “Algorithms for Hyper-Parameter Optimization”. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems*. NIPS’11. Red Hook, NY, USA: Curran Associates Inc., Dec. 12, 2011, pp. 2546–2554. ISBN: 978-1-61839-599-3.
- [14] V. Borisov et al. “Deep Neural Networks and Tabular Data: A Survey”. In: *IEEE Transactions on Neural Networks and Learning Systems* 35.6 (June 2024), pp. 7499–7519. ISSN: 2162-2388. DOI: [10.1109/TNNLS.2022.3229161](https://doi.org/10.1109/TNNLS.2022.3229161). URL: <https://ieeexplore.ieee.org/abstract/document/9998482>.
- [15] L. Breiman. “Random Forests”. In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32. ISSN: 0885-6125, 1573-0565. DOI: [10.1023/A:1010933404324](https://doi.org/10.1023/A:1010933404324). URL: <https://link.springer.com/10.1023/A:1010933404324>.
- [16] J. D. Brown. “Choosing the Right Number of Components or Factors in PCA and EFA”. In: *JALT Testing & Evaluation SIG Newsletter* 13.2 (2009), pp. 9–23. ISSN: 1881-5537.
- [17] M. Brunette et al. “Adaptation to Climate Change in Forestry: A Multiple Correspondence Analysis (MCA)”. In: *Forests* 9.1 (Jan. 6, 2018), p. 20. ISSN: 1999-4907. DOI: [10.3390/f9010020](https://doi.org/10.3390/f9010020). URL: <https://www.mdpi.com/1999-4907/9/1/20>.
- [18] K. Budholiya et al. “An Optimized XGBoost Based Diagnostic System for Effective Prediction of Heart Disease”. In: *Journal of King Saud University - Computer and Information Sciences* 34.7 (July 2022), pp. 4514–4523. ISSN: 13191578. DOI: [10.1016/j.jksuci.2020.10.013](https://doi.org/10.1016/j.jksuci.2020.10.013). URL: <https://linkinghub.elsevier.com/retrieve/pii/S1319157820304936>.
- [19] L. Cao et al. “A Comparison of PCA, KPCA and ICA for Dimensionality Reduction in Support Vector Machine”. In: *Neurocomputing* 55.1–2 (Sept. 2003), pp. 321–336. ISSN: 09252312. DOI: [10.1016/S0925-2312\(03\)00433-8](https://doi.org/10.1016/S0925-2312(03)00433-8). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0925231203004338>.
- [20] Y.-c. I. Chang. *A Survey: Potential Dimensionality Reduction Methods*. Feb. 18, 2025. DOI: [10.48550/arXiv.2502.11036](https://doi.org/10.48550/arXiv.2502.11036). arXiv: [2502.11036 \[stat\]](https://arxiv.org/abs/2502.11036). URL: <http://arxiv.org/abs/2502.11036>. Pre-published.

- [21] T. Chen and C. Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16: The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Francisco California USA: ACM, Aug. 13, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: [10.1145/2939672.2939785](https://doi.org/10.1145/2939672.2939785). URL: <https://dl.acm.org/doi/10.1145/2939672.2939785>.
- [22] A. Creswell et al. *On Denoising Autoencoders Trained to Minimise Binary Cross-Entropy*. Oct. 9, 2017. DOI: [10.48550/arXiv.1708.08487](https://doi.org/10.48550/arXiv.1708.08487). arXiv: [1708.08487 \[cs\]](https://arxiv.org/abs/1708.08487). URL: <http://arxiv.org/abs/1708.08487>. Pre-published.
- [23] P. Domingos. “A Few Useful Things to Know about Machine Learning”. In: *Communications of the ACM* 55.10 (Oct. 2012), pp. 78–87. ISSN: 0001-0782, 1557-7317. DOI: [10.1145/2347736.2347755](https://doi.org/10.1145/2347736.2347755). URL: <https://dl.acm.org/doi/10.1145/2347736.2347755>.
- [24] G. Dutta and K. Mitra. “A Literature Review on Dynamic Pricing of Electricity”. In: *Journal of the Operational Research Society* 68.10 (Oct. 2017), pp. 1131–1145. ISSN: 0160-5682, 1476-9360. DOI: [10.1057/s41274-016-0149-4](https://doi.org/10.1057/s41274-016-0149-4). URL: <https://www.tandfonline.com/doi/full/10.1057/s41274-016-0149-4>.
- [25] Y. Elor and H. Averbuch-Elor. *To SMOTE, or Not to SMOTE?* Version 3. 2022. DOI: [10.48550/ARXIV.2201.08528](https://doi.org/10.48550/ARXIV.2201.08528). URL: <https://arxiv.org/abs/2201.08528>. Pre-published.
- [26] P. I. Frazier. *A Tutorial on Bayesian Optimization*. July 8, 2018. DOI: [10.48550/arXiv.1807.02811](https://doi.org/10.48550/arXiv.1807.02811). arXiv: [1807.02811 \[stat\]](https://arxiv.org/abs/1807.02811). URL: <http://arxiv.org/abs/1807.02811>. Pre-published.
- [27] G. Gallego and G. Van Ryzin. “Optimal Dynamic Pricing of Inventories with Stochastic Demand over Finite Horizons”. In: *Management Science* 40.8 (Aug. 1994), pp. 999–1020. ISSN: 0025-1909, 1526-5501. DOI: [10.1287/mnsc.40.8.999](https://doi.org/10.1287/mnsc.40.8.999). URL: <https://pubsonline.informs.org/doi/10.1287/mnsc.40.8.999>.
- [28] M. J. Greenacre. “Correspondence Analysis”. In: *WIREs Computational Statistics* 2.5 (Sept. 2010), pp. 613–619. ISSN: 1939-5108, 1939-0068. DOI: [10.1002/wics.114](https://doi.org/10.1002/wics.114). URL: <https://wires.onlinelibrary.wiley.com/doi/10.1002/wics.114>.
- [29] C. Guo and F. Berkhahn. *Entity Embeddings of Categorical Variables*. Apr. 22, 2016. DOI: [10.48550/arXiv.1604.06737](https://doi.org/10.48550/arXiv.1604.06737). arXiv: [1604.06737 \[cs\]](https://arxiv.org/abs/1604.06737). URL: <http://arxiv.org/abs/1604.06737>. Pre-published.
- [30] J. Haessner et al. “Dynamic Pricing in Different Industries”. In: *Journal of Marketing Development and Competitiveness* 17.4 (Dec. 31, 2023). ISSN: 2155-2843. DOI: [10.33423/jmdc.v17i4.6735](https://doi.org/10.33423/jmdc.v17i4.6735). URL: <https://articlegateway.com/index.php/JMDC/article/view/6735>.
- [31] H. Hajibaba et al. “Preventing Tourists from Canceling in Times of Crises”. In: *Annals of Tourism Research* 60 (Sept. 2016), pp. 48–62. ISSN: 01607383. DOI: [10.1016/j.annals.2016.06.003](https://doi.org/10.1016/j.annals.2016.06.003). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0160738316300858>.



- [32] *History of Hospitality: The Industry's Remarkable Journey*. URL: <https://www.shms.com/en/news/history-of-hospitality/>.
- [33] *Hotels - Worldwide — Statista Market Forecast*. Statista. URL: <http://frontend.xmo.prod.aws.statista.com/outlook/mmo/travel-tourism/hotels/worldwide>.
- [34] *How to Prevent Hotel No-Show and Last-Minute Cancellations?* D-EDGE. Oct. 29, 2024. URL: <https://www.d-edge.com/avoid-guests-ghosting/>.
- [35] H.-C. Huang et al. "Using Artificial Neural Networks to Establish a Customer-cancellation Prediction Model". In: *PRZEGLAD ELEKTROTECHNICZNY* 89 (1b 2013), pp. 178–180. URL: [https://www.researchgate.net/publication/290226043\\_Using\\_Artificial\\_Neural\\_Networks\\_to\\_Establish\\_a\\_Customer-cancellation\\_Prediction\\_Model](https://www.researchgate.net/publication/290226043_Using_Artificial_Neural_Networks_to_Establish_a_Customer-cancellation_Prediction_Model).
- [36] N. S. Keskar et al. *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima*. Feb. 9, 2017. DOI: [10.48550/arXiv.1609.04836](https://doi.org/10.48550/arXiv.1609.04836). arXiv: [1609.04836](https://arxiv.org/abs/1609.04836) [cs]. URL: <http://arxiv.org/abs/1609.04836>. Pre-published.
- [37] E. J. Kim et al. "Post-Pandemic Hotel Cancellation Policy: Situational Cues as Perceived Risk Triggers". In: *Journal of Hospitality and Tourism Management* 55 (June 2023), pp. 153–160. ISSN: 14476770. DOI: [10.1016/j.jhtm.2023.03.019](https://doi.org/10.1016/j.jhtm.2023.03.019). URL: <https://linkinghub.elsevier.com/retrieve/pii/S1447677023000499>.
- [38] W. E. Marcilio and D. M. Eler. "From Explanations to Feature Selection: Assessing SHAP Values as Feature Selection Mechanism". In: *2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*. 2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI). Recife/Porto de Galinhas, Brazil: IEEE, Nov. 2020, pp. 340–347. ISBN: 978-1-7281-9274-1. DOI: [10.1109/SIBGRAPI51738.2020.00053](https://doi.org/10.1109/SIBGRAPI51738.2020.00053). URL: <https://ieeexplore.ieee.org/document/9265985/>.
- [39] J. Matayoshi and S. Karumbaiah. "Investigating the Validity of Methods Used to Adjust for Multiple Comparisons in Educational Data Mining". In: (June 2021).
- [40] M. A. Mediavilla et al. "Review and Analysis of Artificial Intelligence Methods for Demand Forecasting in Supply Chain Management". In: *Procedia CIRP* 107 (2022), pp. 1126–1131. ISSN: 22128271. DOI: [10.1016/j.procir.2022.05.119](https://doi.org/10.1016/j.procir.2022.05.119). URL: <https://linkinghub.elsevier.com/retrieve/pii/S2212827122004036>.
- [41] D. R. Morales and J. Wang. "Forecasting Cancellation Rates for Services Booking Revenue Management Using Data Mining". In: *European Journal of Operational Research* 202.2 (Apr. 2010), pp. 554–562. ISSN: 03772217. DOI: [10.1016/j.ejor.2009.06.006](https://doi.org/10.1016/j.ejor.2009.06.006). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0377221709004494>.

- [42] T. M. Oshiro et al. “How Many Trees in a Random Forest?” In: *Machine Learning and Data Mining in Pattern Recognition*. Ed. by P. Perner. Red. by D. Hutchison et al. Vol. 7376. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 154–168. ISBN: 978-3-642-31536-7 978-3-642-31537-4. DOI: [10.1007/978-3-642-31537-4\\_13](https://doi.org/10.1007/978-3-642-31537-4_13). URL: [http://link.springer.com/10.1007/978-3-642-31537-4\\_13](http://link.springer.com/10.1007/978-3-642-31537-4_13).
- [43] K. Pearson. “LIII. On Lines and Planes of Closest Fit to Systems of Points in Space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (Nov. 1901), pp. 559–572. ISSN: 1941-5982, 1941-5990. DOI: [10.1080/14786440109462720](https://doi.org/10.1080/14786440109462720). URL: <https://www.tandfonline.com/doi/full/10.1080/14786440109462720>.
- [44] N. Phumchusri and P. Maneesophon. “Optimal Overbooking Decision for Hotel Rooms Revenue Management”. In: *Journal of Hospitality and Tourism Technology* 5.3 (Oct. 14, 2014), pp. 261–277. ISSN: 1757-9880. DOI: [10.1108/JHTT-03-2014-0006](https://doi.org/10.1108/JHTT-03-2014-0006). URL: <https://www.emerald.com/insight/content/doi/10.1108/JHTT-03-2014-0006/full/html>.
- [45] P. Probst et al. “Hyperparameters and Tuning Strategies for Random Forest”. In: *WIREs Data Mining and Knowledge Discovery* 9.3 (May 2019), e1301. ISSN: 1942-4787, 1942-4795. DOI: [10.1002/widm.1301](https://doi.org/10.1002/widm.1301). arXiv: [1804.03515](https://arxiv.org/abs/1804.03515) [stat]. URL: <http://arxiv.org/abs/1804.03515>.
- [46] F. Rosenblatt. “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.” In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 1939-1471, 0033-295X. DOI: [10.1037/h0042519](https://doi.org/10.1037/h0042519). URL: <https://doi.org/10.1037/h0042519>.
- [47] B. M. Salih Hasan et al. “A Review of Principal Component Analysis Algorithm for Dimensionality Reduction”. In: *Journal of Soft Computing and Data Mining* 02.01 (Apr. 15, 2021). ISSN: 2716621X. DOI: [10.30880/jscdm.2021.02.01.003](https://doi.org/10.30880/jscdm.2021.02.01.003). URL: <https://publisher.uthm.edu.my/ojs/index.php/jscdm/article/view/8032>.
- [48] A. J. Sánchez-Medina and E. C-Sánchez. “Using Machine Learning and Big Data for Efficient Forecasting of Hotel Booking Cancellations”. In: *International Journal of Hospitality Management* 89 (Aug. 1, 2020), p. 102546. ISSN: 0278-4319. DOI: [10.1016/j.ijhm.2020.102546](https://doi.org/10.1016/j.ijhm.2020.102546). URL: <https://www.sciencedirect.com/science/article/pii/S0278431920300980>.
- [49] R. Shwartz-Ziv and A. Armon. *Tabular Data: Deep Learning Is Not All You Need*. Nov. 23, 2021. DOI: [10.48550/arXiv.2106.03253](https://doi.org/10.48550/arXiv.2106.03253). arXiv: [2106.03253](https://arxiv.org/abs/2106.03253) [cs]. URL: <http://arxiv.org/abs/2106.03253>. Pre-published.
- [50] L. N. Smith. *A Disciplined Approach to Neural Network Hyper-Parameters: Part 1 – Learning Rate, Batch Size, Momentum, and Weight Decay*. Apr. 24, 2018. DOI: [10.48550/arXiv.1803.09820](https://doi.org/10.48550/arXiv.1803.09820). arXiv: [1803.09820](https://arxiv.org/abs/1803.09820) [cs]. URL: <http://arxiv.org/abs/1803.09820>. Pre-published.
- [51] J. Sommer et al. *Learning to Tune XGBoost with XGBoost*. Nov. 21, 2019. DOI: [10.48550/arXiv.1909.07218](https://doi.org/10.48550/arXiv.1909.07218). arXiv: [1909.07218](https://arxiv.org/abs/1909.07218) [cs]. URL: <http://arxiv.org/abs/1909.07218>. Pre-published.

- [52] C. O. S. Sorzano et al. *A Survey of Dimensionality Reduction Techniques*. Mar. 12, 2014. DOI: [10.48550/arXiv.1403.2877](https://doi.org/10.48550/arXiv.1403.2877). arXiv: [1403.2877 \[stat\]](https://arxiv.org/abs/1403.2877). URL: <http://arxiv.org/abs/1403.2877>. Pre-published.
- [53] L. van der Maaten et al. “Dimensionality Reduction: A Comparative Review”. In: *Journal of Machine Learning Research - JMLR* 10 (Jan. 2007). URL: [https://www.researchgate.net/publication/228657549\\_Dimensionality\\_Reduction\\_A\\_Comparative\\_Review](https://www.researchgate.net/publication/228657549_Dimensionality_Reduction_A_Comparative_Review).
- [54] R. Van Leeuwen. “Data-Driven Strategies in Hospitality”. dr. Vrije Universiteit Amsterdam, Feb. 21, 2024, HFRLN21120240313. ISBN: 9789464733976. DOI: [10.5463/thesis.600](https://doi.org/10.5463/thesis.600). HDL: [1871.1/0cefb27b-5ff4-4bb6-b7f3-a7af8c087526](https://hdl.handle.net/1871.1/0cefb27b-5ff4-4bb6-b7f3-a7af8c087526). URL: <https://hdl.handle.net/1871.1/0cefb27b-5ff4-4bb6-b7f3-a7af8c087526>.
- [55] B. Verot. *Hotel Cancellations: What You Should Know*. URL: <https://www.hotelminder.com/everything-you-need-to-know-about-hotel-cancellations>.
- [56] D. Visbal-Cadavid et al. “Use of Factorial Analysis of Mixed Data (FAMD) and Hierarchical Cluster Analysis on Principal Component (HCPC) for Multivariate Analysis of Academic Performance of Industrial Engineering Programs”. In: *Journal of Southwest Jiaotong University* 55.5 (2020), p. 34. ISSN: 0258-2724. DOI: [10.35741/issn.0258-2724.55.5.34](https://doi.org/10.35741/issn.0258-2724.55.5.34). URL: <http://jsju.org/index.php/journal/article/view/745>.
- [57] S. R. Volovăț et al. “Comparative Performance of Autoencoders and Traditional Machine Learning Algorithms in Clinical Data Analysis for Predicting Post-Staged GKRS Tumor Dynamics”. In: *Diagnostics* 14.18 (Sept. 21, 2024), p. 2091. ISSN: 2075-4418. DOI: [10.3390/diagnostics14182091](https://doi.org/10.3390/diagnostics14182091). URL: <https://www.mdpi.com/2075-4418/14/18/2091>.
- [58] J. Vrábel et al. “Restricted Boltzmann Machine Method for Dimensionality Reduction of Large Spectroscopic Data”. In: *Spectrochimica Acta Part B: Atomic Spectroscopy* 167 (May 2020), p. 105849. ISSN: 05848547. DOI: [10.1016/j.sab.2020.105849](https://doi.org/10.1016/j.sab.2020.105849). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0584854720300410>.
- [59] Y. Wang et al. “Auto-Encoder Based Dimensionality Reduction”. In: *Neurocomputing* 184 (Apr. 2016), pp. 232–242. ISSN: 09252312. DOI: [10.1016/j.neucom.2015.08.104](https://doi.org/10.1016/j.neucom.2015.08.104). URL: <https://linkinghub.elsevier.com/retrieve/pii/S0925231215017671>.
- [60] P. H. Westfall et al. “Multiple McNemar Tests”. In: *Biometrics* 66.4 (Dec. 2010), pp. 1185–1191. ISSN: 0006-341X, 1541-0420. DOI: [10.1111/j.1541-0420.2010.01408.x](https://doi.org/10.1111/j.1541-0420.2010.01408.x). URL: <https://academic.oup.com/biometrics/article/66/4/1185-1191/7333578>.
- [61] S. Yüksel. “An Integrated Forecasting Approach to Hotel Demand”. In: *Mathematical and Computer Modelling*. Decision Making with the Analytic Hierarchy Process and the Analytic Network Process 46.7 (Oct. 1, 2007), pp. 1063–1070. ISSN: 0895-7177. DOI: [10.1016/j.mcm.2007.03.008](https://doi.org/10.1016/j.mcm.2007.03.008). URL: <https://www.sciencedirect.com/science/article/pii/S0895717707000866>.

- [62] Z. Zang et al. *EVNet: An Explainable Deep Network for Dimension Reduction*. Nov. 21, 2022. DOI: [10.48550/arXiv.2211.15478](https://doi.org/10.48550/arXiv.2211.15478). arXiv: [2211.15478](https://arxiv.org/abs/2211.15478) [cs]. URL: <http://arxiv.org/abs/2211.15478>. Pre-published.