Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

MSc Quantitative Finance and Data Science

Wintersemester 2025/2026

Explainable Booking Cancellations Prediction with MLP and Ensemble Methods

Seminar Deep Learning

Evan Leonard
582554

Contents

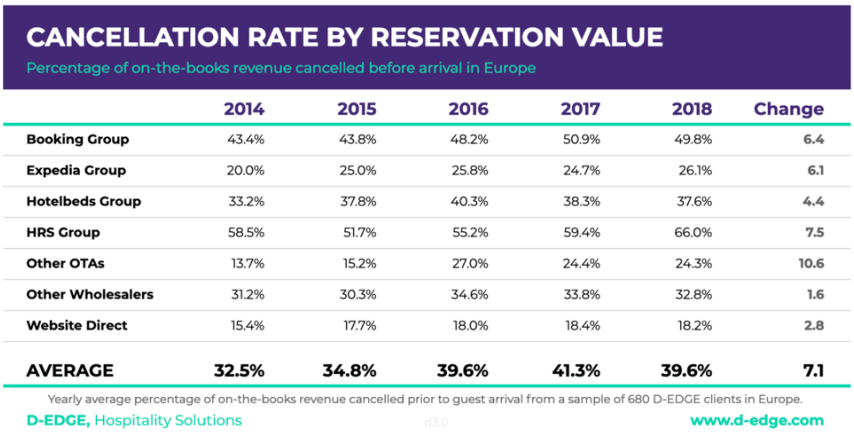
List of Figures

List of Tables

1 Introduction

Hotel booking cancellations present a major challenge for the hospitality industry, leading to revenue loss, inefficient resource allocation, and reduced customer satisfaction. The ability to accurately predict booking cancellations allows hotels to adopt proactive strategies such as optimized overbooking policies, targeted promotional offers, and improved demand forecasting, thereby reducing financial and operational risks.

Recent industry data further highlights the severity of this issue. Figure 1.1 illustrates cancellation rates by reservation value across major booking channels in Europe between 2014 and 2018. On average, approximately 40% of on-the-books revenue is canceled prior to guest arrival, with some online travel agencies (OTAs) exhibiting cancellation rates exceeding 50%. The consistently high cancellation levels across channels and over time underline the structural nature of the problem rather than a temporary market anomaly.



CANCELLATION RATE BY RESERVATION VALUE

Percentage of on-the-books revenue cancelled before arrival in Europe

	2014	2015	2016	2017	2018	Change
Booking Group	43.4%	43.8%	48.2%	50.9%	49.8%	6.4
Expedia Group	20.0%	25.0%	25.8%	24.7%	26.1%	6.1
Hotelbeds Group	33.2%	37.8%	40.3%	38.3%	37.6%	4.4
HRS Group	58.5%	51.7%	55.2%	59.4%	66.0%	7.5
Other OTAs	13.7%	15.2%	27.0%	24.4%	24.3%	10.6
Other Wholesalers	31.2%	30.3%	34.6%	33.8%	32.8%	1.6
Website Direct	15.4%	17.7%	18.0%	18.4%	18.2%	2.8
AVERAGE	**32.5%**	**34.8%**	**39.6%**	**41.3%**	**39.6%**	**7.1**

Yearly average percentage of on-the-books revenue cancelled prior to guest arrival from a sample of 680 D-EDGE clients in Europe.

D-EDGE, Hospitality Solutions www.d-edge.com

Figure 1.1: Cancellation rate by reservation value across booking channels in Europe
Source: D-EDGE Hospitality Solutions

This study is motivated by recent work by Sun [S2025], who demonstrated the effectiveness of machine learning techniques for predicting hotel booking cancellations and emphasized the importance of aligning predictive models with revenue management objectives. Building on this research, the present work focuses on improving cancellation prediction accuracy while ensuring that model outputs remain interpretable and actionable for hotel practitioners.

Three machine learning models are evaluated: Multilayer Perceptron (MLP), Random Forest, and EXtreme Gradient Boosting (XGBoost). The MLP captures complex nonlinear relationships in booking behavior, Random Forest improves robustness through ensemble learning, and XGBoost applies gradient boosting to enhance predictive performance on structured data.

Beyond overall accuracy, particular emphasis is placed on precision and recall for Class 1, representing canceled bookings. High recall is essential to minimize missed cancellations that could result in empty rooms and lost revenue, while high precision ensures that predicted cancellations are reliable, preventing unnecessary overbooking or costly interventions. Given the high cancellation rates observed in Figure 1.1, optimizing these metrics is critical for effective hotel revenue and capacity management.

To enhance model transparency, explainable artificial intelligence (XAI) techniques are employed, including SHapley Additive ExPlanations (SHAP), Local Interpretable Model-agnostic Explanations (LIME), and Permutation Feature Importance. These methods provide insights into the key factors driving cancellation behavior, supporting data-driven and trustworthy decision-making in hotel management.

2 Theoretical Framework

2.1 Multilayer Perceptron (MLP)

Cf. [A2023] Ch. 1, [DS2013] Ch. 2.3, 5, [S2023] Ch. 3.3, 14, [Z2021] Ch. 5

A Multilayer Perceptron (MLP) is a type of neural network consisting of an input layer, one or more hidden layers, and an output layer. Each layer is composed of interconnected neurons that apply linear transformations followed by nonlinear activation functions, making it capable of capturing nonlinear structures in the data. Let the input be

$$x \in \mathbb{R}^d.$$

The MLP maps inputs to outputs through a sequence of layers. The initial activation is simply the input:

$$a^{(0)} = x.$$

For each layer $j = 1, \ldots, L$, the pre-activation is computed as

$$z^{(j)} = W^{(j)} a^{(j-1)} + b^{(j)},$$

where $W^{(j)}$ and $b^{(j)}$ are the weights and biases of layer j. A nonlinear activation function $\sigma^{(j)}$ is then applied to obtain the output of the layer:

$$a^{(j)} = \sigma^{(j)}(z^{(j)}).$$

Nonlinear activations are essential because if all activations were linear, the MLP would reduce to a single linear transformation regardless of its depth. Common choices for σ include:

- Sigmoid:
$$\sigma(x) = \frac{1}{1 + e^{-x}}, \quad \sigma'(x) = \sigma(x)(1 - \sigma(x)).$$
- ReLU:
$$\mathrm{ReLU}(x) = \max(0, x), \quad \mathrm{ReLU}'(x) = \begin{cases} 1, & x > 0, \\ 0, & x \leq 0. \end{cases}$$
- Hyperbolic tangent:
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \tanh'(x) = 1 - \tanh^2(x).$$

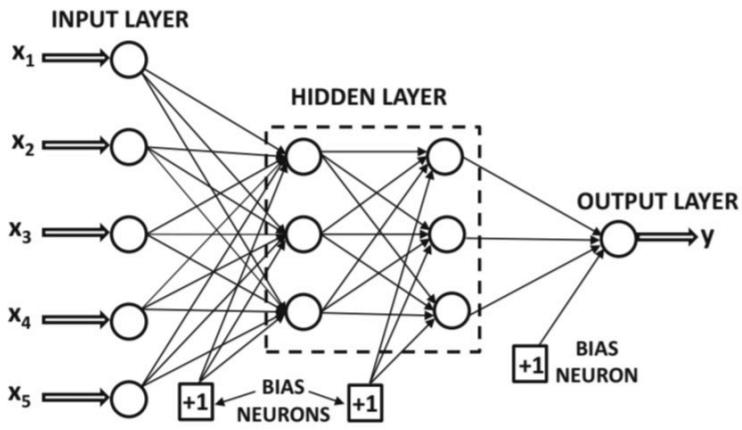


Figure 2.1: Architecture of the MLP model with two hidden layers

Training an MLP involves minimizing a task-dependent loss over a dataset $\{(x_i, y_i)\}_{i=1}^n$:

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i),$$

where ℓ is the loss function. For regression, a common choice is

$$\ell(\hat{y}, y) = \frac{1}{2} \|\hat{y} - y\|^2,$$

while for classification with k classes, the categorical cross-entropy is often used:

$$\ell(\hat{y}, y) = -\sum_{c=1}^{k} y_c \log \hat{y}_c.$$

The learnable parameters of the network are

$$\theta = \{W^{(j)}, b^{(j)}\}_{j=1}^{L},$$

and these weights and biases are updated using gradient descent:

$$W^{(j)} \leftarrow W^{(j)} - \eta \frac{\partial L}{\partial W^{(j)}}, \quad b^{(j)} \leftarrow b^{(j)} - \eta \frac{\partial L}{\partial b^{(j)}},$$

where the gradients are efficiently computed using backpropagation:

$$\delta^{(j)} = (W^{(j+1)})^\top \delta^{(j+1)} \odot \sigma'^{(j)}(z^{(j)}), \quad \frac{\partial L}{\partial W^{(j)}} = \delta^{(j)} (a^{(j-1)})^\top, \quad \frac{\partial L}{\partial b^{(j)}} = \delta^{(j)}.$$

During training, input data is first propagated through the network to compute activations and outputs. The loss is then evaluated, and the error is propagated backward through the layers to calculate the gradients with respect to weights and biases. These gradients are subsequently used to update the parameters. Variants of gradient descent, such as Stochastic Gradient Descent (SGD), Adam, or RMSProp, are commonly employed to improve convergence speed and stability.

To improve generalization and prevent overfitting, regularization techniques are often applied. L2 regularization adds a penalty term

$$\lambda \sum_j \|W^{(j)}\|_F^2$$

to the loss, dropout randomly zeroes activations during training, and batch normalization normalizes layer inputs to stabilize and accelerate training.

In essence, an MLP learns hierarchical representations of the data by alternating linear transformations and nonlinear activations. Training adjusts the weights and biases to minimize a task-specific loss through backpropagation and gradient-based optimization. This combination of depth, nonlinearity, and parameterization enables MLPs to model complex patterns.

2.2 Random Forest

Cf. [R2019] Ch. 6, [S2023] Ch. 3.3, 10, [Z2021] Ch. 8

Random Forests are an ensemble learning method used for both classification, which predicts categories, and regression, which predicts numerical values. They work by combining many decision trees to improve accuracy and reduce overfitting. A decision tree splits the input data into branches based on feature values and assigns a prediction at each leaf, which is the terminal node of the tree. Formally, a decision tree is a function:

$$f : \mathcal{X} \to \mathcal{Y},$$

where $\mathcal{X} \subset \mathbb{R}^p$ is the input space with p features, and $\mathcal{Y}$ is the output space, consisting of numeric values for regression or classes for classification.

The prediction at each leaf L_m depends on the type of task. For regression, the prediction is the average of the target values y_i of all training samples x_i in that leaf:

$$\hat{y}_m = \frac{1}{|L_m|} \sum_{x_i \in L_m} y_i,$$

where $|L_m|$ is the number of samples in leaf L_m. For classification, the prediction is the class $c \in \mathcal{C}$ that occurs most frequently in the leaf:

$$\hat{y}_m = \arg\max_{c \in \mathcal{C}} \frac{1}{|L_m|} \sum_{x_i \in L_m} \mathbf{1}\{y_i = c\},$$

where $\mathcal{C}$ is the set of all classes, and $\mathbf{1}\{\text{condition}\}$ is an indicator function equal to 1 if the condition is true and 0 otherwise.

Random Forests use bootstrap aggregation (bagging) to reduce variance. From a training dataset

$$\{(x_i, y_i)\}_{i=1}^n,$$

where n is the number of training samples, multiple bootstrap datasets D_b^* are created by sampling n instances with replacement. A decision tree f_b is trained on each bootstrap sample D_b^*, and the predictions from all trees are combined. In regression, the Random Forest prediction for an input x is the average of the predictions from all B trees:

$$\hat{f}_{\text{RF}}(x) = \frac{1}{B} \sum_{b=1}^{B} f_b(x),$$

and in classification, the prediction is the class that receives the majority vote:

$$\hat{C}_{\text{RF}}(x) = \text{mode}\{f_1(x), f_2(x), \ldots, f_B(x)\},$$

where $f_b(x)$ is the prediction of the b-th tree.

To further improve performance and reduce correlation between trees, Random Forests randomly select a subset of features at each split. For each candidate split, only a subset

$$\mathcal{F}_j \subset \{1, 2, \ldots, p\}$$

of features is considered, and the split that minimizes an impurity function is chosen. In regression, this impurity is usually the variance of the target values in the leaf:

$$I(L) = \frac{1}{|L|} \sum_{x_i \in L} (y_i - \bar{y}_L)^2,$$

where

$$\bar{y}_L = \frac{1}{|L|} \sum_{x_i \in L} y_i$$

is the mean target value in the leaf. In classification, the impurity is often measured using the Gini index:

$$I(L) = \sum_{c \in \mathcal{C}} \hat{p}_{L,c}(1 - \hat{p}_{L,c}),$$

where

$$\hat{p}_{L,c} = \frac{1}{|L|} \sum_{x_i \in L} \mathbf{1}\{y_i = c\}$$

is the proportion of samples in the leaf that belong to class c.

2.3 EXtreme Gradient Boosting (XGBoost)

Cf. [R2019] Ch. 6, [S2023] Ch. 3.3, 11

EXtreme Gradient Boosting (XGBoost) is an advanced ensemble learning method used for both regression and classification. It builds an additive model iteratively, adding a new function at each step to reduce a differentiable loss function.

Given a dataset

$$\{(x_i, y_i)\}_{i=1}^{n},$$

where $x_i \in \mathbb{R}^p$ is the feature vector of the i-th sample, y_i is the target value, n is the number of samples, and p is the number of features, the model predicts outputs as a sum of K functions (trees):

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F},$$

where $\mathcal{F}$ is the space of all possible regression trees. Each f_k is a decision tree, which maps an input x_i to a leaf index and returns the corresponding leaf weight as the prediction:

$$f_k(x_i) = w_{q_k(x_i)},$$

where q_k is the structure of the k-th tree mapping input x_i to a leaf index j, and w_j is the weight of leaf j.

The objective function in XGBoost combines a training loss term and a regularization term to prevent overfitting:

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k),$$

where l is a differentiable convex loss function (e.g., squared error for regression, logistic loss for classification), and the regularization term $\Omega(f)$ penalizes tree complexity:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2,$$

with T being the number of leaves in the tree, w_j the weight of leaf j, γ the complexity penalty per leaf, and λ the L_2 regularization on leaf weights.

XGBoost optimizes the objective using a second-order Taylor expansion. At step t, given predictions $\hat{y}_i^{(t-1)}$ from previous trees, the objective is approximated as:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t),$$

where

$$g_i = \frac{\partial l(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}}, \quad h_i = \frac{\partial^2 l(y_i, \hat{y}_i^{(t-1)})}{\partial (\hat{y}_i^{(t-1)})^2}.$$

Here, g_i and h_i are the first and second derivatives of the loss function with respect to the previous prediction.

The optimal weight w_j^* for a leaf j is computed by minimizing the approximate objective:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda},$$

where I_j is the set of instances assigned to leaf j. This weight determines the prediction for that leaf in the tree.

The tree structure is learned greedily, meaning the algorithm chooses splits that most improve the objective function at each node. By combining gradient-based optimization, regularization, and greedy tree learning, XGBoost achieves high predictive accuracy while controlling overfitting.

2.4 SHapley Additive ExPlanations (SHAP)

Cf. [KL2021] Ch. 5.3.4-5.3.6, [M2019] P. 177-188

SHapley Additive exPlanations (SHAP) is a model-agnostic framework for interpreting the output of machine learning models by attributing the contribution of each feature to a particular prediction. It is grounded in cooperative game theory, specifically the concept of Shapley values, which were originally introduced to fairly distribute the payout of a game among its players. In the context of machine learning, the "game" is the prediction task, the "players" are the features, and the "payout" is the difference between the predicted value and a baseline value, typically the expected value of the model output.

Formally, for a model $f : \mathbb{R}^p \to \mathbb{R}$, the prediction for an instance x can be expressed as a sum of contributions from each feature plus a baseline term:

$$f(x) = \phi_0 + \sum_{i=1}^{p} \phi_i,$$

where ϕ_0 is the expected value of the model output over the training data:

$$\phi_0 = \mathbb{E}[f(X)],$$

and ϕ_i represents the contribution of feature i to the prediction $f(x)$.

The Shapley value for feature i is computed as the weighted average of the change in the model's output when the feature is included in all possible subsets of features. Let $S \subseteq F \setminus \{i\}$ be a subset of all features F not containing i. Then, the Shapley value ϕ_i is defined as:

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!\,(p - |S| - 1)!}{p!} \Big[f_{S\cup\{i\}}(x_{S\cup\{i\}}) - f_S(x_S) \Big],$$

where $f_S(x_S)$ denotes the model output when only features in subset S are known, and the remaining features are marginalized. The combinatorial factor $\frac{|S|!\,(p-|S|-1)!}{p!}$ ensures that each ordering of features is weighted equally, providing a fair contribution according to cooperative game theory.

SHAP satisfies several criteria:

- Local Accuracy (Additivity): The sum of feature contributions equals the model output:

$$f(x) = \phi_0 + \sum_{i=1}^{p} \phi_i$$

- Missingness (Null Player): If a feature does not contribute to the model, its Shapley value is zero:

$$f_{S\cup\{i\}}(x_{S\cup\{i\}}) = f_S(x_S) \implies \phi_i = 0$$

- Consistency (Monotonicity): If a model changes so that a feature's contribution increases or stays the same for all subsets, the Shapley value does not decrease.

Computing exact Shapley values is computationally expensive, requiring evaluation over 2^p subsets of features. SHAP uses efficient approximations, including TreeSHAP for tree-based models and sampling-based methods for other models, which reduce computation while retaining theoretical guarantees.

SHAP explains a model's predictions by showing how much each feature contributes to the output. It can explain individual predictions as well as provide an overall view of which features are most important across the model.

2.5 Local Interpretable Model-agnostic Explanations (LIME)

Cf. [KL2021] Ch. 5.3.9, [M2019] P. 168-176

Local Interpretable Model-agnostic Explanations (LIME) is a framework for interpreting individual predictions of machine learning models by approximating the model locally with a simpler, interpretable model. Unlike global interpretability methods, LIME focuses on explaining a specific prediction by capturing the model's behavior in the neighborhood of the instance being explained.

Given a complex, possibly black-box model $f : \mathbb{R}^p \to \mathbb{R}$ and a specific instance $x \in \mathbb{R}^p$, LIME constructs an interpretable model $g \in G$ (e.g., a linear model or decision tree) that approximates f in the vicinity of x. The goal is to minimize the following objective:

$$\xi(x) = \arg\min_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g),$$

where:

- $\mathcal{L}(f, g, \pi_x)$ is a loss function measuring how well g approximates f in the local neighborhood of x,
- $\pi_x(z)$ is a proximity measure that assigns higher weight to points z closer to x,
- $\Omega(g)$ is a complexity penalty ensuring g remains interpretable.

To generate the local neighborhood, LIME perturbs the input x to create a set of synthetic samples $\{z_j\}$, and evaluates the black-box model on these perturbed instances to obtain $\{f(z_j)\}$. The local model g is then trained on the weighted dataset $(z_j, f(z_j), \pi_x(z_j))$. Typically, for tabular data, a linear model is used, so that the explanation is given by the feature weights:

$$g(z) = w_0 + \sum_{i=1}^{p} w_i z_i,$$

where w_i indicates the contribution of feature i to the prediction of the instance x.

LIME satisfies several characteristics:

- Local fidelity: The interpretable model g accurately approximates f in the vicinity of x, providing an accurate explanation of the specific prediction.
- Model-agnostic: LIME can be applied to any predictive model, including tree-based models, neural networks, and ensembles, since it treats the model as a black box.
- Sparsity and interpretability: By including a complexity penalty $\Omega(g)$, LIME ensures that explanations remain human-interpretable, often using only a subset of features in the local model.

By focusing on local approximations rather than global behavior, LIME provides intuitive and actionable explanations for individual predictions, enabling better trust and understanding of complex machine learning models.

2.6 Permutation Feature Importance (PFI)

Cf. [KL2021] Ch. 5.3.2, [M2019] P. 154-162

Permutation Feature Importance (PFI) is a model-agnostic method to measure the importance of features for a trained machine learning model. The idea is to quantify how much a model's predictive performance decreases when the values of a specific feature are randomly permuted, breaking the relationship between that feature and the target variable. A large decrease in performance indicates that the feature is important, whereas little or no change implies that the feature has low importance.

Formally, let $f : \mathbb{R}^p \to \mathbb{R}$ be a trained model, and $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ be a test dataset. The baseline performance of the model is evaluated using a scoring function $s(f, \mathcal{D})$, such as accuracy for classification or mean squared error for regression:

$$s_{\text{base}} = s(f, \mathcal{D}).$$

For a feature j, we create a permuted dataset $\mathcal{D}^{(j)}$ by randomly shuffling the values of feature j across all samples, while keeping other features unchanged. The model is then evaluated on this permuted dataset:

$$s_{\text{perm},j} = s(f, \mathcal{D}^{(j)}).$$

The permutation importance of feature j is defined as the difference between the baseline performance and the permuted performance:

$$\text{PFI}_j = s_{\text{base}} - s_{\text{perm},j}.$$

A positive PFI_j indicates that permuting feature j reduces the model performance, implying the feature is important. Conversely, a PFI_j close to zero suggests that the feature has little effect on model predictions.

Permutation Feature Importance has several advantages:

- Model-agnostic: It can be applied to any predictive model, including black-box models such as neural networks or ensembles.
- Relative importance: Features are ranked according to their impact on model performance, making it easy to compare their importance.
- Global interpretability: By computing permutation importance over the entire dataset, PFI provides a measure of the overall relevance of each feature.

However, PFI has some limitations. It assumes features are independent, so correlated features may lead to underestimated importance. It also requires repeated model evaluation, which can be computationally expensive for large datasets.

3 Dataset and Data Preprocessing

Cf. [B2023] Ch. 1.6.3-1.6.4, [KL2021] Ch. 2, [S2023] Ch. 3.3.1, 3.6, 3.7, [S2024] Ch 2.2.5, 2.2.6, 3.2.3, 3.2.4, 5.2, [SW2021] Ch. 1.2

The dataset used in this project is a hotel booking cancellation dataset containing 42,100 reservation records, with each entry describing customer booking details, hotel stay characteristics, spending information, and past booking behavior. The target variable is `booking_status`, which indicates whether a reservation was canceled (1) or not canceled (0). A summary of the original features is shown in Table 1.

Table 1: Original features of the booking cancellation dataset

Feature Name	Description	Type
id	Unique booking identifier	Numerical
no_of_adults	Number of adults in the booking	Numerical
no_of_children	Number of children in the booking	Numerical
no_of_weekend_nights	Nights stayed during weekends	Numerical
no_of_week_nights	Nights stayed during weekdays	Numerical
type_of_meal_plan	Type of meal selected (encoded)	Categorical
required_car_parking_space	Parking required or not (0/1)	Binary
room_type_reserved	Reserved room type (encoded)	Categorical
lead_time	Days between booking and arrival	Numerical
arrival_year	Year of arrival	Numerical
arrival_month	Month of arrival	Numerical
arrival_date	Date of arrival	Numerical
market_segment_type	Booking channel / market segment (encoded)	Categorical
repeated_guest	Whether the customer is a returning guest (0/1)	Binary
no_of_previous_cancellations	Number of past cancellations	Numerical
no_of_previous_bookings_not_canceled	Number of past successful bookings	Numerical
avg_price_per_room	Average room price	Numerical
no_of_special_requests	Number of special requests	Numerical
booking_status	Target variable: 1=canceled, 0=not canceled	Binary

An initial review showed that the dataset contained no missing values or duplicate entries. The attribute **id**, which serves solely as a unique identifier, was removed. The fields `arrival_year`, `arrival_month`, and `arrival_date` represent components of a calendar date and were converted and validated using a custom timestamp check. This resulted in the detection and removal of 50 invalid dates, ensuring reliable temporal information for model learning.

To enrich data representation and to capture behavioral, temporal, and financial patterns, several new features were engineered based on domain relevance and booking behavior. These are listed in Table 2.

Table 2: Engineered features and their descriptions

Engineered Feature	Description	Purpose
total_guests	Sum of adults and children	Represents group size
total_nights	Total nights stayed (weekday + weekend)	Measures booking duration
avg_guests_per_night	total_guests / total_nights	Group size by stay length
children_ratio	Proportion of children among guests	Captures family or adult trips
lead_time_bucket	Categorized lead time into 5 ranges	Indicates planning behavior
stay_length_category	Categorized total nights into duration types	Defines short/long travel patterns
price_per_guest	avg_price_per_room / total_guests	Spending per person
price_per_night	avg_price_per_room / total_nights	Spending per day
has_special_request	Binary indicator for special requests (0/1)	Indicates whether the customer made special requests

To avoid multicollinearity, a correlation heatmap of numeric attributes was generated, which revealed strong redundancy between original guest and night count attributes and their engineered counterparts. Accordingly, the features `no_of_adults, no_of_children, no_of_week_nights, and no_of_weekend_nights` were removed because their information was fully represented in the engineered variables.

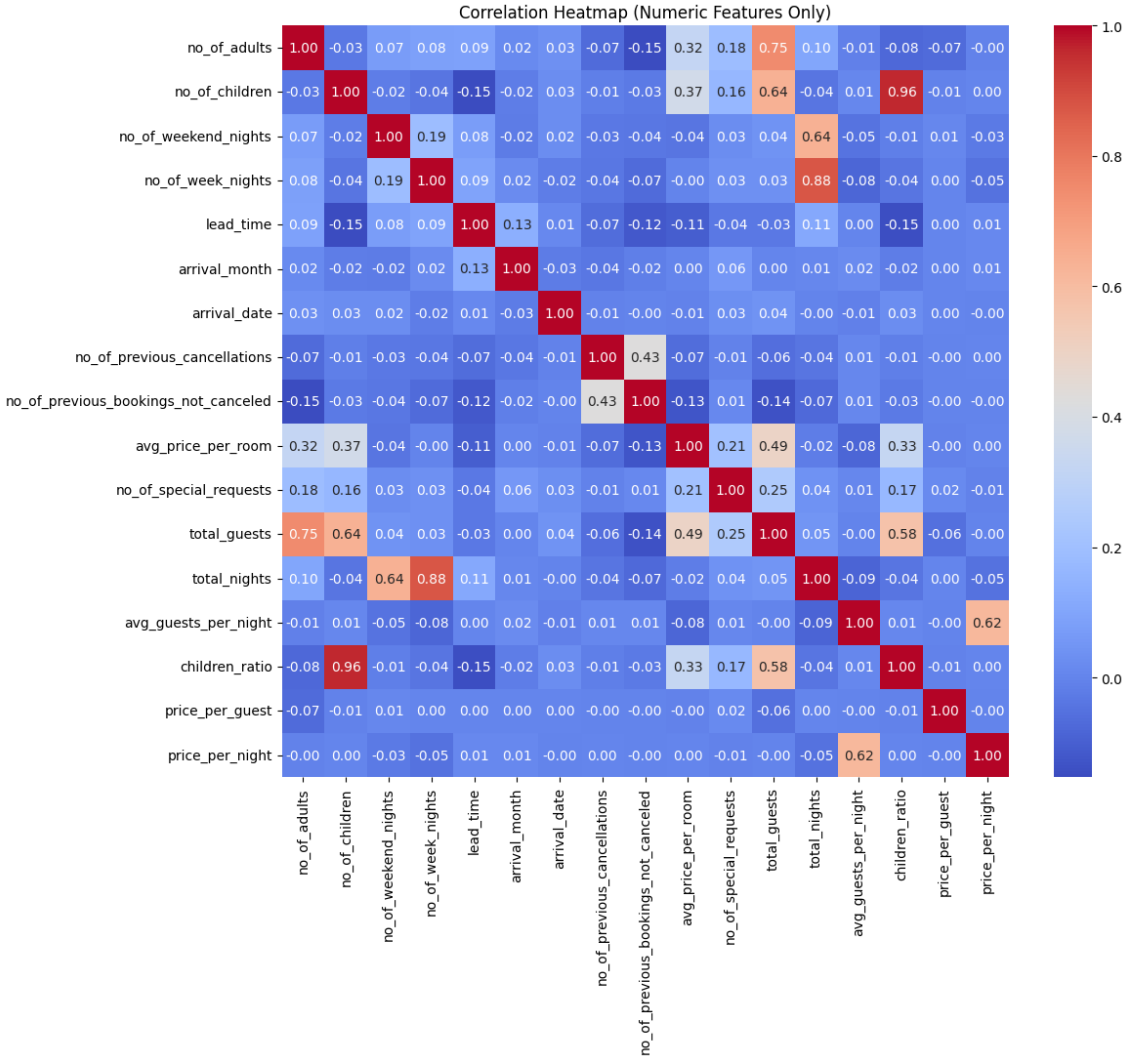


Figure 3.1: Correlation heatmap of numeric features after feature engineering

The dataset was then split using an 80:20 stratified train-test split, ensuring that class proportions remained consistent. Numeric variables were standardized using Z-score normalization via `StandardScaler`, while binary and one-hot encoded variables were left unchanged. Z-score normalization, also known as standardization, transforms a variable to have a mean of 0 and a standard deviation of 1, ensuring that all numeric features contribute equally to the model. The Z-score of a value x is calculated as:

$$z = \frac{x - \mu}{\sigma}$$

where μ is the mean and σ is the standard deviation of the variable. In Python, `StandardScaler` from `scikit-learn` performs this transformation by automatically computing these statistics and scaling the values accordingly.

Since the target distribution was imbalanced, the training set was resampled using **SMOTENC**, a method designed for datasets with both numeric and categorical attributes, to generate balanced synthetic minority samples. This produced a fully balanced training set, enabling reliable model training and interpretability in booking cancellation prediction. Subsequently, all categorical features, including engineered categories such as `lead_time_bucket` and `stay_length_category`, were encoded using one-hot encoding.

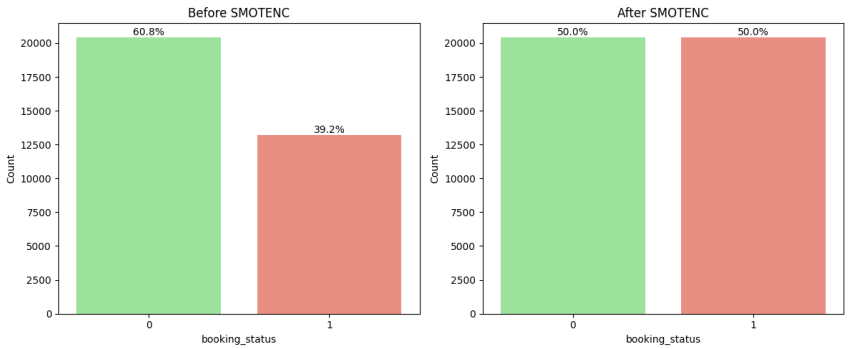


Figure 3.2: Distribution of the target variable (`booking_status`) in the training set before and after applying SMOTENC

4 Model Evaluation

The predictive model was evaluated on the preprocessed and balanced dataset using standard classification metrics. Two main evaluation tools used in this project were the confusion matrix and the classification report.

The confusion matrix provides a detailed breakdown of the model's predictions against the actual labels. It shows the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). This allows identification of patterns in misclassification and highlights whether the model is biased toward a particular class.

From the confusion matrix, several evaluation metrics are calculated:

- **Precision:** the proportion of correctly predicted positive cases among all predicted positives.

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall (Sensitivity):** the proportion of correctly predicted positive cases among all actual positives.

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1-score:** a single score that balances precision and recall.

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

The classification report summarizes these metrics for each class, allowing a comprehensive evaluation of model performance beyond overall accuracy. Combining the confusion matrix and classification report enables assessment of both the model's general predictive ability and its effectiveness in correctly identifying cancellations, which is particularly critical in datasets with imbalanced classes.

5 Model Implementation

5.1 Multilayer Perceptron (MLP)

A Multilayer Perceptron (MLP) was applied for binary booking cancellation prediction. The network consists of an input layer corresponding to the number of input features, followed by one or two fully connected hidden layers. Two hidden layer configurations were considered: a single hidden layer with 64 neurons and a two-layer architecture with 64 and 32 neurons. Nonlinear

activation functions were applied in the hidden layers, with ReLU and hyperbolic tangent (tanh) evaluated during model selection.

The output layer contains a single neuron with a sigmoid activation function, producing a probability that represents the likelihood of a booking cancellation. Binary predictions were obtained using a threshold of 0.5. The model was trained using the Adam optimizer with learning rates of 0.001, 0.01, and 0.1, and the binary cross-entropy loss function. Training was performed for up to 100 epochs using a batch size of 32. Early stopping based on the training loss was applied to reduce overfitting.

Hyperparameter tuning was conducted using Randomized Search with five-fold cross-validation, optimizing classification accuracy. The final model was evaluated on an independent test set that was not used during training or hyperparameter tuning, using standard classification metrics and a confusion matrix. The best-performing hyperparameter configuration was saved and reused for model training.

For the complete feature set, the optimal hyperparameters were a learning rate of 0.01, two hidden layers with 64 and 32 neurons and ReLU activation. Early stopping occurred at epoch 8, and the model weights were restored from the best epoch, which was epoch 3. Evaluation on the independent test set resulted in an accuracy of 0.775, with a precision of 0.75 and recall of 0.64 for class 1, corresponding to canceled bookings.

To interpret the predictions of the trained multilayer perceptron (MLP) model, explainable AI (XAI) techniques were applied using SHAP and LIME. SHAP values were computed for 1,000 test instances, and both summary and bar plots were generated to visualize the overall contribution and relative importance of features across model predictions. The SHAP analysis identifies `no_of_special_requests` as the most influential feature, where higher values mainly decrease the model output. The second most important feature is `lead_time`, with higher values increasing the predicted probability of the target class (cancellation) and shorter lead times associated with a lower probability. The third relevant feature is `avg_price_per_room`, where higher prices increase the predicted probability of cancellation, while lower prices are associated with a reduced likelihood of the target class.

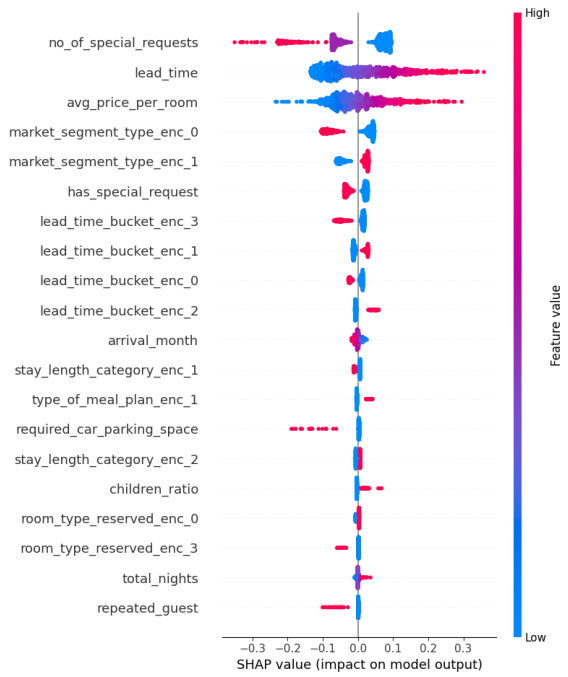


Figure 5.1: SHAP summary plot for the Multilayer Perceptron

In addition, LIME was employed to provide instance-level explanations for the first five test instances. For the first test instance, the model predicts class 1 with a probability of 0.64. This prediction is primarily influenced by `lead_time`, where a longer booking lead time increases the predicted probability of cancellation. Other contributing factors include `required_car_parking_space`, which increases the likelihood of cancellation when no parking space is requested, while a lower `avg_price_per_room` contributes toward the non-cancellation class.

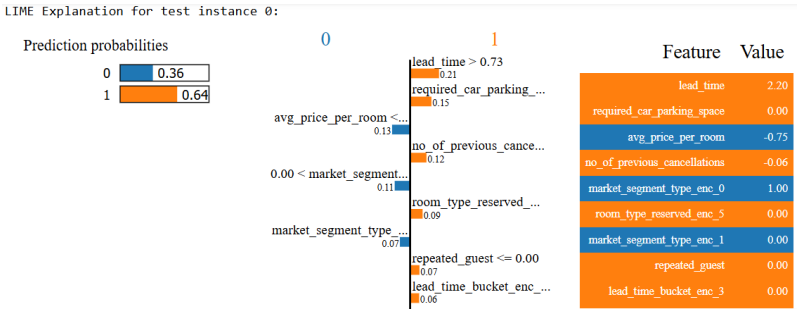


Figure 5.2: LIME explanation for the first test instance of the Multilayer Perceptron

To further simplify the model and focus on the most influential features, the MLP was retrained using only the ten most important features identified by SHAP. A new hyperparameter search was performed using the same search space, and the selected best hyperparameters were a learning rate of 0.01, two hidden layers with 64 and 32 neurons, and ReLU activation. The retrained model achieved a test accuracy of 0.800, with a precision of 0.74 and a recall of 0.75 for class 1, demonstrating that a reduced feature set slightly improved predictive performance while simplifying the model.

After retraining, feature importance was visualized using permutation feature importance, illustrating the relative contribution of the top ten features to the model's predictive performance. As shown in the bar plot, `lead_time` remains the most influential feature, resulting in the largest decrease in model accuracy when permuted. This is followed by `no_of_special_requests` and `lead_time_bucket_enc_3`, indicating that both booking lead time characteristics and customer request behavior play a significant role in the model's predictions.

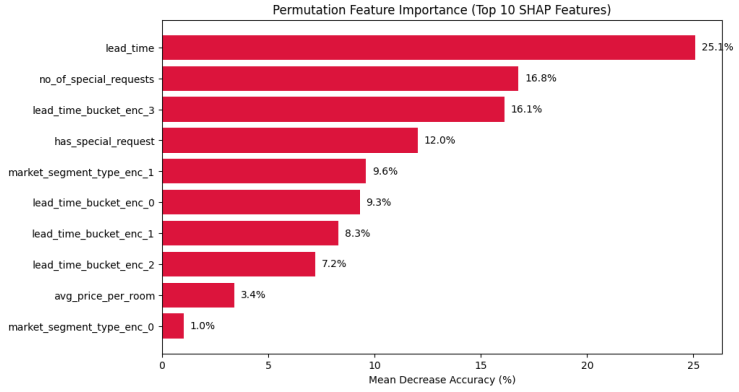


Figure 5.3: Permutation feature importance of the top 10 features based on SHAP values for the Multilayer Perceptron

5.2 Random Forest

A Random Forest classifier was applied for predicting hotel booking cancellations. The model consists of an ensemble of decision trees. Hyperparameter tuning was conducted using Grid Search with five-fold cross-validation. The search space included maximum tree depths of 20, 30, 40, and

50, minimum samples required to split an internal node set to 5, 10, or 15, and minimum samples required at a leaf node set to 1, 2, or 4. The best-performing configuration for the full feature set had a maximum depth of 50, minimum samples per leaf of 1, and minimum samples to split an internal node of 10.

Evaluation of the Random Forest model on the independent test set resulted in an accuracy of 0.817. For class 1, representing canceled bookings, the precision was 0.76 and the recall was 0.77, indicating reliable identification of cancellations.

SHAP values were computed for 1,000 test instances to analyze the contribution of individual features to the Random Forest model predictions. Summary and bar plots were used to illustrate the relative importance of features across predictions. The SHAP analysis identifies `lead_time` as the most influential feature, with higher values increasing the predicted outcome, followed by `no_of_special_requests`, where higher values mainly decrease the model output, and `market_segment_type_enc_1`, which contributes positively to the prediction.

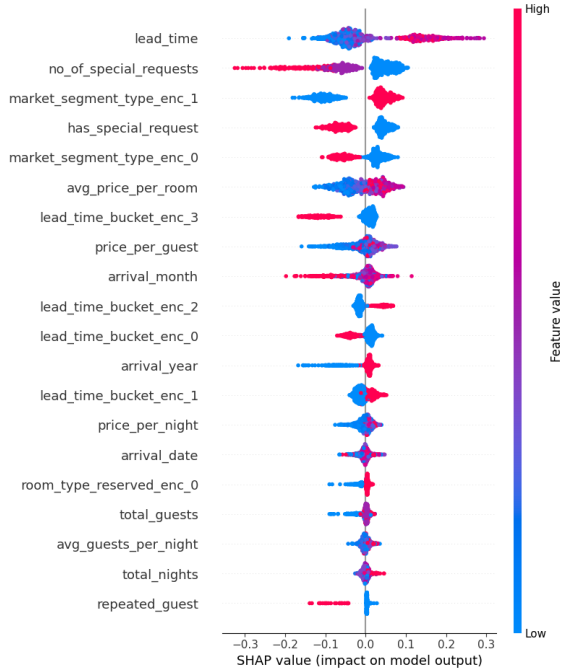


Figure 5.4: SHAP summary plot for the Random Forest

LIME was applied to provide instance-level explanations for the first five test instances. For the first test instance, the model predicts class 0 with a probability of 0.93, mainly influenced by `market_segment_type_enc_1`, followed by `required_car_parking_space` and `has_special_request`.

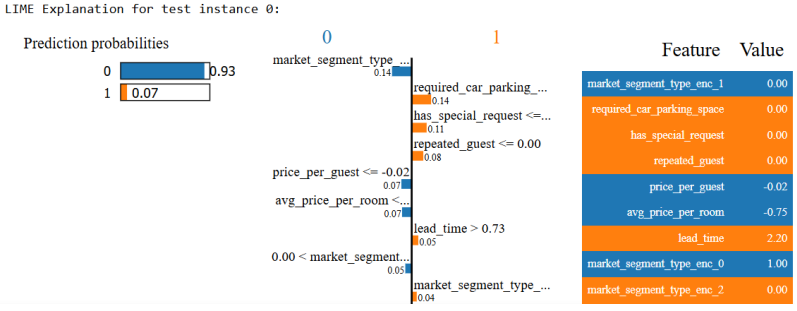


Figure 5.5: LIME explanation for the first test instance of the Random Forest

To simplify the model, a reduced dataset containing only the ten most important features identified by SHAP was used to retrain the Random Forest. A new Grid Search was performed with the same hyperparameter search space. The best configuration for the reduced feature set had a maximum depth of 20, minimum samples per leaf of 1, and minimum samples to split an internal node of 15. The retrained model achieved a test accuracy of 0.816. For class 1, the precision was 0.75 and the recall was 0.79, demonstrating that using only the top ten features maintained strong predictive performance.

Permutation feature importance was computed for the top ten features, and a bar plot was generated to highlight their contribution to the model's predictions. After retraining the Random Forest model using only these ten SHAP-selected features, permutation importance again ranked `lead_time` as the most influential variable, followed by `no_of_special_requests` and `has_special_request`, confirming strong consistency with the SHAP-based feature ranking.

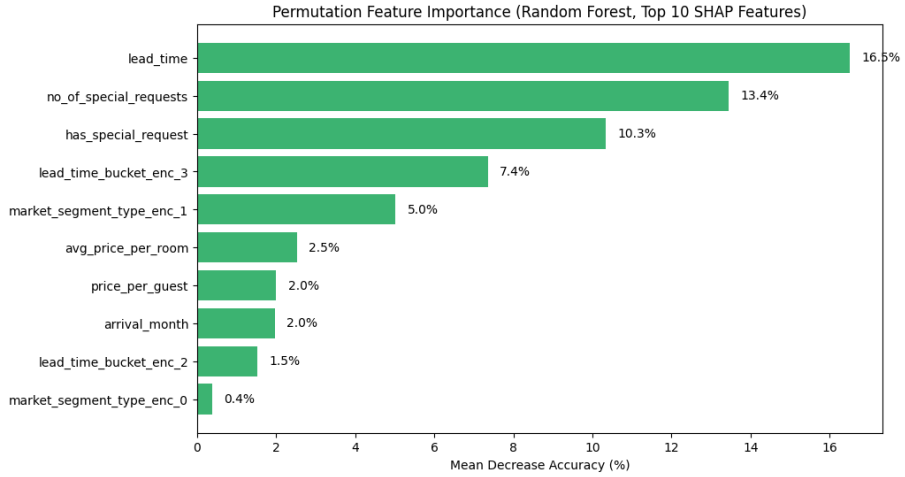


Figure 5.6: Permutation feature importance of the top 10 features based on SHAP values for the Random Forest

5.3 EXtreme Gradient Boosting (XGBoost)

An XGBoost classifier was employed to predict binary hotel booking cancellations. This model uses gradient boosting, where each tree is added sequentially to correct errors made by prior trees. Hyperparameter optimization was conducted using Grid Search with five-fold cross-validation. The search included the number of trees (100, 200, 300), maximum tree depth (3, 5, 10, 20), and learning rates (0.1, 0.3). For the complete feature set, the optimal configuration consisted of 200 trees, a maximum depth of 10, and a learning rate of 0.1.

The model's performance on the independent test set achieved an accuracy of 0.822. For class 1, representing canceled bookings, the precision was 0.77 and the recall was 0.78, showing that the model effectively identified cancellations.

SHAP values were calculated for 1,000 test instances to evaluate the contribution of each feature to the XGBoost model predictions. The analysis shows that `lead_time` is the most important predictor, with higher values increasing the predicted probability of cancellation and lower values reducing it, indicating that bookings made further in advance are more likely to be canceled. The second most influential feature is `market_segment_type_enc_1`, where higher values increase the predicted probability. The third feature is `no_of_special_requests`, which has an inverse effect, with higher numbers of requests reducing the predicted probability of cancellation, while few or no requests increase it.

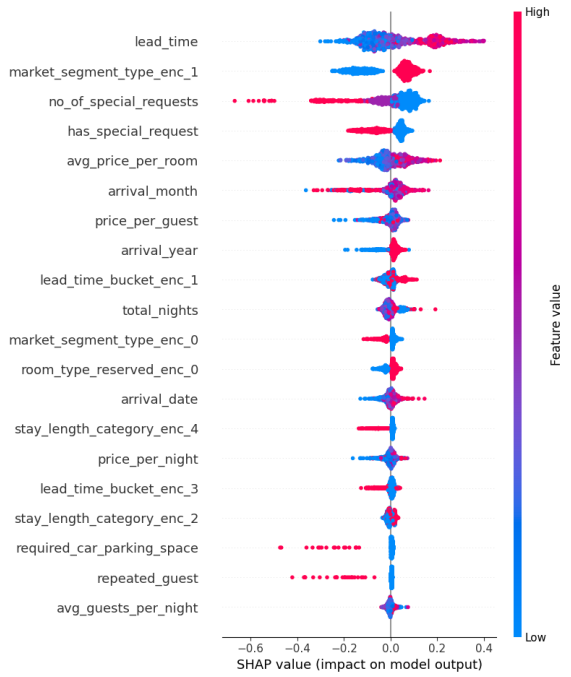


Figure 5.7: SHAP summary plot for the XGBoost

LIME was used to explain individual predictions for the first five test instances. For the first instance, the model predicts Class 0 (non-cancellation) with probability 0.94. The most influential local features are `repeated_guest`, `required_car_parking_space`, and `market_segment_type_enc_3`, which push the prediction toward cancellation. However, their combined effect is outweighed by other features, resulting in a final prediction of non-cancellation.

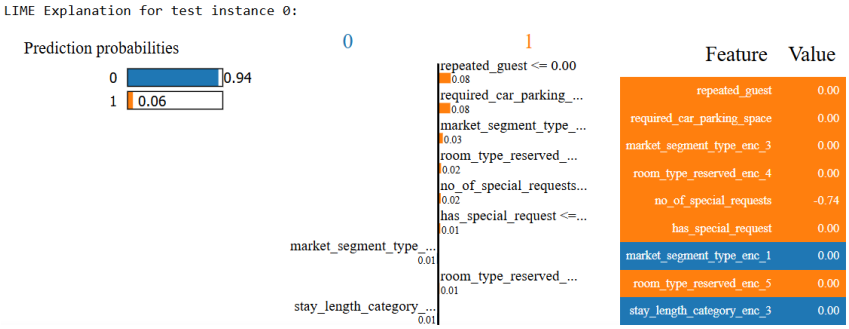


Figure 5.8: LIME explanation for the first test instance of the XGBoost

To simplify the model, training was repeated using only the ten most influential features identified by SHAP. A new Grid Search using the same hyperparameter space identified the best configuration for this reduced feature set, which included 200 trees, a maximum depth of 5, and a learning rate of 0.3. This retrained model reached a test accuracy of 0.824, with a precision of 0.77 and a recall of 0.78 for class 1, indicating that a smaller set of key features maintained strong predictive capability.

Permutation feature importance was calculated for the ten selected features, and a bar plot highlighted their relative impact on the model's predictions. After retraining the XGBoost model using only these top ten SHAP features, the permutation importance results show both consistency and reordering among the most influential predictors. The three most important features after retraining are `lead_time`, `no_of_special_requests`, and `lead_time_bucket_enc_1`.

`lead_time` remains the most important feature, consistent with the original SHAP results, confirming its dominant influence on the model's predictions. `no_of_special_requests` also remains among the top contributors, indicating its continued key role in model performance. In contrast, `lead_time_bucket_enc_1`, which ranked lower in the original SHAP analysis, becomes one of the top three features after retraining, suggesting that this representation of lead time captures additional predictive information in the reduced model.

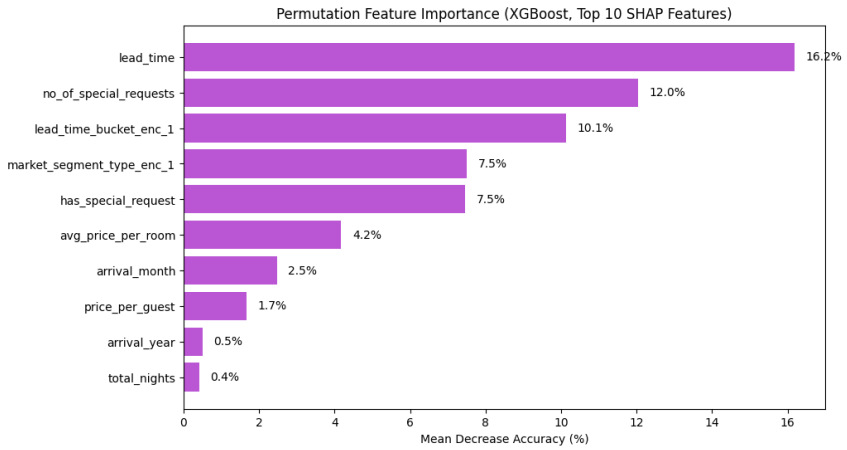


Figure 5.9: Permutation feature importance of the top 10 features based on SHAP values for the XGBoost

6 Conclusion and Future Directions

This study investigated the prediction of hotel booking cancellations using three machine learning models: Multilayer Perceptron (MLP), Random Forest, and XGBoost. Each model was trained on a preprocessed dataset, including engineered features that capture temporal, behavioral, and financial patterns of bookings. Hyperparameter optimization was performed for all models, and independent test sets were used to evaluate predictive performance.

Among the models, XGBoost achieved the highest test accuracy, followed closely by Random Forest and MLP. Reducing the feature set to the ten most important features, as identified by SHAP, slightly improved or maintained predictive performance while simplifying the models. For the reduced feature sets, precision and recall for canceled bookings (class 1) remained high across all models, demonstrating that a small number of key features captures most of the predictive signal.

Explainable AI techniques, including SHAP, LIME, and permutation feature importance, provided insights into feature contributions and the factors influencing individual predictions. These methods consistently identified `lead_time` and `no_of_special_requests` as the most important features across all models, highlighting their strong influence on predicted booking cancellations and supporting interpretable, actionable insights for hotel management.

Overall, combining machine learning with explainable AI allows accurate prediction of cancellations while providing interpretable information to support operational decisions. The results highlight that effective prediction can be achieved using both complete and reduced feature sets, enabling more efficient and transparent decision-making in the hospitality industry.

Future research could extend this work by incorporating additional data sources, such as customer demographics, seasonal patterns, pricing dynamics, and external factors including macroeconomic indicators or local events, to further improve predictive performance. Moreover, exploring advanced deep learning architectures or ensemble stacking approaches may enhance model robustness and generalization.

The implementation of real-time prediction systems within hotel management platforms could enable dynamic pricing strategies and proactive customer engagement. Finally, evaluating model fairness, bias mitigation strategies, and transferability across different hotel segments or geographic regions would strengthen the practical applicability and ethical reliability of cancellation prediction systems.

7 Appendix

The complete code developed for this seminar project is publicly available in the GitHub repository `https://github.com/HTW-WM-FAR/DeepLearningWS2025`. The specific submission is located in the folder `Seminar_DeepLearning_WiSe25_Evan_Leonard_582554`.

8 Academic integrity declaration

I hereby confirm that I have authored this report independently and without use of others than the indicated sources. All passages which are literally or in general matter taken out of publications or other sources are marked as such.

All verbatim or in-sentence copies and quotations, as well as all sections that were designed, written and/or edited with the help of AI-based tools, have been identified and verified.

Information on the use of AI-based tools: In the appendix of my work, I have listed all AI-based resources with product names and prompts used. I assure that I have not used any AI-based tools whose use has been explicitly excluded in writing by the examiner. I understand that AI- generated content does not guarantee correctness. I remain fully responsible for the content of this work.

Berlin, 20.02.2026

Evan Leonard

9 References

[A2023] Aggarwal, C. C. (2023) Neural Networks and Deep Learning, Springer. eISBN 978-3-031-29642-0

[B2023] Botsch, B. (2023). Maschinelles Lernen - Grundlagen und Anwendungen, Springer. eISBN 978-3-662-67277-8

[DS2013] Du, K., Swamy, M. N. S. (2013). Neural Networks and Statitical Learning, Springer. eISBN 978-1-4471-7452-3

[KL2021] Kamath, U., Liu, J. (2021). Explainable Artificial Intelligence: An Introduction to Interpretable Machine Learning, Springer. eISBN 978-3-030-83356-5

[M2019] Molnar, C. (2019). Interpretable Machine Learning, Leanpub. ISBN 978-0-244-76852-2

[R2019] Richter, S. (2019) Statistisches und Maschinelles Lernen, SpringerSpektrum. eISBN 978-3-662-59354-7

[S2023] Schonlau, M. (2023). Applied Statistical Learning, Springer. eISBN 978-3-031-33390-3

[S2024] Selle, S. (2024). Data Science Training - Supervised Learning, Springer. eISBN 978-3-662-67960-9

[S2025] Sun, J. (2025). Hotel Booking Cancellation and Machine Learning, Research Gate

[SW2021] Shardt, Y. A. W., Weiß, H. (2021). Methoden der Statistik und Prozessanalyse, Springer. eISBN 978-3-662-61626-0

[Z2021] Zhou, Z. (2021) Machine Learning, Springer. eISBN 978-981-15-1967-3