Predicting Hotel Cancellations

DATA 1030 Final Project Report - Fall 2024
Sociology Ph.D. Student, Valerie Zixi Li
Public GitHub Repo: https://github.com/ZixiLi76/Hotel-Cancellation

Introduction

Research Question: What factors influence hotel cancellations, and how can these factors be used to predict whether a booking will be canceled?

Motivation: Hotel cancellations pose significant challenges for businesses in revenue forecasting and resource planning. Understanding the key factors driving cancellations, such as customer behavior and characteristics, special requests, booking channels, and more, can help optimize operations, improve customer satisfaction, and enhance revenue management strategies.

Dataset & Collection Methodology: I am using the *Hotel Booking Demand* dataset from Kaggle [1]. This dataset contains booking information for city and resort hotels, including details like booking dates, length of stay, number of guests, and parking space availability. The data, originally published by Nuno Antonio et al. and cleaned by Thomas Mock and Antoine Bichat for #TidyTuesday, provides anonymized insights into booking patterns. It is ideal for exploratory analysis and predictive modeling for us to understand the dynamics of hotel cancellations.

Target Variable (y): is_canceled (1 if canceled, 0 if not canceled)

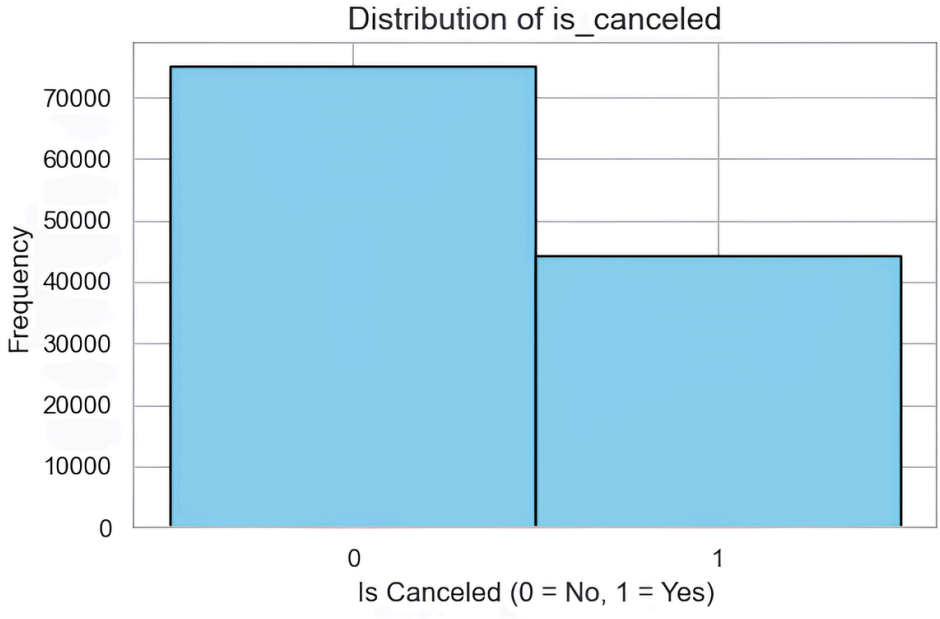

Figure 1: Balance of Target Variable (y): is_canceled

Previous Work: Previous tryouts, often using this same dataset on Kaggle, have explored various ML algorithms to predict hotel cancellations, often achieving high accuracy [2]. However, these approaches frequently lack detailed preprocessing or thorough hyperparameter tuning, and they tend to combine multiple workflows in one setting, making them harder to follow. My approach is different by focusing on careful preprocessing and thoughtful optimization for clarity and effectiveness.

EDA

I first loaded the dataset, explored its structure, and addressed missing values. Non-predictive columns (company ID, agent ID) and rows with null children values (only 4 missing) or no guests were removed. "Undefined" in the meal column was replaced with "SC," and missing country values were set for handling in preprocessing with the one-hot encoder.

Figure 2 shows distribution of hotels by two hotel types (city hotel vs. resorts).

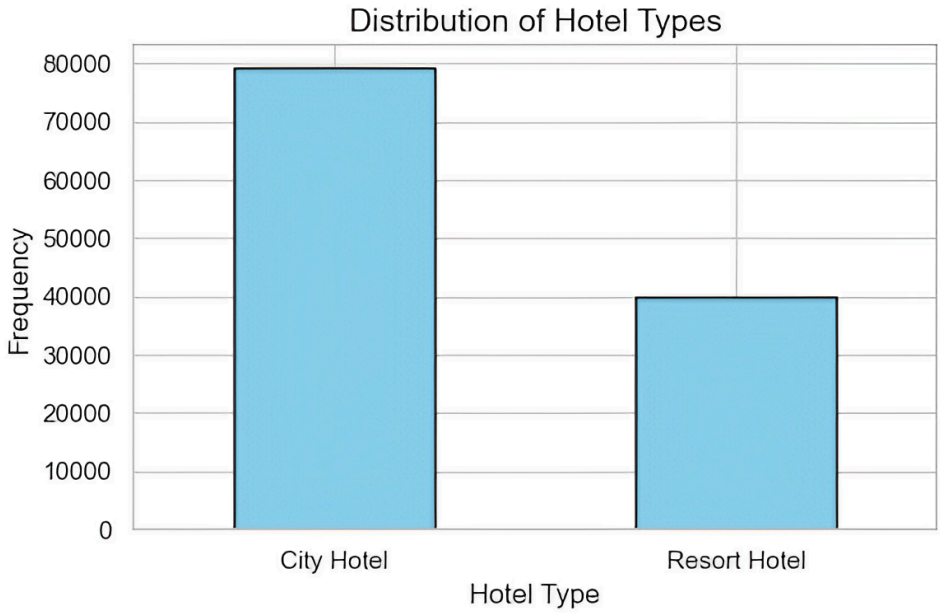

Figure 2: Hotels by Hotel Type (City or Resort)

Figure 3 explores cancellation patterns by hotel type and month. While no clear patterns emerge, City hotels show higher cancellation rates during winter months, and resorts show lower cancellations in summer months.

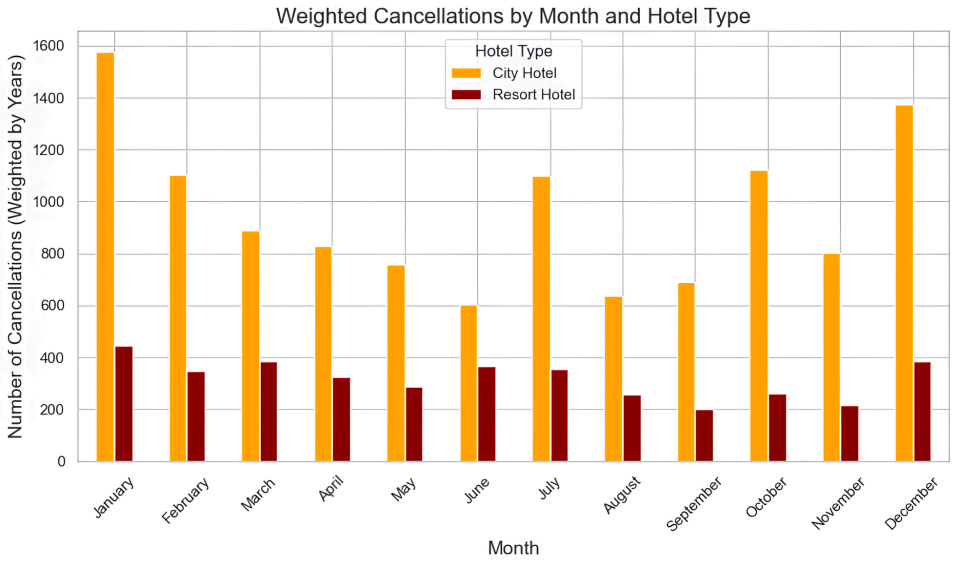

Figure 3: Cancellations by Month and Hotel Type, Weighted by Year Frequency

Figure 4 shows that being a repeated guest would normally cancel less for both types of hotels.

Figure 4: Cancellations by Repeated Guest Status and Hotel Type

Figure 5 shows that guests who have made more special requests along with the booking will largely cancel less for both types of hotels.

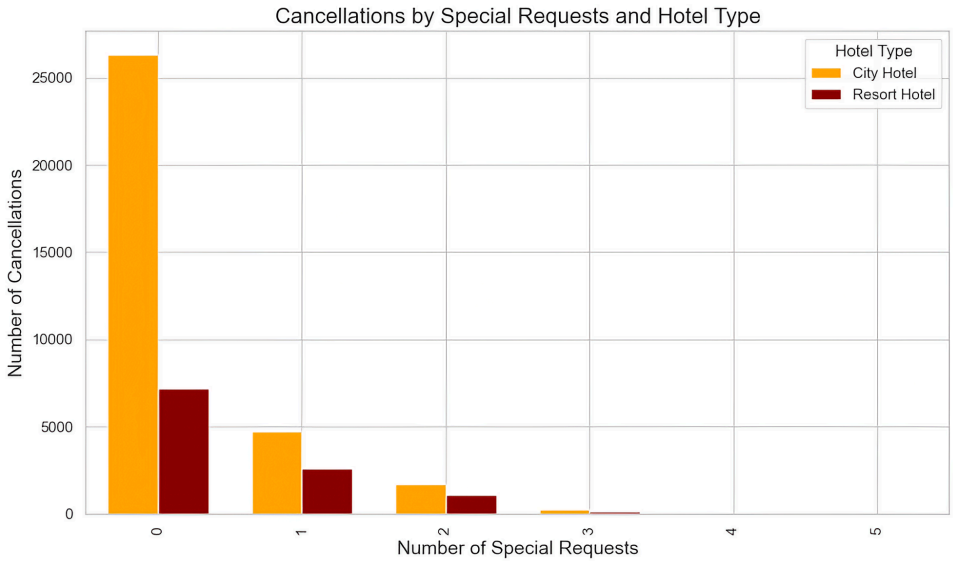

Figure 5: Cancellations by Special Requests and Hotel Type

Figure 6 tells us that both city and resorts tend to have higher average daily prices for canceled bookings compared to non-canceled ones. I took care of outliers in daily prices by following the IQR rule (Q3-Q1).

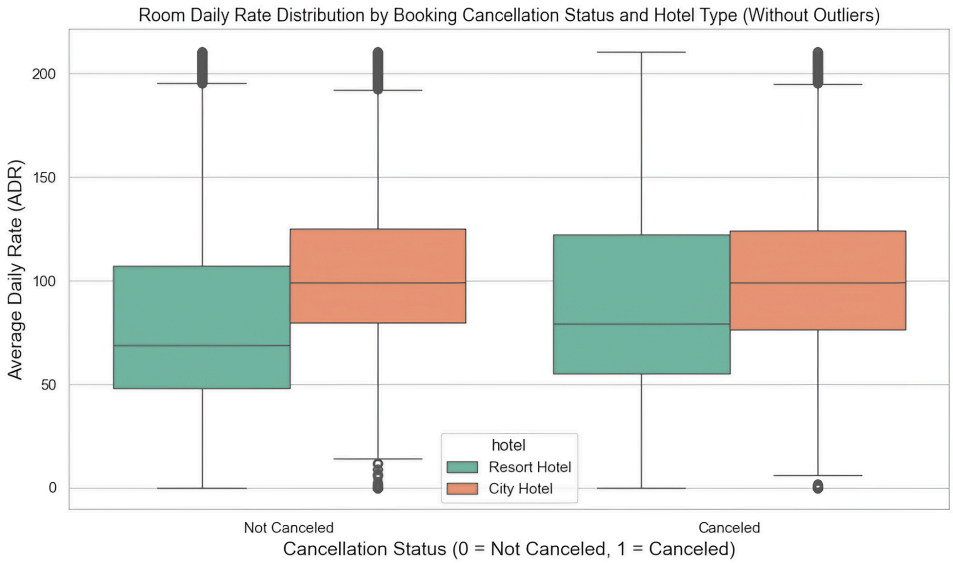

Figure 6: Average Daily Price by Cancellation Status and Hotel Type

Now, we can get to know more descriptives: Where do guests come from? (Figure 7) From which channel did they make the bookings? (Figure 8) Which months are more popular for city hotels and resorts? (Figure 9)

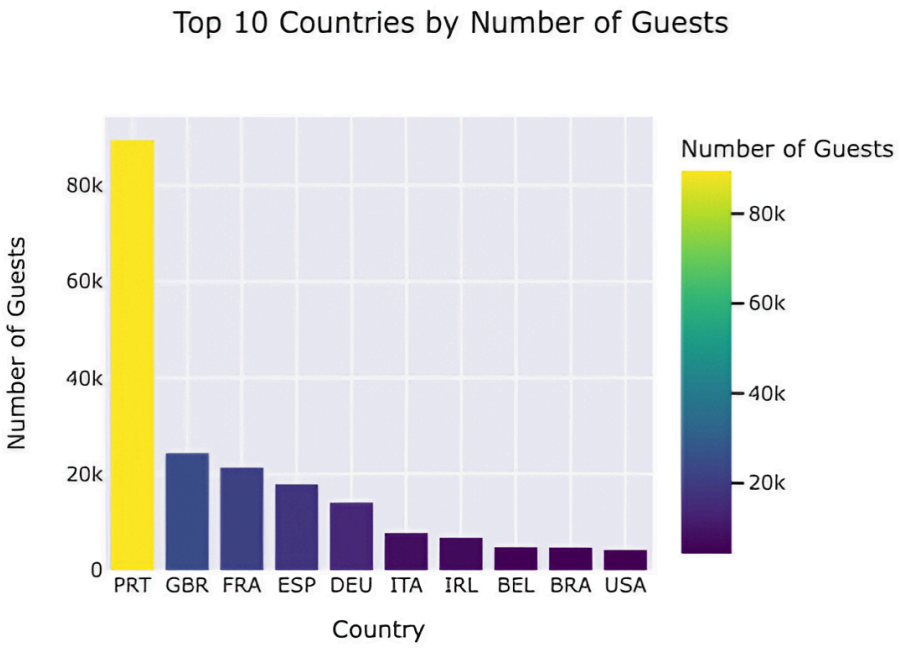

Figure 7: Likely a Portuguese Hotel

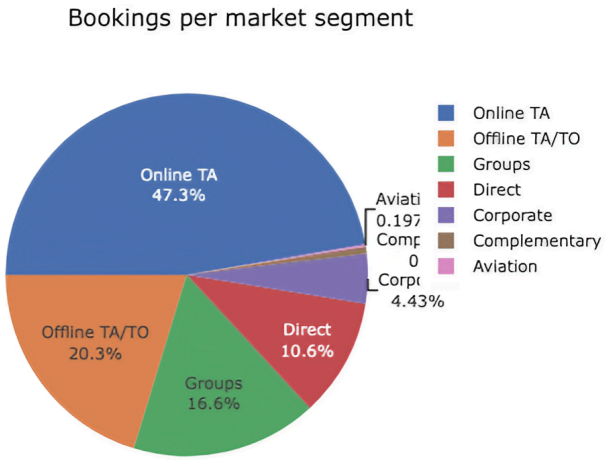

Figure 8: Most were Online Bookings

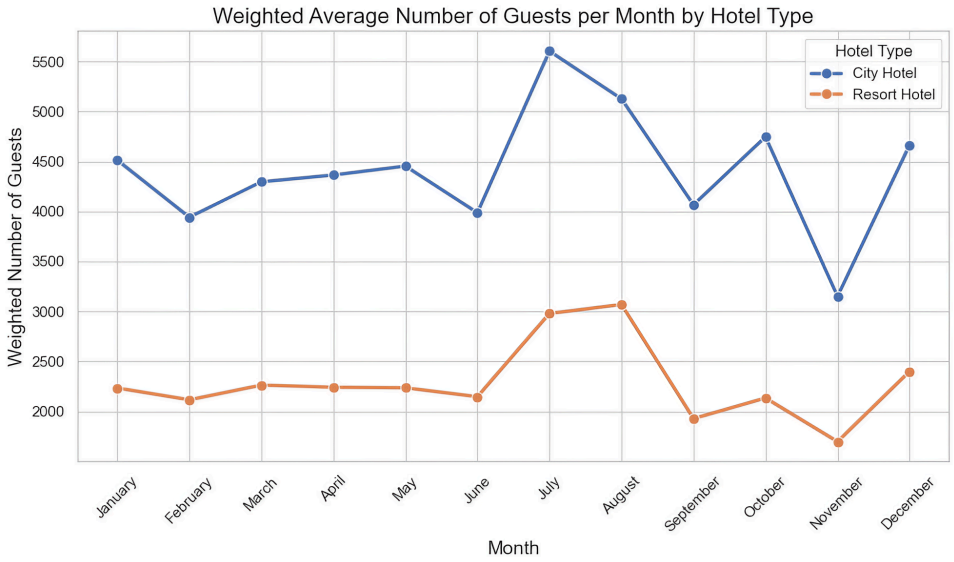

Figure 9: Popular Summer Months

More EDA graphs are produced in my codes.

Methods

Splitting Strategy: The dataset is split into three subsets: 60% training, 20% validation, and 20% testing. The training set is used to train the model, the validation set helps fine-tune hyperparameters and prevent overfitting, and the test set provides an unbiased evaluation of the final model. I set a consistent random_state to ensure the reproducibility of the splits.

Features & Preprocessors: The preprocessing handled 16 numerical, 11 categorical, and 1 ordinal feature. Numerical features were standardized, categorical features were one-hot encoded after imputing missing values with "Unknown," and the ordinal feature (arrival_date_month) was encoded using a defined month order. Feature engineering introduced total_guests (sum of adults, children, and babies), total_nights (sum of weekend and weeknight stays), and differs_in_reserved_assigned (discrepancy between reserved and assigned room types).

ML Algorithms (Overall): I evaluated model performance primarily using **accuracy** since correct predictions, false positives, and false negatives carry similar costs for hotel cancellations, and our target variable (*y* = is_canceled) is largely balanced (see Figure 1). I tested four ML algorithms: Logistic Regression, K-Neighbors Classifier, Random Forest, and XGBoost. While I also attempted SVM, it proved computationally expensive and failed to complete even after running overnight.

Model	**Parameters & Values**
LogisticRegression	Penalty: a grid search L1/L2, Regularization Strength (C): [0.1, 1, 10], Solver: liblinear (for

	faster convergence)
KNeighborsClassifier	n_neighbors: [3, 5, 10], Weights: uniform/distance, Metric: minkowski [3]
RandomForest Classifier	Depth: [5, 10, 50, None], Min Split: [2, 5, 10, 50], Min Leaf: [1, 2, 4, 10]
XGBoost	Max Depth: [3, 5, 10, 20], n_estimators: [50, 100, 500, 1000], Subsample: [0.5, 0.8, 1.0], Colsample By Tree: [0.5, 0.8, 1.0], Metric: ["logloss"], Objective: ["binary:logistic"].

Table 1: Parameters Tuned & Values Tried for 4 Different ML Algorithms

ML Algorithms (Details):
LogisticRegression:
I identified the overall best hyperparameters as {C: 1, penalty: l1, solver: liblinear}. To address convergence warnings, I increased *max_iter* to provide sufficient iterations for the solver to converge, and I set *n_jobs=1* to resolve warnings since the liblinear solver does not support parallel processing. The training (X_train) and test (X_test) datasets were preprocessed using a consistent pipeline to ensure proper scaling, encoding, and transformation of features. After preprocessing, the training data was split further into a *sub-train* and *validation* set, reserving 20% for validation while maintaining class balance through stratification. The validation set was used for hyperparameter tuning to identify the best-performing configurations.

KNeighborsClassifier:
The overall best hyperparameters identified were: {metric: minkowski, n_neighbors: 10, weights: distance}

RandomForestClassifier:
For the RandomForestClassifier, I tuned key hyperparameters using an expanded range to explore broader possibilities: max_depth ([5, 10, 50, None]), min_samples_split ([2, 5, 10, 50]), and min_samples_leaf ([1, 2, 4, 10]). To align with guidelines, I set n_estimators=100 for all models and removed its tuning. The best hyperparameters identified based on validation accuracy were {max_depth: 50, min_samples_leaf: 1, min_samples_split: 2}. Given Random Forest's non-deterministic nature, I accounted for uncertainties by running the model across multiple random splits, capturing variability from both data splitting and the model's stochastic elements.

XGBoost:
For the XGBoost model, I tuned key hyperparameters again using an expanded range to explore broader possibilities: max_depth ([3, 5, 10, 20]), n_estimators ([50, 100, 500, 1000]), subsample ([0.5, 0.8, 1.0]), and colsample_bytree ([0.5, 0.8, 1.0]). To simplify the grid search, I fixed the learning_rate at a reasonable default (0.1) instead of tuning it. I added logic to extract and highlight the best hyperparameters based on validation accuracy, which were identified as

{colsample_bytree: 0.5, eval_metric: logloss, max_depth: 10, n_estimators: 50, objective: binary:logistic, subsample: 1.0}. Early stopping with a patience of 50 rounds was maintained to optimize training efficiency and avoid overfitting by halting training when validation performance plateaued. Additionally, I split the training data into sub-train and validation sets during hyperparameter tuning to ensure robust evaluation and generalization.

Each model algorithm was run using 5 different random state values during data splitting to measure uncertainties in evaluation metrics like accuracy and F1-score due to data variability. Essentially, I am systematically evaluating each ML model by calculating key performance metrics, accuracy, precision, recall, and F1 score, on the test set for different random states. For each state, I log the best hyperparameters, validation metrics, and test results into a structured DataFrame. To measure the robustness of the models, I compute the averages and standard deviations of these metrics across random states. Finally, I print the results and save them to CSV files.

Results

Model	**Mean Test Accuracy**	**Standard Deviation**	**Accuracy-Baseline/ Std**
Log. Reg.	0.826786	0.001808	(0.826786-0.6293)/0.001808 = 109.228982301
KNeighbors	0.894166	0.001678	(0.894166-0.6293)/0.001678 = 157.84624553
RandomForest	0.916530	0.001025	(0.916530-0.6293)/0.001025 = 280.224390244
XGBoost	0.952452	0.001390	(0.952452-0.6293)/0.001390 = 232.483453237

Table 2: ML Models Accuracy & Uncertainties Comparison

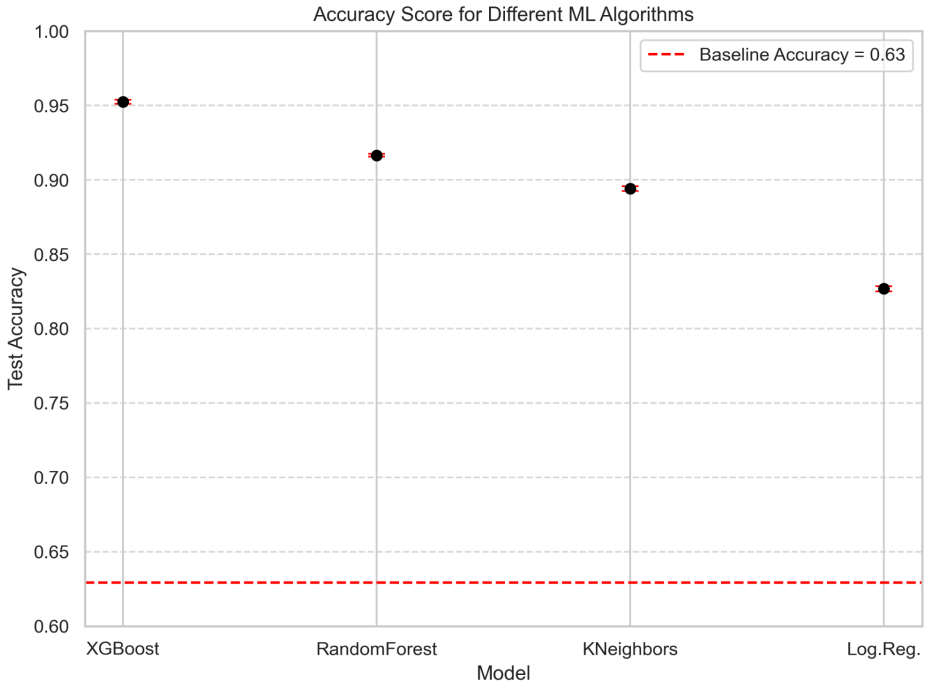

Figure 10: Model Performance Test Accuracy Scores & Uncertainties

From looking at Table 2 and Figure 10, we can clearly see that XGBoost is the best-performing model with a mean test accuracy reaching above 95% and a very small mean standard deviation. The baseline accuracy of 0.6293 is calculated by determining the proportion of the majority class in the target variable *y* = is_canceled.

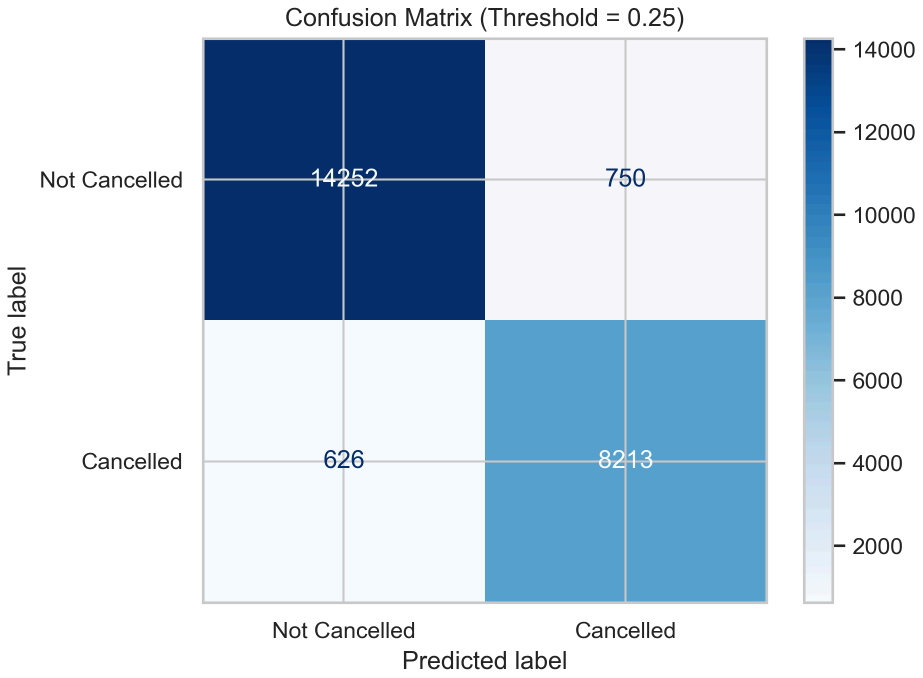

Figure 11: Confusion matrix using the best performing XGBoost model (threshold = 0.25)

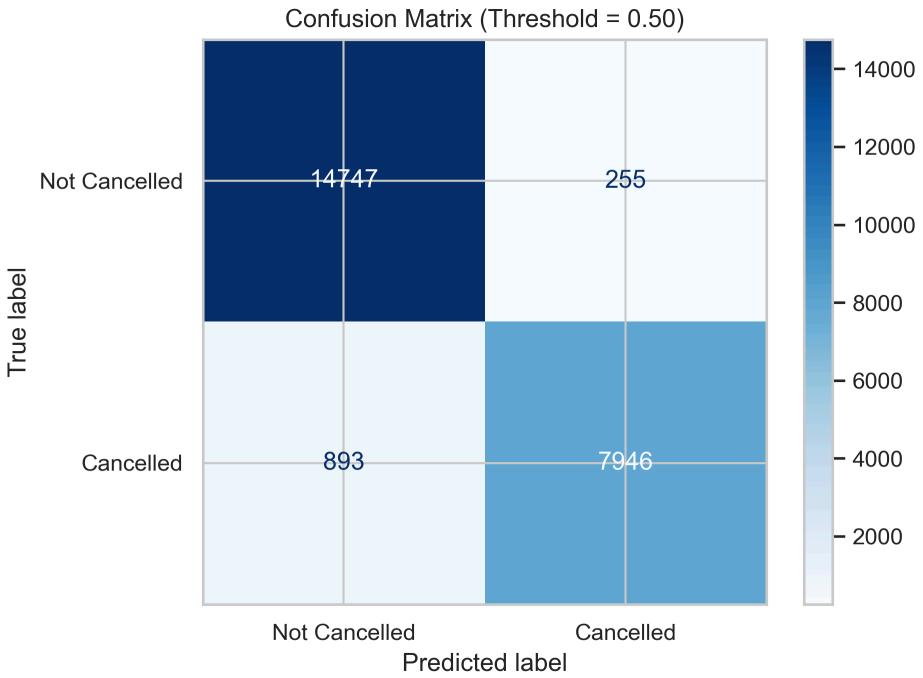

Figure 12: Confusion matrix using the best performing XGBoost model (threshold = 0.50)

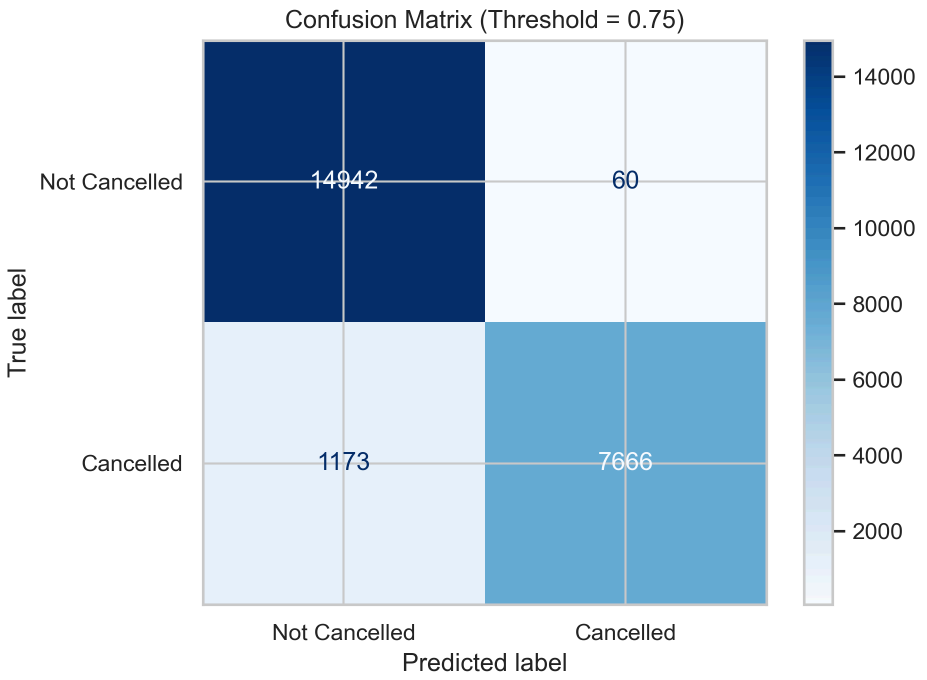

Figure 13: Confusion matrix using the best performing XGBoost model (threshold = 0.75)

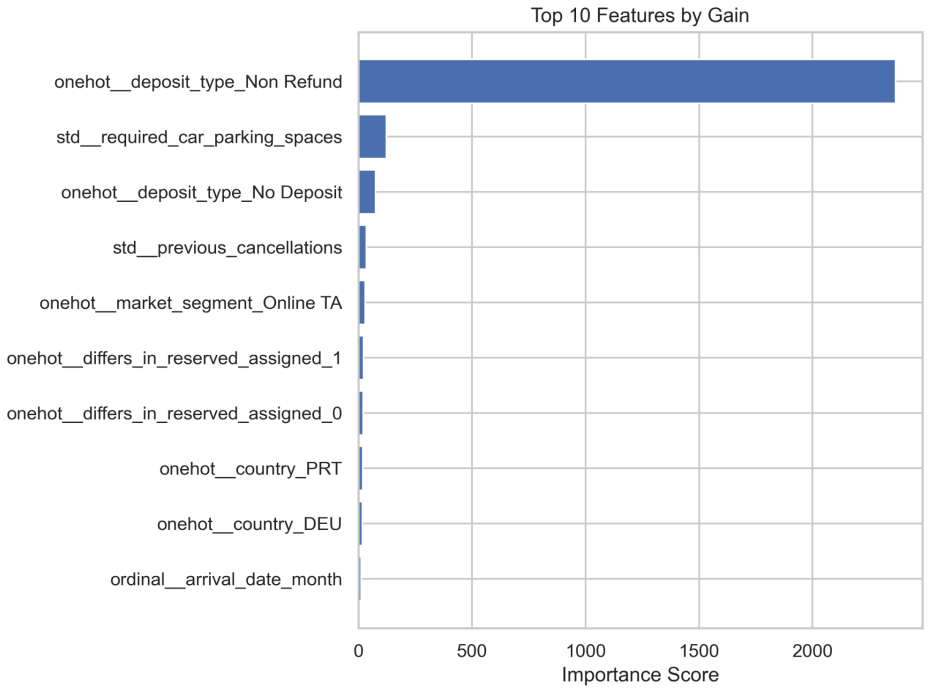

Figure 14: Top 10 Features by Gain - Global Feature Importance

Why does 'Non-Refund' have such a high gain? It indicates whether the customer made a non-refundable deposit to guarantee the booking. A value of 1 (non-refundable) strongly

correlates with cancellations, as such bookings are less likely to be canceled due to financial commitment. This strong predictive power leads to its high gain.

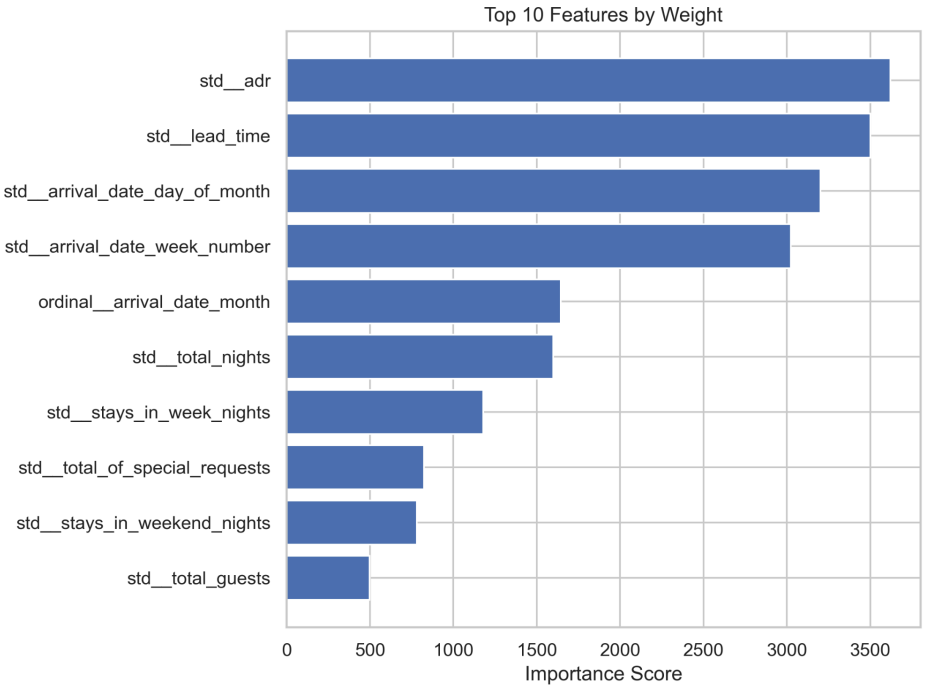

Figure 15: Top 10 Features by Weight - Global Feature Importance

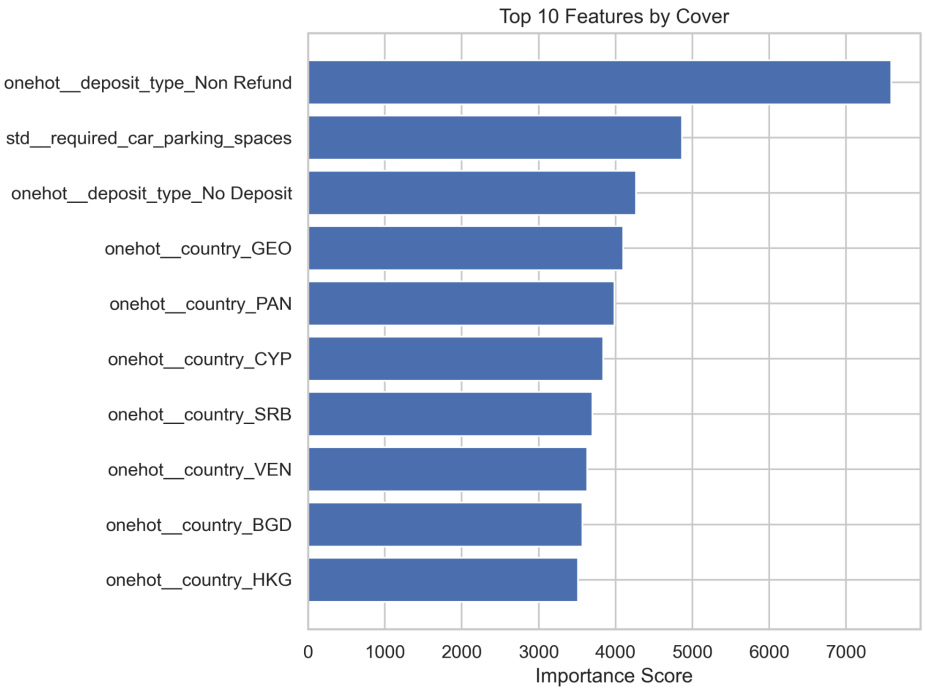

Figure 16: Top 10 Features by Cover - Global Feature Importance

Lead time (the time between booking and stay/checking in) and *average daily rate (ADR)* also stand out, indicating that bookings made far in advance or with higher rates are stronger predictors of cancellations. Interestingly, *the number of required car parking spaces* appears significant, suggesting a link between customer needs (like traveling with vehicles) and unique cancellation behaviors, potentially tied to group or family travel. Geographic features, such as the country of origin (e.g., “GEO”, “PAN”), play a notable role in the model’s coverage, hinting at regional patterns in cancellation behavior that could be influenced by economic, logistical, or cultural factors. Overall, these results align with intuitive expectations.

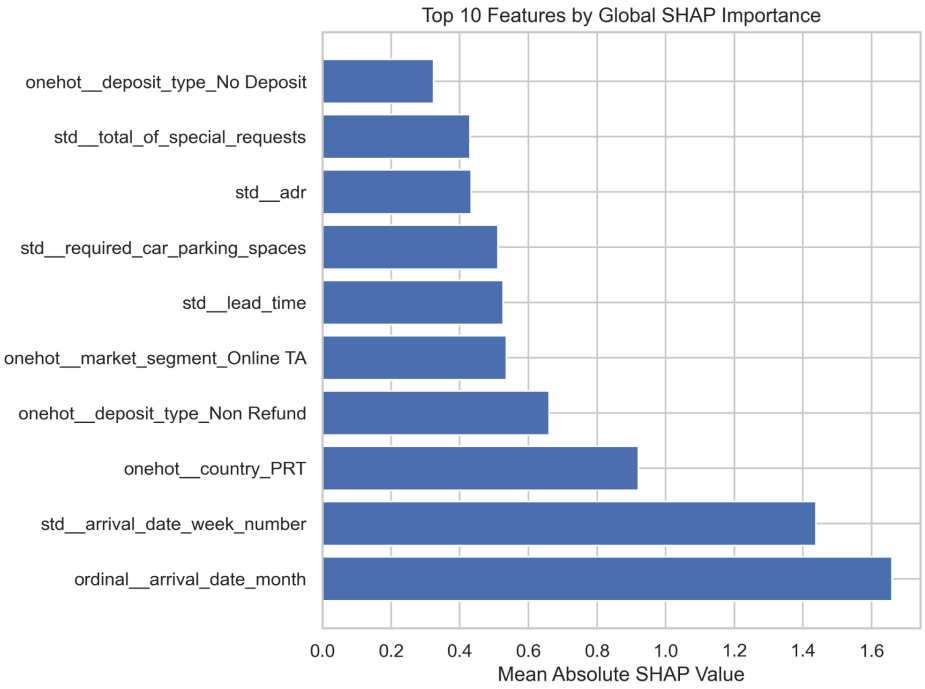

Figure 17 a

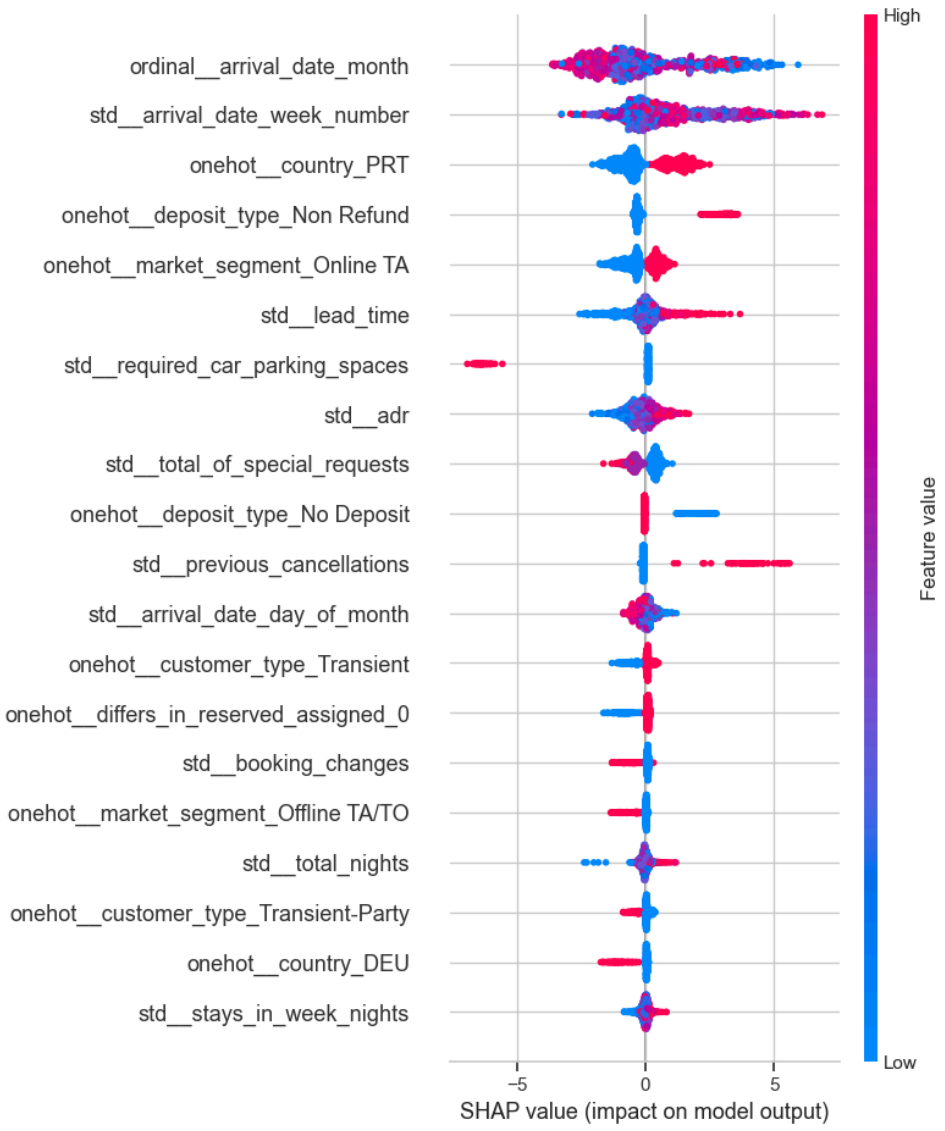

Figure 17 b

Figure 17a, b: Top 10 Features by Global SHAP Importance

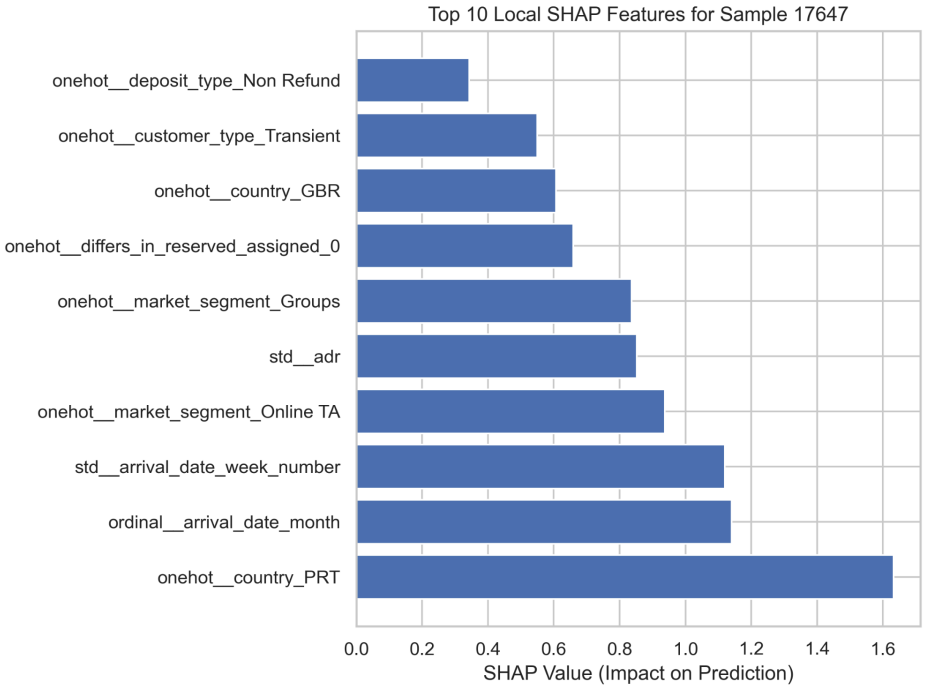

Figure 18 a

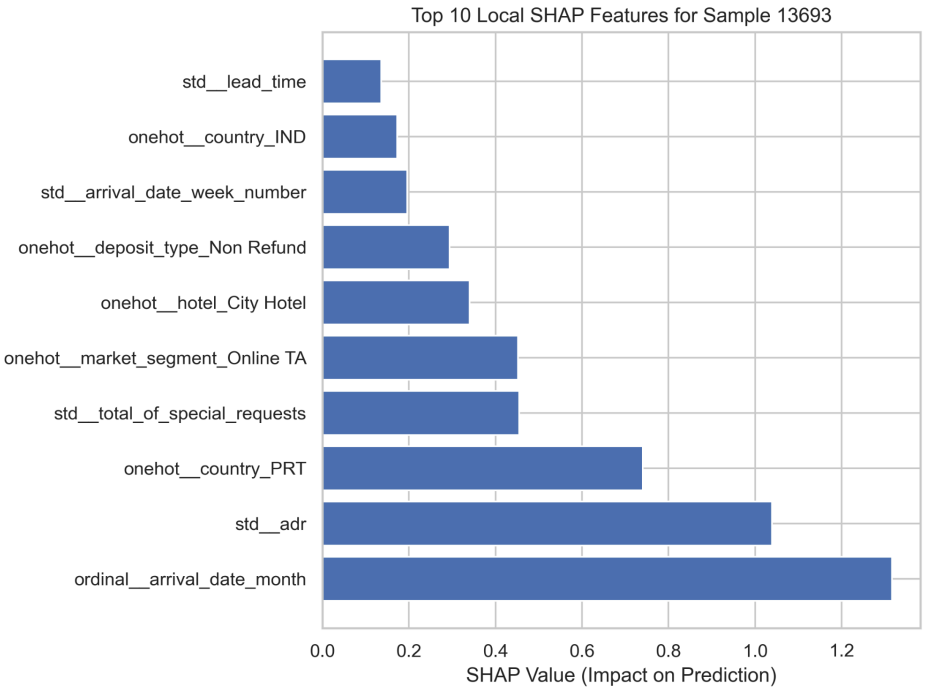

Figure 18 b

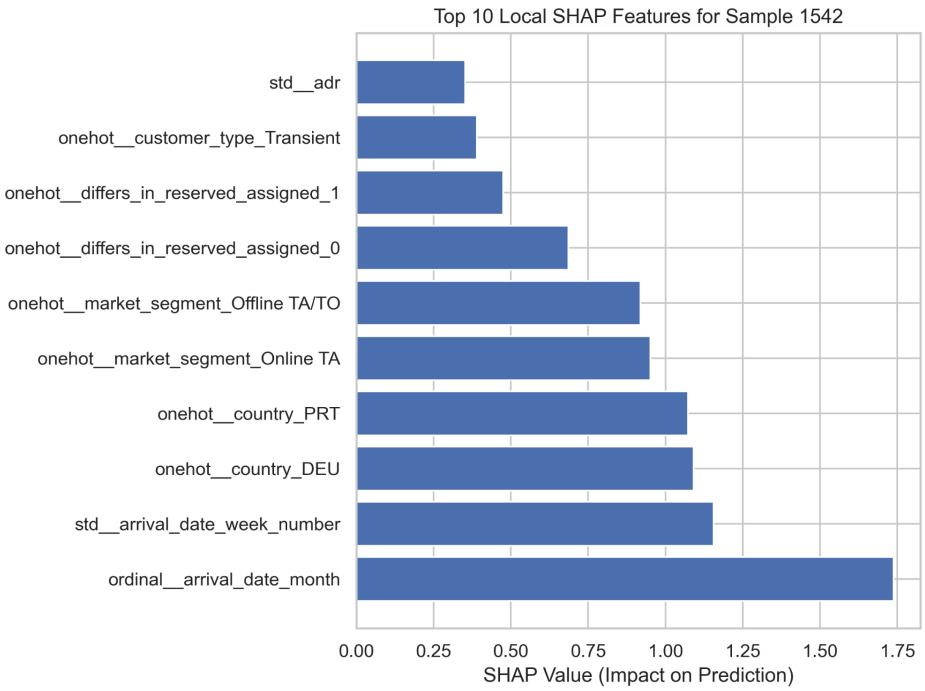

Figure 18 c

Figure 18 a,b,c: Top 10 Local SHAP Features for 3 Unique Smaples

Based on the SHAP analysis, the most important features identified globally and locally were arrival_date_month (ordinal encoded), adr (average daily rate), lead_time (time between booking and stay/checking in), deposit_type_Non Refund, and total_of_special_requests. Notably, Non Refund deposit types consistently ranked high, indicating that bookings requiring non-refundable deposits strongly influence cancellation predictions. Locally, the top SHAP features for individual samples varied slightly but often aligned with the global patterns.

To ensure comprehensive SHAP interpretations, I selected three unique rows for local analysis using a custom function. The function randomly chose one row, stratified by class to include one row from each cancellation class (y_test = 0 and y_test = 1), and added a row where the model made an incorrect prediction to highlight areas where it struggles. By ensuring unique selections through a set, this method captured diverse scenarios: random, class-stratified, and misclassified, to provide valuable insights into how feature importance varies across predictions.

Outlook

While the current modeling approach achieves strong predictive performance, there is still room for improvement in efficiency, interpretability, and data quality. To enhance time efficiency, I could narrow the hyperparameter grid search space, particularly for tree-based models like XGBoost and Random Forest. Replacing *GridSearchCV* with *RandomizedSearchCV* or tools like *Optuna* for Bayesian Optimization would further optimize the search process while saving computational time.

For more robust evaluation, incorporating *k-fold cross-validation* with early stopping would better assess model performance across diverse splits and help reduce overfitting risks.

On the data side, collecting external factors like seasonal patterns, public holidays, or economic indicators (e.g., inflation) could capture variability influencing cancellations. Improved feature engineering, such as aggregating customer behaviors into segmentation groups or analyzing cancellation trends over time, might improve model performance.

Lastly, addressing deployment challenges will potentially ensure scalability and real-world usability.

References

[1] Mostipak, J. (2018). Hotel booking demand. Kaggle. Retrieved from https://www.kaggle.com/datasets/jessemostipak/hotel-booking-demand/data

[2] Yadav, N. (2020). Hotel booking prediction (99.5% acc). Kaggle. Retrieved from https://www.kaggle.com/code/niteshyadav3103/hotel-booking-prediction-99-5-acc/notebook

[3] Scikit-learn developers. (n.d.). KNeighborsClassifier. Scikit-learn Documentation. Retrieved from https://scikit-learn.org/1.5/modules/generated/sklearn.neighbors.KNeighborsClassifier.html