Stat 109 Project
Dai-Phuong Ngo and Brandon Henley

Introduction

In the complex and competitive landscape of the hospitality industry, effective revenue management is crucial for sustaining profitability and optimizing resource utilization. The project outlined here leverages advanced predictive analytics techniques to forecast daily room rates (Average Daily Rate or ADR) and booking cancellations (CNC), utilizing a rich dataset sourced from hotel bookings. The goal is to harness data-driven insights and the capabilities of machine learning models to enhance decision-making processes in hotel revenue management, thereby automating and refining the strategies employed by hotels and resorts.

The objective of this project is twofold: firstly, to predict the daily rates of hotel rooms based on various influencing factors, and secondly, to forecast the likelihood of cancellations. These predictions aim to support hotel managers and stakeholders in crafting strategies that maximize revenue and optimize operational efficiency.

The realm of hotel revenue management is significantly influenced by pricing strategies and the ability to predict cancellations effectively. An adept pricing strategy ensures that room rates are set to optimize revenue based on demand fluctuations, thereby enhancing the profitability of hotel accommodations. Conversely, reliable cancellation forecasts allow for better inventory management, efficient resource allocation, and minimization of revenue losses due to booking cancellations. This dual approach not only improves financial outcomes but also enhances customer service by managing expectations and availability more effectively.

The dataset used in this analysis, available on Kaggle and derived from real hotel bookings, encompasses a variety of variables that impact room rates and cancellation probabilities. These variables include, but are not limited to, lead time, arrival dates, length of stay, the number of guests, booking channels, customer types, and market segments. Additionally, the dataset is enriched with derived metrics critical for revenue management, such as date-related features and other statistical measurements.

The implications of this analysis are profound for hotel management, particularly in the domains of pricing and cancellation forecasting. Accurate predictions of daily room rates enable hotels to adjust prices dynamically, ensuring that revenue is maximized during periods of high demand and maintained during off-peak times. For instance, during peak travel seasons, hotels might increase prices to capitalize on higher demand, whereas in slower periods, they might reduce rates to attract more guests, thus avoiding the pitfalls of empty rooms.

Moreover, predicting cancellations with high accuracy allows hotels to implement proactive strategies aimed at mitigating revenue loss. This can include overbooking practices within acceptable limits, adapting cancellation policies to encourage non-refundable bookings, and deploying targeted promotions designed to reduce the likelihood of cancellations.

This predictive analysis equips hoteliers with actionable insights that enable the refinement of pricing strategies and the optimization of revenue management practices. Through the application of sophisticated predictive models, hotel and resort managers can navigate the challenges of a fluctuating market landscape, making informed decisions that bolster profitability and enhance operational efficiencies. Ultimately, this project demonstrates the transformative potential of data science in revolutionizing traditional approaches to hotel management, making it an indispensable tool in the modern hospitality industry.

With the increasing availability of large datasets, neural networks are becoming more common for tackling complex problems. These networks are especially useful in scenarios where relationships within the data are complicated and not easily modeled with traditional statistical methods. Neural networks can learn these relationships deeply through their layered structures, making them an excellent choice for our hotel booking data, which includes various interacting factors like customer behavior and seasonal influences.

In this context, we test the following hypotheses:
Hypothesis 1 (H1): Factors such as how early a booking is made, the type of customer, and the booking season are key predictors of daily rates.
Hypothesis 2 (H2): The chance of a booking being canceled is likely affected by factors like the amount of time between booking and staying, past cancellations by the customer, and any special requests made during booking.
Hypothesis 3 (H3): Neural networks, with their ability to model complex data through multiple layers, are expected to outperform traditional machine learning models in predicting outcomes related to hotel bookings.

I. **Methodology**

Data Preprocessing

In the initial stage of data preprocessing, the focus was on cleaning the data to ensure that it was suitable for analysis. This involved several key steps:

Handling Missing Values: Missing values were identified across different columns in the dataset. To maintain the integrity of the dataset without losing significant information, these missing values were replaced with the mode of the respective columns.

Filtering ADR and Lead Times: The dataset included entries with non-positive values for 'Average Daily Rate' (ADR) and lead times. These values were not logical within the context of hotel bookings, as they represent scenarios where bookings either did not generate revenue or were booked retroactively. There were three rows with ADR values greater than $500 were also removed. Bookings with a lead time of zero or less were removed from the dataset. These entries might represent walk-ins or same-day reservations, which do not provide useful information for forecasting future bookings, which require lead time analysis.

Feature Engineering

In addition to cleaning the data, we created new variables and augmenting existing ones in an attempt to provide more insights into the booking patterns. Each booking was expanded into multiple records according to the number of nights between the check-in and check-out dates to account for the total rooms sold by date. The total number of room nights was calculated by summing the length of stay for each booking. This metric was then used to compute the 'Average Length of Stay' (ALOS) by dividing the total room nights by the total rooms sold per date. These metrics were intended to provide insights into the actual room usage and help in making informed decisions regarding room pricing and promotional strategies during different periods.

Model Development

We explored various models to forecast daily rates and cancellation probabilities for hotel bookings. The models selected for regression analysis included Linear Regression (LinR), Random Forest (RF), XGBoost (XGB), and Neural Networks (NN), each offering unique strengths in handling the dataset complexities. Linear Regression served as a baseline model, while more sophisticated models like Random Forest and XGBoost were utilized for their robustness and ability to manage overfitting. A Neural Network were also included to see if it could capture deeper patterns within the data the other simpler models might miss. A logistic regression model (LogR) was selected as a baseline classification model. RF and XGB models were also obtained for the classification task. The letter “R” is used to denote a regressor and “C” a classifier. The NN performs regression and classification simultaneously.

For the regression models, we assessed performance using the metrics R^2 and Root Mean Squared Error (RMSE). The classification models were evaluated using accuracy, precision, recall, F1 score and area under the curve (AUC) of the receiver operating characteristic (ROC) curve, with a particular emphasis on recall to reduce the risk of false negatives.

We employed a k-fold cross-validation grid search to search for optimal hyperparameters for the RF and XGB models.

II. **Results**

We began by fitting LinR to the training data, in which all of the original and engineering features are used to predict ADR. This first step allowed for the investigation of multicollinearity via Variance Inflation Factor analysis. The categorical features *market_segment*, *country*, and *distribution_channel_direct* and numeric features *total_room_nights*, *rooms_sold*, and *avg_length_stay* have VIF scores > 10, with *market_segment* showing the most significant collinearity (Figure 1).

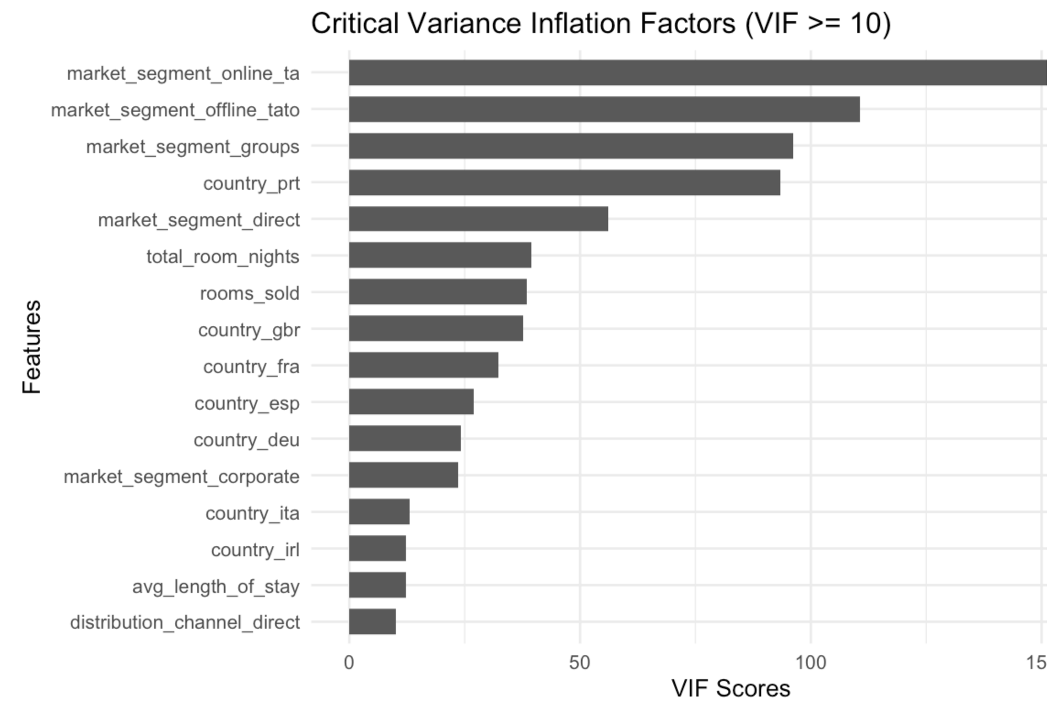

Figure 1. Significant VIF scores (VIF >= 10).

To satisfy LinR's assumption of feature independence, we removed drop *market_segment*, *rooms_sold*, *total_room_nights*, and *avg_length_of_stay*. *country* was replaced with a binary variable *from_prt* representing whether the guest is from Portugal. *distribution_channel* was modified to a binary variable representing whether the booking was made through a travel agency or not. After applying these changes, multicollinearity was removed (all VIF scores < 10).

LinR was fit again using ADR and LogR was fit using CNC on the training set. Since our approach was to treat the regression and classification tasks as a dual modeling task, we excluded only those features that were not statistically significant in both two models. It was found that *babies* was insignificant in both models, while some of the month classes in *booking_month* and *checkin_month* were both significant and insignificant. To handle this, we removed *babies* and replaced the month variables to *booking_season* and *checkin_season*, where December to February is mapped to Winter, March to May to Spring, June to August to Summer, and September to November to Fall. These changes resulted in # total numeric and categorical features, where each numeric and class feature being statistically significant in either the LinR or LogR models.

We fit yet again LinR and LogR on the training data and examined the residuals of LinR to check the normality assumption. Diagnostic plots for LinR are shown below in Figure 2. There was no systematic pattern in the model residuals (top left graph), with only a few moderate outliers. The residuals follow the Q-Q reference line for theoretical quantiles between about -2 and 2 (top right), however this is deviation away from the line in the tail, indicating potential issues with extreme values. There is a consistent spread of variance in residuals across the fitted values, validating the homoscedasticity assumption (bottom left). Finally, there are no clearly problematic leverage points (bottom right). These results suggest that LinR is valid for non-extreme values.

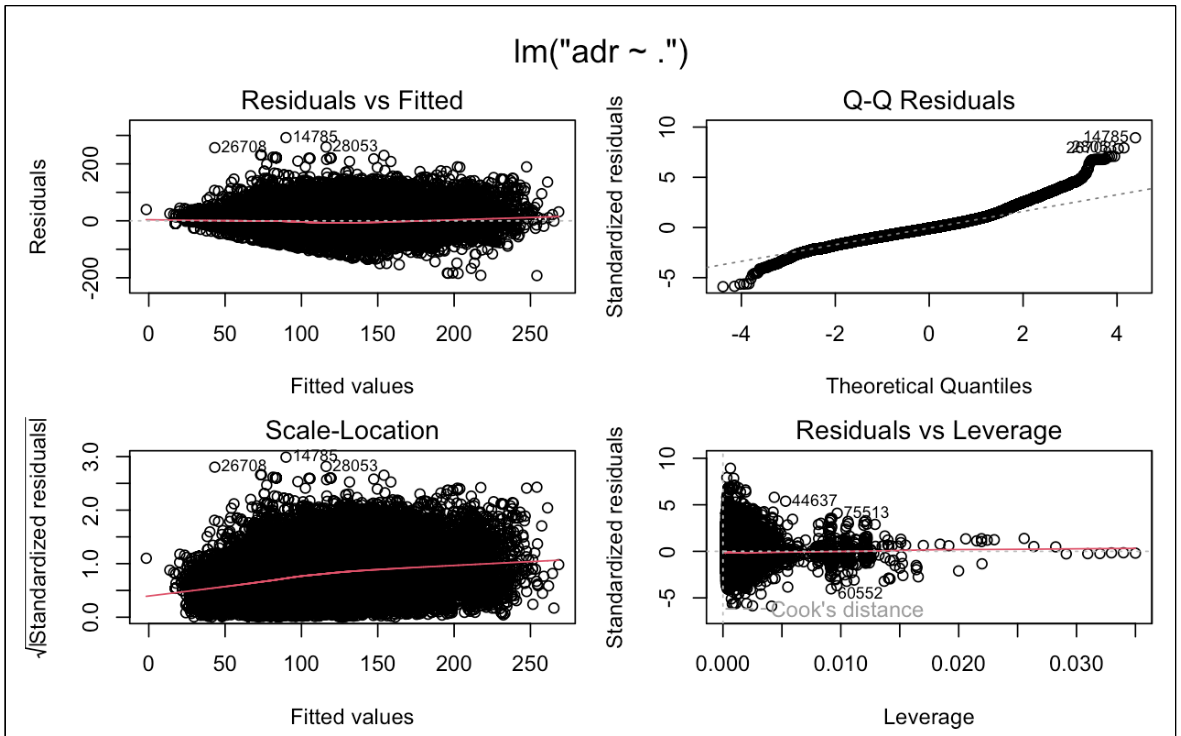

Figure 2. Diagnostic plots for LinR.

The random forest and XGBoost regressors and classifiers were obtained using the same data preprocessed for the linear models. While residual normality and feature independence are not required constraints for these ensemble models, we still use the same preprocessed dataset used to fit the linear models for comparison. The use of tree-based models will also give us a way to identify important features for ADR and CNC prediction. A CV grid search found that the optimal RFR and RFC had an ensemble size of 100, maximum depth of 30, and minimum leaf size of 30. The XGBR and EXGBC were fixed at 100 estimators, and their L1 regularization parameter was searched to limit overfitting. It was found that a regularization value of 1.0 for the XGBR and 0.4 for the XGBC was optimal.

The NN proposed in this project was designed to perform simultaneously the regression and classification task (Figure 3). To do this, we employed two distinct heads: a regression head and a classification head. The regression head, configured with a 64-neuron layer followed by ReLU activation and a single neuron output with ReLU activation, is designed to predict continuous variables such as ADR. Conversely, the classification head, which includes a 128-neuron layer with ReLU activation and has a single output neuron with a sigmoid activation, aims to predict binary outcomes, specifically CNC. The training and validation loss history for this model shows a rapid decrease in loss within the first few epochs, demonstrating quick learning and adaptation by the model. The loss stabilizes as the epochs increase, with the validation loss closely tracking the training loss, which indicates a well-fitted model without significant overfitting or underfitting, highlighting the model's effective learning and generalization capability over 200 epochs.

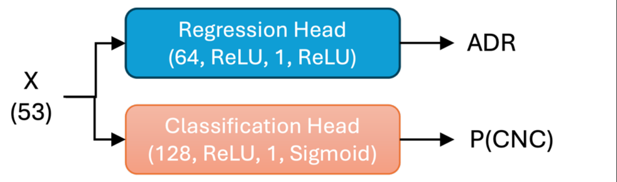

Figure 3. Diagram of proposed neural network.

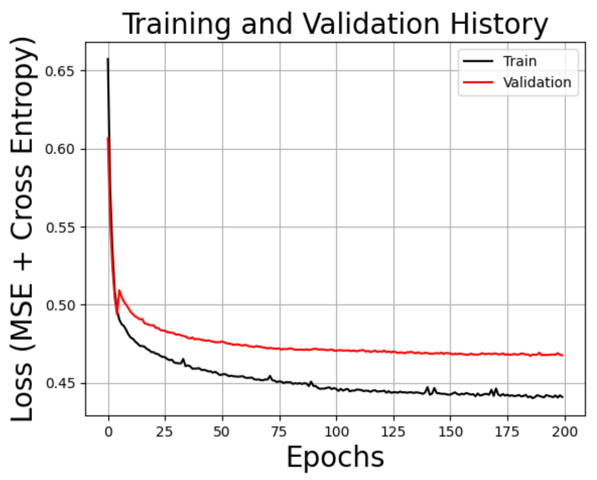

Figure 4. History of training and validation loss, which is the sum of the regression loss (MSE) and classification loss (binary cross entropy).

Figure 5 displays the comparison of R^2 and RMSE for the regression models predicting ADR. All models show relatively high R^2 values, indicating a good fit, with the NN and XGBC performing slightly better than LogR and RFC. The RMSE values are lowest for XGBC and NN, suggesting these models have the highest accuracy in predicting daily rates.

The ROC curve, shown in the Figure 6, illustrates each model's ability to classify cancellations. The curve closer to the top-left corner indicates a better performance. Here, the XGBC and NN models exhibit the highest AUC, followed closely by RFC and LogR, indicating strong discriminative ability for cancellation predictions. As shown in Table 1, XGBC emerges slightly superior across most classification metrics, especially in recall and F1-score, which are crucial for minimizing false negatives in cancellation predictions. This is particularly important as high recall ensures that most actual cancellations are correctly identified, allowing hotel management to adjust bookings and manage revenue loss proactively.

While all models perform commendably, XGBC and NN show particular promise, balancing high performance across all evaluated metrics. These results underscore the potential of advanced ensemble methods and neural networks in refining predictive accuracy in a complex, dynamic industry like hospitality. This capability to predict outcomes accurately can significantly enhance strategic decision-making in hotel revenue management.

Since XGBoost was the top performing model, we examined these models further. Figure 7 show the top 10 important features for the XGBR and XBGC. The features for XGBR mainly pertain to the type of stay and season. Specifically *checkin_season_summer* emerges as the most influential feature, suggesting higher rates during the summer likely due to increased demand. The importance of specific accommodations or offerings such as hotel_resort and various room types like F, G, D, and E also highlight how certain characteristics of a booking can significantly influence pricing. Additionally, the presence of features related to the number of occupants (children and adults) and other seasonal influences like checkin_season_winter and meal plans (meal_half_board) indicates their impact on the cost of the stay.

In XGBC, the most critical feature is *deposit_type_non_refundable* and is very dominant, indicating that bookings with non-refundable deposits have a lower likelihood of cancellation. This is followed by features that track a guest's booking history, such as *previous_cancellations* and *previous_bookings_not_canceled*, suggesting that past behavior is a strong predictor of future actions. The list also includes *customer_type_transient* and various room types, pointing to how different customer segments and room preferences correlate with cancellation probabilities. Operational aspects of bookings, like meal plans (meal_self_catering) and changes made post-booking (booking_changes), are also significant in determining the likelihood of a cancellation.

These insights underscore the different ways in which both internal booking features and external customer behaviors impact hotel management strategies. By understanding the importance of these features, hotel management can better tailor their pricing and operational strategies to optimize revenue and reduce the risks associated with cancellations. This approach allows for more dynamic and effective revenue management in the ever-changing landscape of the hospitality industry.

III. Discussion

Our analysis attempted to provide some insight into predicting daily rates and cancellations in hotel bookings. For daily rates, factors such as the time of booking, customer type, and seasonal variations emerged as significant predictors. The advanced machine learning models, particularly the neural networks, demonstrated enhanced predictive accuracy over traditional regression models, aligning with our initial hypothesis that complex models would perform better due to their ability to capture intricate patterns in the data. For cancellations, the significant predictors included advance booking duration, previous cancellations, and special requests, confirming our hypothesis that these features play crucial roles. These findings indicate that both sets of features have a profound impact on booking dynamics, which can be leveraged to optimize pricing strategies and cancellation policies effectively.

The practical implications of these findings are substantial for hotel management. Accurate predictions of daily rates enable dynamic pricing strategies, ensuring optimal revenue during different demand periods. On the cancellation front, understanding the likelihood of cancellations can aid in better inventory and contingency planning, thus minimizing potential revenue loss and enhancing customer service. Given these insights, we recommend that hotels adopt advanced predictive models to refine their revenue management systems. Further research could explore the integration of external data sources, such as economic indicators or competitive pricing, to enhance the predictive accuracy of the models.

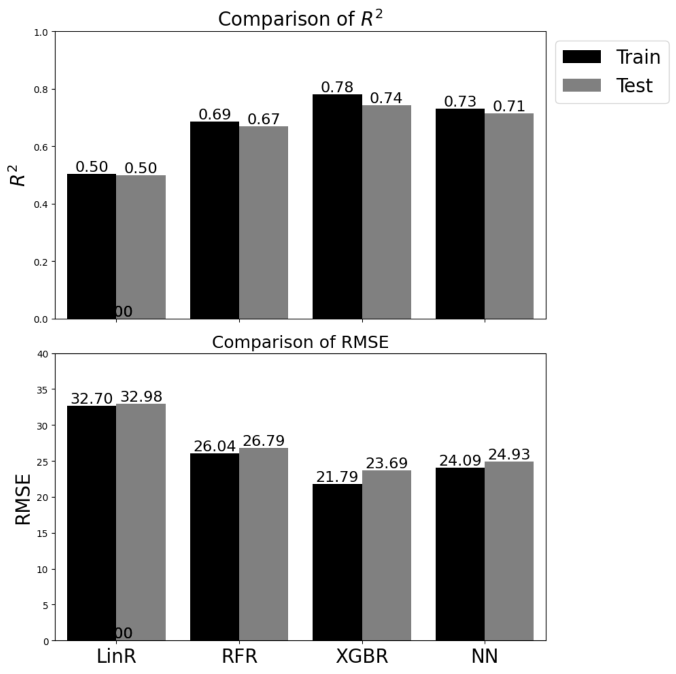

Figure 5: Comparison of R^2 and RMSE for the LinR, RFR, XGBR, and NN models.

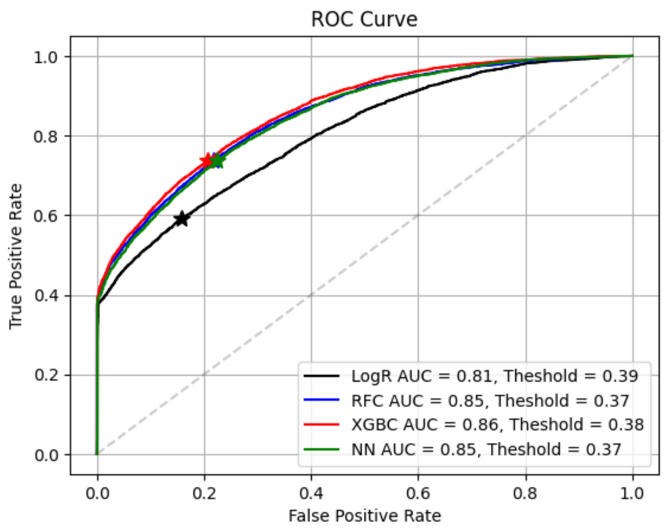

Figure 6: Comparison of Receiver Operating Characteristic (ROC) Curves for the LogR, RFC, XGBC, and NN models.

	LogR	**RFC**	**XGBC**	**NN**
Accuracy	0.74	0.76	0.77	0.76
Precision	0.71	0.69	0.70	0.68
Recall	0.59	0.74	0.74	0.74
F1	0.64	0.71	0.72	0.71
AUC	0.81	0.85	0.86	0.85

Bottom: Table 1. Classification metrics for the LogR, RFC, XGBC, and NN models. Metrics are based on each model's optimal threshold.

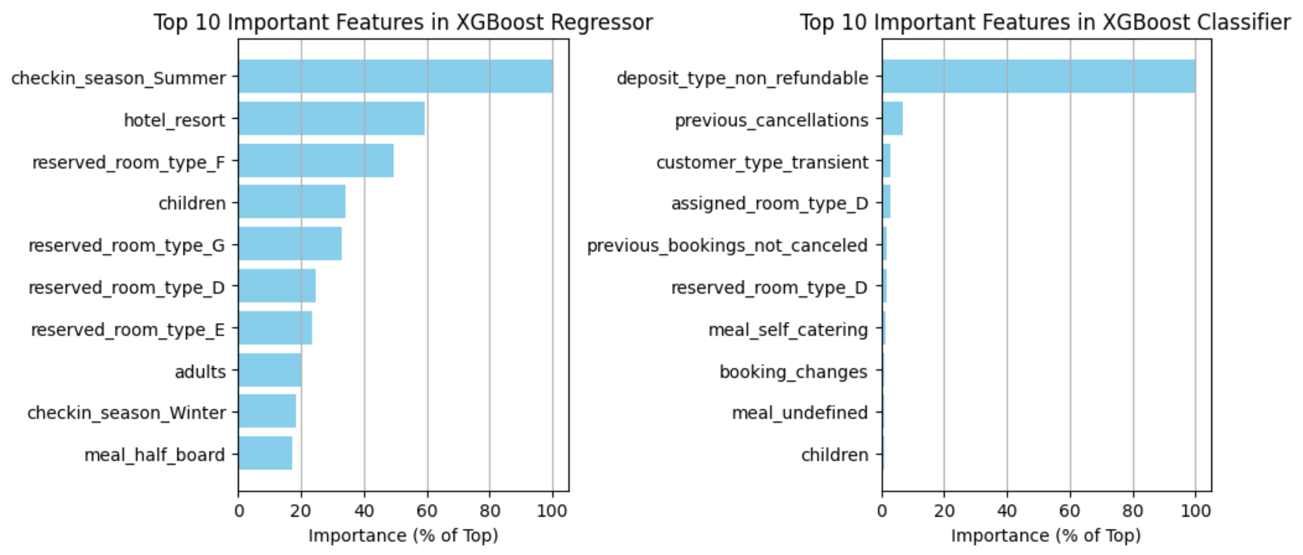

Figure 7. Top 10 important features for XGBR and XGBC.

A key limitation of the project is the raw format of the dataset sourced from Kaggle, which catalogs individual bookings without aggregating data necessary for advanced hotel revenue management metrics such as revenue per available room, total revenue per available room, and revenue per occupied room. For more effective machine learning model development, the dataset should be restructured to include grouped data by dimensions such as date, sales segments, and customer behaviors. This transformation will enable more accurate reporting and forecasting by leveraging historical data to better inform future revenue management strategies. Another limitation was that we assume independence of observations over time. This ignores possible time correlations in the probability of canceling a booking, which is likely not insignificant.

In conclusion, this project highlights the importance of utilizing advanced analytics in the hospitality industry to forecast key outcomes like daily rates and cancellations. The findings from this study not only reinforce the capability of sophisticated machine learning models in enhancing decision-making processes but also contribute valuable insights into the field of hospitality management. By embracing these technological advancements, hoteliers can ensure sustainable growth and continued relevance in a highly competitive market, ultimately leading to more informed and effective management practices in the dynamic landscape of hospitality.

IV. Appendix

Data Dictionary

Data Dictionary of Original Variables	
hotel	Type: Resort Hotel or City Hotel
is_canceled	Booking canceled (1) or not (0)
lead_time	Days between booking and arrival
arrival_date_year	Arrival year
arrival_date_month	Arrival month
arrival_date_week_number	Arrival week number
arrival_date_day_of_month	Arrival day
stays_in_weekend_nights	Weekend nights stayed
stays_in_week_nights	Week nights stayed
adults	Number of adults
children	Number of children
babies	Number of babies
meal	Meal type (Undefined/SC, BB, HB, FB)
country	Guest's country of origin
market_segment	Market segment (e.g., Online TA, Offline TA/TO)
distribution_channel	Distribution channel (e.g., TA/TO, Direct)
is_repeated_guest	Repeated guest (1) or not (0)
previous_cancellations	Number of prior cancellations
previous_bookings_not_canceled	Number of prior bookings not canceled
reserved_room_type	Room type reserved
assigned_room_type	Room type assigned
booking_changes	Number of booking changes
deposit_type	Deposit type (No Deposit, Non Refund, Refundable)
agent	Booking agent ID
company	Company ID
days_in_waiting_list	Days booking was on waiting list
customer_type	Booking type (Contract, Group, Transient, Transient-party)
adr	Average Daily Rate
required_car_parking_spaces	Required parking spaces
total_of_special_requests	Number of special requests
reservation_status	Booking status (Canceled, Check-Out, No-Show)
reservation_status_date	Date of last reservation status

LinR and LogR ®

```
LinR <- lm(formula = "adr ~ .", data = data_train_adr)
LogR <- glm(formula = "factor(is_canceled) ~ .", family = 'binomial', data =
data_train_is_canceled)

LinR_summary <- summary(LinR)
LogR_summary <- summary(LogR)

LinR_LogR_summary <- cbind(LinR_summary$coefficients[,'Pr(>|t|)'],
LogR_summary$coefficients[,'Pr(>|z|)'])
colnames(LinR_LogR_summary) <- c("LinR", "LogR")
```

VIF scores and plotting (R)

```
vif_scores <- vif(LinR)
vif_scores_5 <- sort(vif_scores[vif_scores >= 5], decreasing = TRUE)
vif_scores_10 <- data.frame(sort(vif_scores[vif_scores >= 10], decreasing = TRUE))
colnames(vif_scores_10) <- "vif_score"
vif_scores_10 <- rownames_to_column(vif_scores_10, var = "feature")
vif_scores_10 <- vif_scores_10[order(vif_scores_10$vif_score, decreasing = FALSE), ]
vif_scores_10$feature <- factor(vif_scores_10$feature, levels = unique(vif_scores_10$feature))

X_names_high_vif <- X_names_transformed[vif_scores >= 10]
```

```
ggplot(vif_scores_10, aes(x = feature, y = vif_score)) +
  geom_bar(stat = "identity", position = "dodge", width = 0.7) + # aes(fill = feature)
  coord_flip() +
  labs(title = "Critical Variance Inflation Factors (VIF >= 10)", x = "Features", y = "VIF Scores")
+
  theme_minimal() +
  # scale_fill_brewer(palette = "Paired") +  # color to the bars
  theme(legend.position = "none")  # Remove legend
```

Diagnostic plots for LinR (R)

```
# Set up the graphics layout for a 2x2 grid with less spacing
par(mfrow = c(2, 2), mar = c(4, 4, 2, 1), oma = c(0, 0, 2, 0))

# Plot the four standard diagnostic plots for the linear model
plot(LinR)

# Reset the graphical parameters (optional, for further plotting)
par(mfrow = c(1, 1), mar = c(5, 4, 4, 2) + 0.1)
```

Cross-Validation grid search for RF and XGB models (Python)

```
shuffle_split = ShuffleSplit(n_splits = 5,
                             train_size = train_size,
                             random_state = random_state)

mean_absolute_error_scorer = make_scorer(mean_absolute_error,
                                         greater_is_better = False)
mean_squared_error_scorer = make_scorer(mean_squared_error,
                                        greater_is_better = False)

root_mean_squared_error_scorer = make_scorer(lambda y_true, y_pred:
np.sqrt(mean_squared_error(y_true, y_pred)),
                                             greater_is_better = False)

r2_scorer = make_scorer(r2_score,
                        greater_is_better = True)
```

Cross validation grid search for RFR (Python)

```
n_features = X_train_transformed_LinR.shape[1]
n_samples = X_train_transformed_LinR.shape[0]

param_grid_rfr = {
    'n_estimators': [100],  # Number of trees in the forest
    'max_depth': [None, 10, 20, 30, 40, 50],  # Maximum depth of the trees
    }

print(param_grid_rfr)

min_samples_leaf = 30
min_samples_split = 30
max_features = 'sqrt'

model_rfr = RandomForestRegressor(bootstrap = True,
                                  max_features = max_features,
                                  min_samples_leaf = min_samples_leaf,
                                  min_samples_split = min_samples_split,
                                  random_state = random_state,
                                  n_jobs = 1)

gscv_rfr = GridSearchCV(estimator = model_rfr,
                        param_grid = param_grid_rfr,
                        return_train_score = True,
                        scoring = root_mean_squared_error_scorer,
                        cv = shuffle_split,
                        n_jobs = 1,
                        verbose = 3)

gscv_rfr.fit(X_train_transformed_LinR.values, y_train['adr'].values.squeeze()) ;
```

Cross validation grid search for RFC (Python)

```python
n_features = X_train_transformed_LinR.shape[1]
n_samples = X_train_transformed_LinR.shape[0]

param_grid_rfc = {
    'n_estimators': [100],  # Number of trees in the forest
    'max_depth': [None, 10, 20, 30, 40, 50],  # Maximum depth of the trees
    }

print(param_grid_rfc)

min_samples_leaf = 30
min_samples_split = 30
max_features = 'sqrt'

model_rfc = RandomForestClassifier(bootstrap = True,
                                   max_features = max_features,
                                   min_samples_leaf = min_samples_leaf,
                                   min_samples_split = min_samples_split,
                                   random_state = random_state,
                                   n_jobs = 1)

gscv_rfc = GridSearchCV(estimator = model_rfc,
                        param_grid = param_grid_rfc,
                        return_train_score = True,
                        scoring = 'accuracy',
                        cv = shuffle_split,
                        n_jobs = 1,
                        verbose = 3)

gscv_rfc.fit(X_train_transformed_LinR.values, y_train['is_canceled'].values.squeeze()) ;
```

Cross validation grid search for XGBR (Python)

```python
param_grid_xgbr = {'n_estimators': [100],  # Number of trees in the forest
                  'max_depth': [None, 2, 4, 6],  # Maximum depth of the trees
                  'reg_alpha': [0, 0.2, 0.4, 0.6, .8, 1.],  # L1 regularization term on weights
                  }

model_xgbr = XGBRegressor(random_state = random_state)

gscv_xgbr = GridSearchCV(estimator = model_xgbr,
                        cv = shuffle_split,
                        param_grid = param_grid_xgbr,
                        scoring = root_mean_squared_error_scorer,
                        return_train_score = True,
                        verbose = 3)

gscv_xgbr.fit(X_train_transformed_LinR.values, y_train['adr'].values.squeeze()) ;
```

Cross validation grid search for XGBC (Python)

```
param_grid_xgbc = {'n_estimators': [100],  # Number of trees in the forest
                  'max_depth': [None, 2, 4, 6],  # Maximum depth of the trees
                  'reg_alpha': [0, 0.2, 0.4, 0.6, .8, 1.],  # L1 regularization term on weights
                  }

model_xgbc = XGBClassifier(random_state = random_state)

gscv_xgbc = GridSearchCV(estimator = model_xgbc,
                         cv = shuffle_split,
                         param_grid = param_grid_xgbc,
                         scoring = 'accuracy',
                         return_train_score = True,
                         verbose = 3)

gscv_xgbc.fit(X_train_transformed_LinR.values, y_train['is_canceled'].values.squeeze()) ;
```

Training NN (Python)

```
def torch_train_regclass(model,
                         loss_fns,
                         optimizers,
                         dl_train,
                         dl_val = None,
                         loss_weights = (1, 1),
                         epochs = 10,
                         device = 'cpu',
                         reg_dtype = torch.float32,
                         cls_dtype = torch.float32):
    """
    Train a model with both regression and classification outputs using separate optimizers.

    Args:
    - model: The PyTorch model to train.
    - loss_fns (tuple): A tuple of two loss functions (loss_reg_fn, loss_cls_fn).
    - optimizers (tuple): A tuple of two optimizer instances (optimizer_reg, optimizer_cls).
    - dl_train: Training dataloader.
    - dl_val: Validation dataloader (optional).
    - epochs: Number of epochs to train for.
    - device: Device to train on ('cpu' or 'cuda').

    Returns:
    - train_loss_history: List of average training losses per epoch.
    - val_loss_history: List of average validation losses per epoch if dl_val is provided.
    """

    loss_reg_fn, loss_cls_fn = loss_fns
    optimizer_reg, optimizer_cls = optimizers
    train_loss_history = []
    val_loss_history = []

    for epoch in range(epochs):
        model.train()
        train_losses = []
        for X_train, (y_train_reg, y_train_cls) in dl_train:
            X_train = X_train.to(device=device, dtype=torch.float32)
            y_train_reg = y_train_reg.to(device=device, dtype=reg_dtype)
            y_train_cls = y_train_cls.to(device=device, dtype=cls_dtype)

            optimizer_reg.zero_grad()
            optimizer_cls.zero_grad()

            y_pred_reg, y_pred_cls = model(X_train)

            loss_reg = loss_reg_fn(y_pred_reg, y_train_reg)
            loss_cls = loss_cls_fn(y_pred_cls.squeeze(), y_train_cls.squeeze())

            loss = loss_weights[0]*loss_reg + loss_weights[1]*loss_cls
            loss.backward()

            optimizer_reg.step()  # Update regression parameters
            optimizer_cls.step()  # Update classification parameters

            train_losses.append(loss.item())

        train_loss_history.append(np.mean(train_losses))

        if dl_val:
            model.eval()
            with torch.no_grad():
                val_losses = []
                for X_val, (y_val_reg, y_val_cls) in dl_val:
                    X_val = X_val.to(device=device, dtype=torch.float32)
                    y_val_reg = y_val_reg.to(device=device, dtype=reg_dtype)
                    y_val_cls = y_val_cls.to(device=device, dtype=cls_dtype)

                    y_pred_reg, y_pred_cls = model(X_val)

                    loss_reg = loss_reg_fn(y_pred_reg, y_val_reg)
                    loss_cls = loss_cls_fn(y_pred_cls.squeeze(), y_val_cls.squeeze())

                    loss = loss_weights[0]*loss_reg + loss_weights[1]*loss_cls
                    val_losses.append(loss.item())

                val_loss_history.append(np.mean(val_losses))
                print(f"Epoch {epoch+1}/{epochs}: Training Loss = {train_loss_history[-1]:.4f},
Validation Loss = {val_loss_history[-1]:.4f}")
        else:
            print(f"Epoch {epoch+1}/{epochs}: Training Loss = {train_loss_history[-1]:.4f}")

    return train_loss_history, val_loss_history if dl_val else train_loss_history
```

NN Model

```
class SimpleRegClassModel(torch.nn.Module):
    def __init__(self,
                 input_features,
                 output_regression = 1,
                 reg_hidden_dim = 32, clf_hidden_dim = 32,
                 dropout = 0.,
                 device = 'cpu', dtype = torch.float32):
```

```
        super(SimpleRegClassModel, self).__init__()

        self.device = device
        self.dtype = dtype

        # Regression head
        self.regression_head = torch.nn.Sequential(
            torch.nn.Linear(input_features, reg_hidden_dim),
            torch.nn.ReLU(),
            torch.nn.Dropout(dropout),
            torch.nn.Linear(reg_hidden_dim, output_regression),
            torch.nn.ReLU()
        ).to(device = self.device, dtype = self.dtype)

        # Classification head
        self.classification_head = torch.nn.Sequential(
            torch.nn.Linear(input_features, clf_hidden_dim),
            torch.nn.ReLU(),
            torch.nn.Dropout(dropout),
            torch.nn.Linear(clf_hidden_dim, 1),
            torch.nn.Sigmoid()
        ).to(device = self.device, dtype = self.dtype)

    def forward(self, x):

        x = x.to(device = self.device, dtype = self.dtype)

        regression_output = self.regression_head(x)
        classification_output = self.classification_head(x)
        return regression_output, classification_output
```

References

Data source: https://www.kaggle.com/datasets/jessemostipak/hotel-booking-demand