Predicting Hotel Booking Cancellations

Ina Holtschmit

iholtschmit3@gatech.edu

12 April, 2023

Abstract

Classification models (Logistic Regression, Naïve Bayes, Random Forest, and Boosting) were trained to predict whether an online hotel room booking from a website will be cancelled before the guests arrival. These were trained on a data set of 3,628 observations containing a mix of categorical and numerical variables. The Naïve Bayes model was found to be significantly more accurate than the alternative models, with an accuracy of 0.0.9733, specificity of 0.9619, and sensitivity of 0.9970. Correctly identifying cancellations would help hotels to minimize the amount of revenue lost by allowing them to overbook in such a way that keeps their rooms filled.

Introduction

Since the inception of online booking of hotels on popular travel websites, the number of booking cancellations has been increasing for several reasons. Many customers have turned to the strategy of "book and search," where a customer will book a room when they find one that is adequate, while still continuing to look for a better one afterwards before the no-fee cancellation period is over. Approximately 33% of online bookings are either cancelled or no-shows. This is a huge number of revenue that is lost in the hospitality and tourism industry. A classification model can be used to determine whether a guest is likely to cancel their hotel booking. If a hotel has many guests in one night that are predicted to cancel or not show up for their reservation, the hotel can over-book rooms for that night in order to lose less revenue. Several classification models will be trained and tested to determine how accurately a guest will be predicted to cancel their booking.

Data Source

The data set, "Hotel Reservations," was obtained from Kaggle. It originally contained 36,280 observations, but for faster computation when running iterations of models, only 10% of the original data was used, for a total of 3,628 observations. While these were selected randomly, the distribution of the categorical response variable was viewed before and after this process to confirm it had remained the same. Approximately two thirds of the response variable were 0, where the booking was not cancelled, and the remaining third was 1, where the booking was cancelled.

The data set was composed of the following categorical predicting variables: type of meal plan, type of room reserved, market segment type, guest arrival year, guest arrival month, whether they were repeated guests, and whether a parking space was required. The following numerical predicting variables were also in the data set: number of adults in reservation, number of children in reservation, number of weekend nights, amount of days the booking was made in advance, number of previous cancellations, number of previous bookings not cancelled, average price per room, and number of special requests made by the guest at the time of the booking. The aforementioned categorical and numerical variables were all of the 17 predicting variables used in the models built.

Methodology

Of the above 3,628 observations in the data set, 75% of them were randomly selected to be used in the training of the models, with the remaining 25% used as the testing set to evaluate the models.

The two classification models that were trained with manual tuning of their parameters to predict whether a booking would be cancelled or not were a Random Forest model and a Boosting

model. A Logistic Regression model and a Naïve Bayes model were also built, but these were not as thoroughly explored in the tuning of their parameters, in that the baseline R function to build these models were trusted to choose all optimal parameters. For these latter two models, the naiveBayes() function from the "e1071" library, and the step() and glm() functions from basic R Studio, were used in training and validation. A seed of 8 was used for entirety of the analysis.

The Random Forest model was built using the randomForest() function from the "RandomForest" library. This function builds a series of decision trees based on the predicting variables, and for each tree, the classification error rate is recorded and averaged over all the trees. To find the optimal tuning parameters, several iterations of the model were trained and cross validated with varying "mtry" values. This parameter specifies the number of variables that would be randomly sampled at each split. The model was trained and cross validated 6 times, from a starting mtry value of 3, up until 9.

The Boosting model was built using the gbm() function from the "gbm" library. It is a greedy algorithm that creates a series of decision trees, where the error rate is improved upon based upon the previous tree. Several iterations of the model were trained and validated with varying parameters, such as number of maximum trees allowed and shrinkage. First the optimal shrinkage value was found, and consequently used in the models that were built to find the optimal number of trees.

The models were cross validated within the functions themselves to find the optimal tuning parameters. Once a final model was chosen, it was used to predict whether a booking would be cancelled or not on the testing data set. Confusion matrices were then calculated for each model, as well as testing error rate, accuracy, sensitivity, and specificity, with the following formulas:

$$Sensitivity = \frac{TP}{(TP+FN)} \qquad Specificity = \frac{TN}{(TN+FP)} \qquad Accuracy = \frac{(TP+TN)}{(TP+TN+FP+FN)}$$

Details for how each of the models were ran can be seen in the Appendix.

Analysis and Results

During the initial exploratory analysis, boxplots for each of the predicting variables were made. From Figure I and Figure II, it was predicted that the number of days the reservation was made in advance, as well as the number of special requests made by the guest at the time of booking, were going to be important. The longer a booking was made in advance, the more likely it would be that it would be cancelled. In figure II, it can be seen that if even just one special request is made by the guest, it is less likely that they will cancel their booking.

Figure I. Cancellations Based on How Many Days Room Was Reserved in Advance

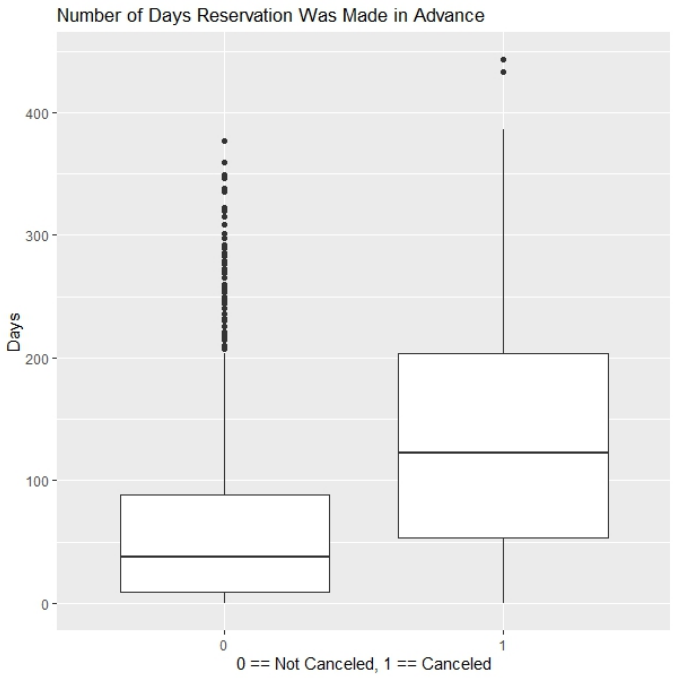
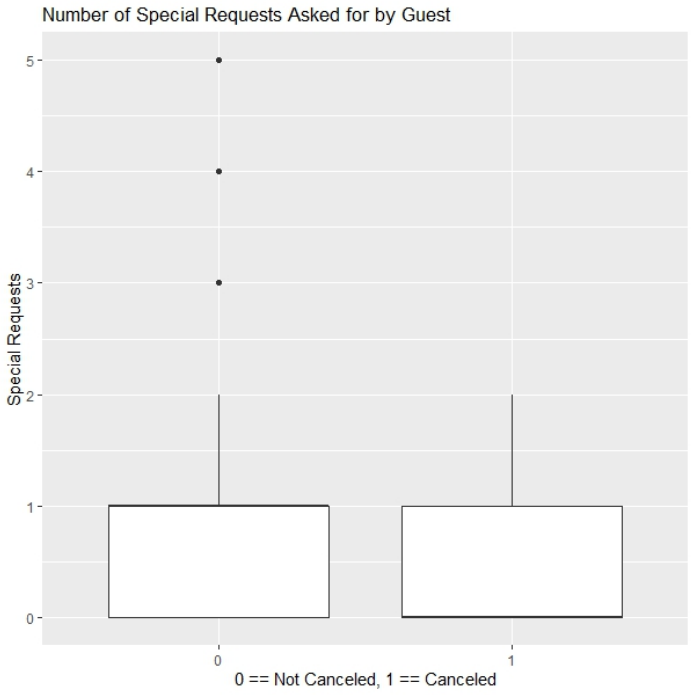


Figure II. Cancellations Based on How Many Special Requests Were Asked by Guest



In tables I and II, we can see that the optimal shrinkage parameter for the Boosting model found during cross validation had a value of 0.05. This value was then used when building and cross validating the Boosting models containing different number of trees to grow (increasing sequentially by 500, starting at 500 and ending at 3500). The optimal number for this parameter was found to be 1500

repeatedly, even when testing under different seeds. The final Boosting model, with shrinkage of 0.05 and 1500 trees, resulted in a testing error of 0.1693.

Table I. Training Errors of Varying Tree Parameter in Boosting Model

Shrinkage	**Training Error**
0.01	0.1444
0.05	0.0859
0.10	0.0898

Table II. Training Errors of Varying Tree Parameter in Boosting Model (Where Shrinkage is 0.05)

Number of Trees	**Training Error**
500	0.0898
1000	0.1005
1500	0.0714
2000	0.0749
2500	0.0848
3000	0.0752
3500	0.0886

In table III below, we see that the optimal parameter for "mtry" in the Random Forest model was 5. The default "mtry" parameter in a classification model, would be the square root of the number of variables in the data, which in this case be 4. After cross validating using values from 3 to 8, the optimal parameter for "mtry" was 5. This was the parameter used in the final model validated on the testing data, which resulted in a testing error of 0.1406.

Table III. Training Errors of Varying Mtry Parameter in Random Forest Model

Mtry	**Training Error**
3	0.0340
4	0.0321
5	0.0317
6	0.0348
7	0.0317
8	0.0328

The difference between the training error and testing error was much greater in the Random Forest model than in the Boosting model. Even though the Random Forest testing error was slightly better than the Boosting testing error (0.1406 versus 0.1693 in table IV below), the accuracy expected from the model is more predictable with the Boosting model. Logistic regression had the highest misclassification rate, with a testing error of 0.2119. On the other hand, the Naïve Bayes model outperformed every other model, with a testing error of 0.0267 (Figure III). Looking at the confusion matrices in Table V, we can see that it only resulted in one false negative and 26 false positives, with a

sensitivity of 0.9970 and specificity of 0.9619. In every sense, the Naive Bayes model is the best model to use when predicting whether a booking will be cancelled. The drastic difference between specificity and sensitivity in the Logistic Regression and Support Vector Machine models can be seen in IV and V.

Table IV. Testing Errors of All Models

Model	Testing Error	Specificity	Sensitivity	Accuracy
Logistic Regression	0.1970	0.5457	0.9267	0.8030
Naïve bayes	0.0267	0.9619	0.9970	0.9733
Boosting	0.1693	0.8856	0.7165	0.8307
Random Forest	0.1406	0.9296	0.7134	0.8594
Support Vector Machine	0.1871	0.4970	0.9648	0.8129

Table V. Confusion Matrices of All Models on Testing Data

Model		**0** ***(Booking Not Canceled – True)***	**1** ***(Booking Canceled – True)***
Boosting	**0** *(Booking Not Canceled—Predicted)*	604	93
	1 *(Booking Canceled—Predicted)*	78	235
Random Forest	**0** *(Booking Not Canceled—Predicted)*	634	94
	1 *(Booking Canceled—Predicted)*	48	234
Logistic Regression	**0** *(Booking Not Canceled—Predicted)*	632	149
	1 *(Booking Canceled—Predicted)*	50	179
Naïve Bayes	**0** *(Booking Not Canceled—Predicted)*	656	1
	1 *(Booking Canceled—Predicted)*	26	327
Support Vector Machine	**0** *(Booking Not Canceled—Predicted)*	658	165
	1 *(Booking Canceled—Predicted)*	24	163

Figure III. Accuracy of Models

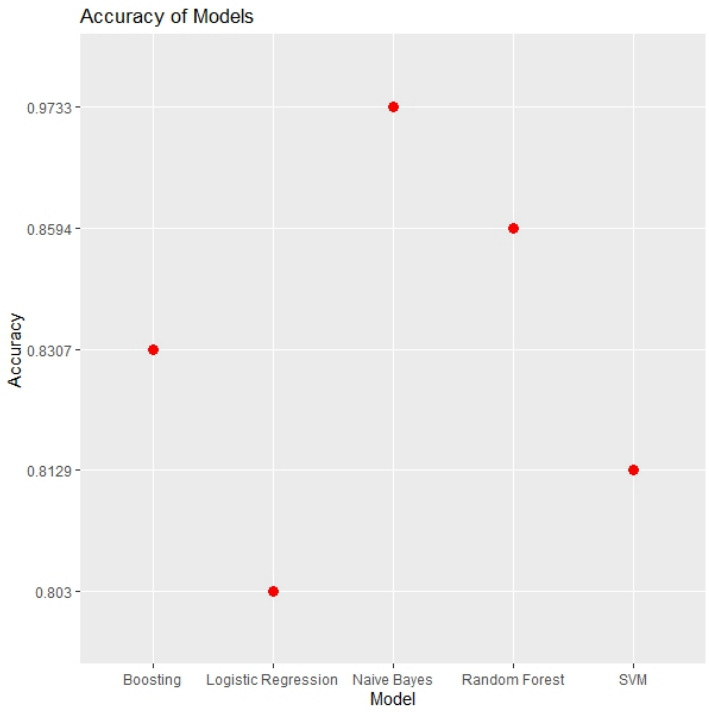


Figure IV. Sensitivity of Models

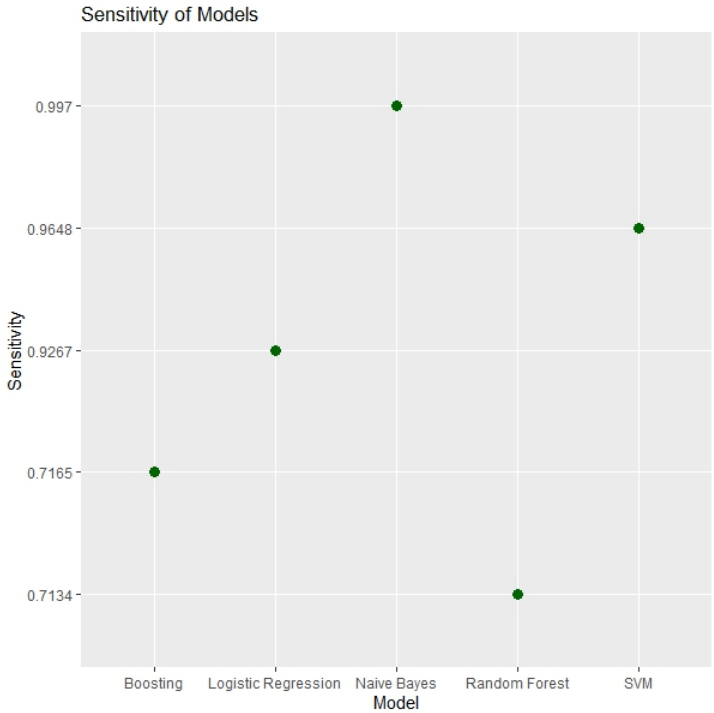
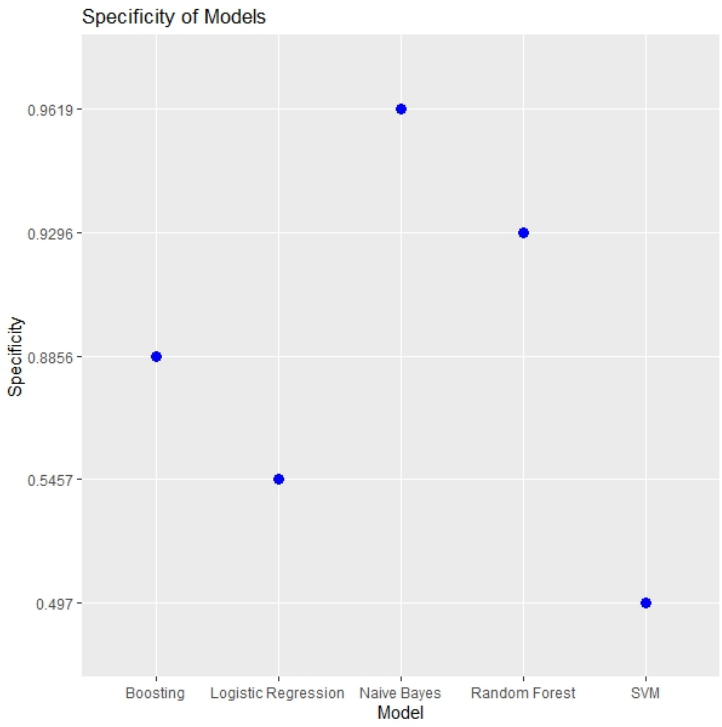


Figure V. Specificity of Models



Conclusion

A Naïve Bayes model should be used to determine whether a future hotel booking will be cancelled. The fact that it had such a high specificity would be extremely valuable in the context of the problem. With a specificity of 0.9619, 96% of people who follow through with their booking were predicted correctly. If a hotel were to overbook, they can use this expected cancellation rate to leave an appropriate margin of error, where even if the hotel overbooks, they only do it to a point where no guest will show up without a room being available to them.

To further research this problem and improve the model, the analysis could be made using all 36,280 observations, rather than just the 3,628. Additionally, the risk of overbooking relative to its consequence can be explored, which could potentially change the probability threshold used when classifying bookings as being cancelled or not. Depending on the reputation of the hotel chain or need for revenue, the way the threshold affects the count of false positives and false negatives needs to be explored. It could be that not having enough rooms ready for the amount of guests that show up for their booking (the day of) could be more costly than having more empty rooms for the night.

Lessons I have learned: One of the major things I learned from this project, is how to try and convey results to someone that has no expertise in the subject (e.g. “fictional boss”). I think that it helps naturally structure the report and what to highlight that will show the audience what the importance is of what you’re looking into. Another aspect of the course I appreciated, is the way the homeworks were structured. It makes it so that you have to thoroughly understand what it is that each of the models is doing, and not just using a function in R Studio. Overall, I feel like I’ve learned a lot in this class, and the

use of practical, real-life examples, makes me more confident that I will be able to provide important insights to future audiences with my work.

Appendix

1) Boosting Model

```
gbm_model <- gbm(booking_status ~ .,data=train,
            distribution = 'bernoulli',
            n.trees = 2500,
            shrinkage = 0.1,
            interaction.depth = 3,
            cv.folds = 10)

perf_gbm = gbm.perf(gbm_model, method="cv")
```

Boosting Model Training Error

```
pred_gbm <- predict(gbm_model,
              newdata = train,
              n.trees=perf_gbm,
              type="response")

pred <- ifelse(pred_gbm < 0.5, 0, 1)
gbm_train_error <- sum(pred != train$booking_status)/length(train$booking_status)
```

Boosting Model Testing Error

```
pred_test <- predict(gbm_model,
               newdata = test[,-18],
               n.trees = 2500,
               shrinkage = 0.1,
               type = "response")

pred_test_new <- ifelse(pred_test < 0.5, 0, 1)
gbm_test_error <- sum(pred_test_new != test$booking_status)/length(test$booking_status)
```

Boosting Optimal Tuning: Number of Trees

```
tree_num <- c(500, 1000, 1500, 2000, 2500, 3000, 3500)
gbm_train_error <- c()
opt_trees <- c()
for (i in tree_num){
  gbm_model <- gbm(booking_status ~ .,data=train,
              distribution = 'bernoulli',
```

```
            n.trees = i,
            shrinkage = 0.1,
            interaction.depth = 3,
            cv.folds = 10)
  perf_gbm <-  gbm.perf(gbm_model, method="cv")
  opt_trees <- c(opt_trees, perf_gbm)

  pred_gbm <- predict(gbm_model,
              newdata = train,
              n.trees=perf_gbm,
              type="response")

  pred <- ifelse(pred_gbm < 0.5, 0, 1)
  gbm_temp_train_error <- sum(pred != train$booking_status)/length(train$booking_status)
  gbm_train_error <- c(gbm_train_error, gbm_temp_train_error);

}

gbm_tree_effect <- as.matrix(cbind(tree_num, gbm_train_error, opt_trees))
gbm_tree_effect
```

Boosting Optimal Tuning: Shrinkage

```
shrink <- c(0.1, 0.05, 0.01)
gbm_shrink_train_error <- c()
for (i in shrink){
  gbm_model <- gbm(booking_status ~ .,data=train,
            distribution = 'bernoulli',
            n.trees = 1000,
            shrinkage = i,
            interaction.depth = 3,
            cv.folds = 10)
  perf_gbm <-  gbm.perf(gbm_model, method="cv")

  pred_gbm <- predict(gbm_model,
              newdata = train,
              n.trees=perf_gbm,
              type="response")

  pred <- ifelse(pred_gbm < 0.5, 0, 1)
  gbm_temp_train_error <- sum(pred != train$booking_status)/length(train$booking_status)
  gbm_shrink_train_error <- c(gbm_shrink_train_error, gbm_temp_train_error);

}

shrinkage_effect <- as.matrix(cbind(shrink, gbm_shrink_train_error))
```

Optimal Boosting Model

```
final_gbm_model <- gbm(booking_status ~ .,data=train,
            distribution = 'bernoulli',
            n.trees = 1000,
            shrinkage = 0.05,
            interaction.depth = 3,
            cv.folds = 10)

final_perf_gbm = gbm.perf(final_gbm_model, method="cv")

final_pred_test <- predict(final_gbm_model,
              newdata = test[,-18],
              n.trees = final_perf_gbm,
              shrinkage = 0.05,
              type = "response")

final_pred_test_new <- ifelse(pred_test < 0.5, 0, 1)
gbm_test_error <- sum(final_pred_test_new != test$booking_status)/length(test$booking_status)
table(final_pred_test_new, test$booking_status)
```

2) Random Forest Model

```
rf_default_train <- randomForest(booking_status ~.,
                    data=train,
                    importance=TRUE)

importance(rf_default_train)
varImpPlot(rf_default_train)
```

Random Forest Training Error

```
rf_pred_train = predict(rf_default_train, train[,-18], type='class')
pred_train_rf <- ifelse(rf_pred_train < 0.5, 0, 1)
rf_train_error <- sum(pred_train_rf != train$booking_status)/length(train$booking_status)

## Prediction on the testing data set
rf_pred = predict(rf_default_train, test[,-18], type='class')
pred_test_rf <- ifelse(rf_pred < 0.5, 0, 1)
rf_test_error <- sum(pred_test_rf != test$booking_status)/length(test$booking_status)
table(pred_test_rf, test$booking_status)
```

Random Forest Optimal Tuning: mtry

```
rf_mtry_train_error <- c()
mtry_num <- c(3, 4, 5, 6, 7, 8)
for (i in mtry_num){
  rf_default_train <- randomForest(booking_status ~.,
                                   data=train,
                                   ntree = 500,
                                   mtry = mtry_num,
                                   importance=TRUE)

  rf_pred_train = predict(rf_default_train, train[,-18], type='class')
  pred_train_rf <- ifelse(rf_pred_train < 0.5, 0, 1)

  rf_temp_train_error <- sum(pred_train_rf != train$booking_status)/length(train$booking_status)
  rf_mtry_train_error <- c(rf_mtry_train_error, rf_temp_train_error);

}
rf_mtry_effect <- as.matrix(cbind(mtry_num, rf_mtry_train_error))
```

Random Forest Testing Error

```
rf_test_model <- randomForest(booking_status ~.,
                              data=train,
                              ntree = 500,
                              mtry = 5,
                              importance=TRUE)

rf_pred_test = predict(rf_default_train, test[,-18], type='class')
pred_test_rf <- ifelse(rf_pred_test < 0.5, 0, 1)
rf_test_error <- sum(pred_test_rf != test$booking_status)/length(test$booking_status)

importance(rf_test_model)
varImpPlot(rf_test_model)
table(pred_test_rf, test$booking_status)
```

3) Logistic Regression Model

```
log_model <- glm(booking_status~., data = train, family = "binomial")
pred_model <- predict(log_model, test, type = "response")

probs <- seq(0.1, 0.95, by = 0.05)
log_test_error <- c()
for (i in probs){
  pred_model <- predict(log_model, test, type = "response")
  pred_model_new <- ifelse(pred_model > i, 1, 0)
```

```
  temp_test_error <- mean(pred_model_new != test$booking_status)
  log_test_error <- c(log_test_error, temp_test_error)
}

pred_model_new <- ifelse(pred_model > 0.6, 1, 0)
log_test_error <- mean(pred_model_new != test$booking_status)

table(pred_model_new, test$booking)
```

4) Naïve Bayes Model

```
modC <- naiveBayes(as.factor(train[,18]) ~. , data = train)
y2hatC <- predict(modC, test, type = 'class')

modC_test_error <- mean( y2hatC != test$booking_status)
table(y2hatC, test$booking)
```

5) Support Vector Machine Model

```
svm_model <- svm(booking_status ~ ., data = hotel)
Yhat_svm <- predict(svm_model, test, type="class")
pred_model_svm <- ifelse(Yhat_svm > 0.5, 1, 0)

modsvm_test_error <- mean( pred_model_svm != test$booking_status)
table(pred_model_svm, test$booking_status)
```

Bibliography

Raza, Ahsan. "Hotel Reservations Dataset." *Kaggle*, 4 Jan. 2023, https://www.kaggle.com/datasets/ahsan81/hotel-reservations-classification-dataset.

Schwartz, Zvi. "Consumers vs. Revenue Managers? the Case of Cancellations and No Shows." *Consumers vs Revenue Managers The Case of Cancellations and No Shows Comments*, School of Hospitality Administration, 29 June 2021, https://www.bu.edu/bhr/2021/06/29/consumers-vs-revenue-managers-the-case-of-cancelations-and-no-shows/.