Random Forest Classification of Hotel Cancellations

Executive Summary

Dakota Contois

Research Question and Hypothesis

The cancellation of hotel reservations is a costly and inevitable expense placed upon the lodging industry. Hotels unable to replace cancelled reservations or fill un-booked rooms lose revenue and profitability. Having the ability to predict if a customer is likely to cancel their reservation gives time to plan ahead allows for hotels to prevent further cancellations and book rooms more efficiently. This study aims to predict the likelihood of a customer cancelling their reservation and reject the null hypothesis that a Random Forest Classifier machine learning model cannot predict hotel cancellations with at least 80% accuracy.

Data Collection

The hotel cancellation data (Mojtaba, 2019) was retrieved in the form of a Comma-Separated Values (CSV) file from Kaggle.com. The unedited file contains 119,390 rows and 36 variables of varying datatypes. The data contains deidentified information about guests and their corresponding reservation cancellation status. The data covers resort and city hotels in 177 different countries over the course of 2016 and 2017. The method of data collection used was searching the internet for a real-world dataset with enough data to sufficiently analyze. One advantage of this data collection method was that I could easily find an open source dataset which matched my criteria instead of having a limited variety of datasets to choose from. One disadvantage to this data collection method is that often times open sourced datasets have imbalanced or incomplete data. In order to handle these issues, I purposely chose a dataset which had a sufficient amount of complete data and a mostly balanced target variable. I also made it a priority to find a dataset which had a binary target variable in order to perform classification.

Data Extraction and Preparation

The program used for this data was the Jupyter Notebook environment through Python. Some advantages of this is that Python code is generally easier to read than other coding programs and Python has a large variety of libraries at its disposal. One disadvantage to Python is that it may be slower in performance to other coding programs. Some libraries used in the data extraction and preparation process were: Pandas, NumPy, Statistics, Matplotlib, and Seaborn. These tools were used in order to store data in a data frame, perform calculations or switch data types, describe the data using summary statistics, or plot and graph the data through visualizations. One advantage to Pandas is that it is very easy to use and one disadvantage is that it may be slower performing than other similar libraries. One advantage to NumPy is that it can easily perform calculations and one disadvantage is that it may not be as intuitive as other programs. One advantage to Statistics is that it can easily describe numeric data with mean, median, standard deviation, and quartiles, but one disadvantage is that its capabilities are limited. One advantage to Matplotlib and Seaborn are that both are extremely versatile and easier to comprehend than other visualization libraries, but one disadvantage is that other libraries may offer more intricate and interesting visualizations, such as ggplot.

The CSV file was uploaded into Python in the form of a Pandas data frame along with necessary packages. Some advantages of this dataset are the large dataset allows for multiple scenarios for which the model to be trained, a wide variety of variables and datatypes, and the target variable has sufficient information for both binary outcomes which aids in accurate prediction. Some disadvantages of the dataset are that there are missing values for several variables, some data entry errors which need to be cleaned, and several columns must be removed prior to analysis.

Importing packages and CSV:

```python
#importing packages and identifying/importing csv file
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import pandas as pd
import numpy as np
import statistics
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc
hotel=pd.read_csv('hotel_booking.csv')
```

Profiling data, testing data sparsity, looking into frequencies:

```python
# profiling dataframe. Large amount of NaNs
hotel.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119390 entries, 0 to 119389
Data columns (total 36 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   hotel                           119390 non-null  object
 1   is_canceled                     119390 non-null  int64
 2   lead_time                       119390 non-null  int64
 3   arrival_date_year               119390 non-null  int64
 4   arrival_date_month              119390 non-null  object
 5   arrival_date_week_number        119390 non-null  int64
 6   arrival_date_day_of_month       119390 non-null  int64
 7   stays_in_weekend_nights         119390 non-null  int64
 8   stays_in_week_nights            119390 non-null  int64
 9   adults                          119390 non-null  int64
 10  children                        119386 non-null  float64
 11  babies                          119390 non-null  int64
 12  meal                            119390 non-null  object
 13  country                         118902 non-null  object
 14  market_segment                  119390 non-null  object
 15  distribution_channel            119390 non-null  object
 16  is_repeated_guest               119390 non-null  int64
 17  previous_cancellations          119390 non-null  int64
 18  previous_bookings_not_canceled  119390 non-null  int64
 19  reserved_room_type              119390 non-null  object
 20  assigned_room_type              119390 non-null  object
 21  booking_changes                 119390 non-null  int64
 22  deposit_type                    119390 non-null  object
 23  agent                           103050 non-null  float64
 24  company                         6797 non-null    float64
 25  days_in_waiting_list            119390 non-null  int64
 26  customer_type                   119390 non-null  object
 27  adr                             119390 non-null  float64
 28  required_car_parking_spaces     119390 non-null  int64
 29  total_of_special_requests       119390 non-null  int64
 30  reservation_status              119390 non-null  object
 31  reservation_status_date         119390 non-null  object
 32  name                            119390 non-null  object
 33  email                           119390 non-null  object
 34  phone-number                    119390 non-null  object
 35  credit_card                     119390 non-null  object
dtypes: float64(4), int64(16), object(16)
memory usage: 32.8+ MB
```

```python
# data sparsity test
#convert each column to SparseArray
spars_test = hotel.apply(pd.arrays.SparseArray)
print (spars_test.sparse.density)
```

```
0.7363893774836903
```

```python
hotel.hotel.value_counts()
```

```
hotel
City Hotel      79330
Resort Hotel    40060
Name: count, dtype: int64
```

```python
hotel.customer_type.value_counts()
```

```
customer_type
Transient          89613
Transient-Party    25124
Contract            4076
Group                577
Name: count, dtype: int64
```

```python
hotel.market_segment.value_counts()
```

```
market_segment
Online TA        56477
Offline TA/TO    24219
Groups           19811
Direct           12606
Corporate         5295
Complementary      743
Aviation           237
Undefined            2
Name: count, dtype: int64
```

```python
hotel.meal.value_counts()
```

```
meal
BB           92310
HB           14463
SC           10650
Undefined     1169
FB             798
Name: count, dtype: int64
```

```
hotel.country.value_counts()
```

```
country
PRT    48590
GBR    12129
FRA    10415
ESP     8568
DEU     7287
       ...
DJI        1
BWA        1
HND        1
VGB        1
NAM        1
Name: count, Length: 177, dtype: int64
```

```
hotel.reserved_room_type.value_counts()
```

```
reserved_room_type
A    85994
D    19201
E     6535
F     2897
G     2094
B     1118
C      932
H      601
P       12
L        6
Name: count, dtype: int64
```

```
hotel.assigned_room_type.value_counts()
```

```
assigned_room_type
A    74053
D    25322
E     7806
F     3751
G     2553
C     2375
B     2163
H      712
I      363
K      279
P       12
L        1
Name: count, dtype: int64
```

```
hotel.reservation_status.value_counts()
```

```
reservation_status
Check-Out    75166
Canceled     43017
No-Show       1207
Name: count, dtype: int64
```

```
hotel.is_canceled.value_counts()
```

```
is_canceled
0    75166
1    44224
Name: count, dtype: int64
```

After initial profiling of the data to look at null values, datatypes, and summary statistics, 12 variables were dropped due to irrelevance, too many null values, or that keeping them in the analysis made the model perform worse than if they were removed. After dropping those superfluous variables, the remaining null values were removed. Some advantages to removing variables with too much missing data and those variables which were irrelevant is that the model will perform better and it will predict only on data which we actually have instead of imputing missing data with the mean or median. One disadvantage is that there is a decrease in the variety of data which means that not all data can be captured and used in the analysis. The variable adr (average daily rate) contained values of $0.00 and below which were deemed as data errors. As a result, values of adr less than $1.00 were removed from the dataset. One advantage to doing this is that the model would be forced to predict on only real data values and one disadvantage is that this could lead to a worse performing model because there could be a missing interaction now as a result of removing the extremely low values.

The resulting dataset contained 24 variables and 117,424 rows of data. The categorical variables were encoded in preprocessing so that the model could interpret them. The advantage to encoding categorical variables is that more data is captured instead of dropping or improperly analyzing the categorical variable. One disadvantage to encoding categorical variables, namely through one-hot encoding, is that it increases the sparsity of the dataset by driving up the zero values. Prior to the encoding, an initial correlation test was run to identify correlations between independent variables and with the dependent variable. The dataset now contains 54 variables as a result of the encoding process. Another correlation test was run with a heatmap to visualize the correlations and identify possible relationships. One advantage to looking at the correlations is that possible relationships and influence can be identified, but one disadvantage to looking at correlation is that it doesn't necessarily mean that the relationship between those two variables is causal.

Dropping variables:

```
hotel=hotel.drop(['name','email','phone-number', 'country', 'agent',
                  'distribution_channel','credit_card', 'company',
                  'reserved_room_type','assigned_room_type',
                  'reservation_status_date','reservation_status'], axis=1)
```

```
hotel.describe()
```

	is_canceled	lead_time	arrival_date_year	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_week_nights	adu
count	119390.000000	119390.000000	119390.000000	119390.000000	119390.000000	119390.000000	119390.000000	119390.0000
mean	0.370416	104.011416	2016.156554	27.165173	15.798241	0.927599	2.500302	1.8564
std	0.482918	106.863097	0.707476	13.605138	8.780829	0.998613	1.908286	0.5792
min	0.000000	0.000000	2015.000000	1.000000	1.000000	0.000000	0.000000	0.0000
25%	0.000000	18.000000	2016.000000	16.000000	8.000000	0.000000	1.000000	2.0000
50%	0.000000	69.000000	2016.000000	28.000000	16.000000	1.000000	2.000000	2.0000
75%	1.000000	160.000000	2017.000000	38.000000	23.000000	2.000000	3.000000	2.0000
max	1.000000	737.000000	2017.000000	53.000000	31.000000	19.000000	50.000000	55.0000

Verifying that missing data has been dropped:

```python
hotel.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119390 entries, 0 to 119389
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   hotel                           119390 non-null  object
 1   is_canceled                     119390 non-null  int64
 2   lead_time                       119390 non-null  int64
 3   arrival_date_year               119390 non-null  int64
 4   arrival_date_month              119390 non-null  object
 5   arrival_date_week_number        119390 non-null  int64
 6   arrival_date_day_of_month       119390 non-null  int64
 7   stays_in_weekend_nights         119390 non-null  int64
 8   stays_in_week_nights            119390 non-null  int64
 9   adults                          119390 non-null  int64
 10  children                        119386 non-null  float64
 11  babies                          119390 non-null  int64
 12  meal                            119390 non-null  object
 13  market_segment                  119390 non-null  object
 14  is_repeated_guest               119390 non-null  int64
 15  previous_cancellations          119390 non-null  int64
 16  previous_bookings_not_canceled  119390 non-null  int64
 17  booking_changes                 119390 non-null  int64
 18  deposit_type                    119390 non-null  object
 19  days_in_waiting_list            119390 non-null  int64
 20  customer_type                   119390 non-null  object
 21  adr                             119390 non-null  float64
 22  required_car_parking_spaces     119390 non-null  int64
 23  total_of_special_requests       119390 non-null  int64
dtypes: float64(2), int64(16), object(6)
memory usage: 21.9+ MB
```

```python
# remove null values
hotel = hotel.dropna(how='any',axis=0)
```

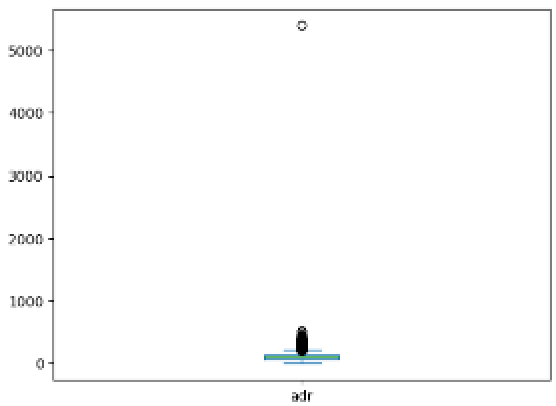
```python
# recheck for nulls and data types
hotel.info()
print(hotel.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 119386 entries, 0 to 119389
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   hotel                           119386 non-null  object
 1   is_canceled                     119386 non-null  int64
 2   lead_time                       119386 non-null  int64
 3   arrival_date_year               119386 non-null  int64
 4   arrival_date_month              119386 non-null  object
 5   arrival_date_week_number        119386 non-null  int64
 6   arrival_date_day_of_month       119386 non-null  int64
 7   stays_in_weekend_nights         119386 non-null  int64
 8   stays_in_week_nights            119386 non-null  int64
 9   adults                          119386 non-null  int64
 10  children                        119386 non-null  float64
 11  babies                          119386 non-null  int64
 12  meal                            119386 non-null  object
 13  market_segment                  119386 non-null  object
 14  is_repeated_guest               119386 non-null  int64
 15  previous_cancellations          119386 non-null  int64
 16  previous_bookings_not_canceled  119386 non-null  int64
 17  booking_changes                 119386 non-null  int64
 18  deposit_type                    119386 non-null  object
 19  days_in_waiting_list            119386 non-null  int64
 20  customer_type                   119386 non-null  object
 21  adr                             119386 non-null  float64
 22  required_car_parking_spaces     119386 non-null  int64
 23  total_of_special_requests       119386 non-null  int64
dtypes: float64(2), int64(16), object(6)
memory usage: 22.8+ MB
hotel                             0
is_canceled                       0
lead_time                         0
arrival_date_year                 0
arrival_date_month                0
arrival_date_week_number          0
arrival_date_day_of_month         0
stays_in_weekend_nights           0
stays_in_week_nights              0
adults                            0
children                          0
babies                            0
meal                              0
market_segment                    0
is_repeated_guest                 0
previous_cancellations            0
previous_bookings_not_canceled    0
booking_changes                   0
deposit_type                      0
days_in_waiting_list              0
customer_type                     0
adr                               0
required_car_parking_spaces       0
total_of_special_requests         0
dtype: int64
```

Identifying and removing adr values less than $1.00:

```
hotel.adr.plot(kind='box')
```

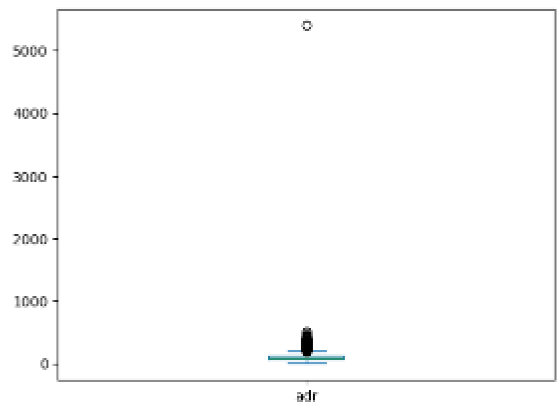
```
<Axes: >
```




```
# remove those with average daily rate of less than 1
hotel = hotel.loc[~(hotel['adr'] < 1)];

hotel.adr.plot(kind='box')
```

```
<Axes: >
```



Looking at frequencies of variables:

```
print(hotel.deposit_type.value_counts())
print(hotel.arrival_date_month.value_counts())
```

```
deposit_type
No Deposit    102675
Non Refund     14587
Refundable       162
Name: count, dtype: int64
arrival_date_month
August       13707
July         12491
May          11611
April        10953
October      10929
June         10819
September    10351
March         9640
February      7920
November      6641
December      6561
January       5801
Name: count, dtype: int64
```

Creating date variable to plot arrival dates by month and year:

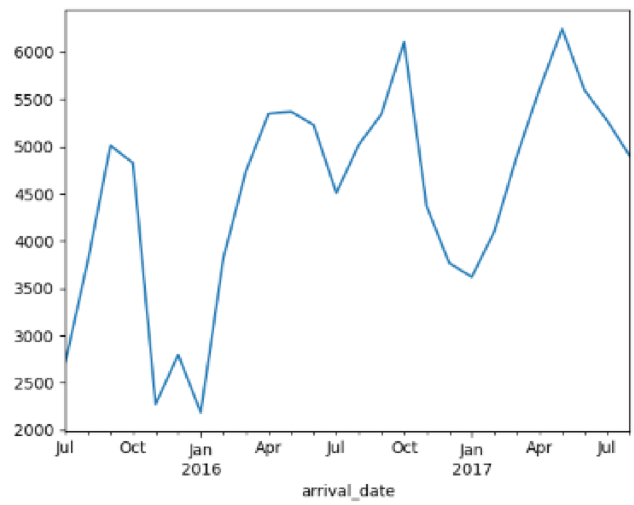
```
hotel['arrival_date'] = pd.to_datetime(hotel.arrival_date_year.astype(str) + '/' + hotel.arrival_date_month.astype(str) + '/01')
hotel.arrival_date.head()
```

```
2    2015-07-01
3    2015-07-01
4    2015-07-01
5    2015-07-01
6    2015-07-01
Name: arrival_date, dtype: datetime64[ns]
```

Plot arrival dates by month and year:

```
# Look at plot of arrival dates
date = hotel['arrival_date'].value_counts().sort_index()
date.plot()
```

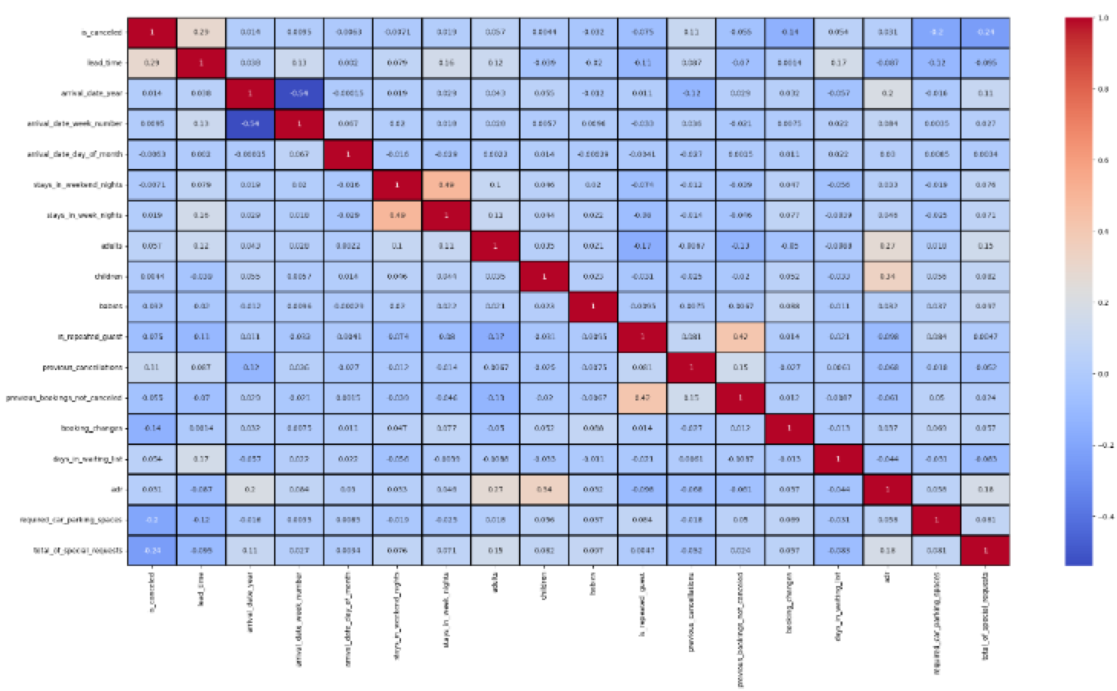
```
<Axes: xlabel='arrival_date'>
```



Testing for correlation between variables:

```
plt.figure(figsize=(28,14))
hotel_forcorr=hotel.select_dtypes(include=np.number)
hotel_corr=hotel_forcorr.corr()
sns.heatmap(hotel_corr,annot=True,linecolor='black', cmap='coolwarm', linewidths=2)
```

```
<Axes: >
```



Encoding categorical variables:

```
hotel = pd.get_dummies(hotel, columns=['deposit_type','arrival_date_month','customer_type','hotel','market_segment',
                                       'meal','arrival_date_year'], prefix='', prefix_sep='', dtype = 'int')
hotel
```

	is_canceled	lead_time	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_week_nights	adults	children	babies	is_repeate
2	0	7	27	1	0	1	1	0.0	0	
3	0	13	27	1	0	1	1	0.0	0	
4	0	14	27	1	0	2	2	0.0	0	
5	0	14	27	1	0	2	2	0.0	0	
6	0	0	27	1	0	2	2	0.0	0	
...	...	...	...	...	...	...	...	...	...	
119385	0	23	35	30	2	5	2	0.0	0	
119386	0	102	35	31	2	5	3	0.0	0	
119387	0	34	35	31	2	5	2	0.0	0	
119388	0	109	35	31	2	5	2	0.0	0	
119389	0	205	35	29	2	7	2	0.0	0	

117424 rows × 54 columns

```
hotel.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 117424 entries, 2 to 119389
Data columns (total 54 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   is_canceled                     117424 non-null  int64
 1   lead_time                       117424 non-null  int64
 2   arrival_date_week_number        117424 non-null  int64
 3   arrival_date_day_of_month       117424 non-null  int64
 4   stays_in_weekend_nights         117424 non-null  int64
 5   stays_in_week_nights            117424 non-null  int64
 6   adults                          117424 non-null  int64
 7   children                        117424 non-null  float64
 8   babies                          117424 non-null  int64
 9   is_repeated_guest               117424 non-null  int64
 10  previous_cancellations          117424 non-null  int64
 11  previous_bookings_not_canceled  117424 non-null  int64
 12  booking_changes                 117424 non-null  int64
 13  days_in_waiting_list            117424 non-null  int64
 14  adr                             117424 non-null  float64
 15  required_car_parking_spaces     117424 non-null  int64
 16  total_of_special_requests       117424 non-null  int64
 17  arrival_date                    117424 non-null  datetime64[ns]
 18  No Deposit                      117424 non-null  int32
 19  Non Refund                      117424 non-null  int32
 20  Refundable                      117424 non-null  int32
 21  April                           117424 non-null  int32
 22  August                          117424 non-null  int32
 23  December                        117424 non-null  int32
 24  February                        117424 non-null  int32
 25  January                         117424 non-null  int32
 26  July                            117424 non-null  int32
 27  June                            117424 non-null  int32
 28  March                           117424 non-null  int32
 29  May                             117424 non-null  int32
 30  November                        117424 non-null  int32
 31  October                         117424 non-null  int32
 32  September                       117424 non-null  int32
 33  Contract                        117424 non-null  int32
 34  Group                           117424 non-null  int32
 35  Transient                       117424 non-null  int32
 36  Transient-Party                 117424 non-null  int32
 37  City Hotel                      117424 non-null  int32
 38  Resort Hotel                    117424 non-null  int32
 39  Aviation                        117424 non-null  int32
 40  Complementary                   117424 non-null  int32
 41  Corporate                       117424 non-null  int32
 42  Direct                          117424 non-null  int32
 43  Groups                          117424 non-null  int32
 44  Offline TA/TO                   117424 non-null  int32
 45  Online TA                       117424 non-null  int32
 46  BB                              117424 non-null  int32
 47  FB                              117424 non-null  int32
 48  HB                              117424 non-null  int32
 49  SC                              117424 non-null  int32
 50  Undefined                       117424 non-null  int32
 51  2015                            117424 non-null  int32
 52  2016                            117424 non-null  int32
 53  2017                            117424 non-null  int32
dtypes: datetime64[ns](1), float64(2), int32(36), int64(15)
memory usage: 33.1 MB
```

```
hotel.describe()
```

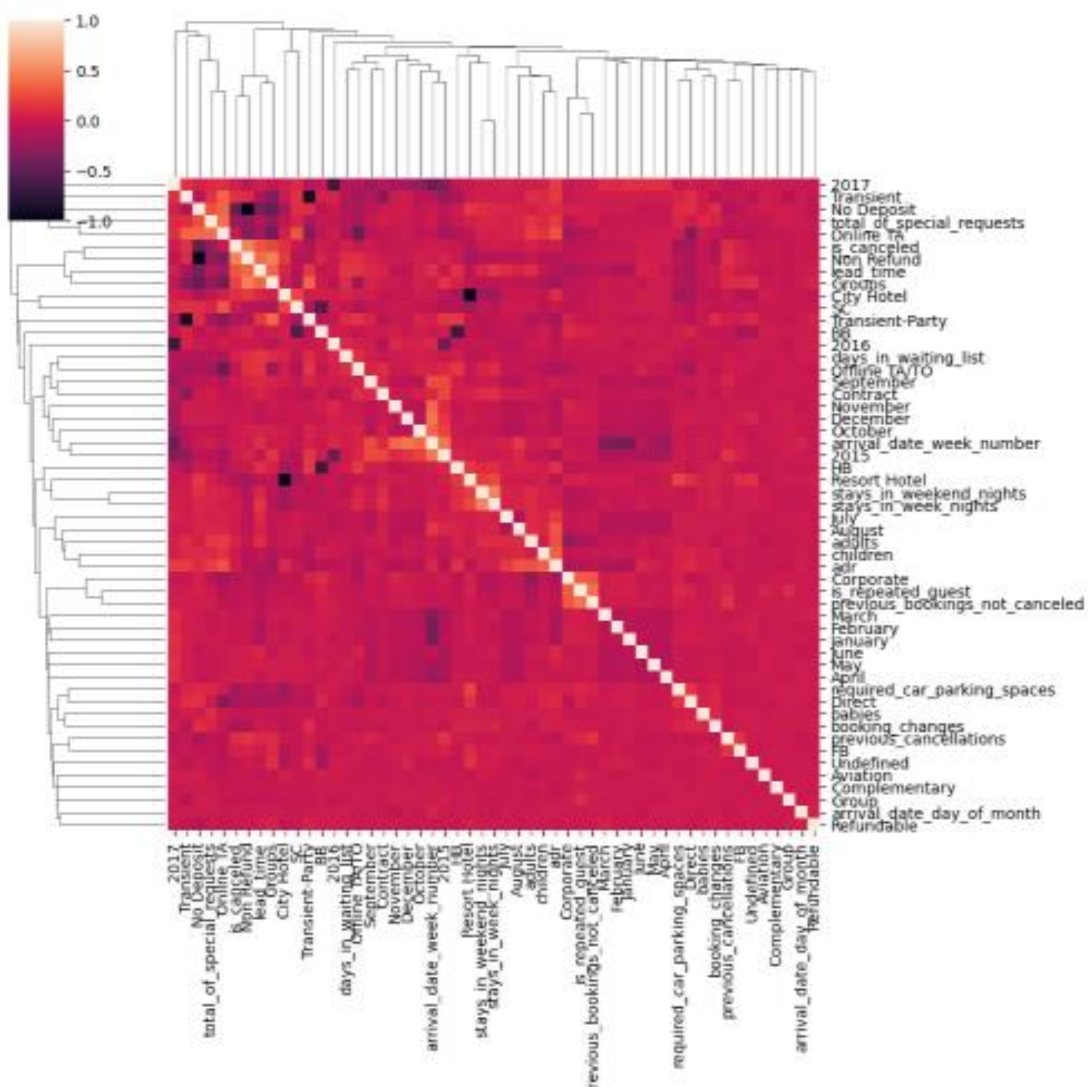
	is_canceled	lead_time	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_week_nights	adults	children
count	117424.000000	117424.000000	117424.000000	117424.000000	117424.000000	117424.000000	117424.000000	117424.000000
mean	0.374762	105.088611	27.136914	15.803192	0.936308	2.520984	1.860625	0.104510
min	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	19.000000	16.000000	8.000000	0.000000	1.000000	2.000000	0.000000
50%	0.000000	71.000000	27.000000	16.000000	1.000000	2.000000	2.000000	0.000000
75%	1.000000	162.000000	38.000000	23.000000	2.000000	3.000000	2.000000	0.000000
max	1.000000	709.000000	53.000000	31.000000	19.000000	50.000000	4.000000	10.000000
std	0.484063	106.907872	13.575787	8.783545	0.995209	1.892453	0.482095	0.399699

8 rows × 54 columns

Running final correlation test with all encoded variables included:

```
hotel_corr=hotel.select_dtypes(include=np.number)
corr=hotel_corr.corr()
sns.clustermap(data=corr, xticklabels=True, yticklabels=True)
```

```
<seaborn.matrix.ClusterGrid at 0x1f2ad9823d8>
```



Data Analysis

Prior to analyzing the data with the Random Forest Classifier, the target variable was removed from the X dataset, along with three other variables, which means that the final number of variables in the X dataset is 50. The y dataset dropped all variables except for the target variable. The data was split into 80% training data and 20% testing data using Scikit-Learn's TrainTestSplit.

```
# Select independent variables and dependent variable
X = hotel.drop(['is_canceled','arrival_date_week_number','arrival_date_day_of_month','arrival_date'], axis=1)

y = hotel['is_canceled']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=50)
```

I decided to run a preliminary training model with no tuning and compare that to a hyperparameter tuned training model before using Scikit-Learn's GridSearchCV function to identify the best hyperparameters for the model. The preliminary model had an accuracy of 99.02% and had precision, recall, and f1-scores of 0.99, except for recall of is_canceled equaling one where it had a score of 0.98. The GridSearchCV model was fit with a list of hyperparameters which showed the best results after several iterations of testing. For the parameter grid, only three values for n_estimators and max_depth and four values for max_features were used in order for the model take less time to run because increasing the number of values resulted in the model taking several hours to finish.

```
# run training model with no hyperparameter tuning
rf = RandomForestClassifier(random_state=50)

#fit model
rf_h=rf.fit(X_train, y_train)

y_train_pred=rf_h.predict(X_train)

model_accuracy = accuracy_score(y_train, y_train_pred)
print(f'Accuracy:', round(model_accuracy*100,2), '%')
print(classification_report(y_train, y_train_pred))
```

```
Accuracy: 99.02 %
              precision    recall  f1-score   support

           0       0.99      0.99      0.99     58778
           1       0.99      0.98      0.99     35161

    accuracy                           0.99     93939
   macro avg       0.99      0.99      0.99     93939
weighted avg       0.99      0.99      0.99     93939
```

The hyperparameter values chosen for testing in GridSearchCV were: n_estimators, 300, 500, 1000; max_features, 5, 10, 20, 40; and max_depth, 10, 20, 40. The advantage of using GridSearchCV is that many scenarios are tested and the best results are chosen, which makes the process of testing different hyperparameter combinations more efficient. The disadvantages of using GridSearchCV is that larger datasets and higher numbers of combinations increase the computational power needed thereby increasing the amount of time that the process will take and that if a particular combination is not specified then it will not be tested (Lee, 2023).

```
# try parameter tuning now
# Defining parameter range
params = {"n_estimators": [300,500,1000],
          "max_features": [5,10,20,40],
          "max_depth": [10,20,40]
          }
rf = RandomForestClassifier(random_state=50)

rf_grid = GridSearchCV(estimator = rf,
                       param_grid= params,
                       cv = 3,
                       scoring = None,
                       n_jobs= -1,
                       verbose = 1,
                       error_score='raise')

#Fitting the model for grid search
rf_grid_search = rf_grid.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
```

The best hyperparameters chosen by GridSearchCV were 300 for n_estimators, 10 for max_features, and 40 for max_depth. Using these parameters on the training model resulted in an accuracy of 99.02% and the same precision, recall, and f-1 scores as the preliminary model. This shows that while the hyperparameters didn't add any value to accuracy, precision, recall, or f-1 scores, it also didn't decrease those values.

```
# check best parameters and scores
print('The best parameters for the RF model are: ')
print(rf_grid_search.best_params_)

y_train_pred=rf_grid_search.predict(X_train)

model_accuracy = accuracy_score(y_train, y_train_pred)
print(f'Accuracy:', round(model_accuracy*100,2), '%')
print(classification_report(y_train, y_train_pred))
```

```
The best parameters for the RF model are:
{'max_depth': 40, 'max_features': 10, 'n_estimators': 300}
Accuracy: 99.02 %
              precision    recall  f1-score   support

           0       0.99      0.99      0.99     58778
           1       0.99      0.98      0.99     35161

    accuracy                           0.99     93939
   macro avg       0.99      0.99      0.99     93939
weighted avg       0.99      0.99      0.99     93939
```

The best hyperparameters were used to create the final Random Forest Classifier model using the training data to predict on the testing data. The advantages to using a Random Forest Classifier are that it is robust to outliers and overfitting, doesn't require scaled data, can utilize both continuous and categorical data (Petkovic et al., 2018), has shown superior accuracy against other classification models such as Logistic Regression (Schonlau & Zou, 2020) and Decision Trees (Cutler et al., 2011), can be tuned through various hyperparameters, and its ability to handle large datasets. Some disadvantages of Random Forest Classifiers are that it can be computationally expensive with large datasets or complex models and they tend to bias predictions towards the majority class when using imbalanced datasets (Petkovic et al., 2018).

Data Summary and Implications

The final model was evaluated using the following metrics: accuracy, the ratio of accurate predictions to all predictions calculated; precision, the ratio of true positive predictions out of all positive predictions (true and false positives); recall, the proportion of true positives out of all actual positive predictions (true positives and false negatives); f-1 score, a measure which accounts for precision and recall; and Receiver Operating Characteristic Area Under the Curve (ROC AUC), which shows how well the model performs at predicting into correct categories (Accuracy, Precision, Recall, and F1-Score - Machine Learning Tutorials, Courses and Certifications, 2025). These metrics are the standard for evaluating classification models (Naidu, G., et al., 2023).

```
# fit RF with new best parameters
rfc = RandomForestClassifier(random_state=50,
                             n_estimators=300,
                             max_features=10,
                             max_depth=40)
#fit model
rfc_h=rfc.fit(X_train, y_train)

y_pred=rfc_h.predict(X_test)

model_accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy:', round(model_accuracy*100,2), '%')
print(classification_report(y_test, y_pred))
```

```
Accuracy: 86.63 %
              precision    recall  f1-score   support

           0       0.87      0.92      0.90     14640
           1       0.86      0.77      0.81      8845

    accuracy                           0.87     23485
   macro avg       0.86      0.85      0.85     23485
weighted avg       0.87      0.87      0.86     23485
```

```
print(confusion_matrix(y_test, y_pred))
```

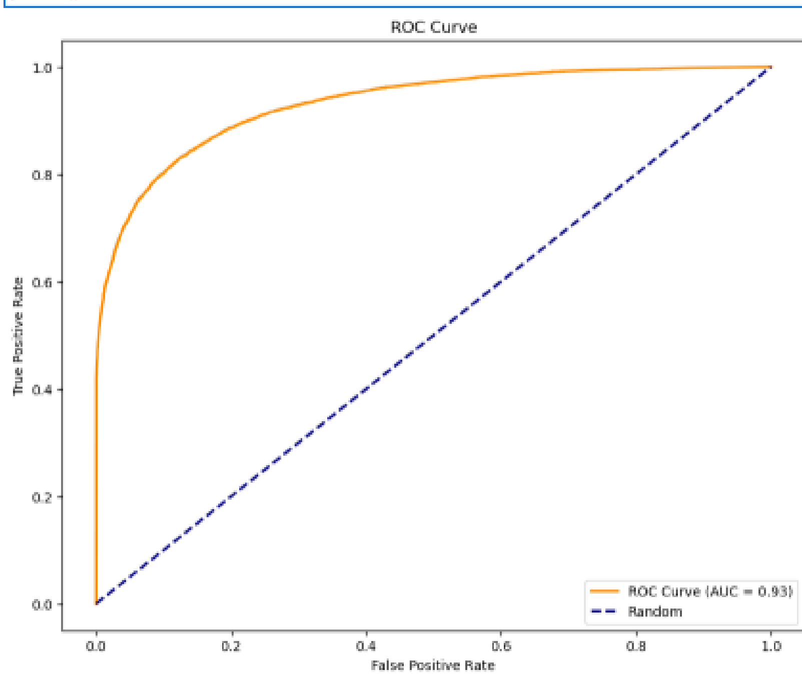
```
[[13523  1117]
 [ 2024  6821]]
```

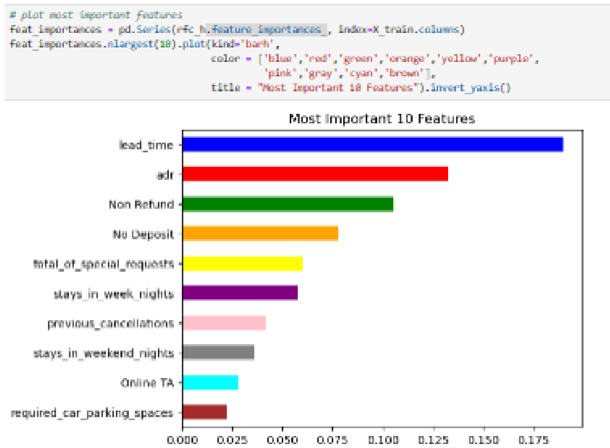
The result of the final model was an accuracy of 86.63% with all precision, recall, and f-1 scores above 0.80, except for recall of the 1 value for ‘is_canceled’, which was 0.77. In other words, the model predicted the correct result against the test set almost 87% of the time, the model mitigated false positive predictions for both cancelled and uncancelled reservations (precision: 0.86 and 0.87, respectively), avoided making false negative predictions (recall: 0.77 and 0.92, respectively), and had a high f-1 score indicating good overall performance (0.81 and 0.90, respectively).

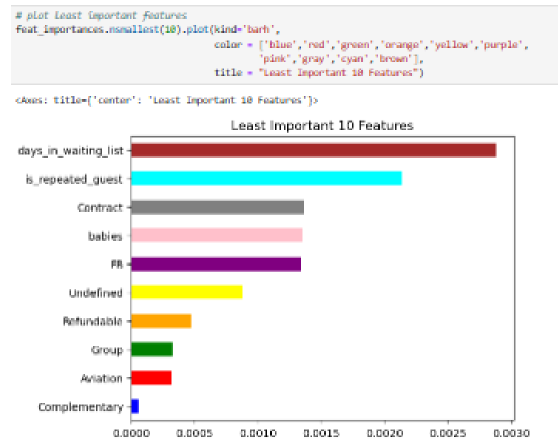
Additionally, I tested and plotted the ROC AUC of the model and identified the ten most and least influential variables on the predictive accuracy of the model. The ROC AUC indicated that the model captured 93% of the data under the curve, which shows that this is a high performing classification model. The code used to create the ROC AUC diagram was sourced from https://www.geeksforgeeks.org/how-to-plot-roc-curve-in-python/.

```
# Plot ROC Curve
y_prob = rfc_h.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(10, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2,
         label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```







The Random Forest Classifier was shown to be an accurate and helpful tool in predicting the cancellation of hotel reservations, but there are limitations and areas which can be improved upon, as well as recommendations for future usage and refinement. One limitation is that there appears to be some overfitting in the model because the accuracy, precision, recall, and f-1 scores are all much higher for the training model than the testing model (accuracy: 99% and 87%, respectively). The overfitting could be lessened by determining better hyperparameter tuning, trying different train/test split ratios, or including more data, which would hopefully give more information for the model to be trained on, thus increasing predictive capabilities. My recommended course of action and recommendations for further study would be to expand the dataset to include more than only two years of data, examine other possibilities of hyperparameters, and identify if any other machine learning models could perform at a higher level with this dataset than the Random Forest Classifier.

References

Accuracy, Precision, Recall, and F1-Score - Machine Learning Tutorials, Courses and Certifications. (2025). *Machine Learning Tutorials, Courses and Certifications.* https://machinelearning.org.in/accuracy-precision-recall-and-f1-score/

Cutler, A., Cutler, D., & Stevens, J. (2011). Random Forests. *ResearchGate.* https://www.researchgate.net/publication/236952762_Random_Forests

GeeksForGeeks. (2024). How to plot ROC curve in Python. *GeeksforGeeks*. https://www.geeksforgeeks.org/how-to-plot-roc-curve-in-python/

Lee, D. E. (2023). Hyperparameter Optimization: Grid Search vs. Random Search vs. Bayesian Optimization in Action. *Medium.* https://drlee.io/hyperparameter-optimization-grid-search-vs-random-search-vs-bayesian-optimization-in-action-106f99b94e32

Mojtaba (2019). Hotel Booking. *Kaggle*. https://www.kaggle.com/datasets/mojtaba142/hotel-booking/data

Naidu, G., Zuva, T., & Sibanda, E.M. (2023). A Review of Evaluation Metrics in Machine Learning Algorithms. In: Silhavy, R., Silhavy, P. (eds) Artificial Intelligence Application in Networks and Systems. CSOC 2023. Lecture Notes in Networks and Systems, vol 724. *Springer, Cham*. https://doi.org/10.1007/978-3-031-35314-7_2

Petkovic, D., Altman, R., Wong, M., & Vigil, A. (2018). Improving the explainability of Random Forest classifier – user centered approach. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing, 23, 204.* https://pmc.ncbi.nlm.nih.gov/articles/PMC5728671/