Final Project Report:
Working with the Hotel-Booking Demand Dataset

Juana Tavera
April 27, 2020

Phase I: Business Understanding

When looking through datasets that could be potential subjects for this project, the one that caught my eye was 'Hotel Booking Demand'. With this dataset, I was able to come up with a business problem that was simple yet challenging. I created a hotel chain, which has an abundance of hotels around Europe. This hotel chain called, Knights Inn, is looking to predict if a customer is likely to cancel according to some select features. Therefore the business problem that was derived from the hotel-booking dataset deals with cancellations and the target variable is 'is_canceled'.

Business Problem

As mentioned above the business problem, that the Knight Inn hotel chain, is currently faced with predicting cancellations to better plan for things like personel and food requirements.

Dataset

This dataset, according to Jesse Mostipak, contains booking information for various city and resort hotels. The booking information includes the length of stay, arrival date, reservation status etc. The hotel booking demand dataset contains 32 columns and is considered to be a small dataset.

[Link to dataset]

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

Proposed Analytics Solution

My proposed analytic solution is to build several models, such as ones that use information-based learning, that will focus on features that have the most effect on the target variable. Those models will then be tested and trained to give the best possible solution to Knights Inn.

Phase II: Data Exploration and Preprocessing

As mentioned above the hotel-booking demand dataset consists of 32 columns. The columns are:

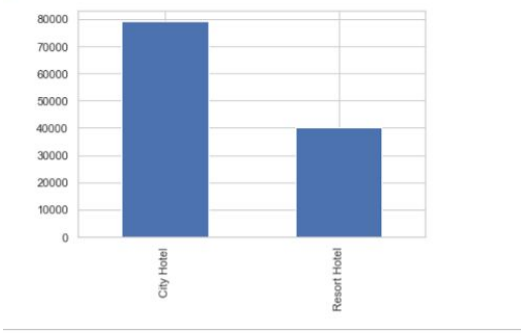
- hotel: H1 = Resort Hotel and H2 = City Hotel
- is_canceled: values indicating if the booking was canceled(1) or not (0)

- lead_time : number of days that elapsed between the entering date of the booking into the PMS and arrival date
- arrival_date_year: year of the arrival date
- arrival_date_month: month of the arrival date
- arrival_date_week_number: week number of the year for arrival date
- arrival_date_day_of_month: day of arrival date
- stays_in_weekend_nights: number of weekend nights (Saturday or Sunday) the guest stayed or booked to stay at the hotel
- stays_in_week_nights: number of week nights (Monday thru Friday) the guest stayed or booked to stay at the hotel
- adults: number of adults
- children: number of children
- babies: number of babies
- meal: type of meal booked
- country: country of origin
- market_segment: market segment designation
- distribution_channel: booking distribution channel
- is_repeated_guest: value indicating if the booking name was from a repeated guest (1) or not (0)
- previous_cancellations: number of previous booking that were cancelled by the customer prior to the current booking
- previous_bookings_not_canceled: number of previous booking not cancelled by the customer prior to the current booking
- reserved_room_type: code of room type reserved
- assigned_room_type: code for the type of room
- booking_changes: number of changes/amendments made to the booking from the moment the booking was entered on the PMS
- deposit_type: indication on if the customer made a deposit to guarantee the booking
- agent: ID of travel agency that made the booking
- company: ID of the company/entity that made the booking responsible for paying the booking
- days_in_waiting_list: number of days the booking was in the waiting list before it was confirmed to the customer
- customer_type: type of booking
- adr: average daily rate
- required_car_parking_spaces: number of car spaces required by the customer
- total_of_special_requests: number of special requests made by the customer
- reservation_status: reservation last status, assuming one of three categories: canceled, check-out, no-show

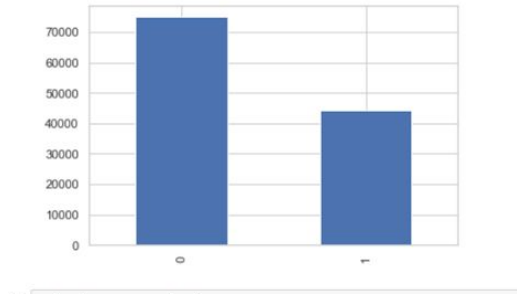
- reservation_status_date: date at which the last status was set

Bar Charts:

hotel:



is_canceled:



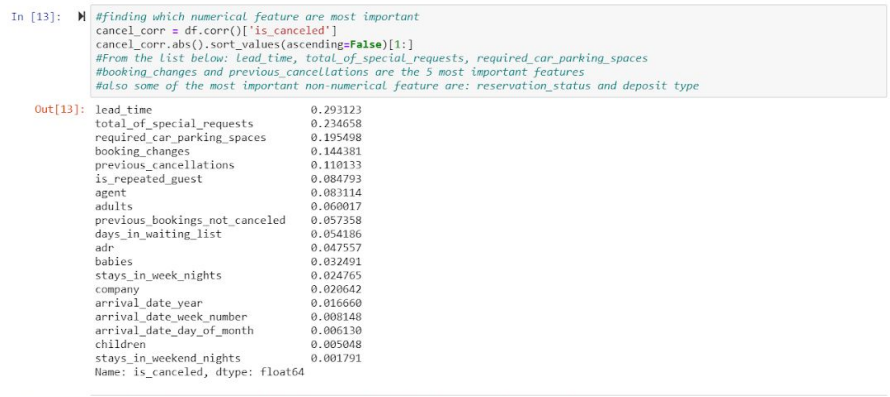
Finding the resort hotel cancelations, city hotel cancelations, and total cancelations

```
#absolute cancelations
total_cancelations = df_update['is_canceled'].sum()
#resort hotel cancelations
rh_cancelations = df_update.loc[df_update['hotel'] == 'Resort Hotel']['is_canceled'].sum()
#city hotel cancelations
ch_cancelations = df_update.loc[df_update['hotel'] == 'City Hotel']['is_canceled'].sum()

print('Total bookings canceled:' ,total_cancelations)
print('Resort hotel bookings canceled: ',rh_cancelations)
print('City hotel booking canceled: ', ch_cancelations)
```

```
Total bookings canceled: 44199
Resort hotel bookings canceled:  11120
City hotel booking canceled:  33079
```

Finding which numerical features are the most important to target variable



```
In [13]: #finding which numerical feature are most important
         cancel_corr = df.corr()['is_canceled']
         cancel_corr.abs().sort_values(ascending=False)[1:]
         #From the list below: lead_time, total_of_special_requests, required_car_parking_spaces
         #booking_changes and previous_cancellations are the 5 most important features
         #also some of the most important non-numerical feature are: reservation_status and deposit type

Out[13]: lead_time                        0.293123
         total_of_special_requests        0.234658
         required_car_parking_spaces      0.195498
         booking_changes                  0.144381
         previous_cancellations           0.110133
         is_repeated_guest                0.084793
         agent                            0.083114
         adults                           0.060017
         previous_bookings_not_canceled   0.057358
         days_in_waiting_list             0.054186
         adr                              0.047557
         babies                           0.032491
         stays_in_week_nights             0.024765
         company                          0.020642
         arrival_date_year                0.016660
         arrival_date_week_number         0.008148
         arrival_date_day_of_month        0.006130
         children                         0.005048
         stays_in_weekend_nights          0.001791
         Name: is_canceled, dtype: float64
```

As shown from the screenshot above the most important numerical features are lead_time, total_of_special_requests, required_car_parking_spaces, booking_changes and previous_cancellations. While the most informative categorical features are reservations_status and deposit_type. Therefore those categorical features will not be included in the ABT.

Selecting features for the Analytic Base Table to use for models

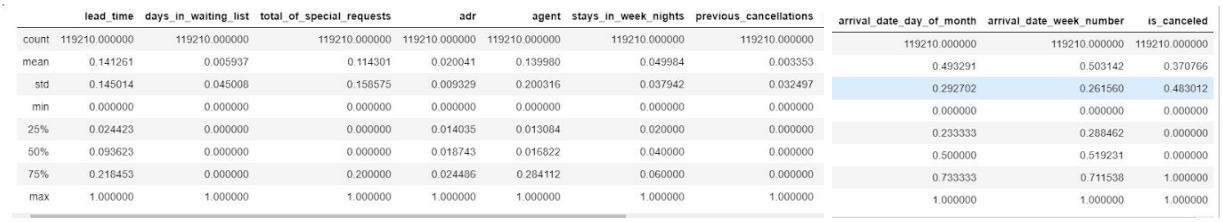
The features that I decided to pick for the analytic base table are mostly numerical features. The features are: lead_time, days_in_waiting_list, total_of_special_requests, adr, agent, stays_in_week_nights, previous_cancellations, arrival_date_day_of_month, and arrival_date_week_number. The target variable, is_canceled, is also included in the ABT.

Data Quality Report

Feature	Amount (after handling missing values)	Data Type (after normalization)	Data Scale
lead_time	119210	float64	ratio
days_in_waiting_list	119210	float64	ratio
total_of_special_requests	119210	float64	ratio
adr	119210	float64	ratio
agent	119210	float64	nominal
stays_in_week_nights	119210	float64	ratio
previous_cancellations	119210	float64	ratio
arrival_date_day_of_month	119210	float64	ordinal

arrival_date_week_number	119210	float64	ordinal

Target variable: is_canceled

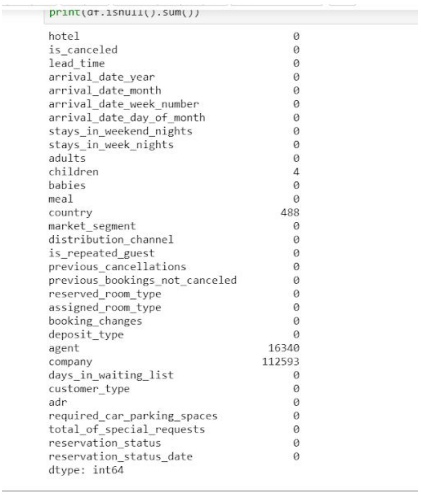


	lead_time	days_in_waiting_list	total_of_special_requests	adr	agent	stays_in_week_nights	previous_cancellations	arrival_date_day_of_month	arrival_date_week_number	is_canceled
count	119210.000000	119210.000000	119210.000000	119210.000000	119210.000000	119210.000000	119210.000000	119210.000000	119210.000000	119210.000000
mean	0.141261	0.005937	0.114301	0.020041	0.139980	0.049984	0.003353	0.493291	0.503142	0.370766
std	0.145014	0.045008	0.158575	0.009329	0.200316	0.037942	0.032497	0.292702	0.261560	0.483012
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.024423	0.000000	0.000000	0.014035	0.013084	0.020000	0.000000	0.233333	0.288462	0.000000
50%	0.093623	0.000000	0.000000	0.018743	0.016822	0.040000	0.000000	0.500000	0.519231	0.000000
75%	0.218453	0.000000	0.200000	0.024486	0.284112	0.060000	0.000000	0.733333	0.711538	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

The chart above contains the count, mean, std, min, 25%, 50%, 75%, and max for the features that I selected and the target variable.

Missing Values and Outliers

The way I handled missing values was by first checking for them by the line: print(df.isnull().sum()). What I found was that the following columns had missing values: children, country, agent, and company.



```
print(df.isnull().sum())

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                        4
babies                          0
meal                            0
country                       488
market_segment                  0
distribution_channel            0
is_repeated_guest               0
previous_cancellations          0
previous_bookings_not_canceled  0
reserved_room_type              0
assigned_room_type              0
booking_changes                 0
deposit_type                    0
agent                       16340
company                    112593
days_in_waiting_list            0
customer_type                   0
adr                             0
required_car_parking_spaces     0
total_of_special_requests       0
reservation_status              0
reservation_status_date         0
dtype: int64
```

As shown in the screenshot above children have 4 missing values, country has 488 missing values, agent has 16,340, and company has 112,593 missing values. I then set default values for the columns. The default values are as follows: children - 0.0, country - Unknown, agent - 0.0, company - 0.

```
In [9]:
#Replacing missing values: for agent, company, country, and children
#setting default values
missingVal_replacement = {'children': 0.0, 'country': 'Unknown', 'agent': 0, 'company': 0}
df_update = df.fillna(missingVal_replacement)

#meal contains values 'Undefined', which is equal to SC (no meal)
df_update['meal'].replace('Undefined', 'SC', inplace=True)

#some rows in the dataset has 0 adults, 0 children, and 0 babies; therefore will be dropped
zero_guests = list(df_update.loc[df_update['adults']+ df_update['children']+df_update['babies']==0].index)
df_update.drop(df_update.index[zero_guests], inplace=True)

In [10]:
#dataset before update
print("Dataset before update: ", df.shape)
#dataset after update
print('Dataset after update: ',df_update.shape)

Dataset before update:  (119390, 32)
Dataset after update:  (119210, 32)
```

In the screenshot above, I also displayed the shape of the dataset before and after the update and removed the rows which had zero guests.

Handling Outliers:

```
In [33]:
#handling outliers for numerical features that aren't binary, IDs, or a date
def outlier_treatment(x, lower, upper):
    if (lower >= x) or (x >= upper):
        return 'Yes'
    else:
        return 'No'

columns_to_check = ['lead_time','total_of_special_requests','required_car_parking_spaces','booking_changes','previous_cancel
                    'adults','previous_bookings_not_canceled', 'days_in_waiting_list','adr','babies','stays_in_week_nights',
                    'children', 'stays_in_weekend_nights']

for col in columns_to_check:
    Q1,Q3 = np.percentile(df[col],[25,75])
    IQR = Q3 - Q1

    lower_bound = Q1 - (1.5 * IQR)
    upper_bound = Q3 + (1.5 * IQR)

    df['outlier'] = df[col].apply(lambda x: outlier_treatment(x, lower_bound,upper_bound))
    print(f"The lower and upper bound of the range for '{col}' respectively is: {lower_bound} and {upper_bound}")
```

```
The lower and upper bound of the range for 'lead_time' respectively is: -195.0 and 373.0
The lower and upper bound of the range for 'total_of_special_requests' respectively is: -1.5 and 2.5
The lower and upper bound of the range for 'required_car_parking_spaces' respectively is: 0.0 and 0.0
The lower and upper bound of the range for 'booking_changes' respectively is: 0.0 and 0.0
The lower and upper bound of the range for 'previous_cancellations' respectively is: 0.0 and 0.0

C:\Users\tvrju\Anaconda3\lib\site-packages\numpy\lib\function_base.py:3826: RuntimeWarning: Invalid value encountered in
percentile
  interpolation=interpolation)

The lower and upper bound of the range for 'agent' respectively is: nan and nan
The lower and upper bound of the range for 'adults' respectively is: 2.0 and 2.0
The lower and upper bound of the range for 'previous_bookings_not_canceled' respectively is: 0.0 and 0.0
The lower and upper bound of the range for 'days_in_waiting_list' respectively is: 0.0 and 0.0
```

```
The lower and upper bound of the range for 'adr' respectively is: -15.774999999999991 and 211.065
The lower and upper bound of the range for 'babies' respectively is: 0.0 and 0.0
The lower and upper bound of the range for 'stays_in_week_nights' respectively is: -2.0 and 6.0
The lower and upper bound of the range for 'company' respectively is: nan and nan
The lower and upper bound of the range for 'arrival_date_year' respectively is: 2014.5 and 2018.5
The lower and upper bound of the range for 'children' respectively is: nan and nan
The lower and upper bound of the range for 'stays_in_weekend_nights' respectively is: -3.0 and 5.0
```

Normalization

Normalization is used to keep columns/attributes to keep bias to a minimum. In the normalization that I performed was to keep all values between 0 and 1.

```
In [14]:
#numerical feature to normalize that have significant impact on is_canceled
from sklearn import preprocessing
df_normalize = df_update._get_numeric_data()
df_concise = df_normalize[['lead_time', 'days_in_waiting_list','total_of_special_requests', 'adr','agent',
                           'stays_in_week_nights','previous_cancellations','arrival_date_day_of_month',
                           'arrival_date_week_number','is_canceled']]

df_minmax = df_concise
print(df_minmax.head())
min_max_scaler = preprocessing.MinMaxScaler()

a_scaled = min_max_scaler.fit_transform(df_minmax)

df_scaled = pd.DataFrame(data=a_scaled, columns = df_minmax.columns)
df_scaled.head()
```

Out[14]:

	lead_time	days_in_waiting_list	total_of_special_requests	adr	agent	stays_in_week_nights	previous_cancellations	arrival_date_day_of_month	arrival_date_week_number	is_canceled
0	0.464043	0.0	0.0	0.001180	0.000000	0.00	0.0	0.0	0.5	0.0
1	1.000000	0.0	0.0	0.001180	0.000000	0.00	0.0	0.0	0.5	0.0
2	0.009498	0.0	0.0	0.015053	0.000000	0.02	0.0	0.0	0.5	0.0
3	0.017639	0.0	0.0	0.015053	0.568224	0.02	0.0	0.0	0.5	0.0
4	0.018996	0.0	0.2	0.019307	0.448598	0.04	0.0	0.0	0.5	0.0

Phase III: Model Selection and Evaluation

The models that were selected for the ABT consisting of the aforementioned features and the target variable - is_canceled are Decision Tree, Logistic Regression, and Random Forest. These models used a train dataset and test dataset. As shown below:

```
In [79]:
#testing/training split
from sklearn.model_selection import train_test_split

df = hotel
X = df[['lead_time','days_in_waiting_list','total_of_special_requests','adr','agent','stays_in_week_nights','previous_cancel
       'arrival_date_week_number']].values
y = df['is_canceled'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=0)
```

```
In [80]:
df1 = pd.DataFrame(data=X_train)
print(df1.head())
print(df1.shape)
df2 = pd.DataFrame(data=y_test)
print(df2.head())
print(df2.shape)
```

```
          0    1    2         3         4     5    6         7         8
0  0.050204  0.0  0.0  0.021526  0.706542  0.04  0.0  0.466667  0.192308
1  0.092266  0.0  0.0  0.014868  0.000000  0.04  0.0  0.466667  0.192308
2  0.248304  0.0  0.0  0.011261  0.557009  0.02  0.0  0.500000  0.288462
3  0.000000  0.0  0.0  0.001180  0.000000  0.02  0.0  0.200000  0.096154
4  0.111262  0.0  0.4  0.001180  0.448598  0.00  0.0  0.033333  0.923077
(79870, 9)
     0
0  1.0
1  0.0
2  0.0
3  0.0
4  1.0
(39340, 1)
```

I also displayed the datasets at the bottom.

Evaluation Metrics

One of the evaluation metrics that was used was classification accuracy which I displayed after performing the Logistic Regression model. I also displayed the confusion matrix for logistic regression and the training and testing score for Decision Tree and Random Forest.

```
In [42]:
#Getting the testing and training score
score = dt.score(X_test, y_test)
print('Testing score: ', score)
score = dt.score(X_train, y_train)
print('Training scoreL ', score)

Testing score:  0.7632689374682258
Training scoreL  0.7684988105671716
```

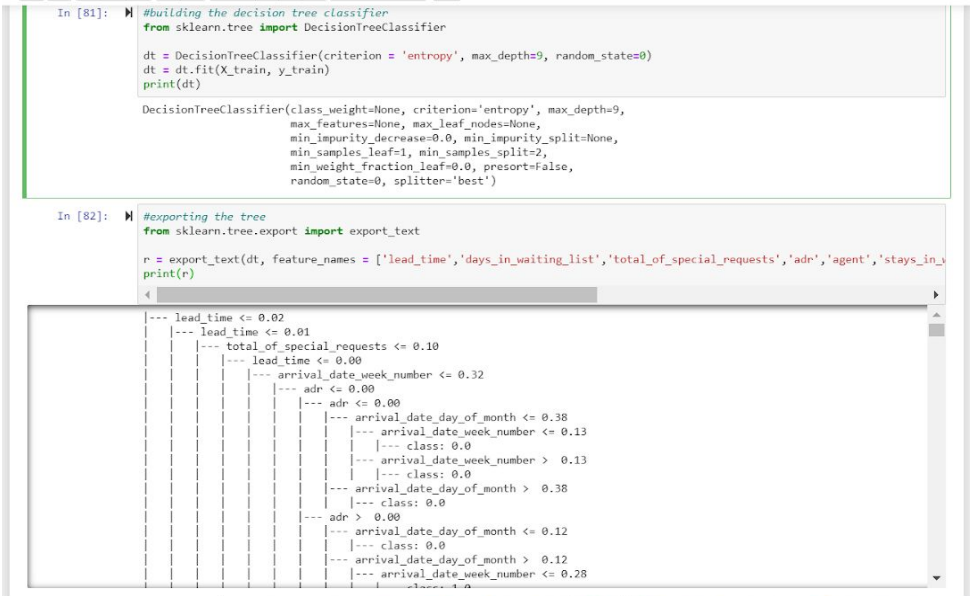
Shown above in the screenshot is the testing and training score for Decision Tree.

Models

As mentioned before the models that were built for this dataset was Decision Tree, Logistic Regression, and Random Forest. I will one by one go over each model.

Decision Tree

The decision tree consists of: a root node, interior nodes, and leaf nodes. The goal of a decision tree is to pick the features which have the most information gain to obtain a shallow tree.



The above screenshot showcases the code used to build the Decision Tree and a snippet of the decision tree that was outputted.

Logistic Regression

Logistic regression is used when the dependent variable (target) is categorical. Although is_canceled has values 0.0 and 1.0. These values represent categorical values: canceled and not canceled.

```
In [54]:
#building logistic regression model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score

log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

C:\Users\tvrju\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver will be chang
ed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)

Out[54]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                            intercept_scaling=1, l1_ratio=None, max_iter=100,
                            multi_class='warn', n_jobs=None, penalty='l2',
                            random_state=None, solver='warn', tol=0.0001, verbose=0,
                            warm_start=False)

In [88]:
#making predictions using the LR model
y_pred = log_reg.predict(X_test)
```

This screenshot above has the code that was used to build the logistic regression model.

Random Forest

Random forests are an ensemble of learning methods for classification, regression, etc. The random forest model uses estimators which is a number of decision tree classifiers on various sub-samples of the dataset.

```
#building random forest classifer
df = hotel

from sklearn.ensemble import RandomForestClassifier
feat_labels = df.columns[:9]
X = df[feat_labels]
y = df['is_canceled']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=0)

forest = RandomForestClassifier(n_estimators=1000, random_state=0, n_jobs=-1)
forest.fit(X_train, y_train)
print("Training accuracy:", forest.score(X_train, y_train))
print("Testing accuracy:", forest.score(X_test, y_test))

importances = forest.feature_importances_
for f in range(X_train.shape[1]):
    print('Feature: ', feat_labels[f], '\tImportance: ', importances[f])
```

This screenshot above has the code that was used to build the random forest model.

Sampling and Evaluation Settings

As mentioned above, the models used two datasets - a train and test one. These datasets were split by 0.33 for all the models - decision tree, logistic regression, and random forest.

Decision Tree

In the decision tree the criterion is entropy, the max_depth is 9 (same amount of columns that I have), and random_state is set to 0.

Logistic Regression

Logistic regression does not have any special evaluation settings other than default that is provided from sklearn.linear_model import LogisticRegression.

Random Forest

In the random forest model it looks through all the columns (other than the target variable) and has n_estimators set to 1000, random_state set to 0, and n_jobs set to -1.

Evaluation

Decision Tree

For the decision tree model, I obtained the testing score and training score for the tree whose max depth is 9.

```
#Getting the testing and training score
score = dt.score(X_test, y_test)
print('Testing score: ', score)
score = dt.score(X_train, y_train)
print('Training score ', score)
```

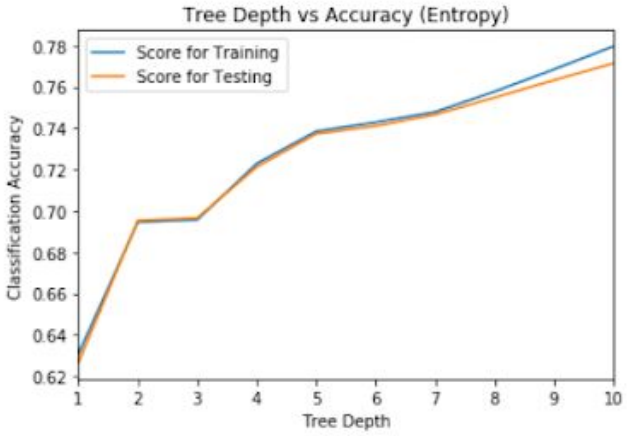
```
Testing score:  0.7632689374682258
Training score  0.7684988105671716
```

I also obtained the training and testing scores for entropy and visualized it.

```
#getting the training and testing scores for entropy
resultsEntropy = pd.DataFrame(columns = ['LevelLimit', 'Score for Training', 'Score for Testing'])
for treeDepth in range (1, 11):
    dt_j = DecisionTreeClassifier(criterion='entropy', max_depth = treeDepth, random_state=0)
    dt_j = dt_j.fit(X_train, y_train)
    dt_j.predict(X_test)
    scoreTrain = dt_j.score(X_train, y_train)
    scoreTest = dt_j.score(X_test, y_test)
    resultsEntropy.loc[treeDepth] = [treeDepth, scoreTrain, scoreTest]

print(resultsEntropy)
```

	LevelLimit	Score for Training	Score for Testing
1	1.0	0.630612	0.626436
2	2.0	0.694541	0.695323
3	3.0	0.695680	0.696594
4	4.0	0.722900	0.721251
5	5.0	0.738488	0.737265
6	6.0	0.742832	0.741052
7	7.0	0.747865	0.746568
8	8.0	0.757731	0.754830
9	9.0	0.768499	0.763269
10	10.0	0.779629	0.771403



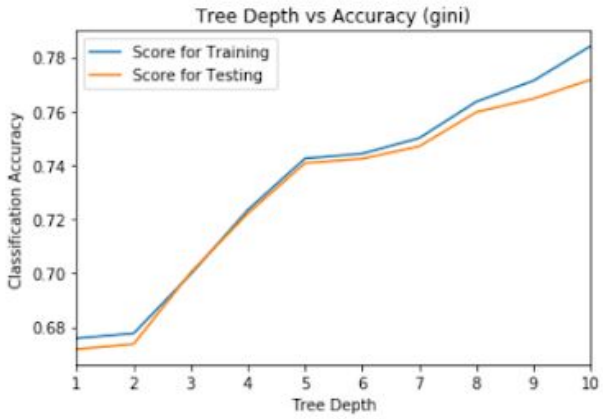
As shown by the table and graph above the further down the tree went the higher the score and accuracy.

Finally I obtained the training and testing scores for gini and visualized it.

```
#getting the training and testing scores for gini index
resultsGini = pd.DataFrame(columns = ['LevelLimit', 'Score for Training', 'Score for Testing'])
for treeDepth in range (1, 11):
    dt_j = DecisionTreeClassifier(criterion='gini', max_depth = treeDepth, random_state=0)
    dt_j = dt_j.fit(X_train, y_train)
    dt_j.predict(X_test)
    scoreTrain = dt_j.score(X_train, y_train)
    scoreTest = dt_j.score(X_test, y_test)
    resultsGini.loc[treeDepth] = [treeDepth, scoreTrain, scoreTest]

print(resultsGini)
```

```
    LevelLimit  Score for Training  Score for Testing
1          1.0            0.675886           0.671810
2          2.0            0.677664           0.673742
3          3.0            0.699649           0.700280
4          4.0            0.723488           0.722166
5          5.0            0.742544           0.740798
6          6.0            0.744310           0.742425
7          7.0            0.750081           0.747026
8          8.0            0.763591           0.759710
9          9.0            0.771353           0.764692
10        10.0            0.784212           0.771708
```



As shown in the table and graph above, the further down the tree the higher the scores and accuracy. The same as entropy; however, gini seems to have gotten slightly better scores at the last level of the tree.

Logistic Regression

For the logistic regression model, I created a confusion matrix which showed me the results of how well or bad the model did in predicting the target variable, is_canceled.

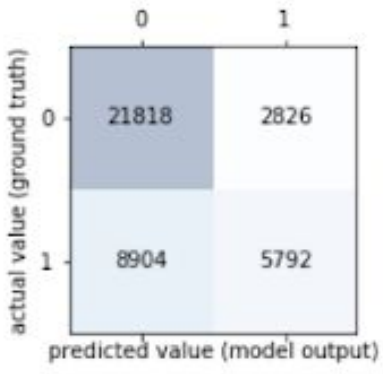
```
#making predictions using the LR model
y_pred = log_reg.predict(X_test)
```

```
#confusion matrix
cf = confusion_matrix(y_test, y_pred)
print('\n', cf)
#According to the confusion matrix: we have 21,818 TP and 2,826 FP
#8,904 FN and 5,792 TN
```

```
[[21818  2826]
 [ 8904  5792]]
```

The above screenshot showcases the confusion matrix with 21,818 true positives and 2, 826 false positives. Also it shows 8,984 false negatives and 5,793 true negatives.

Here is the confusion matrix as a graph:



I also calculated the misclassification error and classification accuracy as shown below:

```
#getting the classification error(missclassification rate)
from sklearn import metrics
print('Missclassification error:', (1 - metrics.accuracy_score(y_test, y_pred)))

#Classification accuracy
print('Classification accuracy: ',metrics.accuracy_score(y_test, y_pred))
```

```
Missclassification error: 0.2981698017285206
Classification accuracy:  0.7018301982714794
```

Random Forest

In the random forest classifier, I obtained the training and testing accuracy. As shown below:

```
#building random forest classifer
df = hotel

from sklearn.ensemble import RandomForestClassifier
feat_labels = df.columns[:9]
X = df[feat_labels]
y = df['is_canceled']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=0)

forest = RandomForestClassifier(n_estimators=1000, random_state=0, n_jobs=-1)
forest.fit(X_train, y_train)
print("Training accuracy:", forest.score(X_train, y_train))
print("Testing accuracy:", forest.score(X_test, y_test))

importances = forest.feature_importances_
for f in range(X_train.shape[1]):
    print('Feature: ', feat_labels[f], '\tImportance: ', importances[f])
```

```
Training accuracy: 0.9889946162514085
Testing accuracy: 0.8543721403152008
```

As well as the importance of all the features on the ABT.

```
Feature:  lead_time     Importance:  0.2518017432074677
Feature:  days_in_waiting_list  Importance:  0.008157936736087917
Feature:  total_of_special_requests     Importance:  0.07746127873070682
Feature:  adr   Importance:  0.1885909119042572
Feature:  agent         Importance:  0.11704120997646894
Feature:  stays_in_week_nights  Importance:  0.06493244314651926
Feature:  previous_cancellations        Importance:  0.0598489714616067
Feature:  arrival_date_day_of_month     Importance:  0.11407813601066806
Feature:  arrival_date_week_number      Importance:  0.1180873688262175
```

Phase IV: Results and Conclusion

After the building, training, and testing the models - here are the results:

Decision Tree (rounding up to 3 significant figures)
For the tree itself:

- Testing score: 0.763
- Training score: 0.768

Here the results for testing and training scores are very similar. However, they are only at 76-77%.

For entropy (in the max depth of the tree):

- Testing score: 0.771
- Training score: 0.779

Once again the results for testing and training scores are very similar and slightly better than the tree itself.

For gini (in the max depth of the tree):

- Testing score: 0.771
- Training score: 0.784

The results for testing and training scores are similar, with the training score close to being 1% higher than the testing score. These scores are better than the tree itself and entropy.

Logistic Regression (rounding up to 3 significant figures)
As mentioned before, the confusion matrix gave 21,818 TP, 2,826 FP, 8904 FN, and 5,792 TN. The model did well with predicting the values in the positive row but not so well in the negative one. Therefore it gave a of:

- Misclassification error: 0.298
- Classification accuracy: 0.701

Therefore this model performed worse than the decision tree model.

Random Forest (rounding up to 3 significant figures)
Random forest calculated accuracy:

- Testing accuracy: 0.854
- Training accuracy: 0.989

Importance (feature ranked highest to lowest):

1. lead_time importance: 0.251
2. adr importance: 0.189
3. arrival_date_week_number importance: 0.118
4. agent importance: 0.117
5. arrival_date_day_of_month importance: 0.114
6. total_of_special_requests importance: 0.077
7. stays_in_week_nights importance: 0.064
8. previous_cancellations importance: 0.060
9. days_in_waiting_list importance: 0.008

Therefore the model which should be implemented by the hotel chain Knights Inn to predict if customers will cancel or not is the Random Forest model. The Random Forest model had significantly better results for testing and training than the other models.