Project_3b

June 14, 2024

1 Project 3b

The final part of the project will ask you to perform your own data science project to classify a new dataset.

1.1 Submission Details

Project is due June 14th at 11:59 pm (Friday Midnight). To submit the project, please save the notebook as a pdf file and submit the assignment via Gradescope. In addition, make sure that all figures are legible and sufficiently large. For best pdf results, we recommend printing the notebook using LaTeX

1.2 Loading Essentials and Helper Functions

[1]:
```
# fix for windows memory leak with MKL
import os
import platform

if platform.system() == "Windows":
    os.environ["OMP_NUM_THREADS"] = "2"
```

[15]:
```
# import libraries
import time
import random
import numpy as np  # linear algebra
import pandas as pd  # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt  # this is used for the plot the graph

# Sklearn classes
from sklearn.model_selection import (
    train_test_split,
    cross_val_score,
    GridSearchCV,
    KFold,
)
from sklearn import metrics
from sklearn.metrics import confusion_matrix, silhouette_score
import sklearn.metrics.cluster as smc
```

```
from sklearn.cluster import KMeans
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.preprocessing import (
    StandardScaler,
    OneHotEncoder,
    LabelEncoder,
    MinMaxScaler,
)
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn import tree
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_blobs

import seaborn as sns
from helper import (
    draw_confusion_matrix,
    heatmap,
    make_meshgrid,
    plot_contours,
    draw_contour,
)

from sklearn.experimental import enable_halving_search_cv
from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import HalvingGridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score

%matplotlib inline
%config InlineBackend.figure_format = 'retina'

# Sets random seed for reproducibility
SEED = 42
random.seed(SEED)
```

2 (100 pts) Putting it all together: Classify your own data

Through the course of this program, you have acquired knowledge and skills in applying various models to tackle supervised learning tasks. Now, we challenge you to harness your cumulative learning and create a model capable of predicting whether a hotel reservation will be canceled or not.

2.0.1 Context

Hotels welcome millions of guests every year, and their primary objective is to keep rooms occupied and paid for. Cancellations can be detrimental to the business, as it may become challenging to rebook a room on short notice. Consequently, it is beneficial for hotels to anticipate which reservations are likely to be canceled. The provided dataset offers a diverse range of information about bookings, which you will utilize to predict cancellations.

2.0.2 Challenge

The goal of this project is to develop a predictive model that can determine whether a reservation will be canceled based on the available input parameters.

While we will provide specific instructions to guide you in the right direction, you have the freedom to choose the models and preprocessing techniques that you deem most appropriate. Upon completion, we request that you provide a detailed description outlining the models you selected and the rationale behind your choices.

2.0.3 Data Description

Refer to https://www.kaggle.com/competitions/m-148-spring-2024-project-3/data for information

2.1 (50 pts) Preprocessing

For the dataset, the following are mandatory pre-processing steps for your data:

- **Use One-Hot Encoding on all categorical features** (specify whether you keep the extra feature or not for features with multiple values)
- Determine which fields need to be dropped
- **Handle missing values** (Specify your strategy)
- **Rescale the real valued features using any strategy you choose** (StandardScaler, MinMaxScaler, Normalizer, etc)
- **Augment at least one feature**
- **Implement a train-test split with 20% of the data going to the test data**. Make sure that the test and train data are balanced in terms of the desired class.

After writing your preprocessing code, write out a description of what you did for each step and provide a justification for your choices. All descriptions should be written in the markdown cells of the jupyter notebook. Make sure your writing is clear and professional.

We highly recommend reading through the scikit-learn documentation to make this part easier.

[3]:
```
# Loading in dataset
df = pd.read_csv("datasets/hotel_booking.csv")

df.head()
```

[3]:

	hotel	is_canceled	lead_time	arrival_date_month	\
0	City Hotel	1	157	May	
1	Resort Hotel	0	167	September	
2	City Hotel	0	124	April	

```
3  Resort Hotel            0          8               July
4    City Hotel            0         43               July

   stays_in_weekend_nights  stays_in_week_nights  adults  children  babies  \
0                        1                     3       2       0.0       0
1                        2                     8       2       0.0       0
2                        1                     1       2       0.0       0
3                        2                     4       2       1.0       0
4                        0                     2       2       0.0       0

  meal  … booking_changes  deposit_type  days_in_waiting_list  \
0   BB  …               0    Non Refund                     0
1   BB  …               0    No Deposit                     0
2   SC  …               0    No Deposit                     0
3   BB  …               0    No Deposit                     0
4   HB  …               1    No Deposit                     0

     customer_type     adr required_car_parking_spaces  \
0        Transient  130.00                           0
1         Contract   62.48                           0
2        Transient   99.00                           0
3        Transient  169.00                           1
4  Transient-Party   43.00                           0

   total_of_special_requests              name                         email  \
0                          0     Taylor Juarez      Juarez.Taylor44@zoho.com
1                          2    Yolanda Taylor  Taylor.Yolanda35@xfinity.com
2                          1       Angie Dixon       Angie_Dixon@hotmail.com
3                          2  Jennifer Higgins   Higgins.Jennifer@yandex.com
4                          0     Jeremy Wilcox     Jeremy_Wilcox@hotmail.com

   phone-number
0  634-458-8010
1  571-733-2380
2  818-661-8987
3  669-803-3888
4  100-100-0744

[5 rows x 24 columns]
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69591 entries, 0 to 69590
Data columns (total 24 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   hotel                           69591 non-null  object
```

```
 1   is_canceled                     69591 non-null  int64
 2   lead_time                       69591 non-null  int64
 3   arrival_date_month              69591 non-null  object
 4   stays_in_weekend_nights         69591 non-null  int64
 5   stays_in_week_nights            69591 non-null  int64
 6   adults                          69591 non-null  int64
 7   children                        69588 non-null  float64
 8   babies                          69591 non-null  int64
 9   meal                            69591 non-null  object
 10  country                         69591 non-null  object
 11  previous_cancellations          69591 non-null  int64
 12  previous_bookings_not_canceled  69591 non-null  int64
 13  reserved_room_type              69591 non-null  object
 14  booking_changes                 69591 non-null  int64
 15  deposit_type                    69591 non-null  object
 16  days_in_waiting_list            69591 non-null  int64
 17  customer_type                   69591 non-null  object
 18  adr                             69591 non-null  float64
 19  required_car_parking_spaces     69591 non-null  int64
 20  total_of_special_requests       69591 non-null  int64
 21  name                            69591 non-null  object
 22  email                           69591 non-null  object
 23  phone-number                    69591 non-null  object
dtypes: float64(2), int64(12), object(10)
memory usage: 12.7+ MB
```

[5]:
```
df.describe()
```

[5]:

	is_canceled	lead_time	stays_in_weekend_nights	stays_in_week_nights	adults	children	babies
count	69591.000000	69591.000000	69591.000000	69591.000000	69591.000000	69588.000000	69591.000000
mean	0.405814	109.181546	0.880746	2.434280	1.839088	0.089081	0.008708
std	0.491052	113.714559	0.983784	1.852226	0.617512	0.369929	0.105919
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	17.000000	0.000000	1.000000	2.000000	0.000000	0.000000
50%	0.000000	71.000000	1.000000	2.000000	2.000000	0.000000	0.000000
75%	1.000000	169.000000	2.000000	3.000000	2.000000	0.000000	0.000000
max	1.000000	709.000000	16.000000	41.000000	55.000000	10.000000	10.000000

```
       previous_cancellations  previous_bookings_not_canceled  \
count            69591.000000                    69591.000000
mean                 0.107571                        0.177566
std                  0.860100                        1.747278
min                  0.000000                        0.000000
25%                  0.000000                        0.000000
50%                  0.000000                        0.000000
75%                  0.000000                        0.000000
max                 26.000000                       72.000000

       booking_changes  days_in_waiting_list           adr  \
count     69591.000000          69591.000000  69591.000000
mean          0.203015              2.795031     98.175805
std           0.597184             19.475713     52.222296
min           0.000000              0.000000     -6.380000
25%           0.000000              0.000000     65.000000
50%           0.000000              0.000000     90.000000
75%           0.000000              0.000000    120.000000
max          21.000000            391.000000   5400.000000

       required_car_parking_spaces  total_of_special_requests
count                 69591.000000                69591.000000
mean                      0.065641                    0.509060
std                       0.248870                    0.767946
min                       0.000000                    0.000000
25%                       0.000000                    0.000000
50%                       0.000000                    0.000000
75%                       0.000000                    1.000000
max                       3.000000                    5.000000
```

3 Data cleaning

From reading the documentation, I found that the following columns are categorical and will be dealt wtih accordingly: * hotel * is_cancelled * arrival_date_month * meal * country * reserved_room_types * deposit_type * customer_type * name * email * phone_number

We can drop name, email, phone_number since these do not provide any useful information

[6]:
```
df = df.drop(columns=['name', 'email', 'phone-number'])
df.head()
```

[6]:
```
          hotel  is_canceled  lead_time arrival_date_month  \
0    City Hotel            1        157                May
1  Resort Hotel            0        167          September
2    City Hotel            0        124              April
3  Resort Hotel            0          8               July
```

```
4    City Hotel            0         43               July

   stays_in_weekend_nights  stays_in_week_nights  adults  children  babies  \
0                        1                     3       2       0.0       0
1                        2                     8       2       0.0       0
2                        1                     1       2       0.0       0
3                        2                     4       2       1.0       0
4                        0                     2       2       0.0       0

  meal  … previous_cancellations  previous_bookings_not_canceled  \
0   BB  …                       0                               0
1   BB  …                       0                               0
2   SC  …                       0                               0
3   BB  …                       0                               0
4   HB  …                       0                               0

   reserved_room_type booking_changes  deposit_type days_in_waiting_list  \
0                   A               0    Non Refund                    0
1                   D               0    No Deposit                    0
2                   A               0    No Deposit                    0
3                   A               0    No Deposit                    0
4                   A               1    No Deposit                    0

     customer_type     adr  required_car_parking_spaces  \
0        Transient  130.00                            0
1         Contract   62.48                            0
2        Transient   99.00                            0
3        Transient  169.00                            1
4  Transient-Party   43.00                            0

   total_of_special_requests
0                          0
1                          2
2                          1
3                          2
4                          0

[5 rows x 21 columns]
```

[7]:
```
# Checking for null values
df.isnull().sum()
```

[7]:
```
hotel                           0
is_canceled                     0
lead_time                       0
arrival_date_month              0
stays_in_weekend_nights         0
```

```
stays_in_week_nights              0
adults                            0
children                          3
babies                            0
meal                              0
country                           0
previous_cancellations            0
previous_bookings_not_canceled    0
reserved_room_type                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
```

I found that there are only 3 null values in the 'children' section, and since the dataset is rather large with almost 70,000 entries, I'm going to just drop these rows.

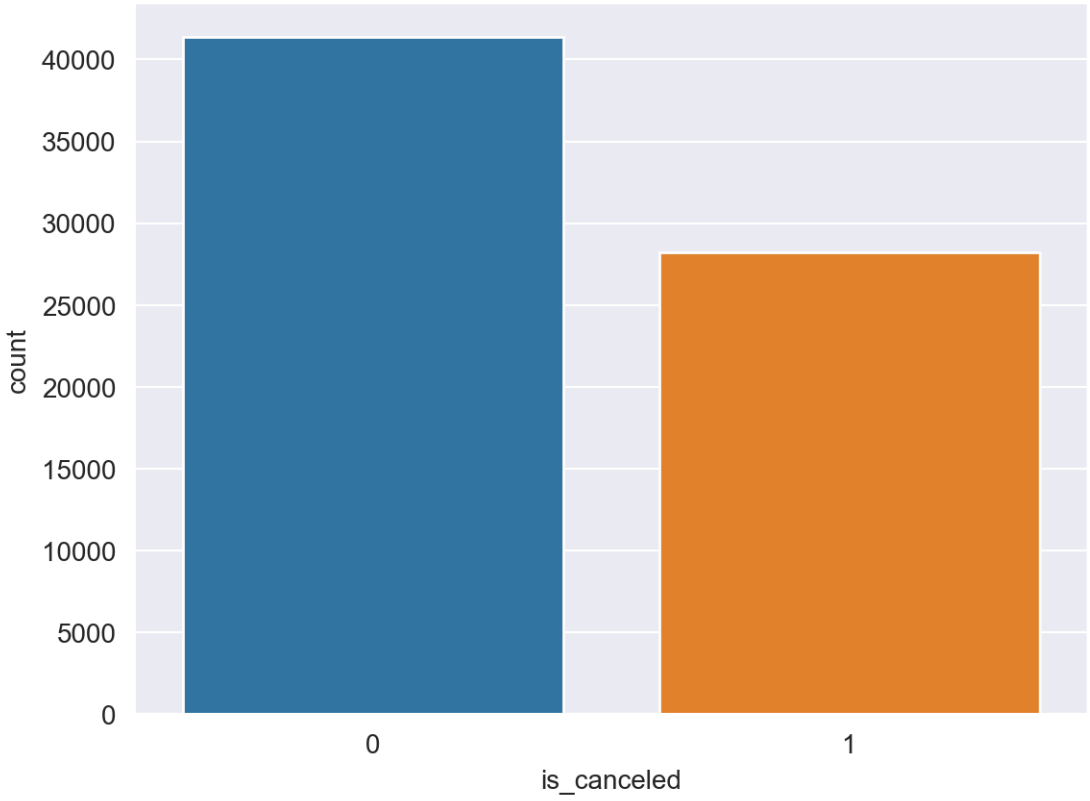
[8]:
```
# Dropping na values
df.dropna(inplace=True)

# Checking to make sure it worked
print(df.isnull().sum())
```

```
hotel                             0
is_canceled                       0
lead_time                         0
arrival_date_month                0
stays_in_weekend_nights           0
stays_in_week_nights              0
adults                            0
children                          0
babies                            0
meal                              0
country                           0
previous_cancellations            0
previous_bookings_not_canceled    0
reserved_room_type                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
```

[9]:
```
# Checking if data is balanced
sns.countplot(x='is_canceled', data=df)
```

[9]:
```
<Axes: xlabel='is_canceled', ylabel='count'>
```



3.0.1 Augmentation

I decided I am going to create a new feature called 'is_family' based on 'children' and 'babies'. According to SHR Group's Hotel Industry Trend, families were the most likely to cancel in 2024, so this augmented feature should provide some good insight.

I am also creating a column called 'stay_duration' which calculates the total number of nights a guest is staying, a column called 'is_repeated_guest' which indicates if a guest has stayed with the hotel before, a column 'has_special_requests' which just indicates if the guest has made special requests, and 'is_high_season' which indicates if a booking is made during a high travel season (usually in the summer), and a column called 'cancellation_rate' which calculates the rate of cancellations for a customer.

[10]:
```
# # Creating new feature indicating if a booking is made for a family
```

```
df['is_family'] = df.apply(lambda row: 1 if row['children'] > 0 or␣
 ↪row['babies'] > 0 else 0, axis=1)

# Stay Duration
df['stay_duration'] = df['stays_in_weekend_nights'] + df['stays_in_week_nights']

# Cancellation rate for each booking
df['cancellation_rate'] = df['previous_cancellations'] /␣
 ↪(df['previous_cancellations'] + df['previous_bookings_not_canceled'])
# Fill any NaN values which might occur due to division by zero
# This occurs when there's a new guest
df['cancellation_rate'].fillna(0, inplace=True)

df['is_repeated_guest'] = np.where((df['previous_cancellations'] > 0) |␣
 ↪(df['previous_bookings_not_canceled'] > 0), 1, 0)

df['has_special_request'] = np.where(df['total_of_special_requests'] > 0, 1, 0)

df['is_high_season'] = df['arrival_date_month'].apply(lambda x: 1 if x in␣
 ↪['June', 'July', 'August'] else 0)


df.head()
```

```
/var/folders/cw/0rcfs23d78sd29z7njmt0yrh0000gn/T/ipykernel_28560/1564307559.py:1
1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.
```

```
  df['cancellation_rate'].fillna(0, inplace=True)
```

[10]:

	hotel	is_canceled	lead_time	arrival_date_month	\
0	City Hotel	1	157	May	
1	Resort Hotel	0	167	September	
2	City Hotel	0	124	April	
3	Resort Hotel	0	8	July	
4	City Hotel	0	43	July	

	stays_in_weekend_nights	stays_in_week_nights	adults	children	babies	\
0	1	3	2	0.0	0	

```
1                        2                    8       2       0.0       0
2                        1                    1       2       0.0       0
3                        2                    4       2       1.0       0
4                        0                    2       2       0.0       0

  meal  …    customer_type     adr  required_car_parking_spaces  \
0   BB  …        Transient  130.00                            0
1   BB  …         Contract   62.48                            0
2   SC  …        Transient   99.00                            0
3   BB  …        Transient  169.00                            1
4   HB  …  Transient-Party   43.00                            0

  total_of_special_requests  is_family stay_duration  cancellation_rate  \
0                         0          0             4                0.0
1                         2          0            10                0.0
2                         1          0             2                0.0
3                         2          1             6                0.0
4                         0          0             2                0.0

  is_repeated_guest  has_special_request  is_high_season
0                 0                    0               0
1                 0                    1               0
2                 0                    1               0
3                 0                    1               1
4                 0                    0               1

[5 rows x 27 columns]
```

[11]:
```
# Splitting features into numerical and categorical
numerical = ['stays_in_weekend_nights', 'stays_in_week_nights', 'adults',␣
 ↪'previous_cancellations', 'previous_bookings_not_canceled',␣
 ↪'booking_changes', 'days_in_waiting_list', 'adr',␣
 ↪'required_car_parking_spaces', 'stay_duration', 'cancellation_rate']

categorical = ['hotel', 'arrival_date_month', 'meal', 'country',␣
 ↪'reserved_room_type', 'deposit_type', 'customer_type', 'is_high_season',␣
 ↪'is_repeated_guest', 'has_special_request','is_family']
```

4 Exploratory Data Analysis

Now I am just going to do some basic exploratory data analysis to look at distributions

I suspect there will be a strong relationship between customer_type and cancellations

[343]:
```
# Checking correlation between customer_type and cancellations
contingency_table = pd.crosstab(df['customer_type'], df['is_canceled'])
print(contingency_table)
```

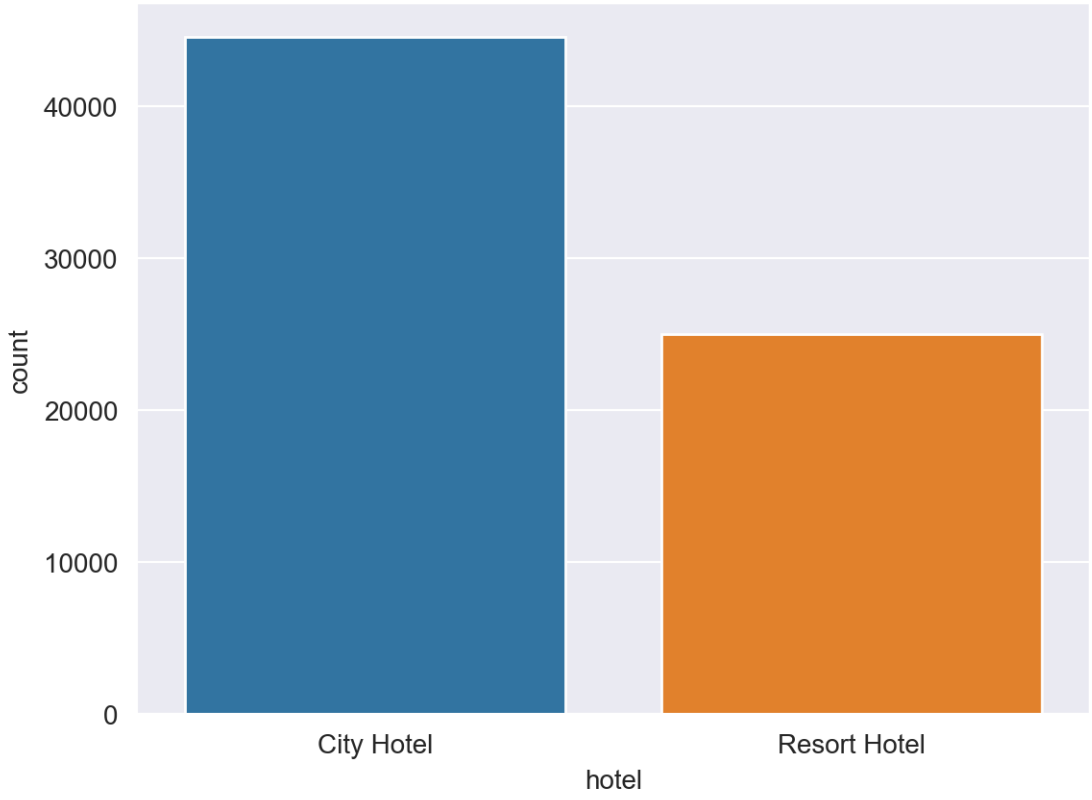
```
is_canceled          0      1
customer_type
Contract          1789   1012
Group              300     46
Transient        28555  22348
Transient-Party  10706   4832
```

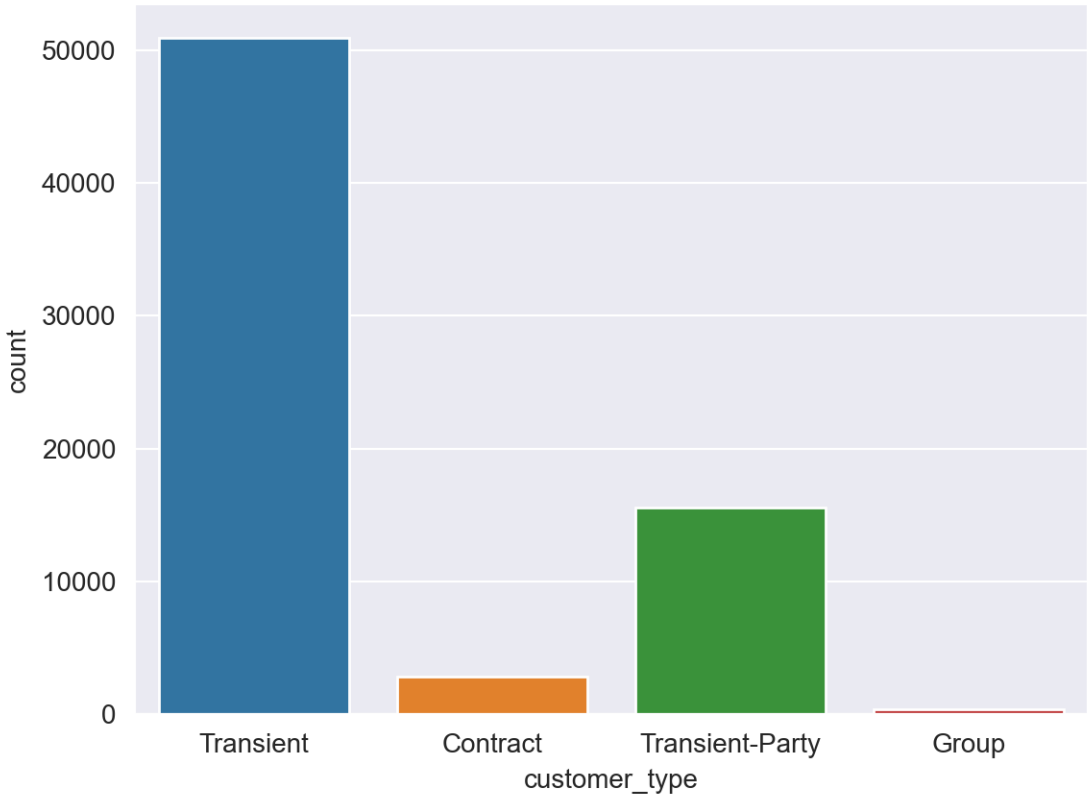
[344]:
```
from scipy.stats import chi2_contingency

chi2, p, dof, ex = chi2_contingency(contingency_table)
print("p-value of chi-square test:", p)
```

```
p-value of chi-square test: 5.8139262209252394e-204
```

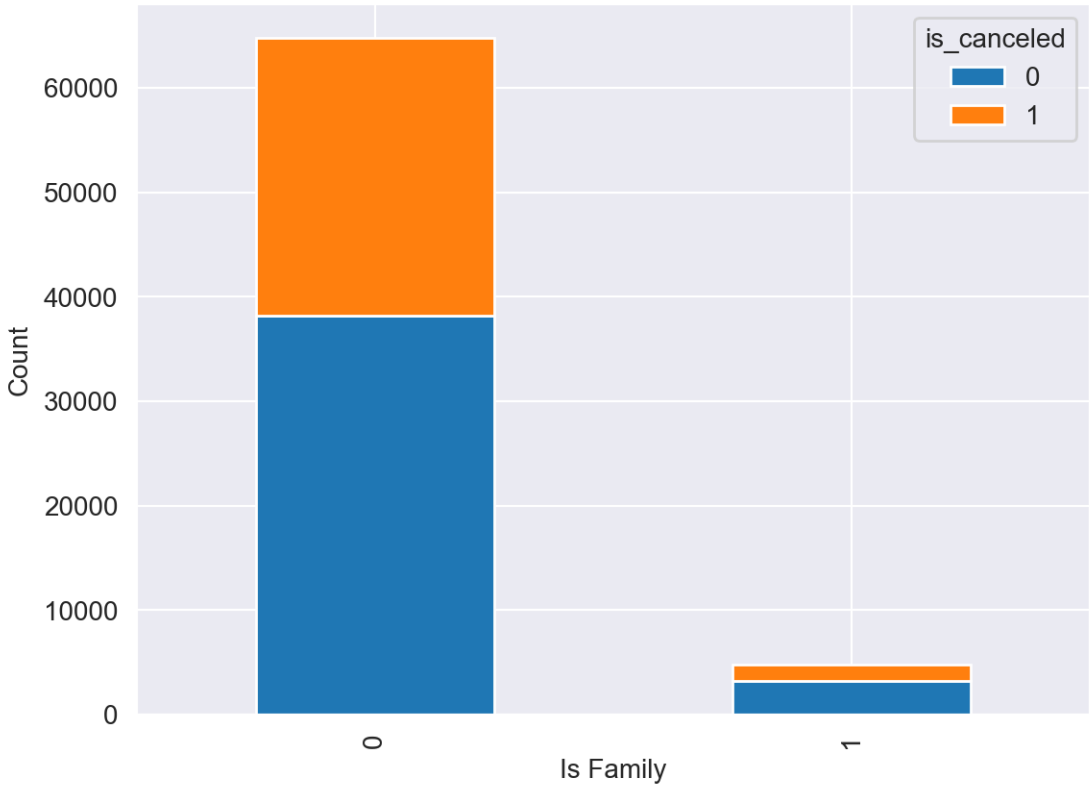
It looks like there is a very strong association. This makes sense as those travelling in groups are less likely to cancel than those travelling alone or as a couple. This will be an important feature in the model.

[345]:
```
# Creating a bar graph for showing number of bookings per hotel

sns.countplot(x='hotel', data=df)
plt.show()

sns.countplot(x='customer_type', data=df)
plt.show()
```

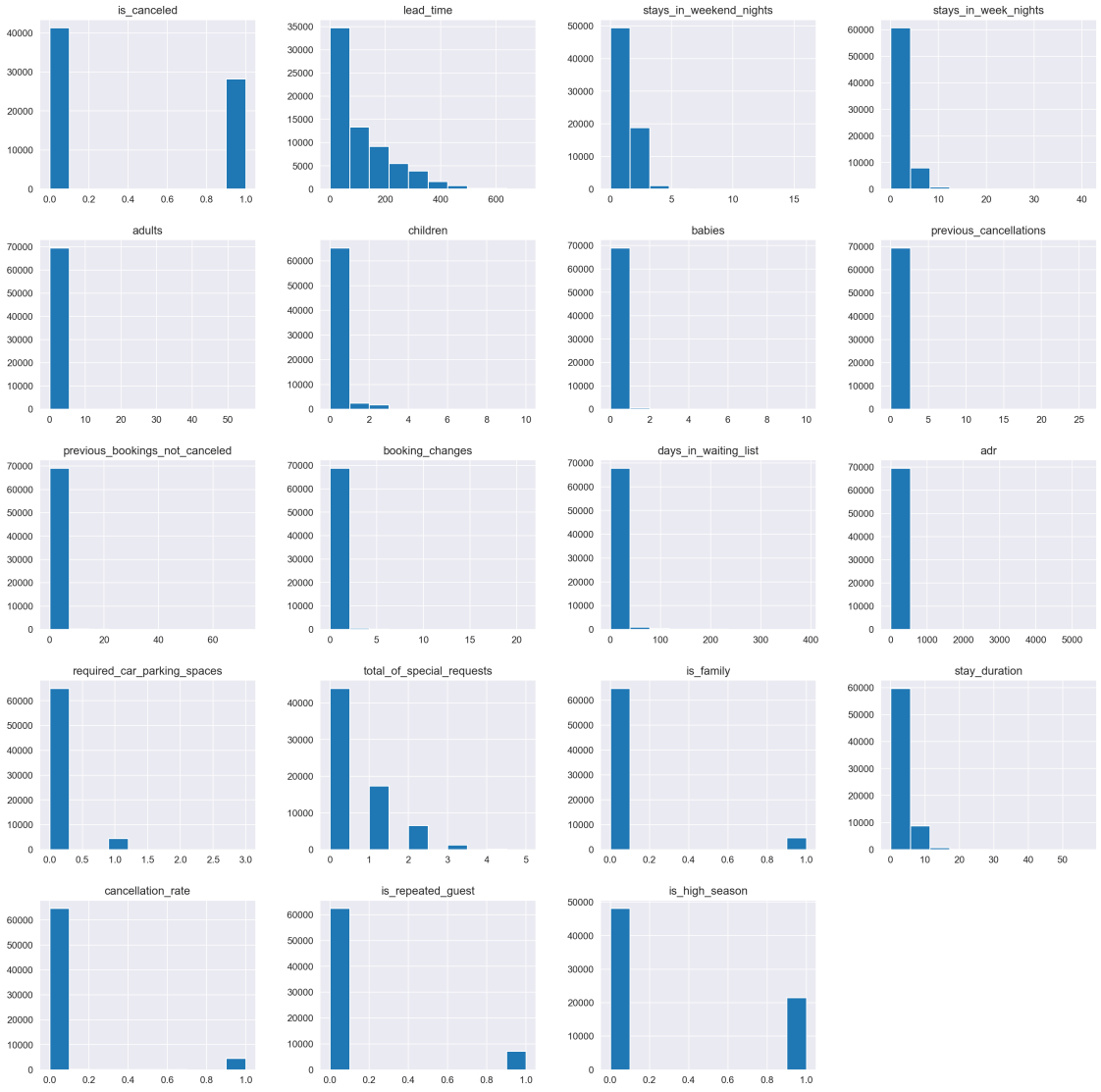




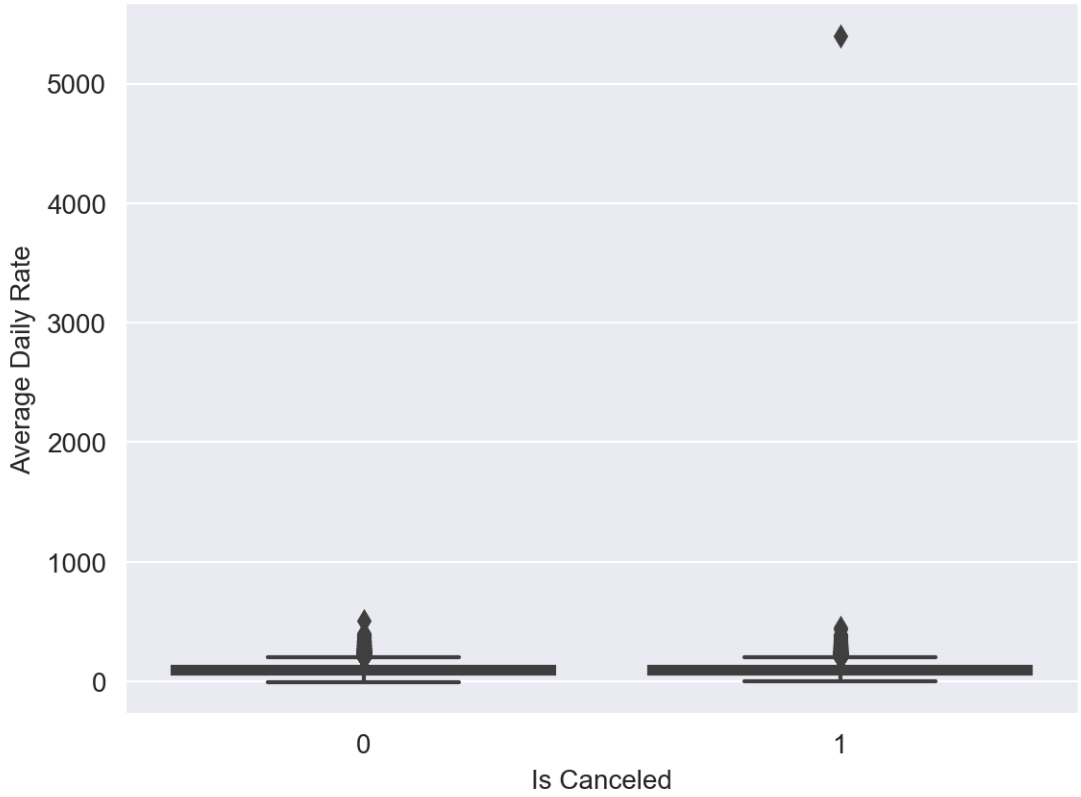
[346]:
```
# Looking at relationship between is_family and cancellations
pd.crosstab(df['is_family'], df['is_canceled']).plot(kind='bar', stacked=True)
plt.xlabel('Is Family')
plt.ylabel('Count')
plt.show()
```



[347]:
```
df.hist(figsize=(20,20))
plt.show()
```



[348]:
```
# Looking at relationship between average daily rate and cancellations
sns.boxplot(x='is_canceled', y='adr', data=df)
plt.xlabel('Is Canceled')
plt.ylabel('Average Daily Rate')
plt.show()
```



5 Data Processing

[12]:
```
# Defining target
y = df['is_canceled']

# Define features
X = df.drop(columns='is_canceled')

# Splitting data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=SEED)
```

[13]:
```
# Creating pipeline for transforming features
numerical_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder())
])
```

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical),
        ('cat', categorical_transformer, categorical)
    ])
```

[14]:
```
train = preprocessor.fit_transform(X_train)
test = preprocessor.transform(X_test)

# Getting feature names
feature_names = preprocessor.get_feature_names_out(list(X.columns))
```

I ended up keeping the extra features after one-hot encoding as I didn't see a reason to not include them.

5.1 (50 pts) Try out a few models

Now that you have pre-processed your data, you are ready to try out different models.

For this part of the project, we want you to experiment with all the different models demonstrated in the course to determine which one performs best on the dataset.

You must perform classification using at least 3 of the following models: - Logistic Regression - K-nearest neighbors - SVM - Decision Tree - Multi-Layer Perceptron

Due to the size of the dataset, be careful which models you use and look at their documentation to see how you should tackle this size issue for each model.

For full credit, you must perform some hyperparameter optimization on your models of choice. You may find the following scikit-learn library on hyperparameter optimization useful.

For each model chosen, write a description of which models were chosen, which parameters you optimized, and which parameters you choose for your best model. While the previous part of the project asked you to pre-process the data in a specific manner, you may alter pre-processing step as you wish to adjust for your chosen classification models.

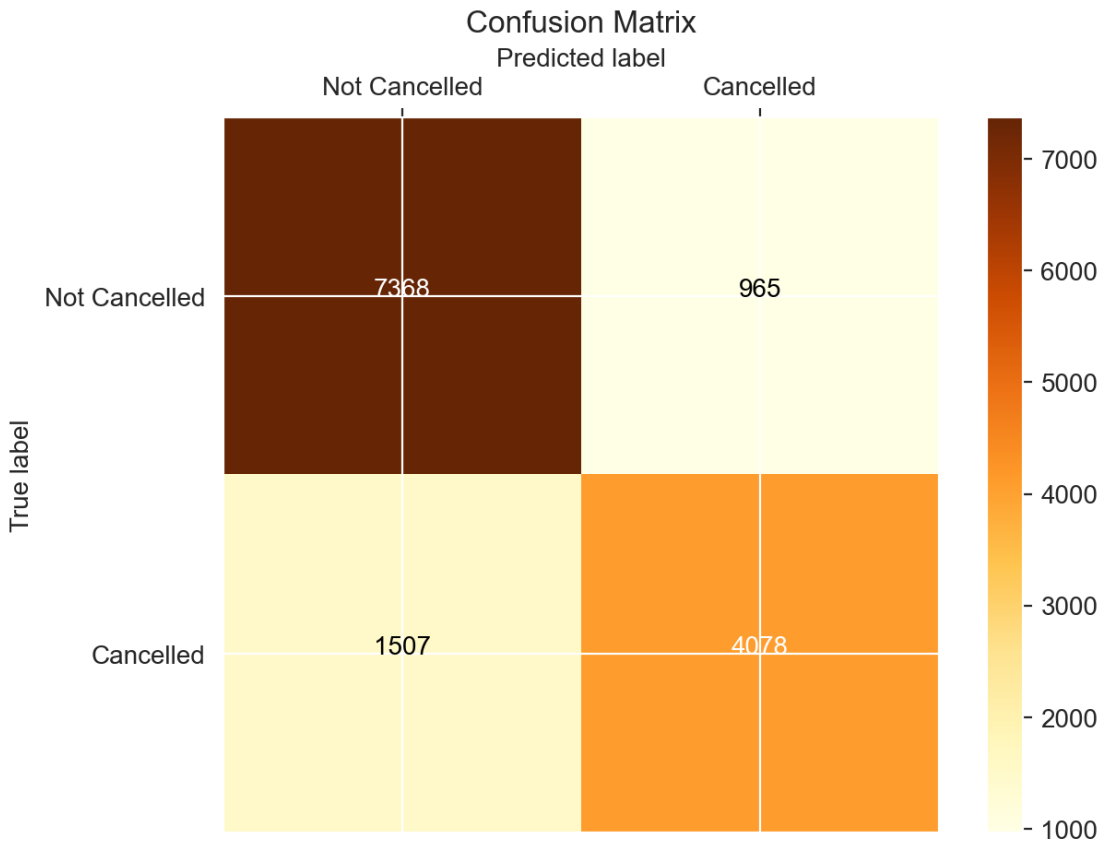
[352]:
```
# Hyperparameter optimization for KNN model
# Define the parameter grid
params = {
    'n_neighbors' : [1, 3, 5, 7, 9],
    'metric' : ["euclidean", "manhattan"]
}

# Instantiate the grid search model
grid_search_knn = GridSearchCV(estimator=KNeighborsClassifier(),␣
 ↪param_grid=params, cv=10, scoring='accuracy')

# Fit the grid search to the data
grid_search_knn.fit(train, y_train)
```

```
# Print the best parameters and the best score
print("Best Parameters: ", grid_search_knn.best_params_)
print("Best Score: ", grid_search_knn.best_score_)
```

```
Best Parameters:  {'metric': 'manhattan', 'n_neighbors': 9}
Best Score:  0.8198131848392313
```

[353]:
```
# Using fitted model to make predictions
test_predictions_knn = grid_search_knn.best_estimator_.predict(test)

# Calculate the accuracy of the model on the test data
test_accuracy = accuracy_score(y_test, test_predictions_knn)

# Print the test accuracy
print(f"Test Accuracy (KNN): {test_accuracy*100:.3f}")

# Classification Report
print(metrics.classification_report(y_test, test_predictions_knn))

draw_confusion_matrix(y_test, test_predictions_knn, ['Not Cancelled',␣
↪'Cancelled'])
```

```
Test Accuracy (KNN): 82.239
              precision    recall  f1-score   support

           0       0.83      0.88      0.86      8333
           1       0.81      0.73      0.77      5585

    accuracy                           0.82     13918
   macro avg       0.82      0.81      0.81     13918
weighted avg       0.82      0.82      0.82     13918
```



with 'has_special_requests' -> 83.2 without ->

5.2 KNN Model Description

This KNN model is pretty accurate, with a test accuracy of 83%. I optimized the hyperparameters using GridSearchCV and found the best parameters to be {'metric': 'euclidean', 'n_neighbors': 9}. It trains very quickly which is I used GridSearchCV instead of HalvingGridSearchCV, which I use in later models.

[354]:
```
# Building Logistic Regression model and hyperparameter optimizing

from sklearn.linear_model import LogisticRegression

# Instantiate the model (using the default parameters)
logreg = LogisticRegression(max_iter=1000)

params = {
    'penalty': ["l1", "l2"],
    'solver': ["liblinear", "saga"],
    'C': [0.001, 0.1, 10]
```

```
}

# Instantiate the grid search model
grid_search_log = HalvingGridSearchCV(estimator=logreg, param_grid=params,␣
↪cv=5, scoring='accuracy', n_jobs=-1)

# Fit the grid search to the data
grid_search_log.fit(train, y_train)

# Print the best parameters and the best score
print("Best Parameters: ", grid_search_log.best_params_)
print("Best Score: ", grid_search_log.best_score_)
```

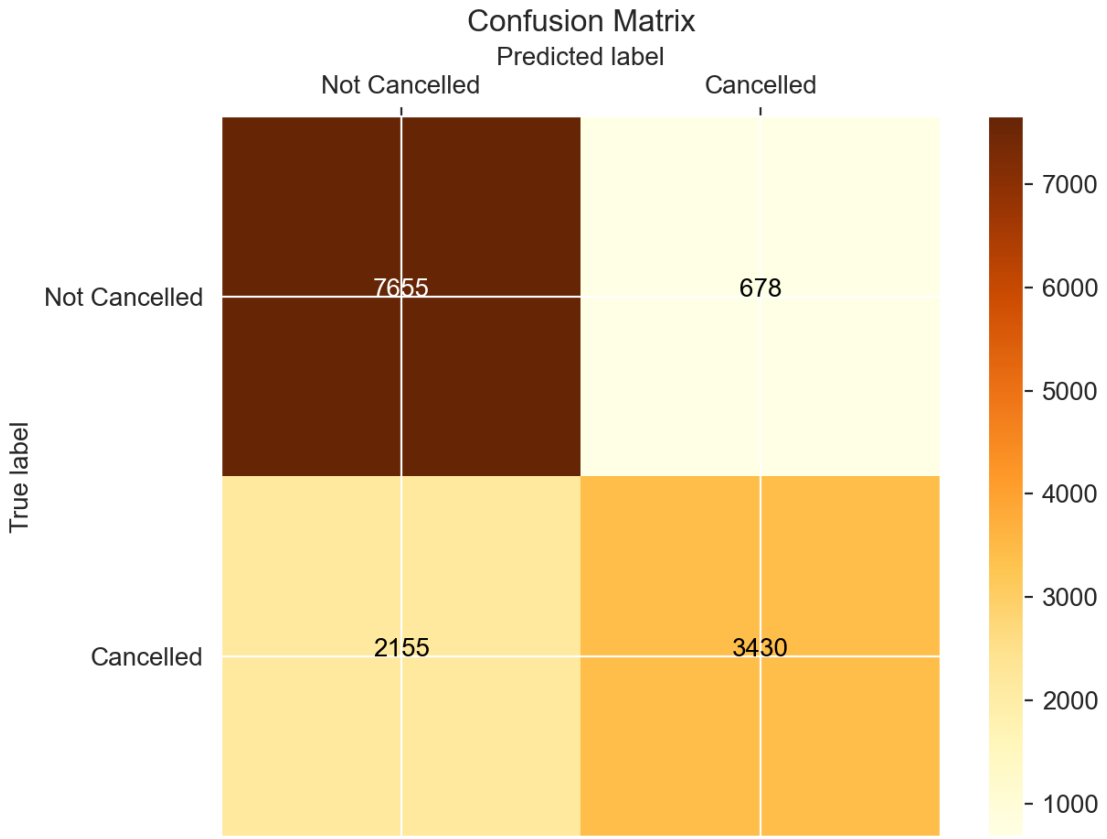
```
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
```

```
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
```

```
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(

Best Parameters:  {'C': 0.1, 'penalty': 'l1', 'solver': 'saga'}
Best Score:  0.7913769873349501

/Users/aidancone/anaconda3/envs/ece148/lib/python3.12/site-
packages/sklearn/linear_model/_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
```

[355]:
```
# Testing best logistic regression model
best_logreg_model = grid_search_log.best_estimator_

test_predictions_logreg = best_logreg_model.predict(test)

test_accuracy_logreg = accuracy_score(y_test, test_predictions_logreg)

print(f"Test Accuracy (Logistic Regression): {test_accuracy_logreg*100:.3f}")

# Classification Report
print(metrics.classification_report(y_test, test_predictions_logreg))

draw_confusion_matrix(y_test, test_predictions_logreg, ['Not Cancelled',␣
 ↪'Cancelled'])
```

```
Test Accuracy (Logistic Regression): 79.645
              precision    recall  f1-score   support

           0       0.78      0.92      0.84      8333
           1       0.83      0.61      0.71      5585

    accuracy                           0.80     13918
   macro avg       0.81      0.77      0.78     13918
weighted avg       0.80      0.80      0.79     13918
```



5.3 Logistic Regression Model Description

This logistic regression model is surprisingly less accurate than a KNN model, with a max test accuracy at 80%. The best parameters were {'C': 10, 'penalty': 'l1', 'solver': 'liblinear'}. It also is very slow to train compared to other models, even with HalvingGridSearchCV.

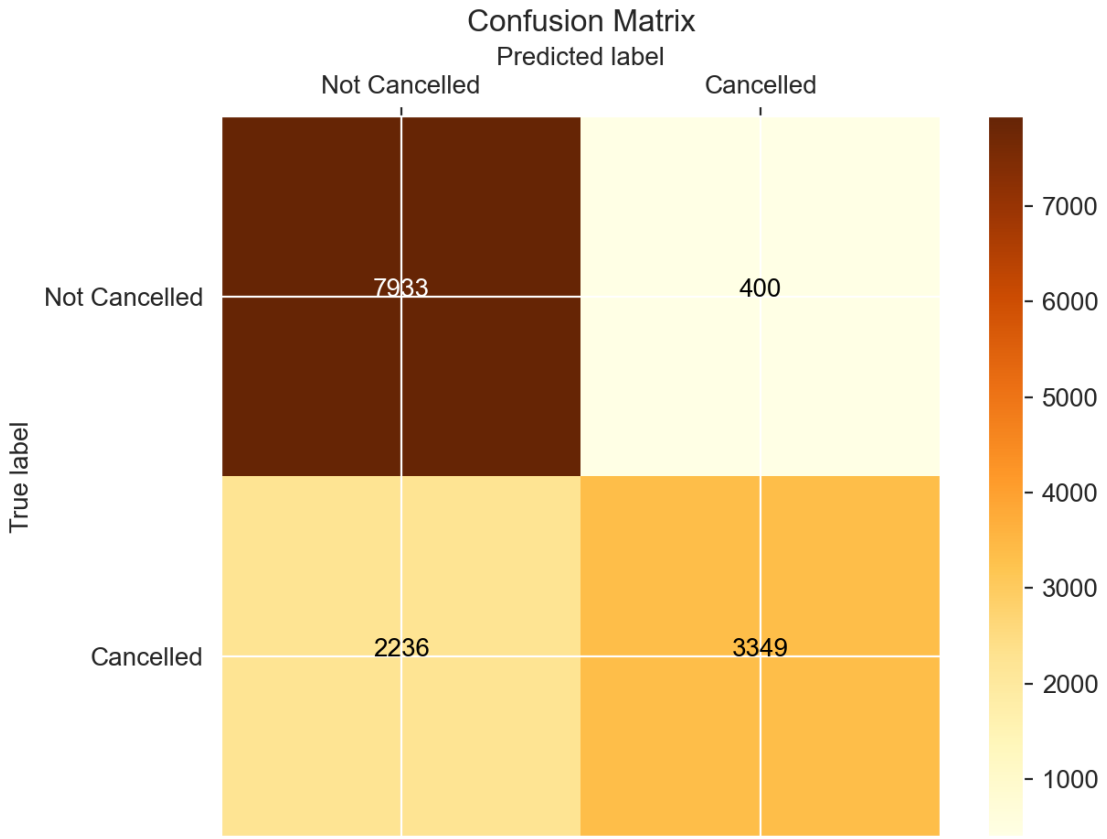
[356]:
```
# Building decision tree model

# Optimizing
params = {
    'max_depth': [10, 20, 30, 40],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

tree = DecisionTreeClassifier()

grid_search_tree = HalvingGridSearchCV(estimator=tree, param_grid=params, cv=3,␣
↪n_jobs=-1)
```

```
# Fitting GridSearch
grid_result_tree = grid_search_tree.fit(train, y_train)

# Best parameters
print("Best parameters found: ", grid_result_tree.best_params_)
print("Highest accuracy: ", grid_result_tree.best_score_)
```

```
Best parameters found:  {'max_depth': 10, 'min_samples_leaf': 1,
'min_samples_split': 2}
Highest accuracy:  0.8065448791453167
```

[357]:
```
# Testing best decision tree model
best_tree_model = grid_search_tree.best_estimator_

test_predictions_tree = best_tree_model.predict(test)

test_accuracy_tree = accuracy_score(y_test, test_predictions_tree)

print(f"Test Accuracy (Decision Tree): {test_accuracy_tree*100:.3f}")

# Classification Report
print(metrics.classification_report(y_test, test_predictions_tree))

draw_confusion_matrix(y_test, test_predictions_tree, ['Not Cancelled',␣
↪'Cancelled'])
```

```
Test Accuracy (Decision Tree): 81.060
              precision    recall  f1-score   support

           0       0.78      0.95      0.86      8333
           1       0.89      0.60      0.72      5585

    accuracy                           0.81     13918
   macro avg       0.84      0.78      0.79     13918
weighted avg       0.83      0.81      0.80     13918
```



5.4 Decision Tree Model Description

This Decision Tree model is around the same accuracy as the best KNN model, with an accuracy of 82%. The best parameters were {'max_depth': 20, 'min_samples_leaf': 4, 'min_samples_split': 10}.

[361]:
```
from sklearn.ensemble import RandomForestClassifier


# Define the parameters // refined
params = {
    'bootstrap': [False],
    'criterion': ['entropy'],
    'max_depth': [45, 50, 55],
    'max_features': ['sqrt'],
    'min_samples_leaf': [1, 2, 3],
    'min_samples_split': [2, 3],
    'n_estimators': [450, 500, 550]
}
```

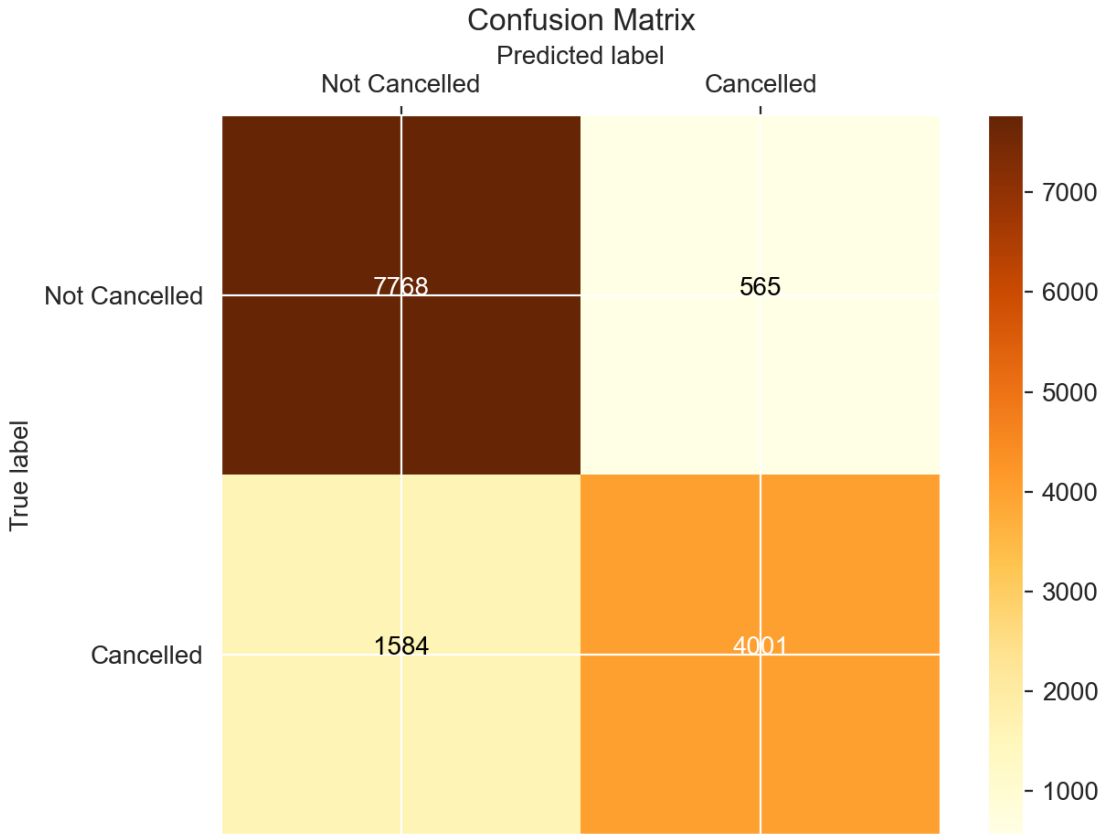
```
rf = RandomForestClassifier()

grid_search_rf = HalvingGridSearchCV(estimator=rf,param_grid=params, cv=5,␣
 ↪n_jobs=-1)

# Run the grid search
grid_search_rf.fit(train, y_train)

# Print out the best parameters
print("Best parameters found: ", grid_search_rf.best_params_)
print("Highest accuracy found: ", grid_search_rf.best_score_)
```

```
Best parameters found:  {'bootstrap': False, 'criterion': 'entropy',
'max_depth': 55, 'max_features': 'sqrt', 'min_samples_leaf': 3,
'min_samples_split': 2, 'n_estimators': 450}
Highest accuracy found:  0.8413334531404439
```

[362]:
```
# Testing the best Random Forest model
best_rf_model = grid_search_rf.best_estimator_

# Predict on the testing data
test_predictions_rf = best_rf_model.predict(test)

# Get the accuracy of the model
test_accuracy_rf = accuracy_score(y_test, test_predictions_rf)

print(f"Test Accuracy (Random Forest): {test_accuracy_rf * 100:.3f}")

# Check the classification report
print(metrics.classification_report(y_test, test_predictions_rf))

# Predict probabilities
probabilities_rf = best_rf_model.predict_proba(test)

# Probabilities for positive class
auc = roc_auc_score(y_test, probabilities_rf[:, 1])

print(f"AUC-ROC score for Random Forest is {auc}")

# Confusion Matrix
draw_confusion_matrix(y_test, test_predictions_rf, ['Not Cancelled',␣
 ↪'Cancelled'])
```

```
Test Accuracy (Random Forest): 84.560
              precision    recall  f1-score   support

           0       0.83      0.93      0.88      8333
```

```
           1       0.88      0.72      0.79      5585

    accuracy                           0.85     13918
   macro avg       0.85      0.82      0.83     13918
weighted avg       0.85      0.85      0.84     13918

AUC-ROC score for Random Forest is 0.9274096550254132
```



5.5 Random Forest Model Description

This Random Forest Classifier was the best model, achieving a test accuracy of 86% and an AUC-ROC score of nearly 94%. The best parameters for it were: {'bootstrap': False, 'criterion': 'entropy', 'max_depth': 45, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 550} Highest accuracy found: 0.8559285291760205. This one was a little slow to train, so to maximize efficiency I used HalvingGridSearchCV which greatly reduces the time to train without affecting the model's performance.

5.6 Extra Credit

We have provided an extra test dataset named `hotel_booking_test.csv` that does not have the target labels. Classify the samples in the dataset with any method of your choosing and save

the predictions into a csv file. Submit the file to our Kaggle contest. The website will specify your classification accuracy on the test set. We will award a bonus point for the project for every percentage point over 75% that you get on your kaggle test accuracy.

To get the bonus points, you must also write out a summary of the model that you submit including any changes you made to the pre-processing steps. The summary must be written in a markdown cell of the jupyter notebook. Note that you should not change earlier parts of the project to complete the extra credit.

Please refer to *Submission and evaluation* section on the contest page for the csv file formatting

5.6.1 Summary

The model I chose to submit is the best bagging classifier model, with parameters Best parameters found: {'bootstrap': False, 'bootstrap_features': True, 'estimator': DecisionTreeClassifier(random_state=42), 'max_features': 0.7, 'max_samples': 0.7, 'n_estimators': 300}. It ended up having an 86% test accuracy.

One thing I noticed in the above parts, was that class 1 (cancelled) was a little underrepresented in the dataset. In the classification reports, class 1 was consistently recalled worse by every model trained. As a result, I used SMOTE (Synthetic Minority Oversampling Technique), which synthetically creates more sample in the minority class so that the classes are balanced. This improved my accuracy a little bit, but also helped improve the models' ability to generalize.

6 Cleaning and processing the testing data

[33]:
```
# Read in hotel_booking_test
hotel = pd.read_csv("datasets/hotel_booking_test.csv")

hotel.head()
```

[33]:

	hotel	lead_time	arrival_date_month	stays_in_weekend_nights	\
0	City Hotel	107	June	0	
1	Resort Hotel	20	May	0	
2	Resort Hotel	125	April	2	
3	Resort Hotel	0	August	1	
4	City Hotel	124	August	0	

	stays_in_week_nights	adults	children	babies	meal	country	…	\
0	2	2	0.0	0	BB	PRT	…	
1	3	2	0.0	0	BB	PRT	…	
2	5	2	0.0	0	BB	GBR	…	
3	1	2	0.0	0	BB	FRA	…	
4	1	2	0.0	0	BB	GBR	…	

	booking_changes	deposit_type	days_in_waiting_list	customer_type	\
0	0	No Deposit	0	Transient-Party	
1	0	No Deposit	0	Transient	

```
2                0    No Deposit                     0         Contract
3                0    No Deposit                     0        Transient
4                0    No Deposit                     0        Transient

      adr  required_car_parking_spaces total_of_special_requests  \
0  130.00                             0                          1
1   91.67                             0                          0
2   42.95                             0                          1
3  106.00                             0                          0
4  127.80                             1                          1

              name                         email  phone-number
0  Dustin Marshall  Dustin.Marshall@xfinity.com  833-801-0855
1  Gregory Roberts       GRoberts17@verizon.com  881-819-0764
2    Dustin Hardin    Dustin_Hardin@verizon.com  560-971-8576
3   Kristy Stewart      Kristy.Stewart@mail.com  783-987-6285
4   Deanna Leblanc   Deanna.Leblanc75@gmail.com  518-112-1761

[5 rows x 23 columns]
```

[34]:
```
hotel.isnull().sum()
```

[34]:
```
hotel                           0
lead_time                       0
arrival_date_month              0
stays_in_weekend_nights         0
stays_in_week_nights            0
adults                          0
children                        0
babies                          0
meal                            0
country                         0
previous_cancellations          0
previous_bookings_not_canceled  0
reserved_room_type              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
name                            0
email                           0
phone-number                    0
dtype: int64
```

[35]:
```
# Applying same preprocessing steps as before
hotel['is_family'] = hotel.apply(lambda row: 1 if row['children'] > 0 or␣
 ↪row['babies'] > 0 else 0, axis=1)

# Stay Duration
hotel['stay_duration'] = hotel['stays_in_weekend_nights'] +␣
 ↪hotel['stays_in_week_nights']

hotel['is_repeated_guest'] = np.where((hotel['previous_cancellations'] > 0) |␣
 ↪(hotel['previous_bookings_not_canceled'] > 0), 1, 0)

# Cancellation rate for each booking
hotel['cancellation_rate'] = hotel['previous_cancellations'] /␣
 ↪(hotel['previous_cancellations'] + hotel['previous_bookings_not_canceled'])
# Fill any NaN values which might occur due to division by zero
# This occurs when there's a new guest
hotel['cancellation_rate'].fillna(0, inplace=True)


hotel['has_special_request'] = np.where(hotel['total_of_special_requests'] > 0,␣
 ↪1, 0)

hotel['is_high_season'] = hotel['arrival_date_month'].apply(lambda x: 1 if x in␣
 ↪['June', 'July', 'August'] else 0)
```

```
/var/folders/cw/0rcfs23d78sd29z7njmt0yrh0000gn/T/ipykernel_28560/328142690.py:13
: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  hotel['cancellation_rate'].fillna(0, inplace=True)
```

[36]:
```
# Dropping email, name, phone numer
hotel = hotel.drop(['email', 'name', 'phone-number'], axis=1)
```

[37]:
```
# Processing dataframe
hotel_transformed = preprocessor.transform(hotel)
```

7 Changing the preprocessing of training dataset

[18]:
```
from imblearn.over_sampling import SMOTE

# Run the preprocessors
train_preprocessed = preprocessor.fit_transform(X_train)
test_preprocessed = preprocessor.transform(X_test)

# Getting the categorical transformer from the pipeline
categorical_transformer = preprocessor.named_transformers_['cat']

# Get the trained OneHotEncoder from the categorical transformer
onehot = categorical_transformer.named_steps['onehot']

# Get the categories from the encoder
transformed_categories = onehot.categories_

# Create feature names for the transformed categories
cat_features_transformed = [f"{feat}_{val}" for feat, vals in zip(categorical,␣
 ↪transformed_categories) for val in vals]

# Combine all feature names
feature_names = numerical + cat_features_transformed



# Now for applying SMOTE
smote = SMOTE(random_state=2)
X_train_resampled, y_train_resampled = smote.fit_resample(train_preprocessed,␣
 ↪y_train)
```

8 Training models

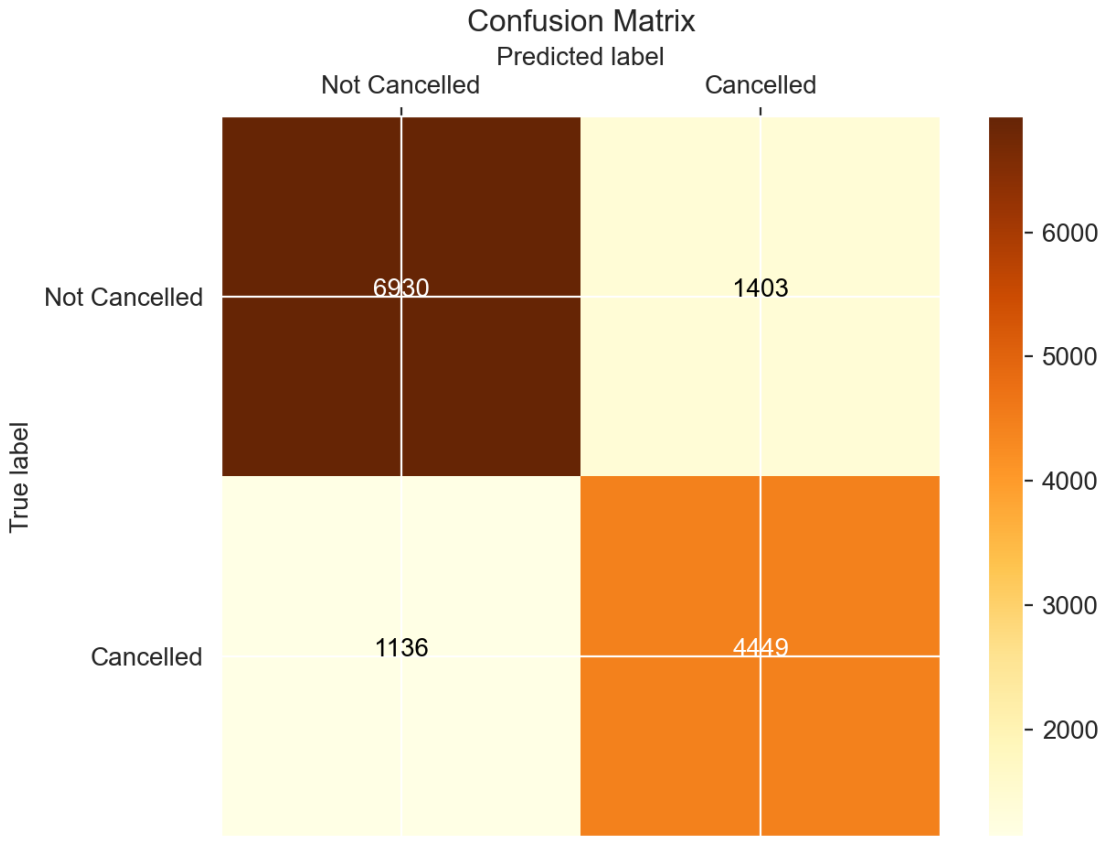
[19]:
```
# Hyperparameter optimization for KNN model

# Define the parameter grid
params = {
    'n_neighbors' : [1, 3, 5, 7, 9],
    'metric' : ["euclidean", "manhattan"]
}

# Instantiate the grid search model
grid_search_knn = GridSearchCV(estimator=KNeighborsClassifier(),␣
 ↪param_grid=params, cv=10, scoring='accuracy')

# Fit the grid search to the data
```

```
grid_search_knn.fit(X_train_resampled, y_train_resampled)

# Print the best parameters and the best score
print("Best Parameters: ", grid_search_knn.best_params_)
print("Best Score: ", grid_search_knn.best_score_)
```

```
Best Parameters:  {'metric': 'euclidean', 'n_neighbors': 1}
Best Score:  0.8568019391365616
```

[20]:
```
# Using fitted model to make predictions
test_predictions_knn = grid_search_knn.best_estimator_.predict(test)

# Calculate the accuracy of the model on the test data
test_accuracy = accuracy_score(y_test, test_predictions_knn)

# Print the test accuracy
print(f"Test Accuracy (KNN): {test_accuracy*100:.3f}")

# Classification Report
print(metrics.classification_report(y_test, test_predictions_knn))

draw_confusion_matrix(y_test, test_predictions_knn, ['Not Cancelled',␣
↪'Cancelled'])
```

```
Test Accuracy (KNN): 81.757
              precision    recall  f1-score   support

           0       0.86      0.83      0.85      8333
           1       0.76      0.80      0.78      5585

    accuracy                           0.82     13918
   macro avg       0.81      0.81      0.81     13918
weighted avg       0.82      0.82      0.82     13918
```



[ ]:
```
from sklearn.ensemble import RandomForestClassifier


# Define the parameters // refined
params = {
    'n_estimators': [500, 525, 550, 575, 600],  # Adjusted around 550
    'criterion': ['gini', 'entropy'],  # Keeping 'entropy' as a search option
    'max_depth': [45, 50, 55, 60],  # Adjusted around 50
    'min_samples_split': [2, 3, 4, 5],  # Adjusted around 3
    'min_samples_leaf': [1, 2, 3],  # Adjusted around 2
    'bootstrap': [False],  # Keeping 'False' as per your best results
    'max_features': ['sqrt', 'log2', None]  # Adding some more options around␣
  ↪'sqrt'
}


rf = RandomForestClassifier()

grid_search_rf = GridSearchCV(estimator=rf,param_grid=params, cv=10, n_jobs=-1)
```

```
# Run the grid search
grid_search_rf.fit(X_train_resampled, y_train_resampled)

# Print out the best parameters
print("Best parameters found: ", grid_search_rf.best_params_)
print("Highest accuracy found: ", grid_search_rf.best_score_)
```

[ ]:
```
# Testing the best Random Forest model
best_rf_model = grid_search_rf.best_estimator_

# Predict on the testing data
test_predictions_rf = best_rf_model.predict(test)

# Get the accuracy of the model
test_accuracy_rf = accuracy_score(y_test, test_predictions_rf)

print(f"Test Accuracy (Random Forest): {test_accuracy_rf * 100:.3f}")

# Check the classification report
print(metrics.classification_report(y_test, test_predictions_rf))

# Predict probabilities
probabilities_rf = best_rf_model.predict_proba(test)

# Probabilities for positive class
auc = roc_auc_score(y_test, probabilities_rf[:, 1])

print(f"AUC-ROC score for Random Forest is {auc}")

# Confusion Matrix
draw_confusion_matrix(y_test, test_predictions_rf, ['Not Cancelled',␣
 ↪'Cancelled'])
```

Best parameters found: {‘bootstrap’: False, ‘criterion’: ‘entropy’, ‘max_depth’: 50, ‘max_features’: ‘sqrt’, ‘min_samples_leaf’: 2, ‘min_samples_split’: 3, ‘n_estimators’: 550}

Test Accuracy (Random Forest): 85.731 precision recall f1-score support

```
       0       0.86      0.91      0.88      8333
       1       0.85      0.78      0.81      5585

accuracy                           0.86     13918
```

macro avg 0.86 0.84 0.85 13918 weighted avg 0.86 0.86 0.86 13918

AUC-ROC score for Random Forest is 0.9336623241115858

<Figure size 640x480 with 2 Axes>

[ ]:
```
# making bagging classifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

# make baseline model
base_estimator_1 = DecisionTreeClassifier(random_state=SEED)
# parameters // refined
params = {
    'n_estimators': [200, 250, 300, 350, 400],  # narrowed around 300
    'max_samples': [0.6, 0.65, 0.7, 0.75, 0.8],  # narrowed around 0.7
    'max_features': [0.6, 0.65, 0.7, 0.75, 0.8],  # narrowed around 0.7
    'bootstrap': [False],
    'bootstrap_features': [True, False],  # Adding False as an option
    'estimator': [base_estimator_1]
    # Keeping the DecisionTreeClassifier since it has been found best
}

bag_clf = BaggingClassifier(random_state=SEED)

hgs_bag = HalvingGridSearchCV(bag_clf, params, scoring='accuracy', cv=10,␣
 ↪n_jobs=-1)

# train model
hgs_bag.fit(X_train_resampled, y_train_resampled)

# print best parameters and score
print("Best parameters found: ", hgs_bag.best_params_)
print("Highest accuracy found: ", hgs_bag.best_score_)
```

[ ]:
```
best_bagging_model = hgs_bag.best_estimator_

test_predictions = best_bagging_model.predict(test)

# Get the accuracy of the model
test_accuracy = metrics.accuracy_score(y_test, test_predictions)

print(f"Test Accuracy: {test_accuracy * 100:.3f}")

# Check the classification report
print(metrics.classification_report(y_test, test_predictions))

# Probabilities for positive class
probabilities = best_bagging_model.predict_proba(test)
auc = roc_auc_score(y_test, probabilities[:, 1])

print(f"AUC-ROC score is {auc}")
```

```
draw_confusion_matrix(y_test, test_predictions,
                      ['Not Cancelled', 'Cancelled'])
```

Best parameters found: {'bootstrap': False, 'bootstrap_features': True, 'estimator': DecisionTreeClassifier(random_state=42), 'max_features': 0.7, 'max_samples': 0.7, 'n_estimators': 300}

Test Accuracy: 86.636 precision recall f1-score support

0	0.85	0.94	0.89	8333
1	0.90	0.76	0.82	5585
accuracy			0.87	13918

macro avg 0.87 0.85 0.86 13918 weighted avg 0.87 0.87 0.86 13918

AUC-ROC score is 0.9367644643117864

<Figure size 640x480 with 2 Axes>

The above two models weren't executed as they took too long after I tried expanding the parameter search and had to interrupt the kernel while executing, but I copied the outputs previously and put them into a markdown cell.

9 Making Predictions

[38]:
```
# Predictions using best random forest model
test_predictions_rf = best_rf_model.predict(hotel_transformed)

# Convert the prediction array into a dataframe with 'target' column
predictions_df = pd.DataFrame(test_predictions_rf, columns=['target'])

# Create 'index' column that contains the row number from 0 to␣
 ↪len(test_predictions)
predictions_df['index'] = range(len(test_predictions_rf))

# Reorder the columns so 'index' is first
predictions_df = predictions_df[['index', 'target']]

# Save the data frame to a .csv file without index column
predictions_df.to_csv('test_predictions.csv', index=False)
```

[39]:
```
# Predictions using bagging classifier
bagging_test_predictions = best_bagging_model.predict(hotel_transformed)

# Convert to dataframe
predictions_bagging_df = pd.DataFrame(bagging_test_predictions,␣
 ↪columns=['target'])

# Create 'index'
```

```
predictions_bagging_df['index'] = range(len(bagging_test_predictions))

# Save data
predictions_bagging_df.to_csv('bagging_test_predictions.csv', index=False)
```

[ ]: