Data Collection and Import

The dataset contains 10000 rows and 17 columns. These variables provide detailed information about each trip, including the origin and destination, booking method, and the target variable, Car_Cancellation, which indicates whether a trip was canceled. The data was imported using pandas.

In [250…

```
import pandas as pd
import numpy as np

df = pd.read_csv('/Users/patriciomartinez/Downloads/Taxi-cancellation-case.c
df
```

Out[250…

	row#	user_id	vehicle_model_id	package_id	travel_type_id	from_area_id	tc
0	1	17712	12	NaN	2	1021.0	
1	2	17037	12	NaN	2	455.0	
2	3	761	12	NaN	2	814.0	
3	4	868	12	NaN	2	297.0	
4	5	21716	28	NaN	2	1237.0	
...	...	...	...	...	...	...	
9995	9996	31877	12	2.0	3	293.0	
9996	9997	28305	12	1.0	3	1017.0	
9997	9998	24007	12	NaN	2	393.0	
9998	9999	33882	12	NaN	2	410.0	
9999	10000	5878	12	1.0	3	1314.0	

10000 rows × 19 columns

Question 1: How can a predictive model based on these data be used by Yourcabs.com?

The predictive model could be used to predict which bookings are likely to be canceled or identify whihch drivers could be more likely to cancel scheduled rides. By knowing this information, Yourcabs.com could take proactive measures to reduce the risk of cancellations. For example, they could assign backup drivers or offer incentives to drivers to reduce cancellations.

Question 2: How can a profiling model (identifying predictors that distinguish canceled/uncanceled trips) be used by Yourcabs.com?

The profiling model can help Yourcabs.com understand which factors are most strongly associated with canceled trips. By understanding these factors, Yourcabs.com can identify patterns in driver behavior, specifically the types of trips certain drivers are more likely to accept or cancel. If a particular driver tends to cancel certain types of trips, Yourcabs.com can adjust the types of bookings offered to those drivers in an effort to get them to make the trip.

Question 3: Explore, prepare, and transform the data to facilitate predictive modeling.

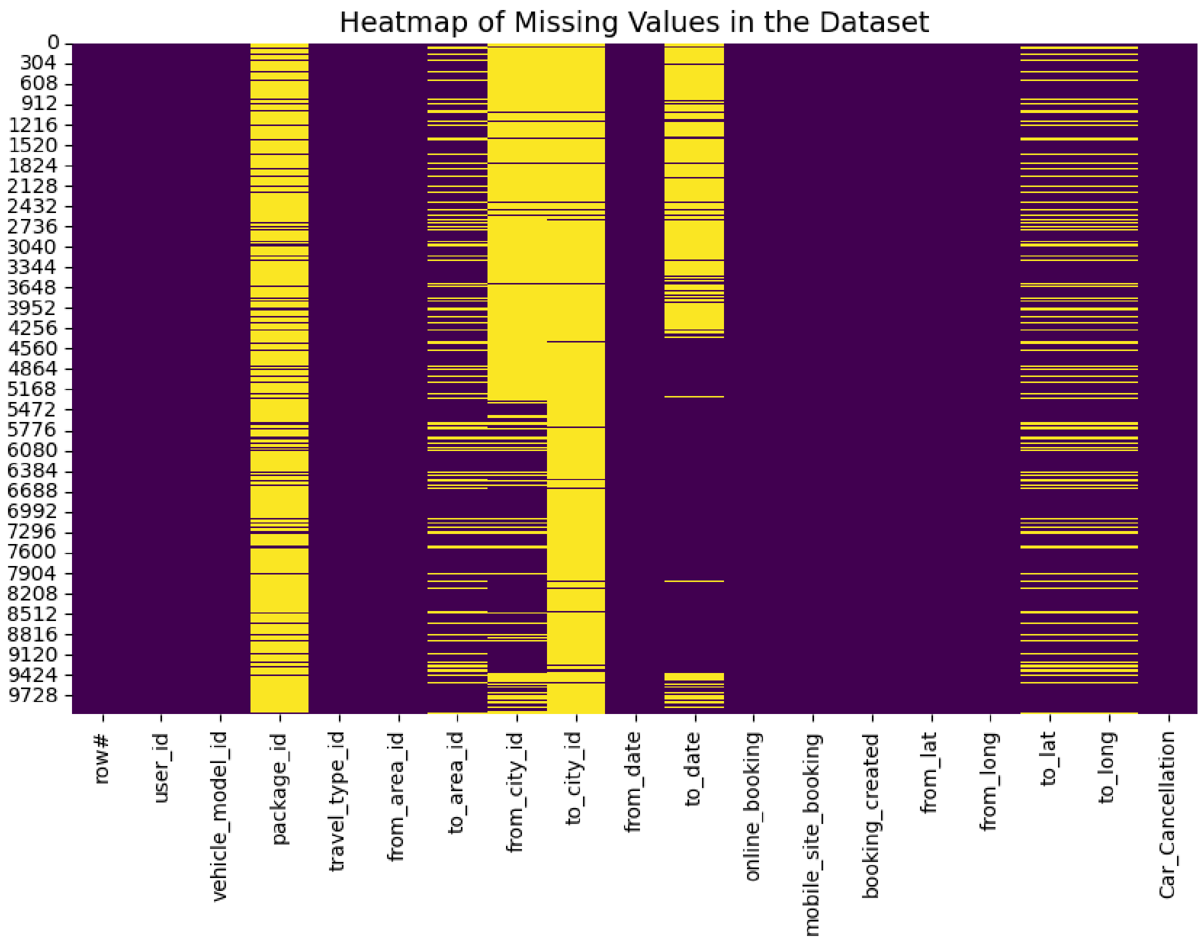
Data Pre-Processing

Several columns contained missing values, most notably package_id, to_area_id, from_city_id, to_city_id, to_date, to_lat and to_long. To handle these missing values, I imputed data for the geographic features using the median. Then for the categorical variables like area_id, I replace missing values with -1. I then extracted the features trip_distance, booking_lead_time, from_hour, from_day_of_week to try to get more valuable information for predictive modeling.

In [251…
```
missing_data_summary = df.isna().sum()
missing_data_summary[missing_data_summary > 0]
```

Out[251…
```
package_id      8248
from_area_id      15
to_area_id      2091
from_city_id    6294
to_city_id      9661
to_date         4178
from_lat          15
from_long         15
to_lat          2091
to_long         2091
dtype: int64
```

In [252…
```
import seaborn as sns
import matplotlib.pyplot as plt

# Visualizing missing values in the dataset
plt.figure(figsize=(10, 6))
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')

plt.title('Heatmap of Missing Values in the Dataset', fontsize=14)
plt.show()
```



In [253…

```
data_cleaned = df.copy()
data_cleaned = data_cleaned.drop(['package_id', 'to_city_id', 'row#', 'user_
# Impute missing latitude and longitude values with the median
data_cleaned['from_lat'].fillna(data_cleaned['from_lat'].median(), inplace=T
data_cleaned['from_long'].fillna(data_cleaned['from_long'].median(), inplace
data_cleaned['to_lat'].fillna(data_cleaned['to_lat'].median(), inplace=True)
data_cleaned['to_long'].fillna(data_cleaned['to_long'].median(), inplace=Tru

remaining_missing_after_median = data_cleaned.isnull().sum()
remaining_missing_after_median
```

Out[253…

```
vehicle_model_id          0
travel_type_id            0
from_area_id             15
to_area_id             2091
from_city_id           6294
from_date                 0
to_date                4178
online_booking            0
mobile_site_booking       0
booking_created           0
from_lat                  0
from_long                 0
to_lat                    0
to_long                   0
Car_Cancellation          0
dtype: int64
```

In [254…
```
data_cleaned['from_area_id'].fillna(-1, inplace=True)
data_cleaned['to_area_id'].fillna(-1, inplace=True)
data_cleaned['from_city_id'].fillna(-1, inplace=True)
most_frequent_to_date = data_cleaned['to_date'].mode()[0]
data_cleaned['to_date'].fillna(most_frequent_to_date, inplace=True)

remaining_missing_area = data_cleaned.isnull().sum()
remaining_missing_area
```

Out[254…
```
vehicle_model_id       0
travel_type_id         0
from_area_id           0
to_area_id             0
from_city_id           0
from_date              0
to_date                0
online_booking         0
mobile_site_booking    0
booking_created        0
from_lat               0
from_long              0
to_lat                 0
to_long                0
Car_Cancellation       0
dtype: int64
```

In [255…
```
# Specify the format explicitly for the given date columns
data_cleaned['from_date'] = pd.to_datetime(data_cleaned['from_date'], format
data_cleaned['to_date'] = pd.to_datetime(data_cleaned['to_date'], format='%m
data_cleaned['booking_created'] = pd.to_datetime(data_cleaned['booking_creat

# Extract features from 'from_date'
data_cleaned['from_day_of_week'] = data_cleaned['from_date'].dt.dayofweek  #
data_cleaned['from_hour'] = data_cleaned['from_date'].dt.hour  # Extract the

# Extract features from 'booking_created' and calculate lead time (differenc
data_cleaned['booking_lead_time'] = (data_cleaned['from_date'] - data_cleane

# Checking if there are any missing or erroneous lead time values
data_cleaned[['from_day_of_week', 'from_hour', 'booking_lead_time']].isnull(
```

Out[255…
```
from_day_of_week     0
from_hour            0
booking_lead_time    0
dtype: int64
```

In [256…
```
data_cleaned
```

Out [256…

	vehicle_model_id	travel_type_id	from_area_id	to_area_id	from_city_id	from_
0	12	2	1021.0	1323.0	-1.0	201 22::
1	12	2	455.0	1330.0	-1.0	201 12:4
2	12	2	814.0	393.0	-1.0	201 00::
3	12	2	297.0	212.0	-1.0	201 13:
4	28	2	1237.0	330.0	-1.0	201 16::
...	...	...	...	...	...	
9995	12	3	293.0	-1.0	-1.0	201: 07::
9996	12	3	1017.0	-1.0	-1.0	201: 13:
9997	12	2	393.0	788.0	-1.0	201: 01:4
9998	12	2	410.0	1026.0	-1.0	201: 14:(
9999	12	3	1314.0	-1.0	-1.0	201 19:4

10000 rows × 18 columns

In [257…

```
from geopy.distance import geodesic

# Function to calculate distance between two sets of lat/long
def calculate_distance(row):
    from_coords = (row['from_lat'], row['from_long'])
    to_coords = (row['to_lat'], row['to_long'])
    # Calculate the geodesic distance in kilometers between the two points
    return geodesic(from_coords, to_coords).kilometers

# Apply the function to calculate distance and create a new column 'trip_dis
data_cleaned['trip_distance'] = data_cleaned.apply(calculate_distance, axis=
```

```
# Display the first few rows to verify the trip distances have been added
data_cleaned[['from_lat', 'from_long', 'to_lat', 'to_long', 'trip_distance']
```

Out[257…

	from_lat	from_long	to_lat	to_long	trip_distance
0	13.028530	77.54625	12.869805	77.653211	21.048611
1	12.999874	77.67812	12.953434	77.706510	5.990251
2	12.908993	77.68890	13.199560	77.706880	32.204802
3	12.997890	77.61488	12.994740	77.607970	0.826682
4	12.926450	77.61206	12.858833	77.589127	7.883644

In [258…

```
# Set negative booking lead times to zero
data_cleaned['booking_lead_time'] = data_cleaned['booking_lead_time'].apply(

# Binning the 'trip_distance' into categories
data_cleaned['trip_distance_category'] = pd.cut(data_cleaned['trip_distance'
                                                labels=['short', 'medium', '

# Binning the 'booking_lead_time' into categories
data_cleaned['booking_lead_time_category'] = pd.cut(data_cleaned['booking_le
                                                    labels=['same_day', '1-2

# Display the first few rows to confirm the new features have been added
data_cleaned[['trip_distance', 'trip_distance_category', 'booking_lead_time'
```

Out[258…

	trip_distance	trip_distance_category	booking_lead_time	booking_lead_time_categ
0	21.048611	long	14.533333	same_
1	5.990251	medium	2.733333	same_
2	32.204802	long	12.233333	same_
3	0.826682	short	0.500000	same_
4	7.883644	medium	1.433333	same_

In [259…

```
data_cleaned = data_cleaned.dropna()
```

In [260…

```
# Check for negative or missing values in trip_distance and booking_lead_tim
print(data_cleaned[['trip_distance', 'booking_lead_time']].describe())
print(data_cleaned[['trip_distance', 'booking_lead_time']].isnull().sum())
```

```
       trip_distance  booking_lead_time
count    9990.000000        9990.000000
mean       15.471801          35.772990
std        10.169474          97.974999
min         0.011063           0.016667
25%         7.189447           3.033333
50%        12.274393           8.816667
75%        24.430018          18.612500
max        53.043130        1222.166667
trip_distance        0
booking_lead_time    0
dtype: int64
```

In [261…

```python
from sklearn.preprocessing import LabelEncoder

# Initialize the label encoder
le = LabelEncoder()

# List of categorical columns to encode
categorical_cols = ['trip_distance_category', 'booking_lead_time_category']

# Apply label encoding using .loc to avoid the warning
for col in categorical_cols:
    data_cleaned.loc[:, col] = le.fit_transform(data_cleaned[col])

# Check the transformed data
data_cleaned[['trip_distance_category', 'booking_lead_time_category']].head(
```

Out[261…

	trip_distance_category	booking_lead_time_category
0	0	2
1	1	2
2	0	2
3	2	2
4	1	2

In [262…

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, classification_report, roc_auc
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

# Define the features (X) and target (y)
X = data_cleaned.drop(['Car_Cancellation', 'from_date', 'to_date', 'booking_
y = data_cleaned['Car_Cancellation']  # Target

# Split the data into training and testing sets (70% training, 30% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rar

# Apply SMOTE to the training data
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

```
from sklearn.tree import DecisionTreeClassifier

# Initialize and fit the Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train_smote, y_train_smote)

# Make predictions on the test data
y_pred_dt = dt_model.predict(X_test)

# Evaluate the model performance
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)
class_report_dt = classification_report(y_test, y_pred_dt)
roc_auc_dt = roc_auc_score(y_test, dt_model.predict_proba(X_test)[:, 1])

# Display the results
print("Decision Tree Confusion Matrix:\n", conf_matrix_dt)
print("\nDecision Tree Classification Report:\n", class_report_dt)
print("\nDecision Tree ROC-AUC Score:", roc_auc_dt)
```

```
Decision Tree Confusion Matrix:
 [[2468  306]
 [ 163   60]]

Decision Tree Classification Report:
               precision    recall  f1-score   support

           0       0.94      0.89      0.91      2774
           1       0.16      0.27      0.20       223

    accuracy                           0.84      2997
   macro avg       0.55      0.58      0.56      2997
weighted avg       0.88      0.84      0.86      2997


Decision Tree ROC-AUC Score: 0.5793741371673548
```

Question 4: Fit several predictive models of your choice. Do they provide information on how the predictor variables relate to cancellations?

In [270…

```
from sklearn.ensemble import RandomForestClassifier

# Fit the Random Forest model to the SMOTE-balanced training data
rf_model_smote = RandomForestClassifier(random_state=42)
rf_model_smote.fit(X_train_smote, y_train_smote)

# Make predictions on the test data
y_pred_smote = rf_model_smote.predict(X_test)

# Evaluate the model performance with SMOTE
conf_matrix_smote = confusion_matrix(y_test, y_pred_smote)
class_report_smote = classification_report(y_test, y_pred_smote)
roc_auc_smote = roc_auc_score(y_test, rf_model_smote.predict_proba(X_test)[:

# Display the results
print("Random Forest Confusion Matrix:\n", conf_matrix_smote)
print("\nRandom Forest Classification Report:\n", class_report_smote)
```

```
print("\nRandom Forest ROC-AUC Score:", roc_auc_smote)

# Feature importance
importances = pd.Series(rf_model_smote.feature_importances_, index=X_train_s
print("\nFeature Importance:\n", importances)
```

```
Random Forest Confusion Matrix:
 [[2668  106]
 [ 184   39]]

Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.96      0.95      2774
           1       0.27      0.17      0.21       223

    accuracy                           0.90      2997
   macro avg       0.60      0.57      0.58      2997
weighted avg       0.89      0.90      0.89      2997


Random Forest ROC-AUC Score: 0.7488781154926754

Feature Importance:
 from_city_id                  0.161881
booking_lead_time            0.111044
from_lat                     0.086079
from_long                    0.085334
from_area_id                 0.076970
to_long                      0.076500
to_area_id                   0.075016
trip_distance                0.074252
to_lat                       0.072693
from_hour                    0.055547
from_day_of_week             0.036793
vehicle_model_id             0.031434
trip_distance_category       0.021459
online_booking               0.012769
travel_type_id               0.011469
booking_lead_time_category   0.007956
mobile_site_booking          0.002803
dtype: float64
```

In [265…

```
from sklearn.neural_network import MLPClassifier

# Initialize and fit the Neural Network model
nn_model = MLPClassifier(random_state=42, max_iter=500)
nn_model.fit(X_train_smote, y_train_smote)

# Make predictions on the test data
y_pred_nn = nn_model.predict(X_test)

# Evaluate the model performance
conf_matrix_nn = confusion_matrix(y_test, y_pred_nn)
class_report_nn = classification_report(y_test, y_pred_nn)
roc_auc_nn = roc_auc_score(y_test, nn_model.predict_proba(X_test)[:, 1])
```

```
# Display the results
print("Neural Network Confusion Matrix:\n", conf_matrix_nn)
print("\nNeural Network Classification Report:\n", class_report_nn)
print("\nNeural Network ROC-AUC Score:", roc_auc_nn)
```

```
Neural Network Confusion Matrix:
 [[2032  742]
 [ 117  106]]

Neural Network Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.73      0.83      2774
           1       0.12      0.48      0.20       223

    accuracy                           0.71      2997
   macro avg       0.54      0.60      0.51      2997
weighted avg       0.88      0.71      0.78      2997


Neural Network ROC-AUC Score: 0.647894769173071
```

In [ ]:

```
from sklearn.linear_model import LogisticRegression

# Initialize and fit the Logistic Regression model
lr_model = LogisticRegression(random_state=42, max_iter=500)
lr_model.fit(X_train_smote, y_train_smote)

# Make predictions on the test data
y_pred_lr = lr_model.predict(X_test)

# Evaluate the model performance
conf_matrix_lr = confusion_matrix(y_test, y_pred_lr)
class_report_lr = classification_report(y_test, y_pred_lr)
roc_auc_lr = roc_auc_score(y_test, lr_model.predict_proba(X_test)[:, 1])

# Display the results
print("Logistic Regression Confusion Matrix:\n", conf_matrix_lr)
print("\nLogistic Regression Classification Report:\n", class_report_lr)
print("\nLogistic Regression ROC-AUC Score:", roc_auc_lr)
```

In [271…

```
coef = pd.Series(lr_model.coef_[0], index=X_train_smote.columns).sort_values
print("\nLogistic Regression Coefficients:\n", coef)
```

```
Logistic Regression Coefficients:
 from_long                    0.346215
online_booking               0.280096
travel_type_id               0.111388
booking_lead_time_category   0.094284
from_city_id                 0.041631
from_hour                    0.035531
to_area_id                   0.000532
from_area_id                -0.000014
booking_lead_time           -0.000288
vehicle_model_id            -0.009281
from_day_of_week            -0.013837
trip_distance               -0.146598
to_long                     -0.198593
to_lat                      -0.252293
mobile_site_booking         -0.439773
from_lat                    -0.482732
trip_distance_category      -1.637976
dtype: float64
```

The Random Forest and the Logistic Regression model provide coefficients and importance scores that tell us how influential each feature is in predicting the cancellations. In the Logistic Regression model, from_long (0.35) tells us that as the longitude of the origin increases, the chance of cancellation increases while trip_distance_category (-1.64) shows that longer trips are likely to be cancelled. In the Random Forest model, from_city_id (0.16), booking_lead_time (0.11) and from_lat (0.09) were identified as the most important predictors. The Neural Network is less interpretable than the other two models.

Question 5: Report the predictive performance of your model in terms of error rates (the confusion matrix). How well does the model perform? Can the model be used in practice?

The logistic regression model had 1854 true positives, 125 true negatives, 920 false positives, and 98 false negatives. The logistic regression model achieved an overall accuracy of 66%, with a recall of 56% for class 1 (canceled trips) and a precision of 12%. The most impactful positive predictor was from_long with a coefficient of 0.35, indicating that as the longitude of the origin increases, the likelihood of cancellation increases.

The neural network model achieved an overall accuracy of 71%. Its confusion matrix shows 2032 true positives and 106 true negatives, with 742 false positives, giving it a recall of 48% for class 1 but a low precision of 12%. While it did better in identifying canceled trips than logistic regression, it still struggled with a high number of false positives, resulting in a low f1-score for class 1. The ROC-AUC score of 0.65 indicates moderate performance.

The random forest model had the highest accuracy at 90%. The confusion matrix shows 2668 true positives and 39 true negatives, but like the other models, it struggled with a high number of false negatives (184), leading to a recall of only 17% for class 1 (canceled trips) and a precision of 27%. The random forest provided insights into feature importance, with from_city_id (0.16), booking_lead_time (0.11), and from_lat (0.09) being the most significant predictors of cancellations.

Out of all of the models, the random forest would be the better option if it were to be used in a practical setting. However, the model would still need some fine-tuning to improve its performance.

Question 6: Examine the predictive performance of your model in terms of ranking (lift). How well does the model perform? Can the model be used in practice?

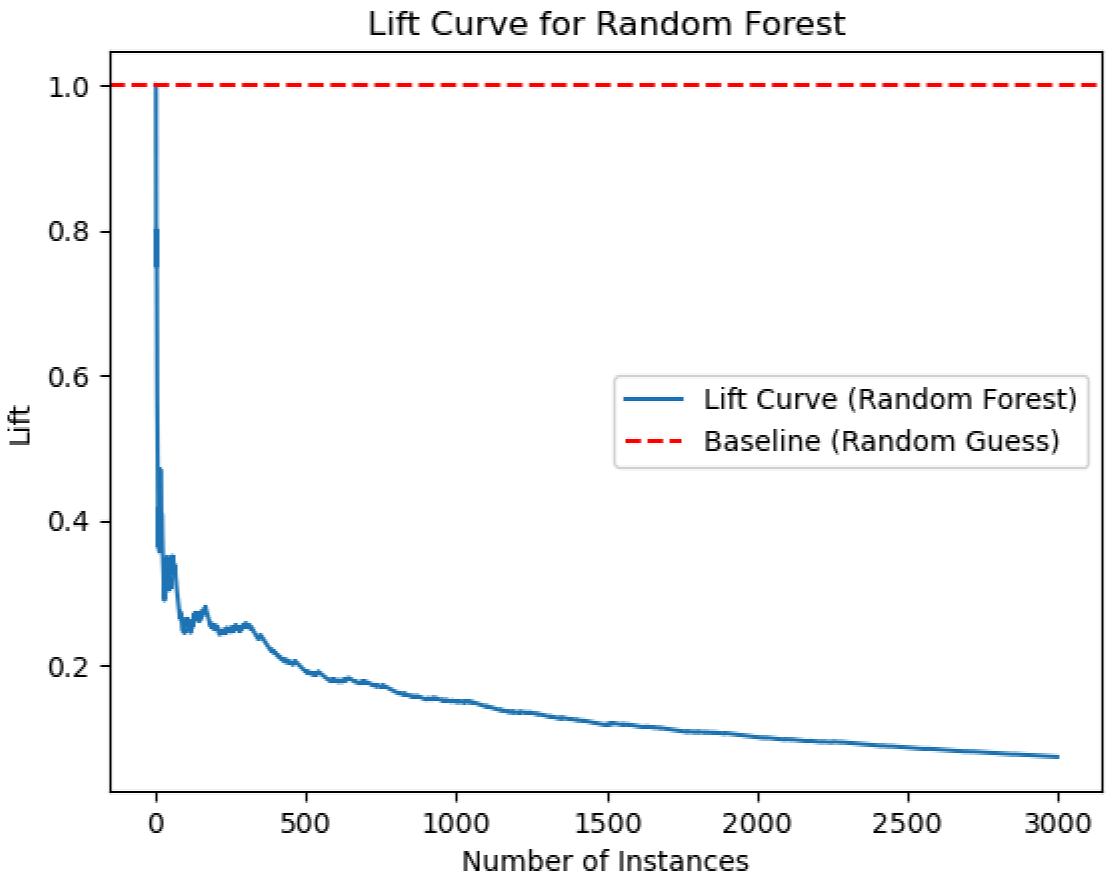
In [272…

```
# Predict probabilities for class 1 (cancellations)
y_probs_rf = rf_model_smote.predict_proba(X_test)[:, 1]

# Create a DataFrame with actual values and predicted probabilities
lift_df_rf = pd.DataFrame({
    'Predicted_Prob': y_probs_rf,
    'Actual': y_test
})

# Sort by predicted probability in descending order
lift_df_rf = lift_df_rf.sort_values(by='Predicted_Prob', ascending=False)

# Calculate cumulative cancellations and cumulative lift
lift_df_rf['Cumulative_Cancellations'] = lift_df_rf['Actual'].cumsum()
lift_df_rf['Lift'] = lift_df_rf['Cumulative_Cancellations'] / (np.arange(ler

# Display the first few rows to see how the lift is calculated
print(lift_df_rf.head())
```

```
      Predicted_Prob  Actual  Cumulative_Cancellations  Lift
8087            0.93       1                         1  1.00
8748            0.92       1                         2  1.00
9128            0.90       1                         3  1.00
415             0.90       0                         3  0.75
3408            0.89       1                         4  0.80
```

In [274…

```
import matplotlib.pyplot as plt

# Plot the lift curve
plt.plot(np.arange(len(lift_df_rf)), lift_df_rf['Lift'], label='Lift Curve (
plt.axhline(y=1, color='r', linestyle='--', label='Baseline (Random Guess)')
plt.title('Lift Curve for Random Forest')
plt.xlabel('Number of Instances')
plt.ylabel('Lift')
plt.legend()
plt.show()
```



Based on the lift of the Random Forest Model, the model performs worse than random guessing. The curve remains below the baseline and it is not effectively ranking the canceled trips higher in probability compared to non-canceled trips. Therefore the model is not suitable for practical use.

Question 7: Briefly explain, in two to three paragraphs, the business objective, the data mining models used, why they were used, the model results, and your recommendations to your non-technical stakeholder team.

The business objective was to help Yourcabs.com identify factors that can contribute to trip cancellations and develop predictive models that could flag high-risk trips. I applied four models for this task: Logistic Regression, Neural Networks, Random Forests, and Decision Trees. Logistic regression and Decision Trees was choses to help understand how predictors can influence cancellations. Neural networks was selected to try to capture non-linear relationships between features. Random Forests was chosen because of its robustness and its ability to provide feature importance. The Random Forest model was better than the other models mentioned in terms of overall accuracy (90%), but all models struggled to identify canceled trips. The features that were indicated to be more important were the location-based features and booking lead time. However, the lift analysis indicated that the Random Forest model is not suitable for practical use.

I recommend further tuning of the models and additional feature engineering. It may also be beneficial to revisit the data collection process to ensure that more details and relevant data is captured.