+ Code + Text

Decision Tree Classifier

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv('booking.csv')
df.head()
```

	Booking_ID	number of adults	number of children	number of weekend nights	number of week nights	type of meal	car parking space	room type	lead time	market segment type	repeated	P-C	P-not-C	average price	special requests
0	INN00001	1	1	2	5	Meal Plan 1	0	Room_Type 1	224	Offline	0	0	0	88.00	0
1	INN00002	1	0	1	3	Not Selected	0	Room_Type 1	5	Online	0	0	0	106.68	1
2	INN00003	2	1	1	3	Meal Plan 1	0	Room_Type 1	1	Online	0	0	0	50.00	0
3	INN00004	1	0	0	2	Meal Plan 1	0	Room_Type 1	211	Online	0	0	0	100.00	1
4	INN00005	1	0	1	2	Not	0	Room_Type	48	Online	0	0	0	77.00	0

Next steps: Generate code with df View recommended plots New interactive sheet

```
df.shape
```

```
(36285, 17)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36285 entries, 0 to 36284
Data columns (total 17 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Booking_ID                36285 non-null  object
 1   number of adults          36285 non-null  int64
 2   number of children        36285 non-null  int64
 3   number of weekend nights  36285 non-null  int64
 4   number of week nights     36285 non-null  int64
 5   type of meal              36285 non-null  object
 6   car parking space         36285 non-null  int64
 7   room type                 36285 non-null  object
 8   lead time                 36285 non-null  int64
 9   market segment type       36285 non-null  object
 10  repeated                  36285 non-null  int64
 11  P-C                       36285 non-null  int64
 12  P-not-C                   36285 non-null  int64
 13  average price             36285 non-null  float64
 14  special requests          36285 non-null  int64
 15  date of reservation       36285 non-null  object
 16  booking status            36285 non-null  object
dtypes: float64(1), int64(10), object(6)
memory usage: 4.7+ MB
```

```
df.describe()
```

	number of adults	number of children	number of weekend nights	number of week nights	car parking space	lead time	repeated	P-C	P-not-C	
count	36285.000000	36285.000000	36285.000000	36285.000000	36285.000000	36285.000000	36285.000000	36285.000000	36285.000000	362
mean	1.844839	0.105360	0.810693	2.204602	0.030977	85.239851	0.025630	0.023343	0.153369	
std	0.518813	0.402704	0.870590	1.410946	0.173258	85.938796	0.158032	0.368281	1.753931	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	2.000000	0.000000	0.000000	1.000000	0.000000	17.000000	0.000000	0.000000	0.000000	
50%	2.000000	0.000000	1.000000	2.000000	0.000000	57.000000	0.000000	0.000000	0.000000	
75%	2.000000	0.000000	2.000000	3.000000	0.000000	126.000000	0.000000	0.000000	0.000000	

```
# Booking Id column is not required therefore drop it.
df = df.drop('Booking_ID', axis=1)
df.head()
```

	number of adults	number of children	number of weekend nights	number of week nights	type of meal	car parking space	room type	lead time	market segment type	repeated	P-C	P-not-C	average price	special requests	date of reservation
0	1	1	2	5	Meal Plan 1	0	Room_Type 1	224	Offline	0	0	0	88.00	0	10/2/2015
1	1	0	1	3	Not Selected	0	Room_Type 1	5	Online	0	0	0	106.68	1	11/6/2018
2	2	1	1	3	Meal Plan 1	0	Room_Type 1	1	Online	0	0	0	50.00	0	2/28/2018
3	1	0	0	2	Meal Plan 1	0	Room_Type 1	211	Online	0	0	0	100.00	1	5/20/2017
4	1	0	1	2	Not	0	Room_Type	48	Online	0	0	0	77.00	0	4/11/2018

Next steps: Generate code with df View recommended plots New interactive sheet

```
# Encoding all categorical columns
from sklearn.preprocessing import LabelEncoder
LE = LabelEncoder()


df['type of meal'] = LE.fit_transform(df['type of meal'])
print(df['type of meal'].unique())
```

```
[0 3 1 2]
```

```
print(LE.inverse_transform([0, 3, 2 ,1]))
```

```
['Meal Plan 1' 'Not Selected' 'Meal Plan 3' 'Meal Plan 2']
```

```
df['room type'] = LE.fit_transform(df['room type'])
print(df['room type'].unique())
```

```
[0 3 1 5 4 6 2]
```

```
print(LE.inverse_transform([0, 3, 1, 5, 4, 6, 2]))
```

```
['Room_Type 1' 'Room_Type 4' 'Room_Type 2' 'Room_Type 6' 'Room_Type 5'
 'Room_Type 7' 'Room_Type 3']
```

```
df['market segment type'] = LE.fit_transform(df['market segment type'])
print(df['market segment type'].unique())
```

```
[3 4 2 0 1]
```

```
print(LE.inverse_transform([3, 4, 2, 0, 1]))
```

```
['Offline' 'Online' 'Corporate' 'Aviation' 'Complementary']
```

```
df['booking status'] = LE.fit_transform(df['booking status'])
print(df['booking status'].unique())
```

```
[1 0]
```

```
print(LE.inverse_transform([1, 0]))
```

```
['Not_Canceled' 'Canceled']
```

```
df.head()
```

	number of adults	number of children	number of weekend nights	number of week nights	type of meal	car parking space	room type	lead time	market segment type	repeated	P-C	P-not-C	average price	special requests	date of reservation	booking status
0	1	1	2	5	0	0	0	224	3	0	0	0	88.00	0	10/2/2015	1
1	1	0	1	3	3	0	0	5	4	0	0	0	106.68	1	11/6/2018	1
2	2	1	1	3	0	0	0	1	4	0	0	0	50.00	0	2/28/2018	0
3	1	0	0	2	0	0	0	211	4	0	0	0	100.00	1	5/20/2017	0

Next steps: Generate code with df View recommended plots New interactive sheet

```
df = df.drop('date of reservation', axis =1)
df.head()
```

	number of adults	number of children	number of weekend nights	number of week nights	type of meal	car parking space	room type	lead time	market segment type	repeated	P-C	P-not-C	average price	special requests	booking status
0	1	1	2	5	0	0	0	224	3	0	0	0	88.00	0	1
1	1	0	1	3	3	0	0	5	4	0	0	0	106.68	1	1
2	2	1	1	3	0	0	0	1	4	0	0	0	50.00	0	0
3	1	0	0	2	0	0	0	211	4	0	0	0	100.00	1	0

Next steps: Generate code with df View recommended plots New interactive sheet

```
# Data Seperation as features and target columns
x = df.drop(columns = ['booking status'])
y = df['booking status']
```

```
# Data splitting in train test split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=4)
```

```
x_train.shape
```

```
(25399, 14)
```

```
x_test.shape
```

```
(10886, 14)
```

```
y_train.shape
```

```
(25399,)
```

```
y_test.shape
```

```
(10886,)
```

```
# Decision Tree Classification Modeling
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
# Model Fitting
model = DecisionTreeClassifier(random_state=5, max_depth=3, criterion='gini')
```

```
model.fit(x_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3, random_state=5)
```

```
# Estimating model performance
training_score = model.score(x_train, y_train) # training score
testing_score = model.score(x_test, y_test) # testing score
print('Training score =', training_score)
print('Testing score =', testing_score)
```

```
Training score = 0.7844797039253514
Testing score = 0.7888113172882601
```

```
# Training score is nearly equal to testing score
# both scores are more than 78%
# Therefore model is good fit
```

```
y_pred_train = model.predict(x_train)
y_pred_test = model.predict(x_test)
```

```
print(classification_report(y_pred_train, y_train)) # report on training data
```

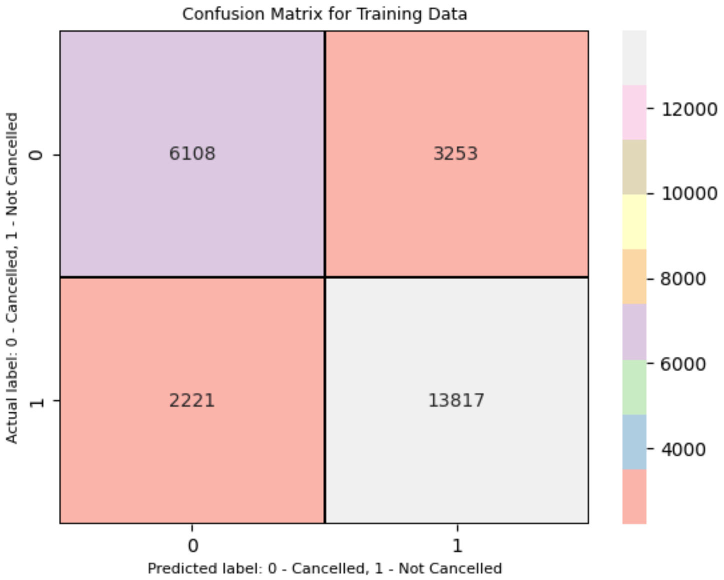
```
              precision    recall  f1-score   support

           0       0.73      0.65      0.69      9361
           1       0.81      0.86      0.83     16038

    accuracy                           0.78     25399
   macro avg       0.77      0.76      0.76     25399
weighted avg       0.78      0.78      0.78     25399
```

```
print(classification_report(y_pred_test, y_test)) # report on testing data
```

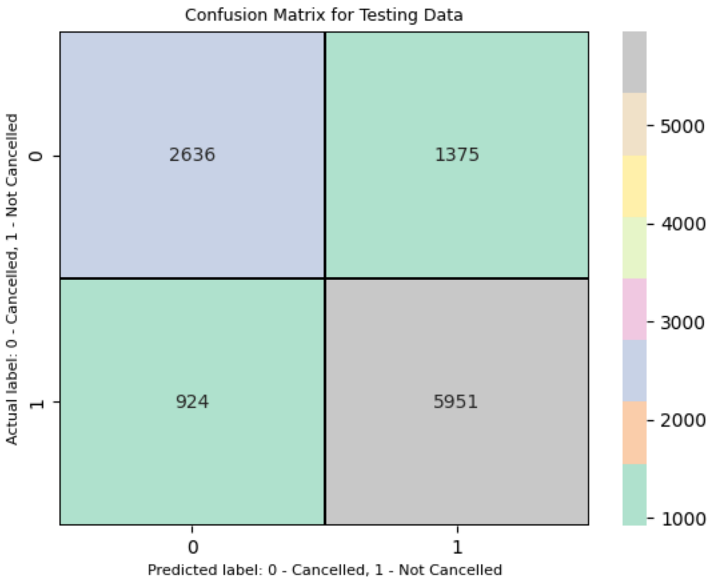
```
              precision    recall  f1-score   support

           0       0.74      0.66      0.70      4011
           1       0.81      0.87      0.84      6875

    accuracy                           0.79     10886
   macro avg       0.78      0.76      0.77     10886
weighted avg       0.79      0.79      0.79     10886
```

```
# Calculating confusion matrix
cm1 = confusion_matrix(y_pred_train, y_train) # confusion matrix for training
cm2 = confusion_matrix(y_pred_test, y_test) # confusion matrix for testing
```

```
# Plotting confusion matrix
sns.heatmap(cm1, annot= True, linewidths = 0.3, linecolor='black', cmap='Pastel1', fmt = 'd')
plt.xlabel('Predicted label: 0 - Cancelled, 1 - Not Cancelled', fontsize=8)
plt.ylabel('Actual label: 0 - Cancelled, 1 - Not Cancelled', fontsize=8)
plt.title('Confusion Matrix for Training Data', fontsize=9)
plt.show()
```



```
sns.heatmap(cm2, annot= True, linewidths = 0.3, linecolor='black', cmap='Pastel2', fmt = 'd')
plt.xlabel('Predicted label: 0 - Cancelled, 1 - Not Cancelled', fontsize=8)
plt.ylabel('Actual label: 0 - Cancelled, 1 - Not Cancelled', fontsize=8)
plt.title('Confusion Matrix for Testing Data', fontsize=9)
plt.show()
```

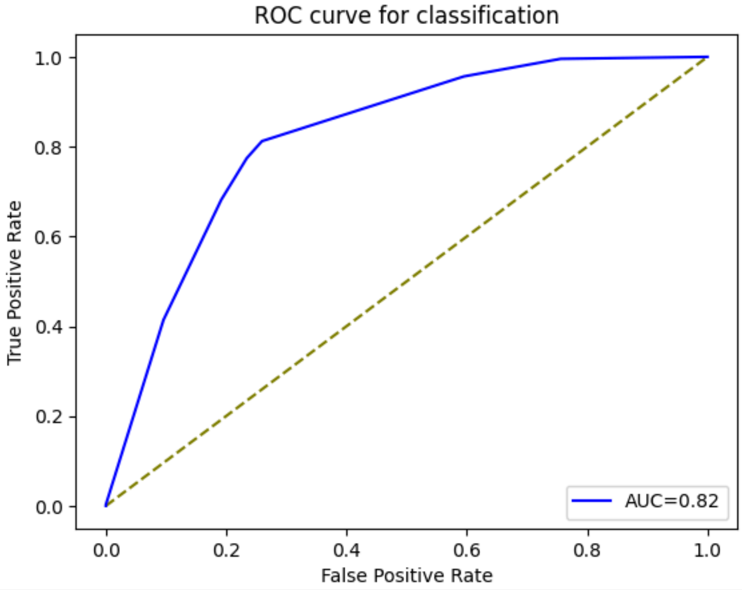


```
from sklearn.metrics import roc_curve, roc_auc_score


y_pred_prob = model.predict_proba(x_test)


auc = roc_auc_score(y_test, y_pred_prob[:, 1], multi_class='ovr')
print('Area under curve =', auc)
```

```
Area under curve = 0.820763530384317
```

Generated code may be subject to a license | arita37/dsa2_code

```
# Plotting Auc graph
fpr, tpr, thresh = roc_curve(y_test, y_pred_prob[:, 1], pos_label=1)
plt.plot([0,1],[0,1], linestyle='--', color='olive')
plt.plot(fpr, tpr, linestyle='-', color='blue', label='AUC={:.2f}'.format(auc))
plt.title('ROC curve for classification')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()
```



Start coding or generate with AI.