Booking Cancellation Prediction Dashboard

Predict whether a booking will be cancelled and the chances of cancellation based on input features.

Lead Time

100

Market Segment Type

Online

Average Price per Room

149

Number of Adults

0

Number of Weekend Nights

2

Number of Week Nights

0

Number of Special Requests

0

Arrival Month

10

Required Car Parking Space

No

Clear

Submit

Will the booking be cancelled?

No

Chances of Cancellation (%)

20.00%

Flag

Use via API · Built with Gradio

```
# libraries to help with reading and manipulating data
import pandas as pd
import numpy as np

# libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# removing the limit for the number of displayed columns
pd.set_option("display.max_columns", None)
# setting the limit for the number of displayed rows
pd.set_option("display.max_rows", 200)
# setting the precision of floating numbers to 2 decimal points
pd.set_option("display.float_format", lambda x: "%.2f" % x)

# library to split data
from sklearn.model_selection import train_test_split

# library to build AI model
from sklearn.tree import DecisionTreeClassifier

# library to tune AI model
from sklearn.model_selection import GridSearchCV

from sklearn.metrics import (
    f1_score,
    accuracy_score,
    recall_score,
    precision_score,
    confusion_matrix,
    roc_auc_score,
    ConfusionMatrixDisplay,
)

# libraries to evaluate the AI model
from sklearn.metrics import f1_score, make_scorer

# libraries to deploy the AI model
import os
import joblib
import gradio as gr
```

```
#importing using Google Colab
import pandas as pd

# # connecting Google Drive to this Python notebook
from google.colab import drive
drive.mount('/content/drive')

#loading the data files into pandas dataframe
hoteldata = pd.read_csv('/content/drive/MyDrive/MLS 1 Notebook/INNHotelsGroup.csv')
```

```
Mounted at /content/drive
```

```
# importing 10records
hoteldata.sample(10, random_state=10)
```

	lead_time	market_segment_type	no_of_special_requests	avg_price_per_room	no_of_adults	no_of_weekend_nights	arrival_date	req
18212	103	Offline	0	115.00	1	0	2018-04-19	
3267	86	Online	2	90.00	2	1	2018-11-04	
28472	20	Online	1	92.67	2	1	2018-12-09	
34178	22	Online	1	126.04	2	4	2018-12-28	
35404	109	Online	4	105.21	2	2	2018-07-06	
31551	30	Offline	0	127.15	1	0	2018-10-28	
30927	192	Offline	0	100.00	2	0	2018-08-25	
30473	278	Online	0	103.95	2	2	2018-07-16	
27445	0	Online	0	85.93	1	0	2018-05-12	
33154	37	Online	1	92.31	1	4	2018-12-15	

```
# creating a copy of the data to avoid any changes to original data
data = hoteldata.copy()
```

```
# checking the statistical summary of the data
data.describe().T
```

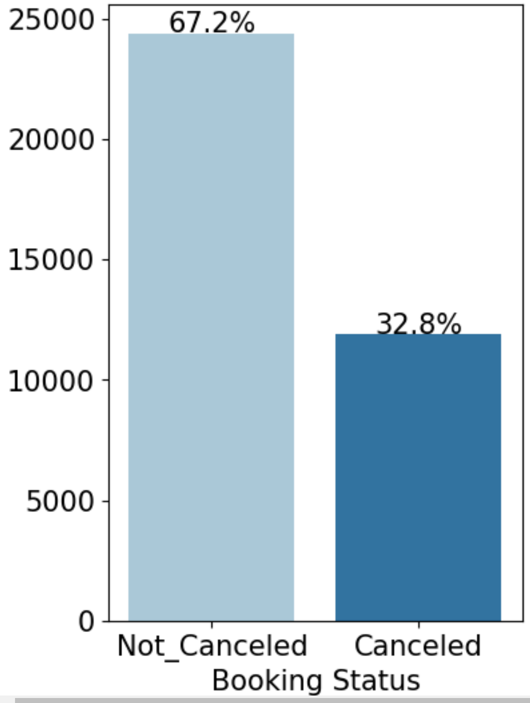
	count	mean	std	min	25%	50%	75%	max
lead_time	36275.0	85.232557	85.930817	0.0	17.0	57.00	126.0	443.0
no_of_special_requests	36275.0	0.619655	0.786236	0.0	0.0	0.00	1.0	5.0
avg_price_per_room	36275.0	103.423539	35.089424	0.0	80.3	99.45	120.0	540.0
no_of_adults	36275.0	1.844962	0.518715	0.0	2.0	2.00	2.0	4.0
no_of_weekend_nights	36275.0	0.810724	0.870644	0.0	0.0	1.00	2.0	7.0
required_car_parking_space	36275.0	0.030986	0.173281	0.0	0.0	0.00	0.0	1.0
no_of_week_nights	36275.0	2.204300	1.410905	0.0	1.0	2.00	3.0	17.0

```
# defining a function to create a bar graph with percentage values

def labeled_barplot(data, feature, perc=False, n=None):

    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all levels)
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 2, 6))
    else:
        plt.figure(figsize=(n + 2, 6))

    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n],
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=15,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage

    # increase the size of x-axis and y-axis scales
    ax.tick_params(axis='x', labelsize=15)
    ax.tick_params(axis='y', labelsize=15)

    # setting axis labels
    ax.set_xlabel(feature.replace('_', ' ').title(), fontsize=15)
    ax.set_ylabel('')

    # show the plot
    plt.show(labeled_barplot)
```

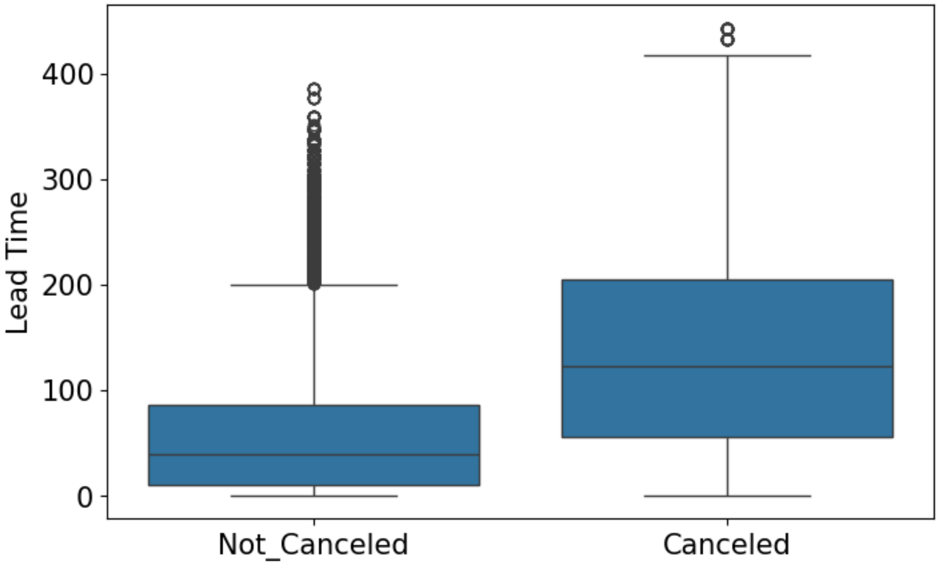
```
def stacked_barplot(data, predictor, target):

    """
    Print the category counts and plot a stacked bar chart

    data: dataframe
    predictor: independent variable
    target: target variable
    """
    count = data[predictor].nunique()
    sorter = data[target].value_counts().index[-1]
    tab1 = pd.crosstab(data[predictor], data[target], margins=True)
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(data[predictor], data[target], normalize="index")
    tab.plot(kind="bar", stacked=True, figsize=(count + 5, 5))
    plt.legend(loc="upper left", fontsize=12, bbox_to_anchor=(1, 1))

    # setting the formatting for x-axis
    plt.xticks(fontsize=15, rotation=0)
    plt.xlabel(predictor.replace('_', ' ').title(), fontsize=15)
    # setting the formatting for y-axis
    plt.yticks(fontsize=15)
    # show the plot
    plt.show(stacked_barplot)
```

```
# visualizing the number of cancelled bookings
import matplotlib.pyplot as plt
import seaborn as sns
labeled_barplot(data, "booking_status", perc=True)
```

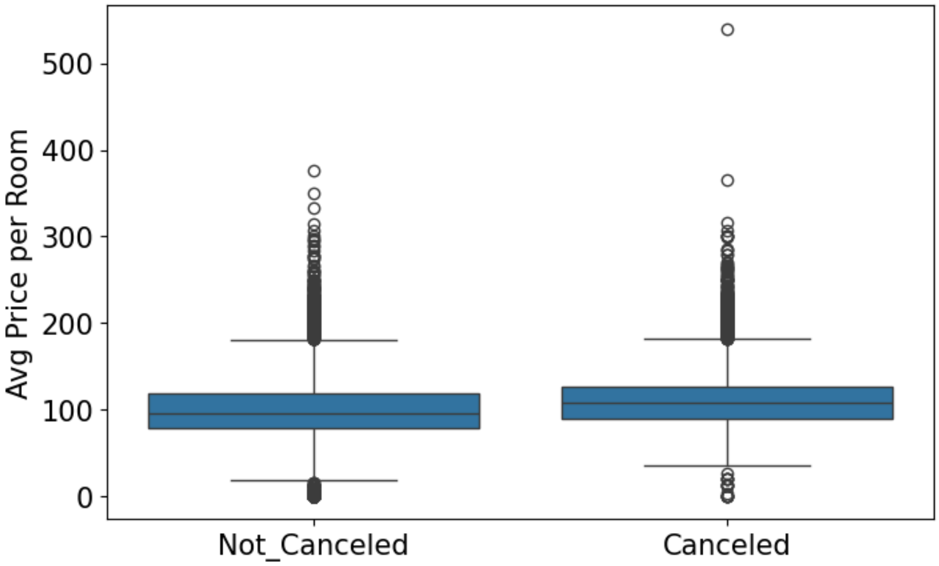
```
<ipython-input-8-d025e5bcb9f7>:21: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  ax = sns.countplot(
```



```
# visualizing the relationship between lead time and booking cancellation
plt.figure(figsize=(8, 5))
sns.boxplot(data=data, x="booking_status", y="lead_time")
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.xlabel('')
plt.ylabel('Lead Time', fontsize=15);
```

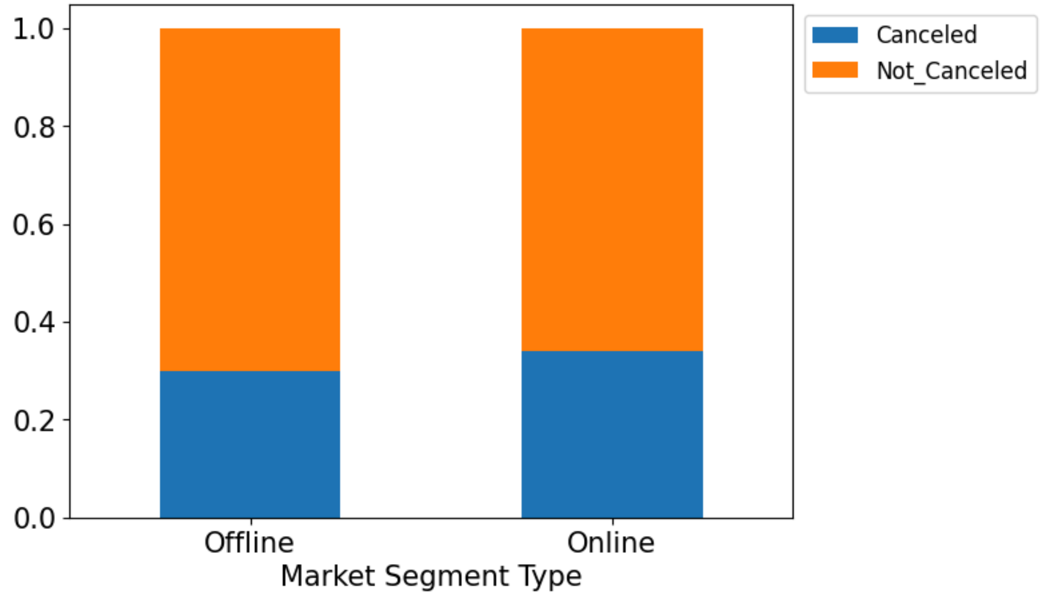


```
# visualizing the relationship between avg room price and booking cancellation
plt.figure(figsize=(8, 5))
sns.boxplot(data=data, x="booking_status", y="avg_price_per_room")
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.xlabel('')
plt.ylabel('Avg Price per Room', fontsize=15);
```



```
stacked_barplot(data, "market_segment_type", "booking_status")
```

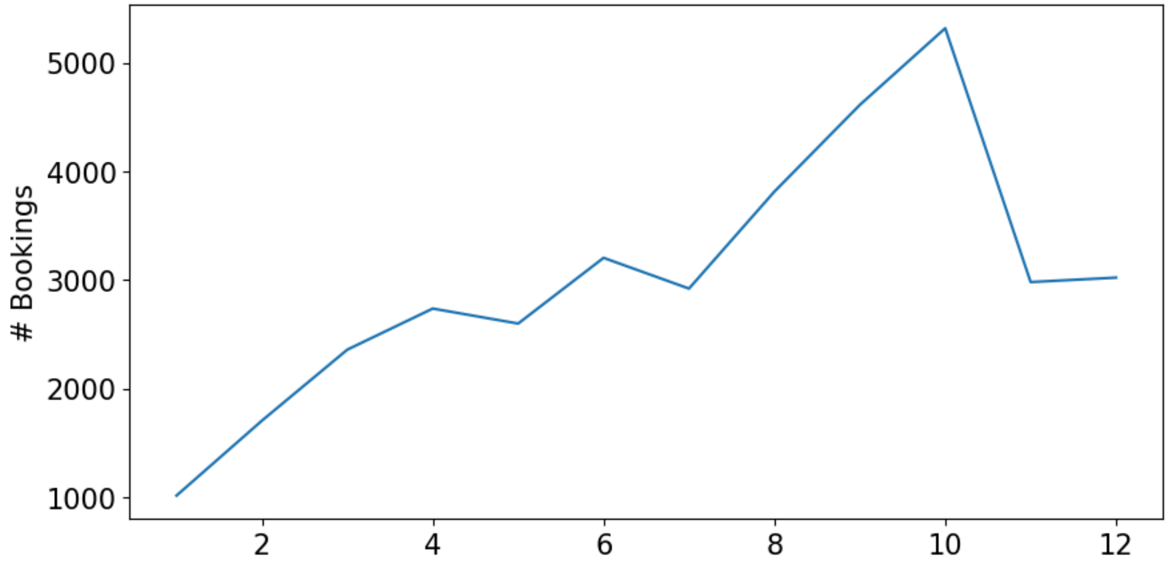
```
booking_status       Canceled  Not_Canceled    All
market_segment_type
Offline                  3153          7375  10528
Online                   8732         17015  25747
All                     11885         24390  36275
------------------------------------------------------------------------------------------------------------------------
```



```
# converting the 'arrival_date' column to datetime type
data['arrival_date'] = pd.to_datetime(data['arrival_date'])

# extracting month from 'arrival_date'
data['arrival_month'] = data['arrival_date'].dt.month


# grouping the data on arrival months and extracting the count of bookings
monthly_data = data.groupby(["arrival_month"])["booking_status"].count().to_frame().reset_index()
monthly_data.columns = ['Month', 'Bookings']
monthly_data
```

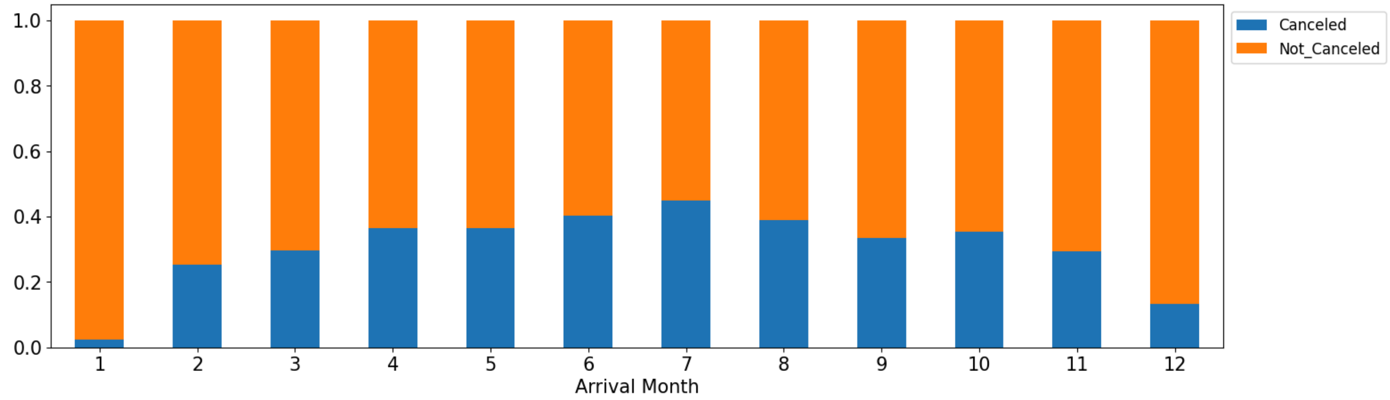
	Month	Bookings
0	1	1014
1	2	1704
2	3	2358
3	4	2736
4	5	2598
5	6	3203
6	7	2920
7	8	3813
8	9	4611
9	10	5317
10	11	2980
11	12	3021

```
# visualizing the trend of number of bookings across months
plt.figure(figsize=(10, 5))
sns.lineplot(data=monthly_data, x="Month", y="Bookings")
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.xlabel('')
plt.ylabel('# Bookings', fontsize=15);
```



```
##Let's check the percentage of bookings canceled in each month
stacked_barplot(data, "arrival_month", "booking_status")
```

```
booking_status  Canceled  Not_Canceled    All
arrival_month
1                     24           990   1014
2                    430          1274   1704
3                    700          1658   2358
4                    995          1741   2736
5                    948          1650   2598
6                   1291          1912   3203
7                   1314          1606   2920
8                   1488          2325   3813
9                   1538          3073   4611
10                  1880          3437   5317
11                   875          2105   2980
12                   402          2619   3021
All                11885         24390  36275
------------------------------------------------------------------------------------------------------------------------
```



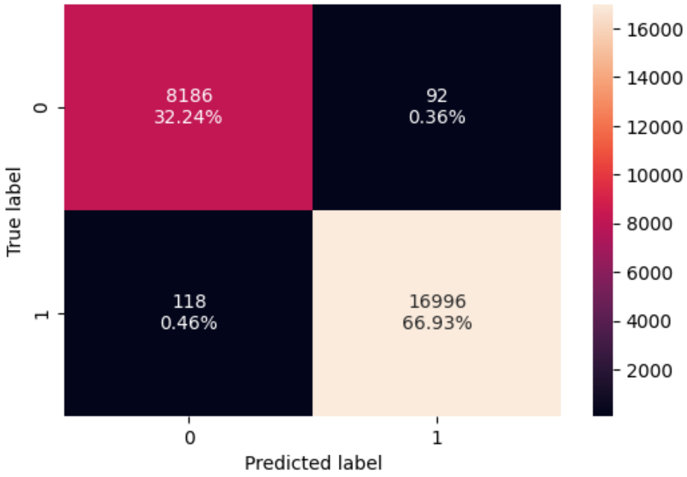
```
 #encoding the output (also called target) attribute
data["booking_status"] = data["booking_status"].apply(
    lambda x: 1 if x == "Canceled" else 0
)
```

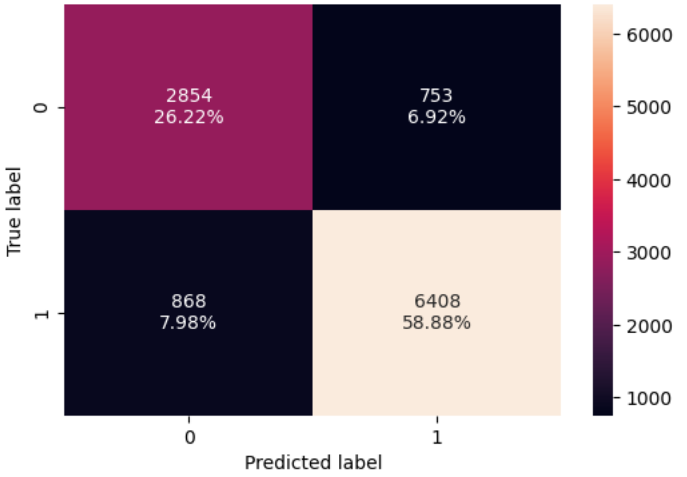
```
 #separating the input and output variables
X = data.drop(["booking_status","arrival_date"], axis=1)
y = data["booking_status"]

# Import the train_test_split function
from sklearn.model_selection import train_test_split

# encoding the categorical variables
X = pd.get_dummies(X, drop_first=True)

# splitting data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

```
X.head()
```

	lead_time	no_of_special_requests	avg_price_per_room	no_of_adults	no_of_weekend_nights	required_car_parking_space	no_of_week_nig
0	224	0	65.00	2	1	0	
1	5	1	106.68	2	2	0	
2	1	0	60.00	1	2	0	
3	211	0	100.00	2	0	0	
4	48	0	94.50	2	1	0	

```
# defining the AI model to build
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn import metrics
model = DecisionTreeClassifier(random_state=1)

#separating the input and output variables
X = data.drop(["booking_status","arrival_date"], axis=1)
y = data["booking_status"]

# Import the train_test_split function
from sklearn.model_selection import train_test_split

# encoding the categorical variables
X = pd.get_dummies(X, drop_first=True)

# splitting data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# training the AI model on the train data
model.fit(X_train, y_train)

def confusion_matrix_sklearn(model, predictors, target):
    """
    To plot the confusion_matrix with percentages

    model: classifier
    predictors: independent variables
    target: dependent variable
    """
    y_pred = model.predict(predictors)
    cm = confusion_matrix(target, y_pred)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten().sum())]
            for item in cm.flatten()
        ]
    ).reshape(2, 2)

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")

    # confusion matrix for train data
confusion_matrix_sklearn(model, X_train, y_train)

# confusion matrix for test data
confusion_matrix_sklearn(model, X_test, y_test)
```


8186
32.24%
92
0.36%
118
0.46%
16996
66.93%
True label
Predicted label
0
1
0
1
16000
14000
12000
10000
8000
6000
4000
2000


2854
26.22%
753
6.92%
868
7.98%
6408
58.88%
True label
Predicted label
0
1
0
1
6000
5000
4000
3000
2000
1000

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, f1_score
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd

model = DecisionTreeClassifier(random_state=1)

# Assuming 'data' is your DataFrame containing the dataset

# Separating the input and output variables
X = data.drop(["booking_status", "arrival_date"], axis=1)
y = data["booking_status"]

# Import the train_test_split function
from sklearn.model_selection import train_test_split

# Encoding the categorical variables
X = pd.get_dummies(X, drop_first=True)

# Splitting data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# Training the AI model on the train data
model.fit(X_train, y_train)

def confusion_matrix_sklearn(model, predictors, target):
    """
    To plot the confusion_matrix with percentages

    model: classifier
    predictors: independent variables
    target: dependent variable
    """
    y_pred = model.predict(predictors)
    cm = confusion_matrix(target, y_pred)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten().sum())]
            for item in cm.flatten()
        ]
    ).reshape(2, 2)

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")



# Evaluate the model performance on the train data
model_train_predictions = model.predict(X_train)

# Calculate F1 score for multiclass classification
model_train_score = f1_score(y_train, model_train_predictions, average='weighted')

print("Model Score on Train Data:", np.round(100 * model_train_score, 2))

# Evaluate the model performance on the test data
model_test_predictions = model.predict(X_test)
model_test_score = f1_score(y_test, model_test_predictions, average='weighted')

print("Model Score on Test Data:", np.round(100 * model_test_score, 2))
```

```
Model Score on Train Data: 99.17
Model Score on Test Data: 85.16
```

```
# defining the AI model to build
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import make_scorer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn import metrics

model = DecisionTreeClassifier(random_state=1)

# choosing the type of AI model
dummy_model = DecisionTreeClassifier(random_state=1, class_weight="balanced")

# defining the grid of parameters of the AI model to choose from
parameters = {
    "max_leaf_nodes": [150,250],
    "min_samples_split": [10,30],
}

# defining the model score on which we want to compare parameter combinations
acc_scorer = make_scorer(f1_score)

# running the model tuning algorithm
grid_obj = GridSearchCV(dummy_model, parameters, scoring=acc_scorer, cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

# selecting the best combination of parameters for the model to create a new model
tuned_model = grid_obj.best_estimator_

# training the new AI model
tuned_model.fit(X_train, y_train)
```

Show hidden output

```
# Importing necessary libraries
from sklearn.tree import DecisionTreeClassifier
```