A random forest model

Step 1: Imports

Import relevant Python libraries and modules, including numpy and pandaslibraries for data processing; the pickle package to save the model; and the sklearn library, containing:

The module ensemble, which has the function RandomForestClassifier The module model_selection, which has the functions train_test_split, PredefinedSplit, and GridSearchCV The module metrics, which has the functions f1_score, precision_score, recall_score, and accuracy_score

Import `numpy`, `pandas`, `pickle`, and `sklearn`.

In [1]:
```
# Import `numpy`, `pandas`, `pickle`, and `sklearn`.
# Import the relevant functions from `sklearn.ensemble`, `sklearn.model_selection`, and `sk


import numpy as np
import pandas as pd

import pickle as pkl

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, PredefinedSplit, GridSearchCV
from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score
```

In [2]:
```


air_data = pd.read_csv("eOfF6I7IRk2nxeiOyIZNmQ_a3d7736dda7244dda3872e1768730df1_Invistico_A
```

Step 2: Data cleaning

In [3]:
```
1 air_data.head(10)
```

Out[3]:

	satisfaction	Customer Type	Age	Type of Travel	Class	Flight Distance	Seat comfort	Departure/Arrival time convenient	Food and drink	Gate location	...	Online support
0	satisfied	Loyal Customer	65	Personal Travel	Eco	265	0	0	0	2	...	2
1	satisfied	Loyal Customer	47	Personal Travel	Business	2464	0	0	0	3	...	2
2	satisfied	Loyal Customer	15	Personal Travel	Eco	2138	0	0	0	3	...	2
3	satisfied	Loyal Customer	60	Personal Travel	Eco	623	0	0	0	3	...	3
4	satisfied	Loyal Customer	70	Personal Travel	Eco	354	0	0	0	3	...	4
5	satisfied	Loyal Customer	30	Personal Travel	Eco	1894	0	0	0	3	...	2
6	satisfied	Loyal Customer	66	Personal Travel	Eco	227	0	0	0	3	...	5
7	satisfied	Loyal Customer	10	Personal Travel	Eco	1812	0	0	0	3	...	2
8	satisfied	Loyal Customer	56	Personal Travel	Business	73	0	0	0	3	...	5
9	satisfied	Loyal Customer	22	Personal Travel	Eco	1556	0	0	0	3	...	2

10 rows × 22 columns

In [4]:
```
1 # Display variable names and types.
2
3 air_data.dtypes
```

Out[4]:
```
satisfaction                       object
Customer Type                      object
Age                                 int64
Type of Travel                     object
Class                              object
Flight Distance                     int64
Seat comfort                        int64
Departure/Arrival time convenient   int64
Food and drink                      int64
Gate location                       int64
Inflight wifi service               int64
Inflight entertainment              int64
Online support                      int64
Ease of Online booking              int64
On-board service                    int64
Leg room service                    int64
Baggage handling                    int64
Checkin service                     int64
Cleanliness                         int64
Online boarding                     int64
Departure Delay in Minutes          int64
Arrival Delay in Minutes          float64
dtype: object
```

Question: What do you observe about the differences in data types among the variables included in the data?

There are three types of variables included in the data: int64, float64, and object. The object variables are satisfaction, customer type, type of travel, and class.

Next, to understand the size of the dataset, identify the number of rows and the number of columns.

In [5]:
```
# Identify the number of rows and the number of columns.

air_data.shape
```

Out[5]:
```
(129880, 22)
```

Now, check for missing values in the rows of the data. Start with .isna() to get Booleans indicating whether each value in the data is missing. Then, use .any(axis=1) to get Booleans indicating whether there are any missing values along the columns in each row. Finally, use .sum() to get the number of rows that contain missing values.

In [6]:
```
# Get Booleans to find missing values in data.
# Get Booleans to find missing values along columns.
# Get the number of rows that contain missing values.

air_data.isna().any(axis=1).sum()
```

Out[6]:
```
393
```

Question: How many rows of data are missing values?**

There are 393 rows with missing values.

Drop the rows with missing values. This is an important step in data cleaning, as it makes the data more useful for analysis and regression. Then, save the resulting pandas DataFrame in a variable named air_data_subset.

In [7]:
```
# Drop missing values.
# Save the DataFrame in variable `air_data_subset

air_data_subset = air_data.dropna(axis=0)
```

In [8]:
```
# Display the first 10 rows

air_data_subset.head(10)
```

Out[8]:

	satisfaction	Customer Type	Age	Type of Travel	Class	Flight Distance	Seat comfort	Departure/Arrival time convenient	Food and drink	Gate location	...	Online support
0	satisfied	Loyal Customer	65	Personal Travel	Eco	265	0	0	0	2	...	2
1	satisfied	Loyal Customer	47	Personal Travel	Business	2464	0	0	0	3	...	2
2	satisfied	Loyal Customer	15	Personal Travel	Eco	2138	0	0	0	3	...	2
3	satisfied	Loyal Customer	60	Personal Travel	Eco	623	0	0	0	3	...	3
4	satisfied	Loyal Customer	70	Personal Travel	Eco	354	0	0	0	3	...	4
5	satisfied	Loyal Customer	30	Personal Travel	Eco	1894	0	0	0	3	...	2
6	satisfied	Loyal Customer	66	Personal Travel	Eco	227	0	0	0	3	...	5
7	satisfied	Loyal Customer	10	Personal Travel	Eco	1812	0	0	0	3	...	2
8	satisfied	Loyal Customer	56	Personal Travel	Business	73	0	0	0	3	...	5
9	satisfied	Loyal Customer	22	Personal Travel	Eco	1556	0	0	0	3	...	2

10 rows × 22 columns

In [9]:
```
# Count of missing values.

air_data_subset.isna().sum()
```

Out[9]:
```
satisfaction                       0
Customer Type                      0
Age                                0
Type of Travel                     0
Class                              0
Flight Distance                    0
Seat comfort                       0
Departure/Arrival time convenient  0
Food and drink                     0
Gate location                      0
Inflight wifi service              0
Inflight entertainment             0
Online support                     0
Ease of Online booking             0
On-board service                   0
Leg room service                   0
Baggage handling                   0
Checkin service                    0
Cleanliness                        0
Online boarding                    0
Departure Delay in Minutes         0
Arrival Delay in Minutes           0
dtype: int64
```

Next, convert the categorical features to indicator (one-hot encoded) features.

Note: The drop_first argument can be kept as default (False) during one-hot encoding for random forest models, so it does not need to be specified. Also, the target variable, satisfaction, does not need to be encoded and will be extracted in a later step.

In [10]:
```
# Convert categorical features to one-hot encoded features.

air_data_subset_dummies = pd.get_dummies(air_data_subset,
                                         columns=['Customer Type','Type of Travel','Class'])
```

Question: Why is it necessary to convert categorical data into dummy variables?**

It is necessary because the sklearn implementation of RandomForestClassifier() requires that categorical features be encoded to numeric, which can be done using dummy variables or one-hot encoding.

Next, display the first 10 rows to review the air_data_subset_dummies.

In [11]:
```
# Display the first 10 rows.
air_data_subset_dummies.head(10)
```

Out[11]:

	satisfaction	Age	Flight Distance	Seat comfort	Departure/Arrival time convenient	Food and drink	Gate location	Inflight wifi service	Inflight entertainment	Online support	...	Onlir boardir
0	satisfied	65	265	0	0	0	2	2	4	2	...	
1	satisfied	47	2464	0	0	0	3	0	2	2	...	
2	satisfied	15	2138	0	0	0	3	2	0	2	...	
3	satisfied	60	623	0	0	0	3	3	4	3	...	
4	satisfied	70	354	0	0	0	3	4	3	4	...	
5	satisfied	30	1894	0	0	0	3	2	0	2	...	
6	satisfied	66	227	0	0	0	3	2	5	5	...	
7	satisfied	10	1812	0	0	0	3	2	0	2	...	
8	satisfied	56	73	0	0	0	3	5	3	5	...	
9	satisfied	22	1556	0	0	0	3	2	0	2	...	

10 rows × 26 columns

Then, check the variables of air_data_subset_dummies.

In [12]:
```
# Display variables.

air_data_subset_dummies.dtypes
```

Out[12]:
```
satisfaction                           object
Age                                     int64
Flight Distance                         int64
Seat comfort                            int64
Departure/Arrival time convenient       int64
Food and drink                          int64
Gate location                           int64
Inflight wifi service                   int64
Inflight entertainment                  int64
Online support                          int64
Ease of Online booking                  int64
On-board service                        int64
Leg room service                        int64
Baggage handling                        int64
Checkin service                         int64
Cleanliness                             int64
Online boarding                         int64
Departure Delay in Minutes              int64
Arrival Delay in Minutes              float64
Customer Type_Loyal Customer            uint8
Customer Type_disloyal Customer         uint8
Type of Travel_Business travel          uint8
Type of Travel_Personal Travel          uint8
Class_Business                          uint8
Class_Eco                               uint8
Class_Eco Plus                          uint8
dtype: object
```

Question: What changes do you observe after converting the string data to dummy variables?**

All of the following changes could be observed:

Customer Type --> Customer Type_Loyal Customer and Customer Type_disloyal Customer Type of Travel --> Type of Travel_Business travel and Type of Travel_Personal travel Class --> Class_Business, Class_Eco, Class_Eco Plus

Step 3: Model building

The first step to building your model is separating the labels (y) from the features (X).

In [13]:
```
# Separate the dataset into labels (y) and features (X).

y = air_data_subset_dummies["satisfaction"]
X = air_data_subset_dummies.drop("satisfaction", axis=1)
```

In [14]:
```
# Separate into train, validate, test sets.

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state =
X_tr, X_val, y_tr, y_val = train_test_split(X_train, y_train, test_size = 0.25, random_stat

```

Tune the model

Now, fit and tune a random forest model with separate validation set. Begin by determining a set of hyperparameters for tuning the model using GridSearchCV.

In [15]:
```
# Determine set of hyperparameters.

cv_params = {'n_estimators' : [50,100],
             'max_depth' : [10,50],
             'min_samples_leaf' : [0.5,1],
             'min_samples_split' : [0.001, 0.01],
             'max_features' : ["sqrt"],
             'max_samples' : [.5,.9]}
```

In [16]:
```
# Create list of split indices.

split_index = [0 if x in X_val.index else -1 for x in X_train.index]
custom_split = PredefinedSplit(split_index)
```

In [17]:
```
# Instantiate model.

rf = RandomForestClassifier(random_state=0)
```

In [18]:
```
# Search over specified parameters.

rf_val = GridSearchCV(rf, cv_params, cv=custom_split, refit='f1', n_jobs = -1, verbose = 1
```

In [19]:
```
%%time

# Fit the model.
rf_val.fit(X_train, y_train)
```

```
Fitting 1 folds for each of 32 candidates, totalling 32 fits
CPU times: total: 15.4 s
Wall time: 2min 39s
```

Out[19]:
```
GridSearchCV(cv=PredefinedSplit(test_fold=array([-1, -1, ..., -1, -1])),
             estimator=RandomForestClassifier(random_state=0), n_jobs=-1,
             param_grid={'max_depth': [10, 50], 'max_features': ['sqrt'],
                         'max_samples': [0.5, 0.9],
                         'min_samples_leaf': [0.5, 1],
                         'min_samples_split': [0.001, 0.01],
                         'n_estimators': [50, 100]},
             refit='f1', verbose=1)
```

In [20]:
```
# Obtain optimal parameters.
rf_val.best_params_
```

Out[20]:
```
{'max_depth': 50,
 'max_features': 'sqrt',
 'max_samples': 0.9,
 'min_samples_leaf': 1,
 'min_samples_split': 0.001,
 'n_estimators': 50}
```

Step 4: Results and evaluation¶

In [21]:
```
# Use optimal parameters on GridSearchCV.

rf_opt = RandomForestClassifier(n_estimators = 50, max_depth = 50,
                                min_samples_leaf = 1, min_samples_split = 0.001,
                                max_features="sqrt", max_samples = 0.9, random_state = 0)
```

Once again, fit the optimal model.

In [22]:
```
# Fit the optimal model

rf_opt.fit(X_train, y_train)
```

Out[22]:
```
RandomForestClassifier(max_depth=50, max_features='sqrt', max_samples=0.9,
                       min_samples_split=0.001, n_estimators=50,
                       random_state=0)
```

And predict on the test set using the optimal model.

In [23]:
```
# Predict on test set.
y_pred = rf_opt.predict(X_test)
```

Obtain performance scores

First, get your precision score.

In [24]:
```
# Get precision score.

pc_test = precision_score(y_test, y_pred, pos_label = "satisfied")
print("The precision score is {pc:.3f}".format(pc = pc_test))
```

```
The precision score is 0.950
```

In [25]:
```
# Get recall score.

rc_test = recall_score(y_test, y_pred, pos_label = "satisfied")
print("The recall score is {rc:.3f}".format(rc = rc_test))
```

```
The recall score is 0.945
```

In [26]:
```
# Get accuracy score.

ac_test = accuracy_score(y_test, y_pred)
print("The accuracy score is {ac:.3f}".format(ac = ac_test))
```

```
The accuracy score is 0.942
```

In [27]:
```
# Get F1 score.

f1_test = f1_score(y_test, y_pred, pos_label = "satisfied")
print("The F1 score is {f1:.3f}".format(f1 = f1_test))
```

```
The F1 score is 0.947
```

F1 scores are calculated using the following formula:

F1 = 2 * (precision * recall) / (precision + recall)

Question: What are the pros and cons of performing the model selection using test data instead of a separate validation dataset?

Pros:

The coding workload is reduced. The scripts for data splitting are shorter. It's only necessary to evaluate test dataset performance once, instead of two evaluations (validate and test). Cons:

If a model is evaluated using samples that were also used to build or fine-tune that model, it likely will provide a biased evaluation. A potential overfitting issue could happen when fitting the model's scores on the test data. Evaluate the model Now that you have results, evaluate the model.

Question: What are the four basic parameters for evaluating the performance of a classification model?

True positives (TP): These are correctly predicted positive values, which means the value of actual and predicted classes are positive.

True negatives (TN): These are correctly predicted negative values, which means the value of the actual and predicted classes are negative.

False positives (FP): This occurs when the value of the actual class is negative and the value of the predicted class is positive.

False negatives (FN): This occurs when the value of the actual class is positive and the value of the predicted class in negative.

Reminder: When fitting and tuning classification modeld, data professioals aim to minimize false positives and false negatives.

Question: What do the four scores demonstrate about your model, and how do you calculate them?

Accuracy (TP+TN/TP+FP+FN+TN): The ratio of correctly predicted observations to total observations.

Precision (TP/TP+FP): The ratio of correctly predicted positive observations to total predicted positive observations.

Recall (Sensitivity, TP/TP+FN): The ratio of correctly predicted positive observations to all observations in actual class.

F1 score: The harmonic average of precision and recall, which takes into account both false positives and false negatives.

In [28]:
```
# Precision score on test data set.
print("\nThe precision score is: {pc:.3f}".format(pc = pc_test), "for the test set,", "\nwh
```

```
The precision score is: 0.950 for the test set,
which means of all positive predictions, 95.0% prediction are true positive.
```

In [29]:
```
# Recall score on test data set.
print("\nThe recall score is: {rc:.3f}".format(rc = rc_test), "for the test set,", "\nwhich
```

```
The recall score is: 0.945 for the test set,
which means of which means of all real positive cases in test set, 94.5% are  predicted positi
ve.
```

In [30]:
```
# Accuracy score on test data set.
print("\nThe accuracy score is: {ac:.3f}".format(ac = ac_test), "for the test set,", "\nwh
```

```
The accuracy score is: 0.942 for the test set,
which means of all cases in test set, 94.2% are predicted true positive or true negative.
```

In [31]:
```
# F1 score on test data set.

print("\nThe F1 score is: {f1:.3f}".format(f1 = f1_test), "for the test set,", "\nwhich mea
```

```
The F1 score is: 0.947 for the test set,
which means the test set's harmonic mean is 94.7%.
```

How does this model perform based on the four scores?

The model performs well according to all 4 performance metrics. The model's precision score is slightly better than the 3 other metrics.

Evaluate the model Finally, create a table of results that you can use to evaluate the performace of your model.

Create table of results.

In [32]:
```
# Create table of results.

### YOUR CODE HERE ###

table = pd.DataFrame()
table = table.append({'Model': "Tuned Decision Tree",
                      'F1':  0.945422,
                      'Recall': 0.935863,
                      'Precision': 0.955197,
                      'Accuracy': 0.940864
                     },
                       ignore_index=True
                    )

table = table.append({'Model': "Tuned Random Forest",
                      'F1':  f1_test,
                      'Recall': rc_test,
                      'Precision': pc_test,
                      'Accuracy': ac_test
                     },
                       ignore_index=True
                    )
table
```

```
C:\Users\Blessing\AppData\Local\Temp\ipykernel_7064\1400616487.py:6: FutureWarning: The frame.
append method is deprecated and will be removed from pandas in a future version. Use pandas.co
ncat instead.
  table = table.append({'Model': "Tuned Decision Tree",
C:\Users\Blessing\AppData\Local\Temp\ipykernel_7064\1400616487.py:15: FutureWarning: The fram
e.append method is deprecated and will be removed from pandas in a future version. Use pandas.
concat instead.
  table = table.append({'Model': "Tuned Random Forest",
```

Out[32]:

	Model	F1	Recall	Precision	Accuracy
0	Tuned Decision Tree	0.945422	0.935863	0.955197	0.940864
1	Tuned Random Forest	0.947306	0.944501	0.950128	0.942450

The random forest model predicted satisfaction with more than 94.2% accuracy. The precision is over 95% and the recall is approximately 94.5%. The random forest model outperformed the tuned decision tree with the best hyperparameters in most of the four scores. This indicates that the random forest model may perform better. Because stakeholders were interested in learning about the factors that are most important to customer satisfaction, this would be shared based on the tuned random forest. In addition, you would provide details about the precision, recall, accuracy, and F1 scores to support your findings.

In [ ]:
```
1
```