In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
sns.set_theme(color_codes=True)
pd.set_option('display.max_columns', None)
```

In [2]:

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

Out[2]:

	Unnamed: 0	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/ time conv
0	0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	
1	1	5047	Male	disloyal Customer	25	Business travel	Business	235	3	
2	2	110028	Female	Loyal Customer	26	Business travel	Business	1142	2	
3	3	24026	Female	Loyal Customer	25	Business travel	Business	562	2	
4	4	119299	Male	Loyal Customer	61	Business travel	Business	214	3	

Remove ID Column

In [3]:

```
df.drop(columns=['Unnamed: 0', 'id'], inplace=True)
df.shape
```

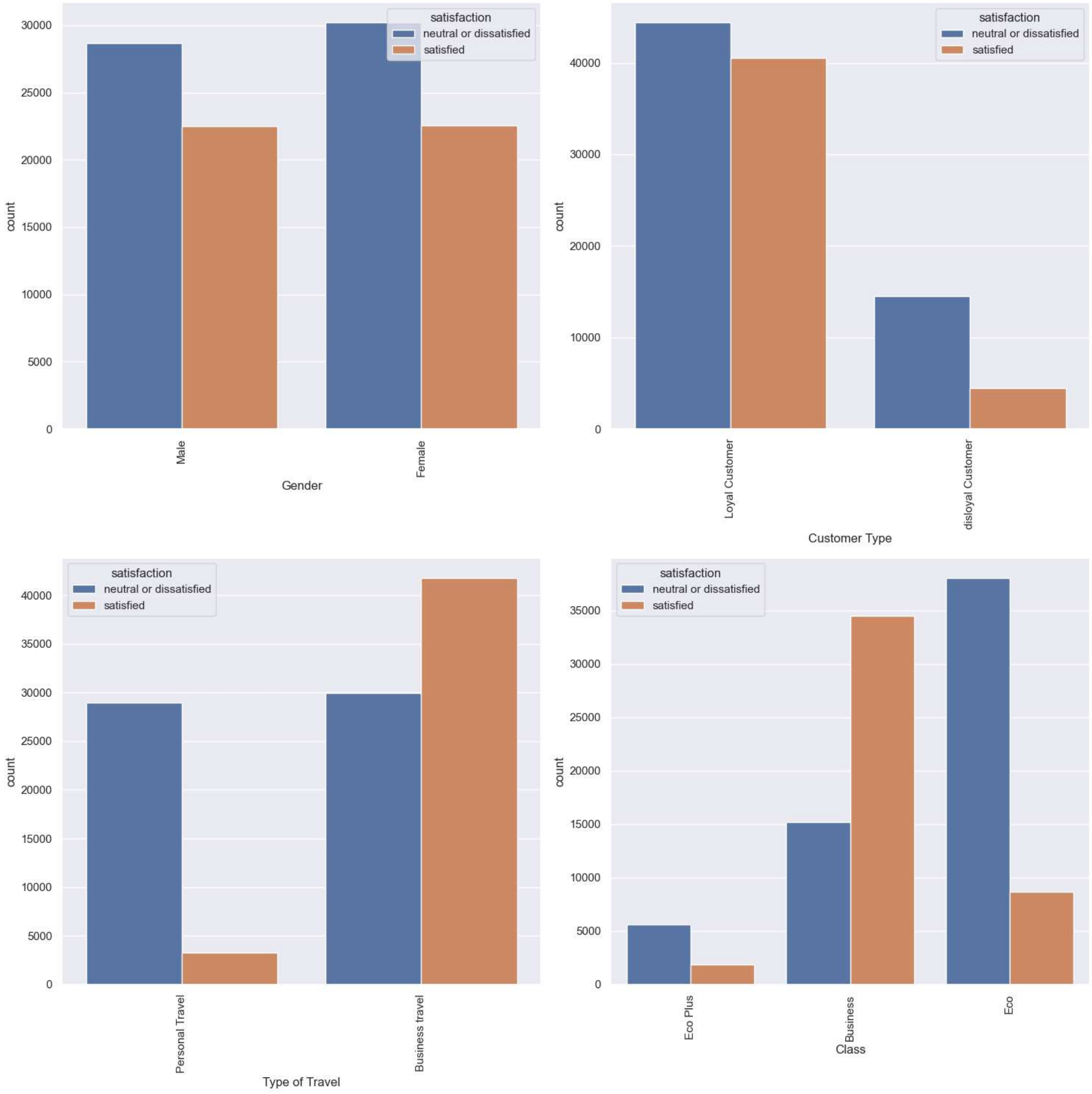
Out[3]:

```
(103904, 23)
```

Exploratory Data Analysis

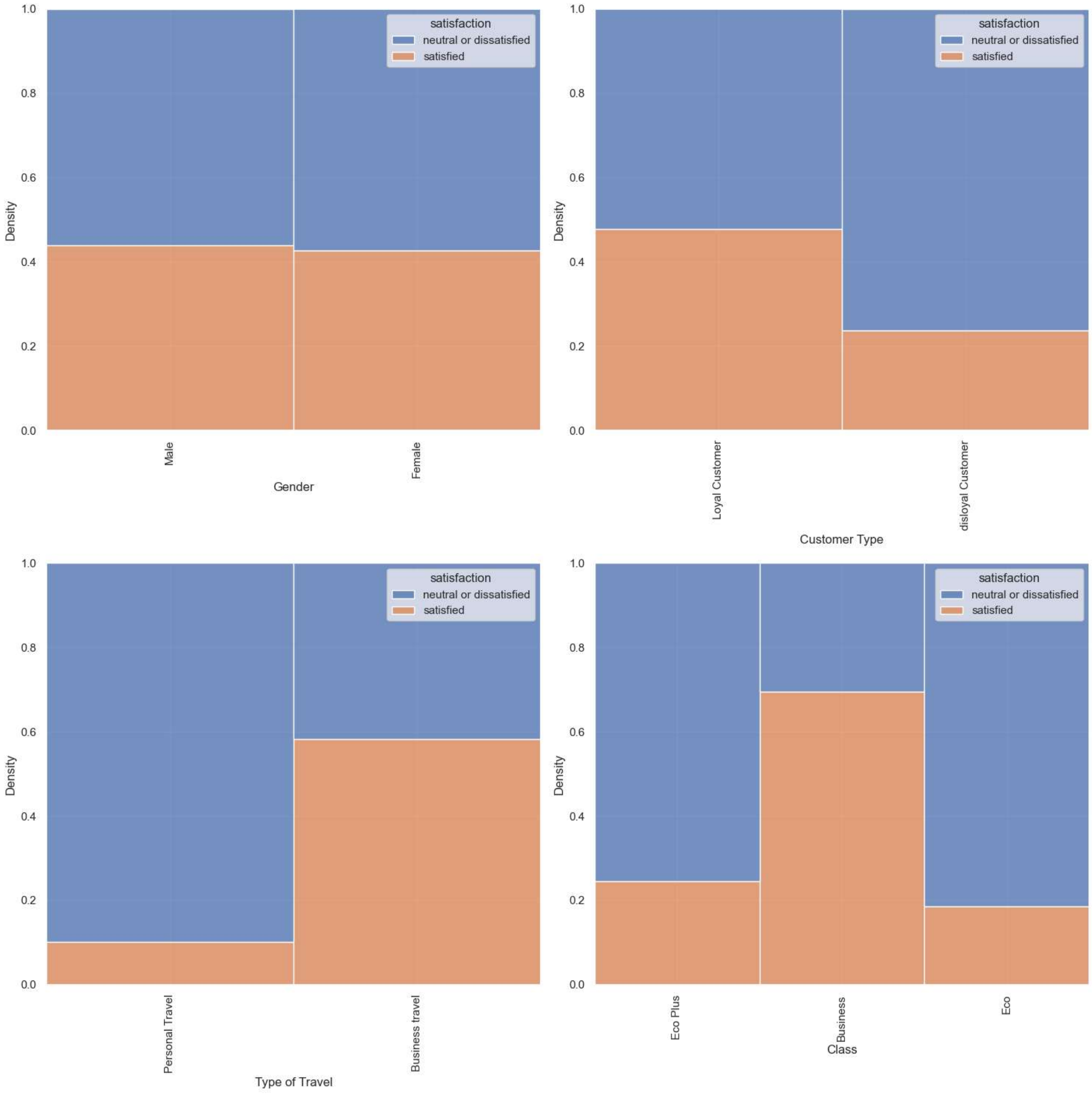
In [4]:

```python
# list of categorical variables to plot
cat_vars = ['Gender', 'Customer Type', 'Type of Travel', 'Class']

# create figure with subplots
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(15, 15))
axs = axs.flatten()

# create barplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.countplot(x=var, hue='satisfaction', data=df, ax=axs[i])
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```



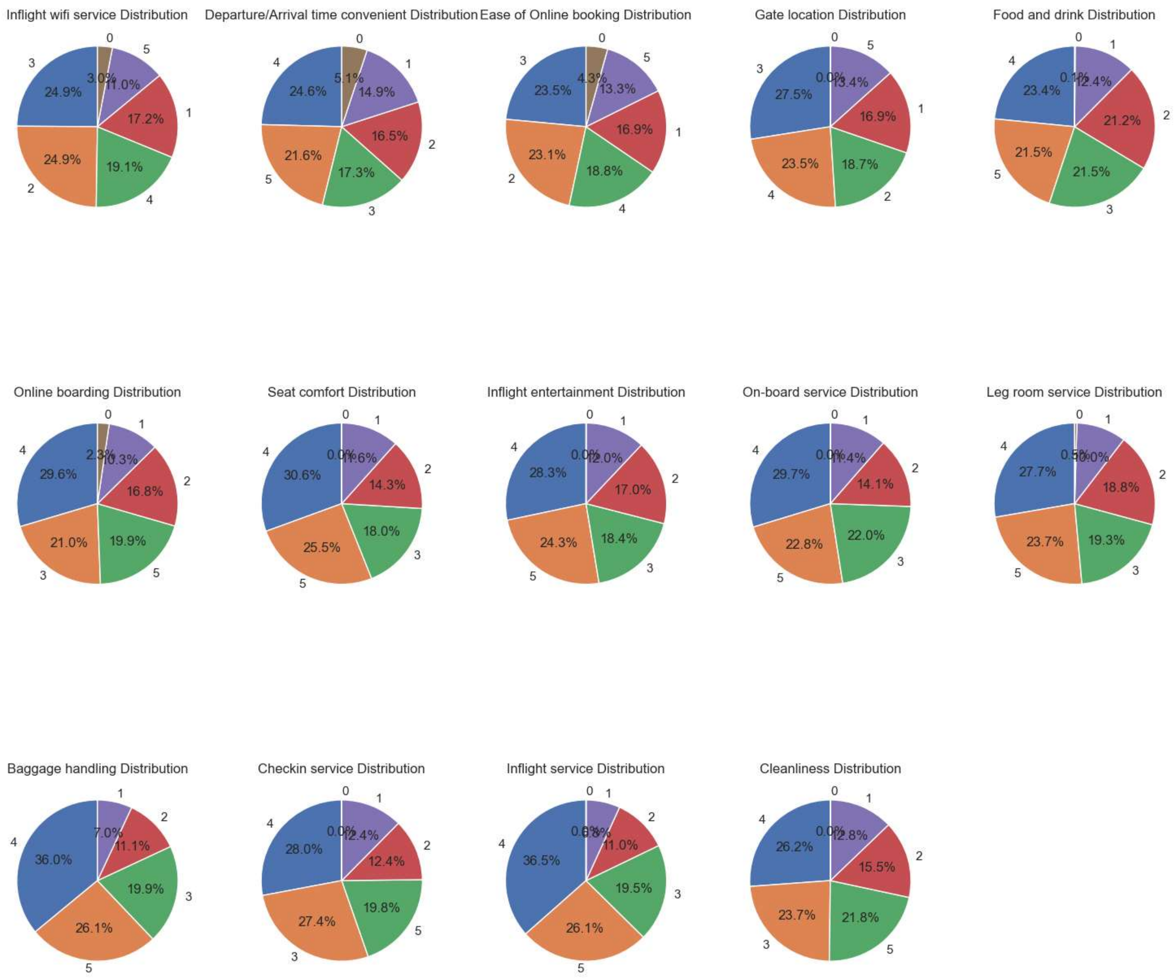
In [5]:

```
import warnings
warnings.filterwarnings("ignore")
# get list of categorical variables
cat_vars = ['Gender', 'Customer Type', 'Type of Travel', 'Class']

# create figure with subplots
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(15, 15))
axs = axs.flatten()

# create histplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.histplot(x=var, hue='satisfaction', data=df, ax=axs[i], multiple="fill
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
    axs[i].set_xlabel(var)

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```

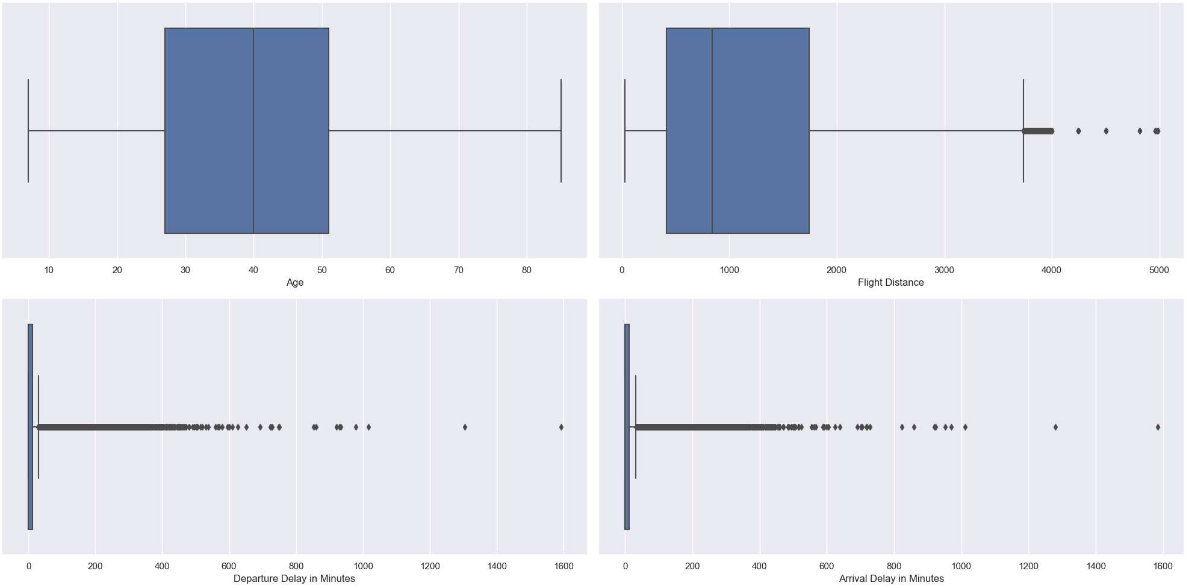

satisfaction
🟦 neutral or dissatisfied
🟧 satisfied
Density
1.0
0.8
0.6
0.4
0.2
0.0
Male
Female
Gender
Loyal Customer
disloyal Customer
Customer Type
Personal Travel
Business travel
Type of Travel
Eco Plus
Business
Eco
Class

In [6]:

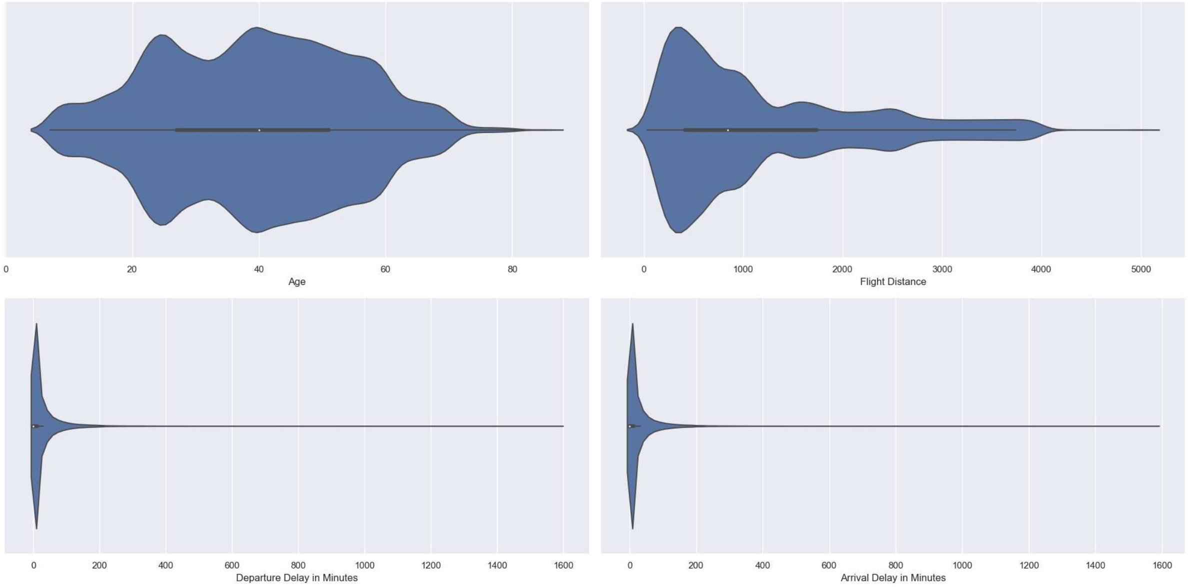
```
cat_vars = ['Inflight wifi service', 'Departure/Arrival time convenient', 'Eas
            'Gate location', 'Food and drink', 'Online boarding', 'Seat comfor
           'On-board service', 'Leg room service', 'Baggage handling', 'Checki
           'Cleanliness']

# create a figure and axes
fig, axs = plt.subplots(nrows=3, ncols=5, figsize=(15, 15))

# create a pie chart for each categorical variable
for i, var in enumerate(cat_vars):
    if i < len(axs.flat):
        # count the number of occurrences for each category
        cat_counts = df[var].value_counts()

        # create a pie chart
        axs.flat[i].pie(cat_counts, labels=cat_counts.index, autopct='%1.1f%%'

        # set a title for each subplot
        axs.flat[i].set_title(f'{var} Distribution')

# adjust spacing between subplots
fig.tight_layout()
fig.delaxes(axs[2][4])
# show the plot
plt.show()
```



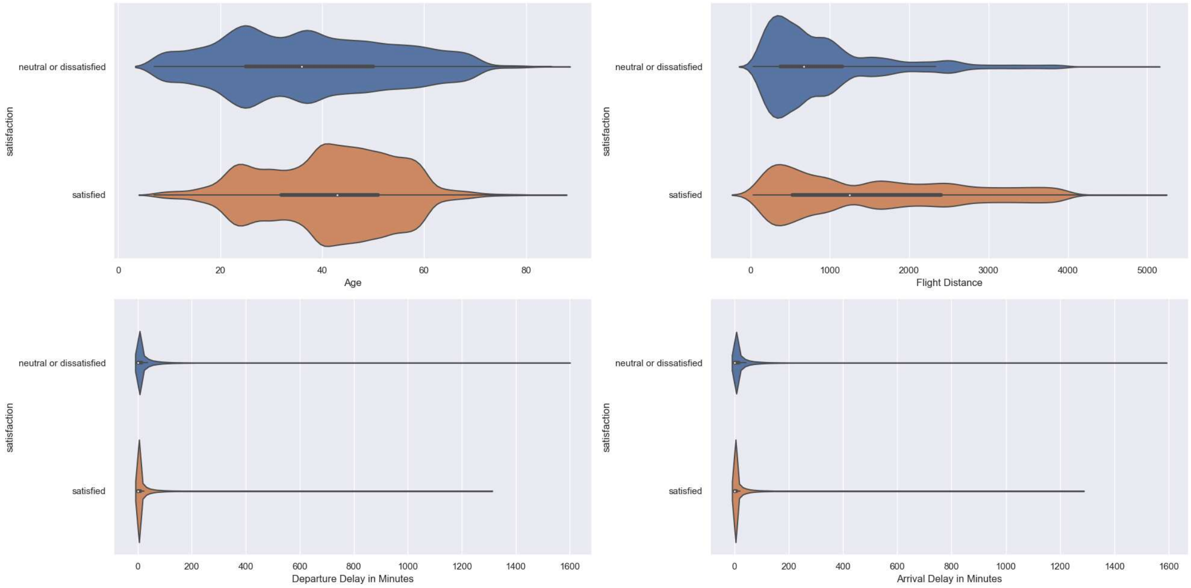
In [7]:
```
num_vars = ['Age', 'Flight Distance', 'Departure Delay in Minutes', 'Arrival De

fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.boxplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



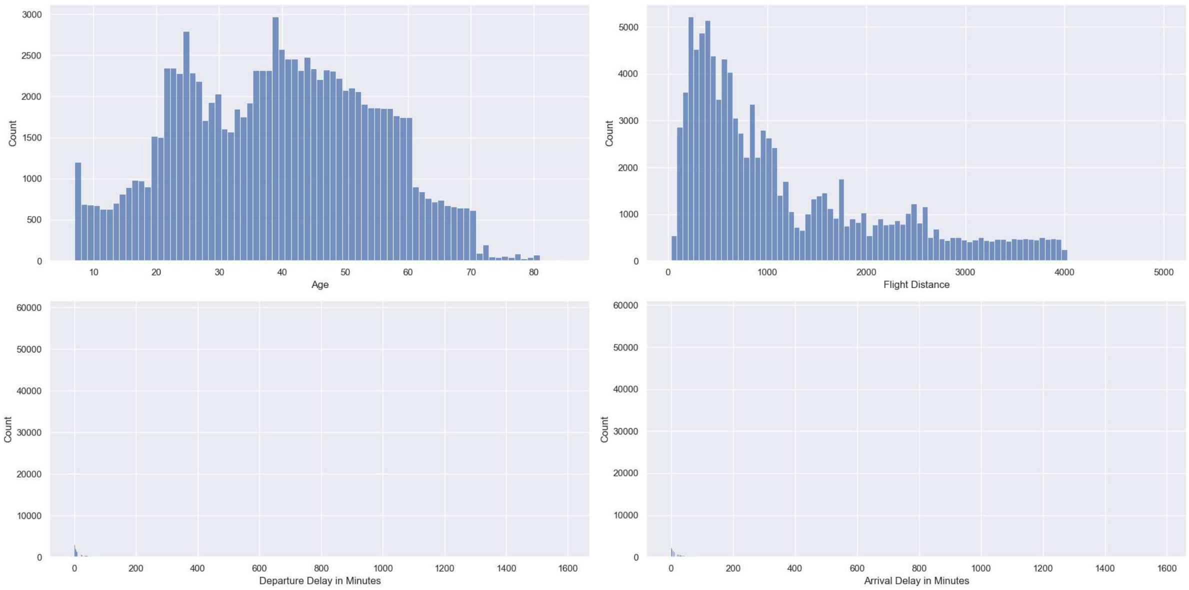
```
In [8]: num_vars = ['Age', 'Flight Distance', 'Departure Delay in Minutes', 'Arrival D

        fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))
        axs = axs.flatten()

        for i, var in enumerate(num_vars):
            sns.violinplot(x=var, data=df, ax=axs[i])

        fig.tight_layout()
        plt.show()
```



In [9]:
```
num_vars = ['Age', 'Flight Distance', 'Departure Delay in Minutes', 'Arrival D

fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.violinplot(x=var, y='satisfaction', data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```

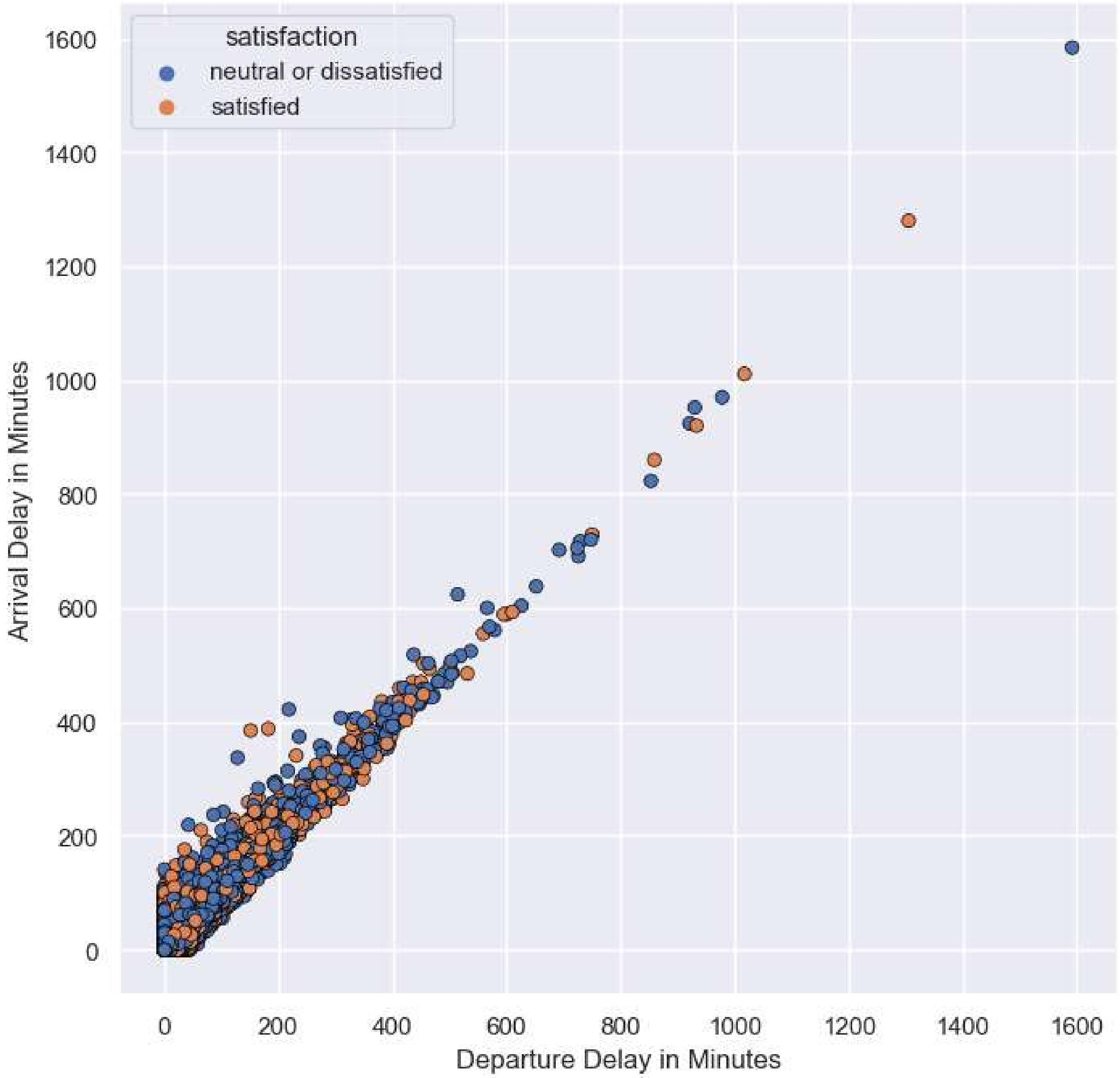


In [10]:
```
num_vars = ['Age', 'Flight Distance', 'Departure Delay in Minutes', 'Arrival De

fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.histplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



In [11]:

```
plt.figure(figsize=(8,8),dpi=100)
sns.scatterplot(x="Departure Delay in Minutes", y="Arrival Delay in Minutes", 
plt.show()
```



Data Preprocessing

Check null value

In [12]:

```
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[12]:

```
Arrival Delay in Minutes    0.298352
dtype: float64
```

In [13]:
```
df['Arrival Delay in Minutes'].fillna(df['Arrival Delay in Minutes'].median(),
```

In [14]:
```
df.dtypes
```

Out[14]:
```
Gender                                object
Customer Type                         object
Age                                    int64
Type of Travel                        object
Class                                 object
Flight Distance                        int64
Inflight wifi service                  int64
Departure/Arrival time convenient      int64
Ease of Online booking                 int64
Gate location                          int64
Food and drink                         int64
Online boarding                        int64
Seat comfort                           int64
Inflight entertainment                 int64
On-board service                       int64
Leg room service                       int64
Baggage handling                       int64
Checkin service                        int64
Inflight service                       int64
Cleanliness                            int64
Departure Delay in Minutes             int64
Arrival Delay in Minutes             float64
satisfaction                          object
dtype: object
```

Show Unique Value each Columns

In [15]:
```
# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Print the column name and the unique values
    print(f"{col}: {df[col].unique()}")
```

```
Gender: ['Male' 'Female']
Customer Type: ['Loyal Customer' 'disloyal Customer']
Type of Travel: ['Personal Travel' 'Business travel']
Class: ['Eco Plus' 'Business' 'Eco']
satisfaction: ['neutral or dissatisfied' 'satisfied']
```

Label encoding each categorial column

In [16]:

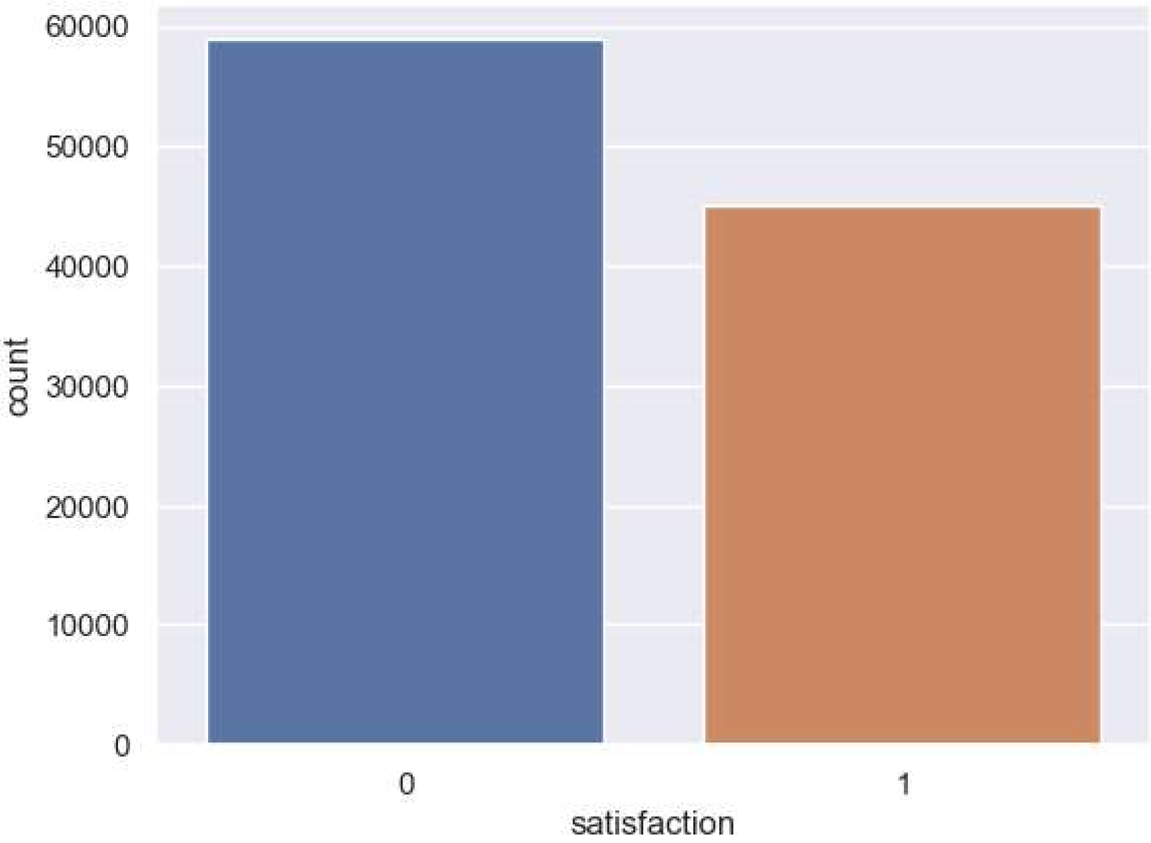
```python
from sklearn import preprocessing

# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Initialize a LabelEncoder object
    label_encoder = preprocessing.LabelEncoder()

    # Fit the encoder to the unique values in the column
    label_encoder.fit(df[col].unique())

    # Transform the column using the encoder
    df[col] = label_encoder.transform(df[col])

    # Print the column name and the unique encoded values
    print(f"{col}: {df[col].unique()}")
```

```
Gender: [1 0]
Customer Type: [0 1]
Type of Travel: [1 0]
Class: [2 0 1]
satisfaction: [0 1]
```

Check class value

In [17]:
```
sns.countplot(df['satisfaction'])
df['satisfaction'].value_counts()
```

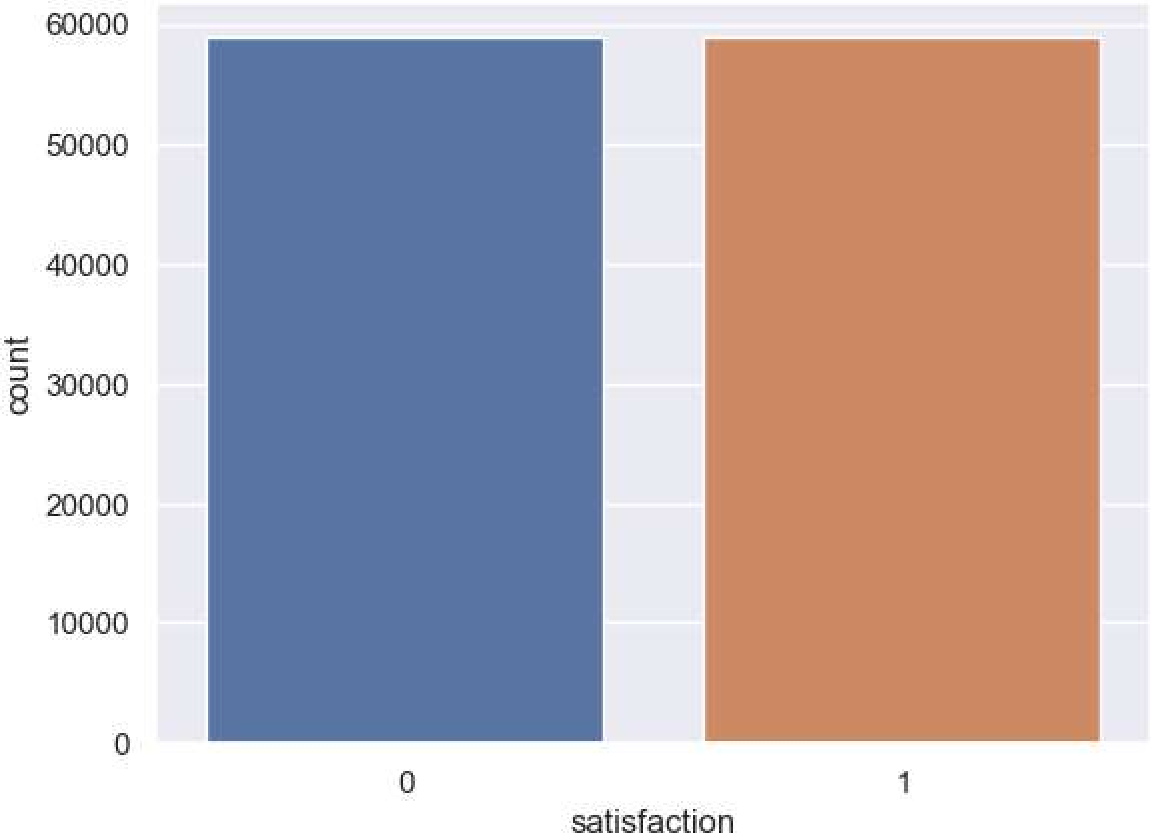
Out[17]:
```
0    58879
1    45025
Name: satisfaction, dtype: int64
```



In [18]:
```
from sklearn.utils import resample
#create two different dataframe of majority and minority class
df_majority = df[(df['satisfaction']==0)]
df_minority = df[(df['satisfaction']==1)]
# upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,    # sample with replacement
                                 n_samples= 58879, # to match majority class
                                 random_state=0)  # reproducible results
# Combine majority class with upsampled minority class
df_upsampled = pd.concat([df_minority_upsampled, df_majority])
```

```
In [19]: sns.countplot(df_upsampled['satisfaction'])
         df_upsampled['satisfaction'].value_counts()
```

```
Out[19]: 1    58879
         0    58879
         Name: satisfaction, dtype: int64
```



Remove Outlier using IQR

In [20]:
```
# specify the columns to remove outliers from dataframe
column_names = ['Age', 'Flight Distance', 'Departure Delay in Minutes', 'Arriva

# remove outliers for each selected column using the IQR method
for column_name in column_names:
    Q1 = df_upsampled[column_name].quantile(0.25)
    Q3 = df_upsampled[column_name].quantile(0.75)
    IQR = Q3 - Q1
    df_upsampled = df_upsampled[~((df_upsampled[column_name] < (Q1 - 1.5 * IQR

df_upsampled.head()
```

Out[20]:

	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	lo
6287	0	0	43	0	0	3603	3	3	3	
100566	1	0	45	0	1	451	4	2	2	
98330	1	0	31	0	0	1334	1	1	1	
48752	0	0	51	0	1	589	4	2	2	
69983	1	0	46	0	0	1400	2	2	5	

In [21]:
```
df_upsampled.shape
```

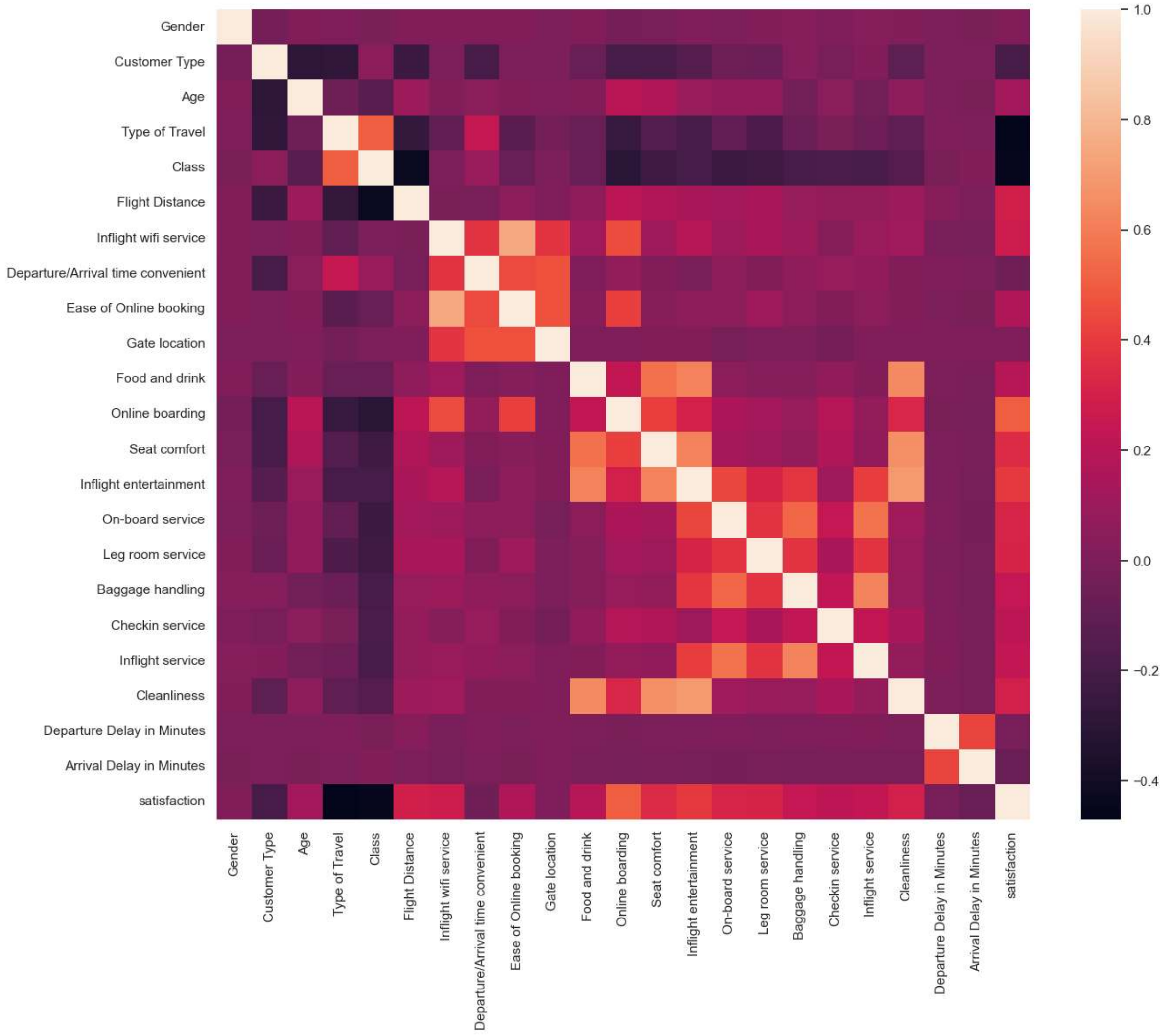
Out[21]:
```
(87448, 23)
```

In [22]:

```
plt.figure(figsize=(15,12))
sns.heatmap(df_upsampled.corr(), fmt='.2g')
```

Out[22]:

```
<AxesSubplot:>
```



Machine Learning Model Building

In [23]:

```
X = df_upsampled.drop('satisfaction', axis=1)
y = df_upsampled['satisfaction']
```

In [24]:

```
#test size 20% and train size 80%
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,random_
```

Decision Tree

In [25]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
dtree = DecisionTreeClassifier()
param_grid = {
    'max_depth': [3, 4, 5, 6, 7, 8],
    'min_samples_split': [2, 3, 4],
    'min_samples_leaf': [1, 2, 3, 4]
}

# Perform a grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(dtree, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'max_depth': 8, 'min_samples_leaf': 4, 'min_samples_split': 2}
```

In [26]:

```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=0, max_depth=8, min_samples_leaf=4
dtree.fit(X_train, y_train)
```

Out[26]:

```
DecisionTreeClassifier(max_depth=8, min_samples_leaf=4, random_state=0)
```

In [27]:

```
y_pred = dtree.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

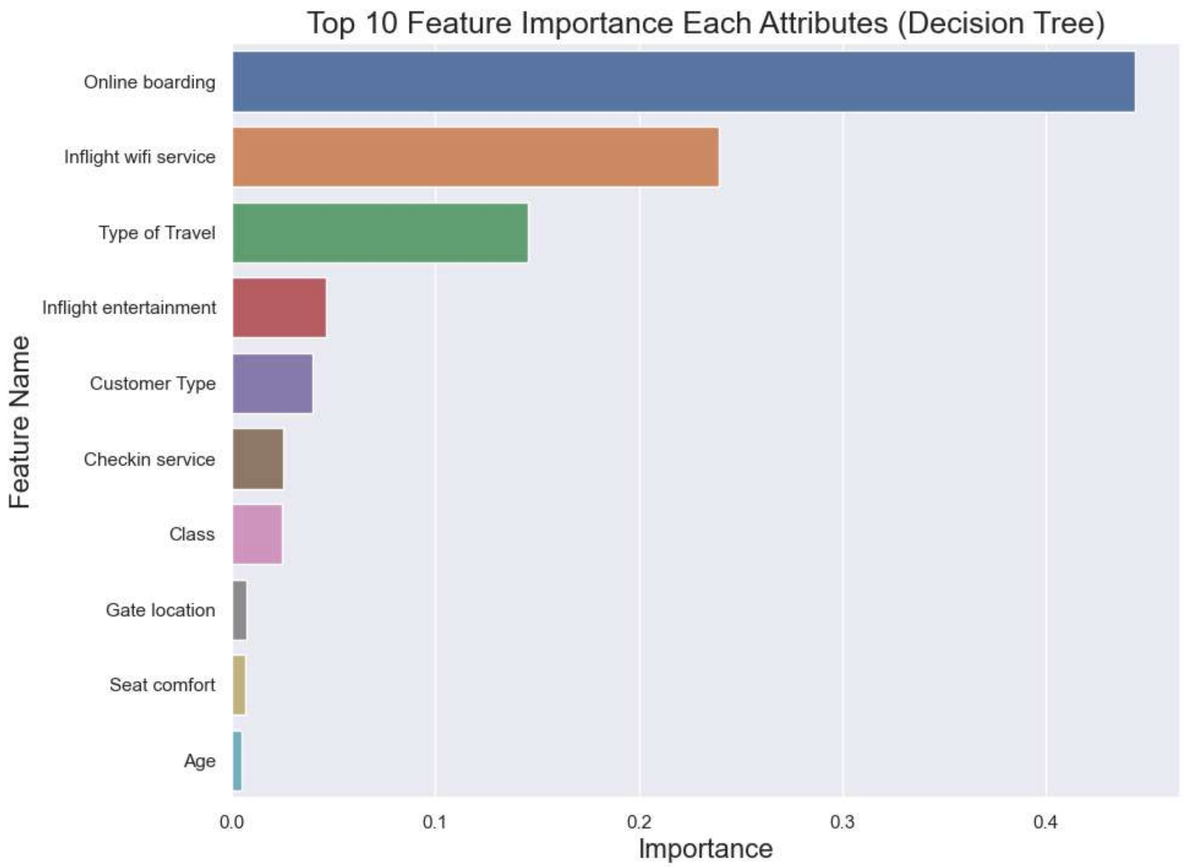
```
Accuracy Score : 93.62 %
```

In [28]:

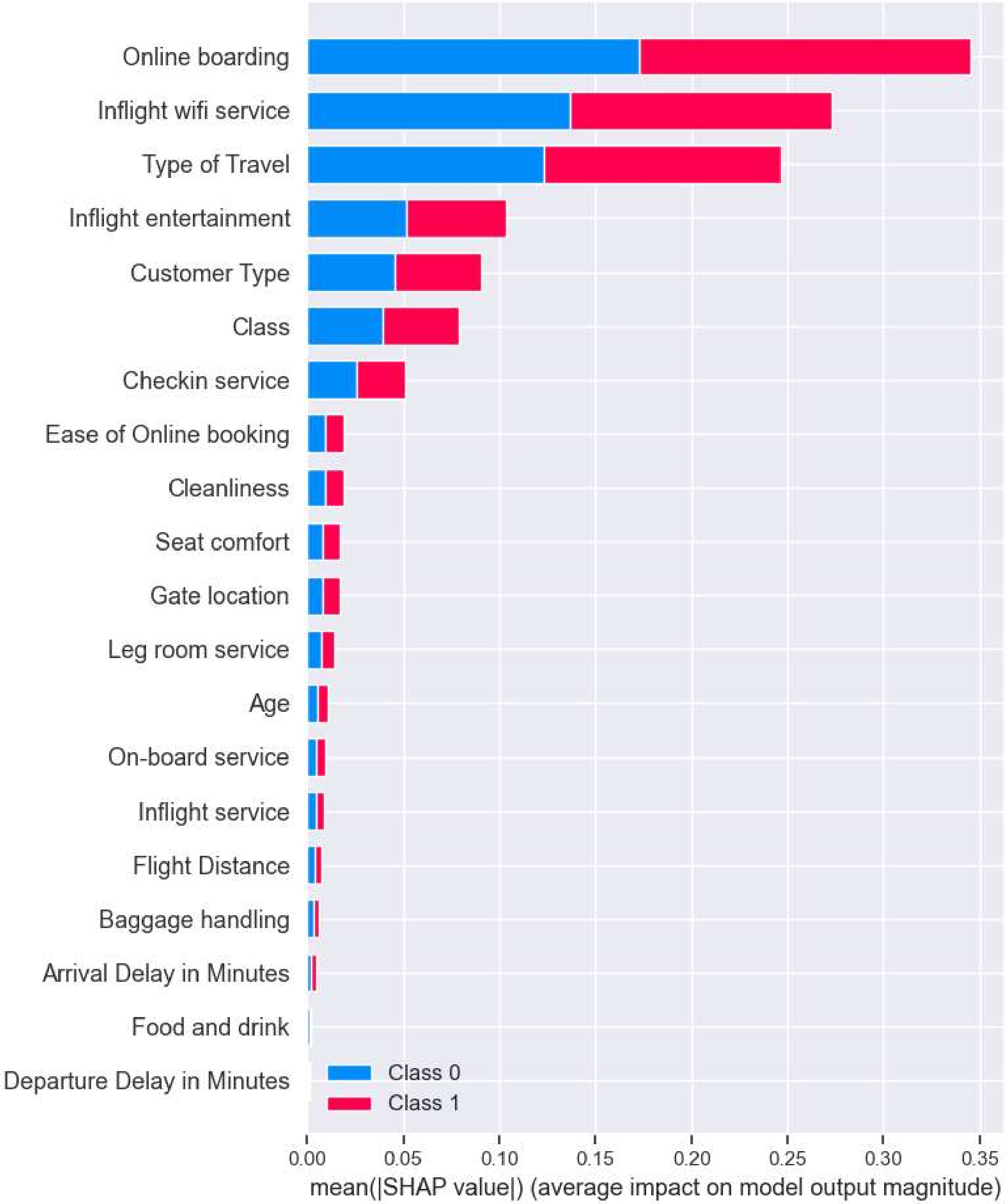
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.9361921097770154
Precision Score :  0.9361921097770154
Recall Score :  0.9361921097770154
Jaccard Score :  0.8800386971944534
Log Loss :  2.2038680769756898
```

```
In [29]: imp_df = pd.DataFrame({
             "Feature Name": X_train.columns,
             "Importance": dtree.feature_importances_
         })
         fi = imp_df.sort_values(by="Importance", ascending=False)

         fi2 = fi.head(10)
         plt.figure(figsize=(10,8))
         sns.barplot(data=fi2, x='Importance', y='Feature Name')
         plt.title('Top 10 Feature Importance Each Attributes (Decision Tree)', fontsiz
         plt.xlabel ('Importance', fontsize=16)
         plt.ylabel ('Feature Name', fontsize=16)
         plt.show()
```

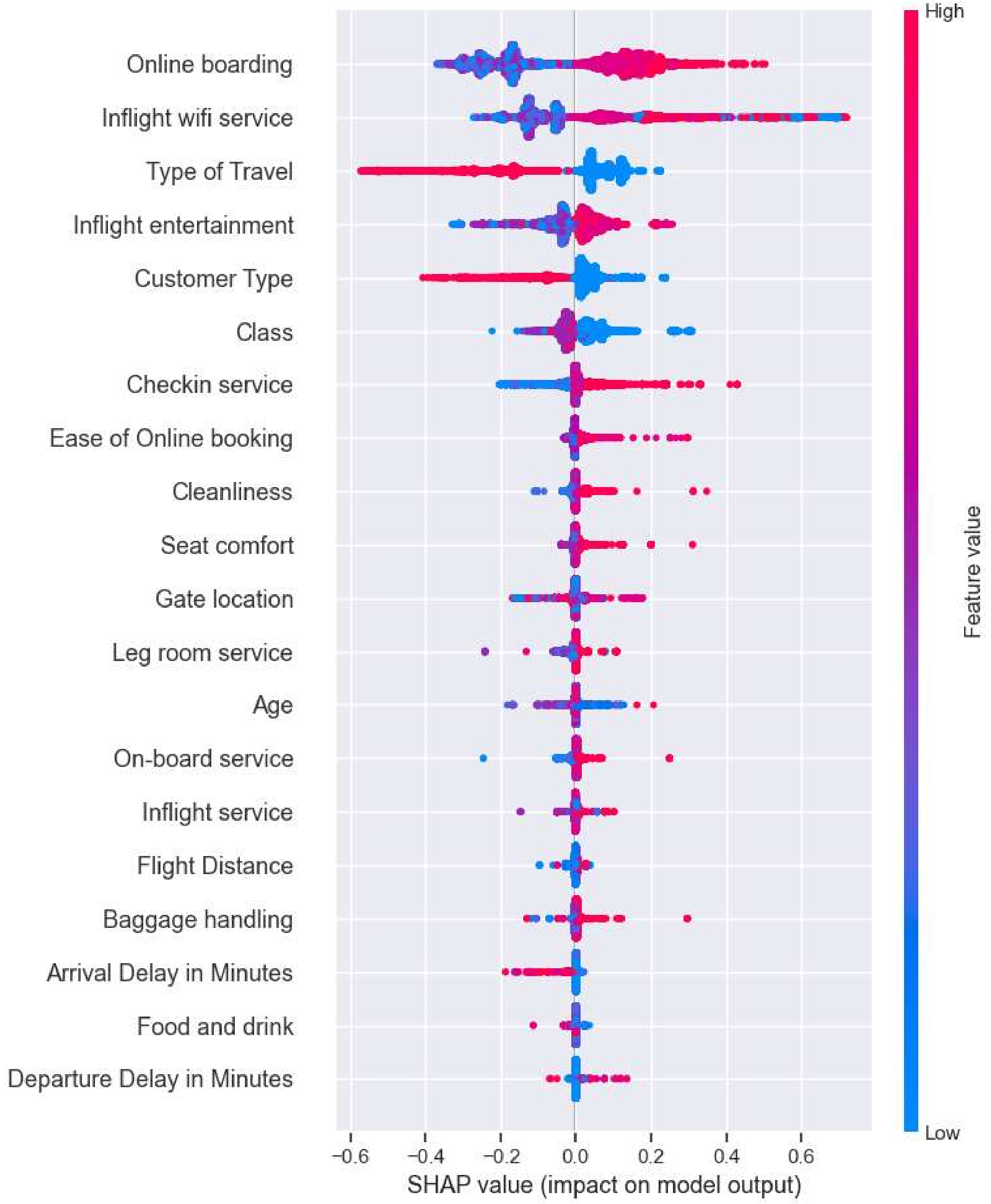


```
In [30]: import shap
         explainer = shap.TreeExplainer(dtree)
         shap_values = explainer.shap_values(X_test)
         shap.summary_plot(shap_values, X_test)
```



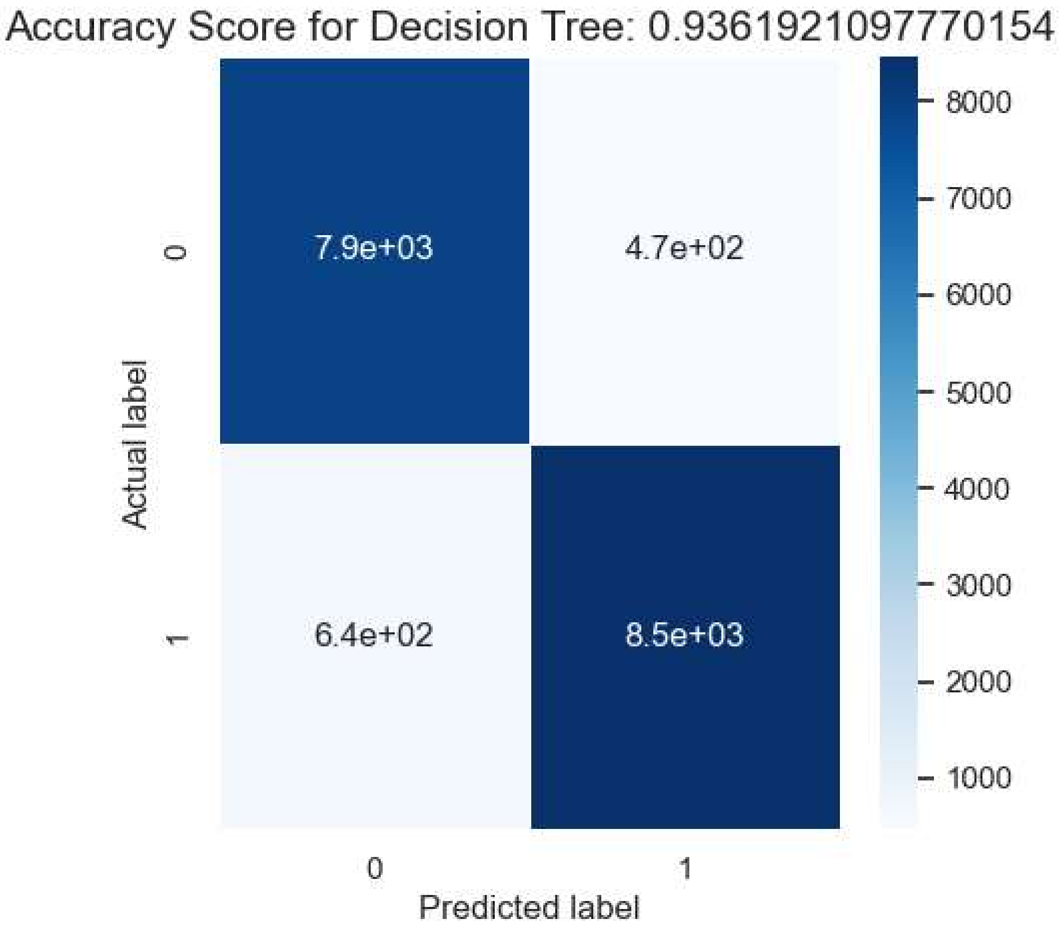
In [31]:

```
# compute SHAP values
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.column
```



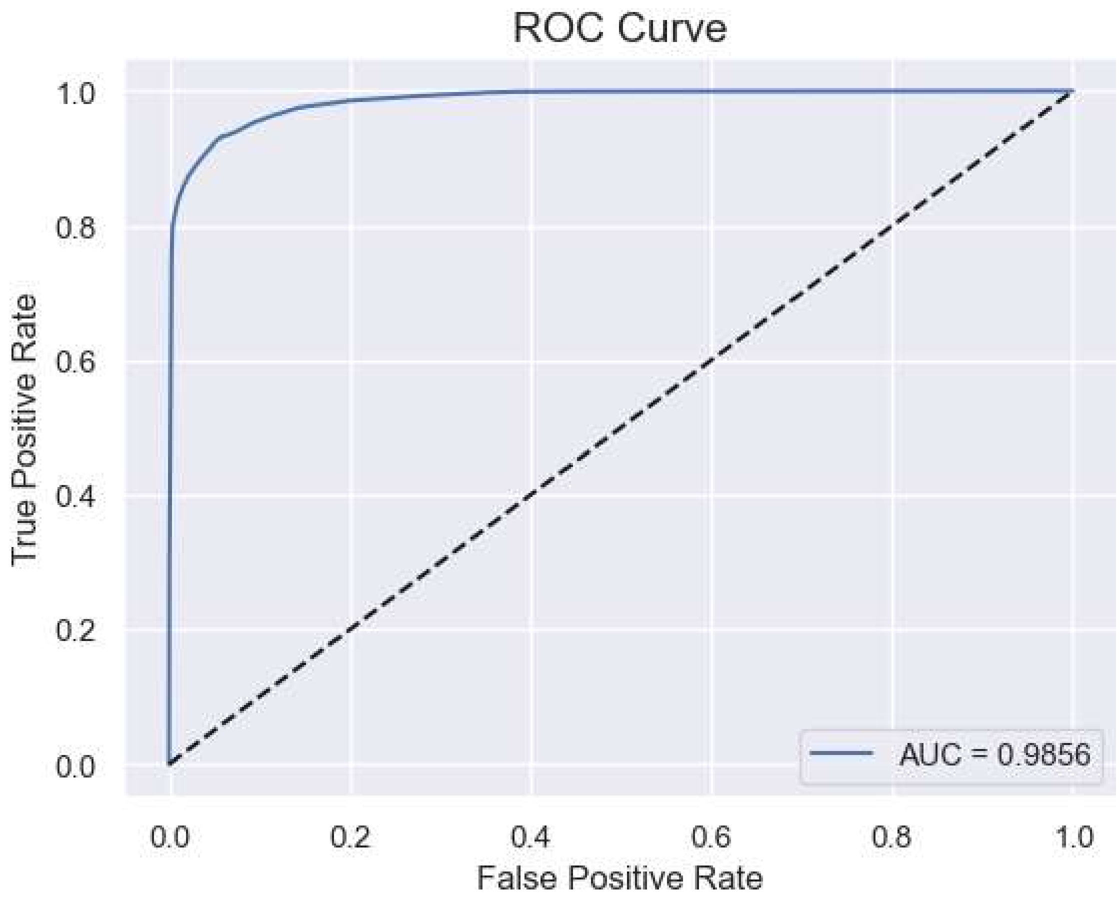
```
In [32]: from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test, y_pred)
         plt.figure(figsize=(5,5))
         sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
         plt.ylabel('Actual label')
         plt.xlabel('Predicted label')
         all_sample_title = 'Accuracy Score for Decision Tree: {0}'.format(dtree.score(
         plt.title(all_sample_title, size = 15)
```

```
Out[32]: Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.9361921097770154')
```



In [33]:
```
from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = dtree.predict_proba(X_test)[:][:,1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test), columns=['y_ac
df_actual_predicted.index = y_test.index

fpr, tpr, tr = roc_curve(df_actual_predicted['y_actual'], df_actual_predicted[
auc = roc_auc_score(df_actual_predicted['y_actual'], df_actual_predicted['y_pr

plt.plot(fpr, tpr, label='AUC = %0.4f' %auc)
plt.plot(fpr, fpr, linestyle = '--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve', size = 15)
plt.legend()
```

Out[33]:
```
<matplotlib.legend.Legend at 0x229b4b3d8e0>
```



Random Forest

In [34]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
rfc = RandomForestClassifier()
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 5, 10],
    'max_features': ['sqrt', 'log2', None]
}

# Perform a grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(rfc, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'max_depth': None, 'max_features': 'sqrt', 'n_estimators': 200}
```

In [35]:

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=0, max_features='sqrt', n_estimators
rfc.fit(X_train, y_train)
```

Out[35]:

```
RandomForestClassifier(max_features='sqrt', n_estimators=200, random_state=0)
```

In [36]:

```
y_pred = rfc.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 97.61 %
```

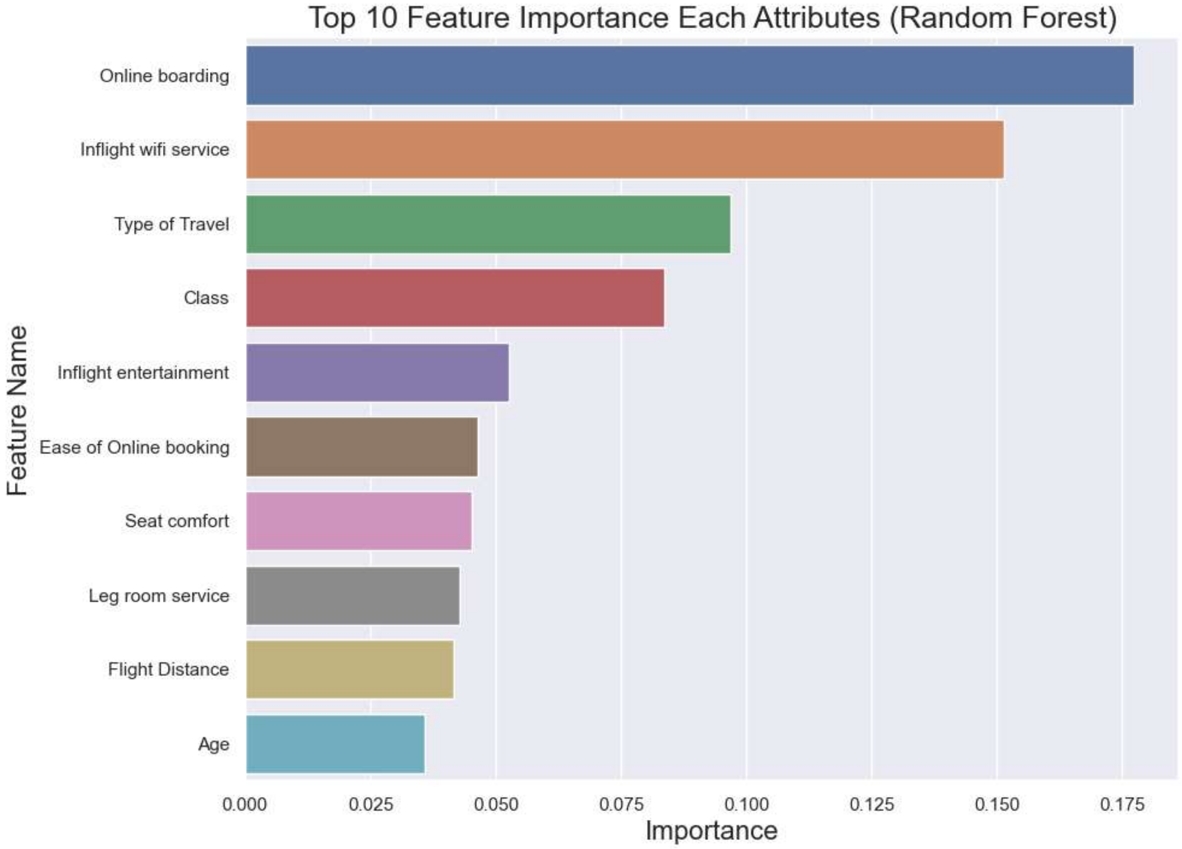
In [37]:

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.9761006289308176
Precision Score :  0.9761006289308176
Recall Score :  0.9761006289308176
Jaccard Score :  0.9533169533169533
Log Loss :  0.8254636282098088
```

In [38]:

```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": rfc.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Top 10 Feature Importance Each Attributes (Random Forest)', fontsiz
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```


Top 10 Feature Importance Each Attributes (Random Forest)
Feature Name
Online boarding
Inflight wifi service
Type of Travel
Class
Inflight entertainment
Ease of Online booking
Seat comfort
Leg room service
Flight Distance
Age
Importance