# Task 2

## Predictive modeling of customer bookings

This Jupyter notebook includes some code to get you started with this predictive modeling task. We will use various packages for data manipulation, feature engineering and machine learning.

### Exploratory data analysis

First, we must explore the data in order to better understand what we have and the statistical properties of the dataset.

In [1]:

```
import pandas as pd
```

In [6]:

```
df = pd.read_csv("D:\Forage\Data Science\data/customer_booking.csv", encoding="ISO-8859-1")
df.head()
```

Out[6]:

| | num_passengers | sales_channel | trip_type | purchase_lead | length_of_stay | flight_hour | flight_day | route | booking_origin | wants_extra_baggage | wants_preferred_seat | wants_in_flight_meals | flight_duration | booking_complete |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2 | Internet | RoundTrip | 262 | 19 | 7 | Sat | AKLDEL | New Zealand | 1 | 0 | 0 | 5.52 | 0 |
| **1** | 1 | Internet | RoundTrip | 112 | 20 | 3 | Sat | AKLDEL | New Zealand | 0 | 0 | 0 | 5.52 | 0 |
| **2** | 2 | Internet | RoundTrip | 243 | 22 | 17 | Wed | AKLDEL | India | 1 | 1 | 0 | 5.52 | 0 |
| **3** | 1 | Internet | RoundTrip | 96 | 31 | 4 | Sat | AKLDEL | New Zealand | 0 | 0 | 1 | 5.52 | 0 |
| **4** | 2 | Internet | RoundTrip | 68 | 22 | 15 | Wed | AKLDEL | India | 1 | 0 | 1 | 5.52 | 0 |

The .head() method allows us to view the first 5 rows in the dataset, this is useful for visual inspection of our columns

In [7]:

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   num_passengers         50000 non-null  int64  
 1   sales_channel          50000 non-null  object 
 2   trip_type              50000 non-null  object 
 3   purchase_lead          50000 non-null  int64  
 4   length_of_stay         50000 non-null  int64  
 5   flight_hour            50000 non-null  int64  
 6   flight_day             50000 non-null  object 
 7   route                  50000 non-null  object 
 8   booking_origin         50000 non-null  object 
 9   wants_extra_baggage    50000 non-null  int64  
 10  wants_preferred_seat   50000 non-null  int64  
 11  wants_in_flight_meals  50000 non-null  int64  
 12  flight_duration        50000 non-null  float64
 13  booking_complete       50000 non-null  int64  
dtypes: float64(1), int64(8), object(5)
memory usage: 5.3+ MB
```

The .info() method gives us a data description, telling us the names of the columns, their data types and how many null values we have. Fortunately, we have no null values. It looks like some of these columns should be converted into different data types, e.g. flight_day.

To provide more context, below is a more detailed data description, explaining exactly what each column means:

- num_passengers = number of passengers travelling
- sales_channel = sales channel booking was made on
- trip_type = trip Type (Round Trip, One Way, Circle Trip)
- purchase_lead = number of days between travel date and booking date
- length_of_stay = number of days spent at destination
- flight_hour = hour of flight departure
- flight_day = day of week of flight departure

- route = origin -> destination flight route
- booking_origin = country from where booking was made
- wants_extra_baggage = if the customer wanted extra baggage in the booking
- wants_preferred_seat = if the customer wanted a preferred seat in the booking
- wants_in_flight_meals = if the customer wanted in-flight meals in the booking
- flight_duration = total duration of flight (in hours)
- booking_complete = flag indicating if the customer completed the booking

Before we compute any statistics on the data, lets do any necessary data conversion

In [8]:

```
df["flight_day"].unique()
```

Out[8]:

```
array(['Sat', 'Wed', 'Thu', 'Mon', 'Sun', 'Tue', 'Fri'], dtype=object)
```

In [9]:

```
mapping = {
    "Mon": 1,
    "Tue": 2,
    "Wed": 3,
    "Thu": 4,
    "Fri": 5,
    "Sat": 6,
    "Sun": 7,
}

df["flight_day"] = df["flight_day"].map(mapping)
```

In [10]:

```
df["flight_day"].unique()
```

Out[10]:

```
array([6, 3, 4, 1, 7, 2, 5], dtype=int64)
```

In [11]:

```
df.describe()
```

Out[11]:

| | num_passengers | purchase_lead | length_of_stay | flight_hour | flight_day | wants_extra_baggage | wants_preferred_seat | wants_in_flight_meals | flight_duration | booking_complete |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 50000.000000 | 50000.000000 | 50000.00000 | 50000.00000 | 50000.000000 | 50000.000000 | 50000.000000 | 50000.000000 | 50000.000000 | 50000.000000 |
| **mean** | 1.591240 | 84.940480 | 23.04456 | 9.06634 | 3.814420 | 0.668780 | 0.296960 | 0.427140 | 7.277561 | 0.149560 |
| **std** | 1.020165 | 90.451378 | 33.88767 | 5.41266 | 1.992792 | 0.470657 | 0.456923 | 0.494668 | 1.496863 | 0.356643 |
| **min** | 1.000000 | 0.000000 | 0.00000 | 0.00000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 4.670000 | 0.000000 |
| **25%** | 1.000000 | 21.000000 | 5.00000 | 5.00000 | 2.000000 | 0.000000 | 0.000000 | 0.000000 | 5.620000 | 0.000000 |
| **50%** | 1.000000 | 51.000000 | 17.00000 | 9.00000 | 4.000000 | 1.000000 | 0.000000 | 0.000000 | 7.570000 | 0.000000 |
| **75%** | 2.000000 | 115.000000 | 28.00000 | 13.00000 | 5.000000 | 1.000000 | 1.000000 | 1.000000 | 8.830000 | 0.000000 |
| **max** | 9.000000 | 867.000000 | 778.00000 | 23.00000 | 7.000000 | 1.000000 | 1.000000 | 1.000000 | 9.500000 | 1.000000 |

The .describe() method gives us a summary of descriptive statistics over the entire dataset (only works for numeric columns). This gives us a quick overview of a few things such as the mean, min, max and overall distribution of each column.

From this point, you should continue exploring the dataset with some visualisations and other metrics that you think may be useful. Then, you should prepare your dataset for predictive modelling. Finally, you should train your machine learning model, evaluate it with performance metrics and output visualisations for the contributing variables. All of this analysis should be summarised in your single slide.

In [13]:

```
# Check for missing values in the dataset
df_missing_values = df.isnull().sum()
df_missing_values
```

Out[13]:

```
num_passengers          0
sales_channel           0
trip_type               0
purchase_lead           0
length_of_stay          0
flight_hour             0
flight_day              0
route                   0
booking_origin          0
wants_extra_baggage     0
```

```
wants_preferred_seat     0
wants_in_flight_meals    0
flight_duration          0
booking_complete         0
dtype: int64
```

In [14]:

```
# Check for any categorical variables that need encoding
categorical_cols = df.select_dtypes(include=['object']).columns
categorical_cols
```

Out[14]:

```
Index(['sales_channel', 'trip_type', 'route', 'booking_origin'], dtype='object')
```

In [15]:

```
# Check for numerical variables that may need scaling
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns
numerical_cols
```

Out[15]:

```
Index(['num_passengers', 'purchase_lead', 'length_of_stay', 'flight_hour',
       'flight_day', 'wants_extra_baggage', 'wants_preferred_seat',
       'wants_in_flight_meals', 'flight_duration', 'booking_complete'],
      dtype='object')
```

In [16]:

```
# Output the missing values, categorical columns, and numerical columns
df_missing_values, categorical_cols.tolist(), numerical_cols.tolist()
```

Out[16]:

```
(num_passengers           0
 sales_channel            0
 trip_type                0
 purchase_lead            0
 length_of_stay           0
 flight_hour              0
 flight_day               0
 route                    0
 booking_origin           0
 wants_extra_baggage      0
```

```
 wants_preferred_seat     0
 wants_in_flight_meals    0
 flight_duration          0
 booking_complete         0
 dtype: int64,
 ['sales_channel', 'trip_type', 'route', 'booking_origin'],
 ['num_passengers',
  'purchase_lead',
  'length_of_stay',
  'flight_hour',
  'flight_day',
  'wants_extra_baggage',
  'wants_preferred_seat',
  'wants_in_flight_meals',
  'flight_duration',
  'booking_complete'])
```

In [17]:

```
from sklearn.preprocessing import StandardScaler, LabelEncoder
```

In [18]:

```
# Initialize the label encoder
label_encoder = LabelEncoder()
```

In [19]:

```
# Encode categorical variables
for col in categorical_cols:
    df[col] = label_encoder.fit_transform(df[col])
```

In [20]:

```
# Initialize the standard scaler
scaler = StandardScaler()
```

In [21]:

```
# Scale numerical variables
for col in numerical_cols:
    df[col] = scaler.fit_transform(df[[col]])
```

In [24]:

```
# Display the first few rows of the updated dataframe
updated_df_head = df.head()
```

In [25]:

```
# Output the updated dataframe head
updated_df_head
```

Out[25]:

| | num_passengers | sales_channel | trip_type | purchase_lead | length_of_stay | flight_hour | flight_day | route | booking_origin | wants_extra_baggage | wants_preferred_seat | wants_in_flight_meals | flight_duration | booking_complete |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.400684 | 0 | 2 | 1.957530 | -0.119353 | -0.381764 | 1.096754 | 0 | 61 | 0.703747 | -0.649919 | -0.863497 | -1.174175 | -0.419359 |
| **1** | -0.579559 | 0 | 2 | 0.299164 | -0.089844 | -1.120780 | 1.096754 | 0 | 61 | -1.420965 | -0.649919 | -0.863497 | -1.174175 | -0.419359 |
| **2** | 0.400684 | 0 | 2 | 1.747470 | -0.030824 | 1.465775 | -0.408687 | 0 | 36 | 0.703747 | 1.538654 | -0.863497 | -1.174175 | -0.419359 |
| **3** | -0.579559 | 0 | 2 | 0.122272 | 0.234761 | -0.936026 | 1.096754 | 0 | 61 | -1.420965 | -0.649919 | 1.158082 | -1.174175 | -0.419359 |
| **4** | 0.400684 | 0 | 2 | -0.187290 | -0.030824 | 1.096267 | -0.408687 | 0 | 36 | 0.703747 | -0.649919 | 1.158082 | -1.174175 | -0.419359 |

In [31]:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
```

In [32]:

```
# Convert the scaled 'booking_complete' back to categorical
# Assuming that the negative value corresponds to 0 and the positive to 1
# We will use the median of the column as the threshold for conversion
median_val = df['booking_complete'].median()
df['booking_complete'] = (df['booking_complete'] > median_val).astype(int)
```

In [33]:

```
# Now let's split the data again and train the model
X = df.drop('booking_complete', axis=1)
y = df['booking_complete']
```

In [34]:

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [35]:

```
# Initialize the Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
```

In [36]:

```
# Train the model
rf_model.fit(X_train, y_train)
```

Out[36]:

```
RandomForestClassifier(random_state=42)
```

In [37]:

```
# Make predictions
y_pred = rf_model.predict(X_test)
```

In [39]:

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

Out[39]:

```
0.8544
```

In [41]:

```
# Generate classification report
report = classification_report(y_test, y_pred)
report
```

Out[41]:

```
'              precision    recall  f1-
score   support\n\n           0       0.86      0.98      0.92      8520\n           1       0.54      0.11      0.18      1480\n\n
accuracy                           0.85     10000\n   macro avg       0.70      0.55      0.55     10000\nweighted
avg       0.82      0.85      0.81     10000\n'
```

In [42]:

```
# Output the accuracy and the classification report
print('Accuracy:', accuracy)
print('Classification Report:\n', report)
```

```
Accuracy: 0.8544
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.98      0.92      8520
           1       0.54      0.11      0.18      1480

    accuracy                           0.85     10000
   macro avg       0.70      0.55      0.55     10000
weighted avg       0.82      0.85      0.81     10000
```

In [43]:

```python
from sklearn.model_selection import cross_val_score
```

In [44]:

```python
# Perform cross-validation
cv_scores = cross_val_score(rf_model, X, y, cv=5, scoring='accuracy')
cv_scores
```

Out[44]:

```
array([0.8509, 0.4393, 0.2513, 0.3701, 0.5139])
```

In [45]:

```python
# Output the cross-validation scores
cv_scores
```

Out[45]:

```
array([0.8509, 0.4393, 0.2513, 0.3701, 0.5139])
```

In [48]:

```python
import matplotlib.pyplot as plt
import numpy as np
```

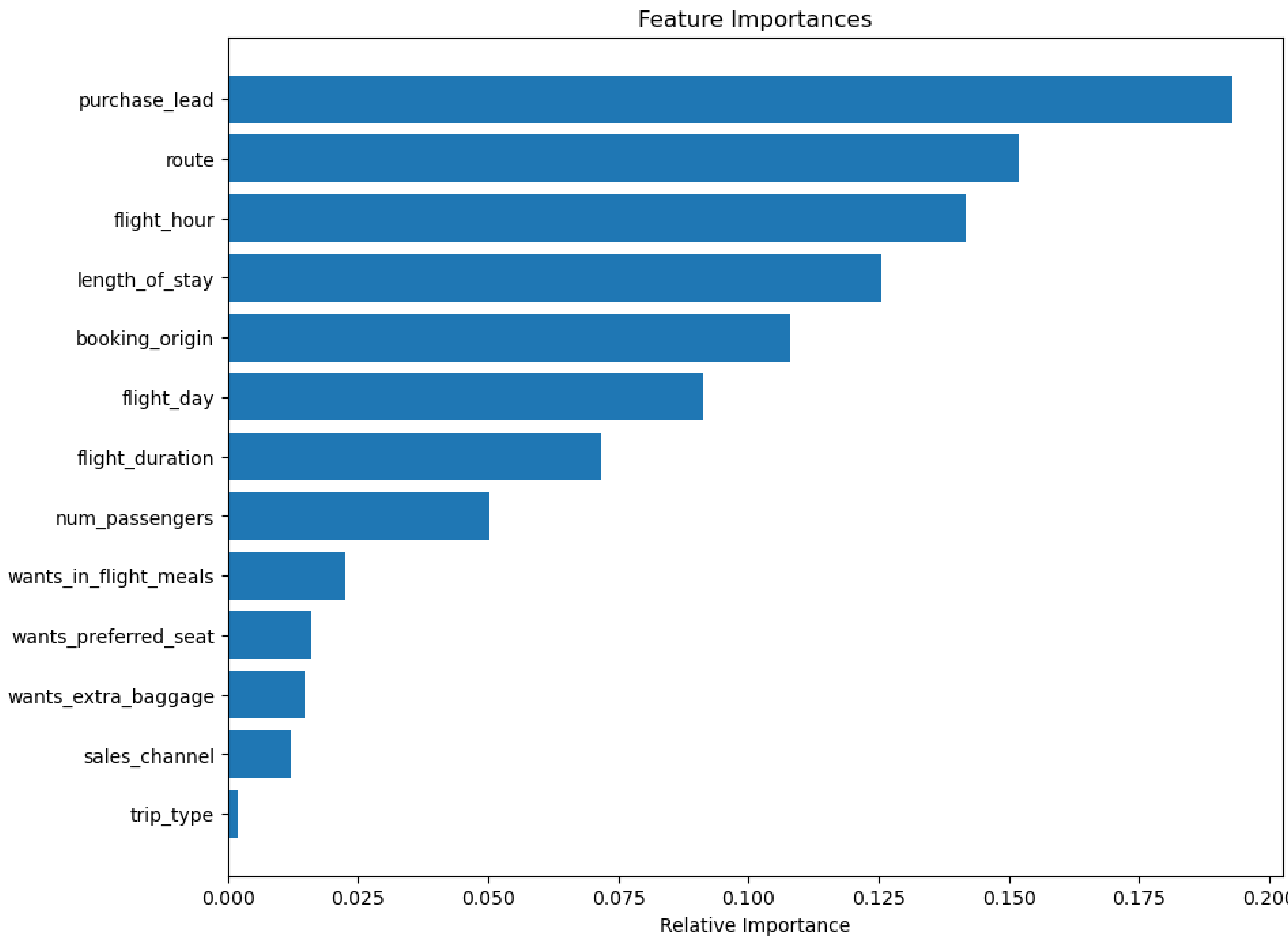
In [50]:

```python
# Get feature importances
importances = rf_model.feature_importances_
features = X.columns
```

```python
indices = np.argsort(importances)
```

In [51]:

```python
# Plot feature importances
plt.figure(figsize=(10, 8))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```


Feature Importances
purchase_lead
route
flight_hour
length_of_stay
booking_origin
flight_day
flight_duration
num_passengers
wants_in_flight_meals
wants_preferred_seat
wants_extra_baggage
sales_channel
trip_type
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
Relative Importance