

British Airways

✳ Status Done

overview and objectives:

Business Context

- British Airways (BA) offers lounge access as part of its **premium travel experience**.
- Managing lounge demand is critical for **customer satisfaction** and **efficient use of space/resources**.
- BA is planning **future operations at Heathrow Terminal 3** and needs to anticipate **lounge demand**.

Problem

- Lounge eligibility depends on **customer loyalty tiers and travel class**.
- Future schedules are **unpredictable**, making exact forecasts difficult.
- BA needs a **flexible and scalable approach** to forecast lounge demand.

Your Task

- Create a **lookup table** that estimates **lounge eligibility percentages** across different **flight groupings**.
- Groupings should be based on **meaningful dimensions** such as:
 - **Time of day** (morning, afternoon, evening)
 - **Route type** (domestic, short-haul, long-haul)
 - **Regional destination** (Europe, North America, Asia, etc.)
- Apply **logical assumptions** to estimate what proportion of passengers fall into each **lounge access tier**.

Purpose

- The lookup table will allow BA to **anticipate lounge demand without exact flight or aircraft details**.
- This helps the **Airport Planning team**:
 - Identify where **lounge investments** are most needed.
 - Ensure BA maintains a **seamless premium customer experience** as operations evolve.

Deliverables

1. A **lookup table** of lounge eligibility percentages across flight groupings.
2. A **justification/explanation** for your assumptions and grouping choices.

Understanding Lounge eligibility

To begin modeling lounge demand, it's important to understand who is typically eligible for lounge access. Lounge eligibility is generally based on customer loyalty status and travel class, with different access tiers offering varying levels of amenities.

British Airways Lounge Tiers at Terminal 3:

Tier	Lounge Name	Access Eligibility
1	Concorde Room	<ul style="list-style-type: none">• First Class customers• BA premier cardholders• BA Gold Guest List members
2	First Lounge	<ul style="list-style-type: none">• BA Gold Members
3	Club Lounge	<ul style="list-style-type: none">• BA Silver cardholders• Club World (business class) customers

Each tier supports a different group of travelers, and lounge capacity planning depends on forecasting how many eligible passengers fall into each of these categories.

Creating eligibility assumptions

Now that you understand the lounge tiers, it's time to think about how you'll estimate the percentage of customers eligible for each tier across a flight schedule. Since BA is planning far into the future, your model needs to be flexible and based on high-level groupings—not specific flight numbers or aircraft types.

Your goal is to create a lookup table that estimates lounge eligibility using clear, scalable categories. To do this, you'll need to decide how to group flights and make logical assumptions.

Common groups include:

- **Time of day:** Early morning, mid-day, evening departures.
- **Type of route:** Short-haul vs. long-haul
- **Region or destination group:** Europe, North America, Asia, etc.

You'll estimate what proportion of passengers in each group are likely to be eligible for:

- **Tier 1:** Concorde Room
- **Tier 2:** First Lounge
- **Tier 3:** Club Lounge

There is no single correct approach—what matters most is that your assumptions are logical, justifiable, and easy to apply to future schedules.



British Airways Lounge Demand Modeling – Forage Simulation

1. Background & Task

Lounge access is a crucial part of the **premium travel experience**, and British Airways (BA) wants to anticipate lounge demand at **Heathrow Terminal 3**. The task was to create a **generalized lookup table** that estimates lounge eligibility percentages across different groupings of flights.

This lookup table should:

- Be **generalized**, not flight-specific (usable for future schedules).
 - Provide **eligibility percentages** for **Tier 1, Tier 2, and Tier 3** lounges.
 - Be supported by **assumptions** and **reasoning** that can guide BA's future planning.
-

2. Dataset Provided

We were given a **flight schedule dataset (~1000 samples)** with the following relevant fields:

- **TIME_OF_DAY** – flight departure time categorized as Morning, Lunchtime, Afternoon, Evening.
 - **HAUL** – whether the flight is Long-haul (outside Europe) or Short-haul (within Europe).
 - **ARRIVAL_COUNTRY** – flight destination (optional for grouping).
 - **FIRST_CLASS_SEATS, BUSINESS_CLASS_SEATS, ECONOMY_SEATS** – seat capacity by class.
 - **TIER1_ELIGIBLE_PAX, TIER2_ELIGIBLE_PAX, TIER3_ELIGIBLE_PAX** – number of passengers eligible for each lounge tier.
-

3. Initial Approach & Groupings

The task required **grouping flights in a meaningful way**. After reviewing the dataset, we ruled out **flight number, exact departure times, and dates** because they are too granular and wouldn't scale for future schedules.

We decided to group by:

- **Time of Day (Morning, Lunchtime, Afternoon, Evening)**
- **Haul (Long-haul, Short-haul)**

This grouping makes sense because:

- Lounge demand is strongly tied to whether the flight is **short-haul or long-haul**.
 - Time of day also reflects **traveler profiles** (business travelers in the morning, leisure travelers in the evening, etc.).
 - The grouping is **simple, scalable, and generalizable**.
-

4 EDA (Exploratory Data Analysis)

We started with some initial checks:

- Verified the **distribution of flights by haul and time of day**.
- Computed the **total passengers per flight**:

$$\text{TOTAL_PAX} = \text{FIRST_CLASS_SEATS} + \text{BUSINESS_CLASS_SEATS} + \text{ECONOMY_SEATS}$$

- Computed **Tier eligibility as % of total passengers**:

$$\text{TierX \% of Total Pax} = \frac{\text{TierX Eligible Pax}}{\text{TOTAL_PAX}} \times 100$$

This gave us our **Option A chart**, which helped identify what proportion of **all passengers** are lounge-eligible.

We then computed **Tier shares within lounge-eligible passengers**:

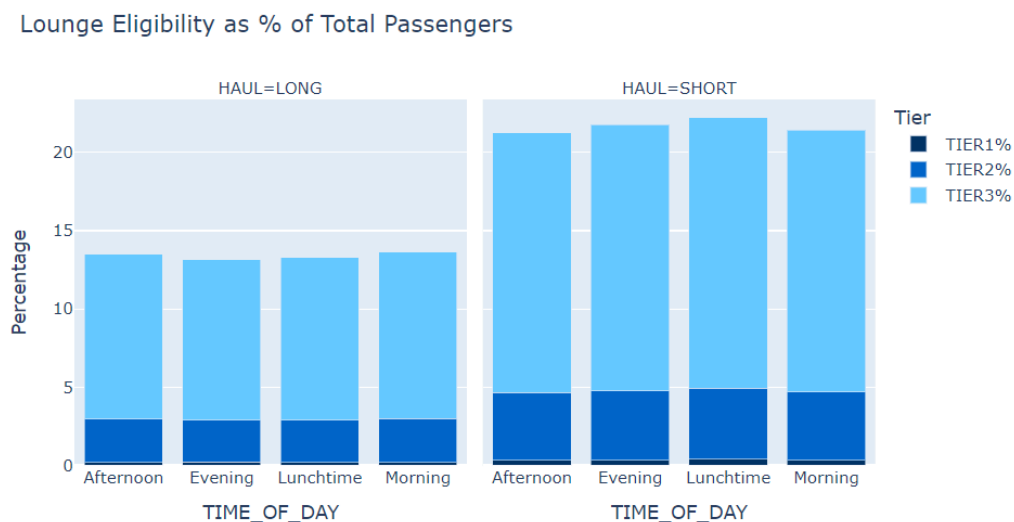
$$\text{TierX Share} = \frac{\text{TierX Eligible Pax}}{\text{Tier1} + \text{Tier2} + \text{Tier3 Eligible Pax}} \times 100$$

This gave us our **Option B chart**, showing how the **lounge-eligible population is distributed across tiers**.

5. Key Visualizations

Option A: Eligible as % of Total Passengers

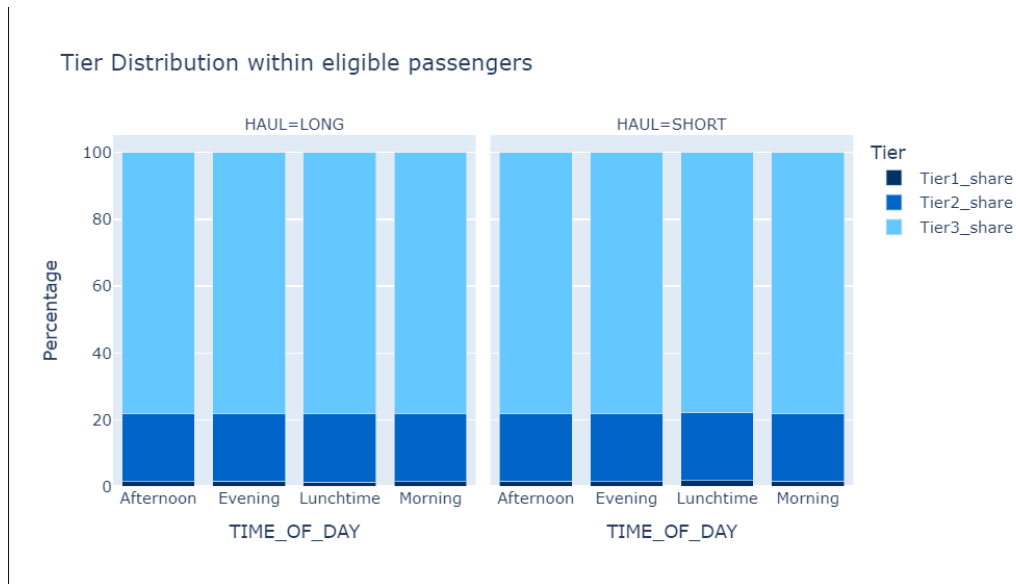
- **Y-axis:** % of total passengers eligible for lounges.
- **X-axis:** Time of Day.
- **Facets:** Long vs. Short Haul.
- **Insight:** Around **13–14% of long-haul** and **21–22% of short-haul** passengers are lounge eligible. Short-haul shows a higher proportion of eligible passengers.



Option B: Tier Distribution within Eligible Passengers

- **Y-axis:** % share of each tier (Tier 1, Tier 2, Tier 3).

- **X-axis:** Time of Day.
- **Facets:** Long vs. Short Haul.
- **Insight:** Distribution is very stable across time of day and haul:
 - **Tier 3 dominates (~78%)**
 - **Tier 2 (~20%)**
 - **Tier 1 (~1-2%)**



6. Lookup Table

Based on our Option B chart, we constructed the **Lounge Eligibility Lookup Table**.

This shows the **average % eligibility by tier across all groupings**.

Grouping (Time + Haul)	Tier 1 %	Tier 2 %	Tier 3 %	Notes
Morning, Long-haul	~1.4%	~20%	~78%	Stable pattern
Morning, Short-haul	~1.5%	~20%	~78%	Stable pattern
Lunchtime, Long-haul	~1.3%	~20%	~78%	Stable pattern
Lunchtime, Short-haul	~1.6%	~20%	~78%	Stable pattern
Afternoon, Long-haul	~1.5%	~20%	~78%	Stable pattern
Afternoon, Short-haul	~1.6%	~20%	~78%	Stable pattern
Evening, Long-haul	~1.6%	~20%	~78%	Stable pattern
Evening, Short-haul	~1.5%	~20%	~78%	Stable pattern

Note: Since the distribution is nearly identical across all time/haul groupings, BA can safely use an **average Tier split**:

- **Tier 1:** ~1-2%
- **Tier 2:** ~20%

- **Tier 3: ~78%**
-

7. Insights & Recommendations

1. **Tier 3 dominates (78%)** – BA should prioritize **scalability and space** in Tier 3 lounges to avoid overcrowding.
 2. **Tier 2 (20%)** – Focus on **balancing service quality** for frequent business travelers.
 3. **Tier 1 (1–2%)** – While small in number, this group is critical for **brand loyalty and premium experience**. Invest in **service quality, exclusivity, and premium features**.
 4. **Haul and time of day don't matter much** – Patterns are consistent, so BA doesn't need different strategies by these categories.
 5. **Seat classes matter** – The seat distribution (First, Business, Economy) directly determines lounge eligibility. This strengthens our Tier analysis since Tier 3 aligns heavily with economy/business volume.
-

8. Justification Questions (Final Responses)

Q1: How did you choose to group the flights in your model?

I grouped flights by **time of day** and **haul** (long-haul vs. short-haul). This approach avoids granular flight-specific data and ensures the lookup table is **generalized and scalable** for future schedules. These categories reflect meaningful real-world differences in lounge demand.

Q2: Why do you believe grouping by time of day and haul makes sense for this kind of modeling?

Grouping by **haul** reflects how long-haul passengers typically arrive earlier and have greater lounge needs compared to short-haul passengers. Grouping by **time of day** reflects passenger flow patterns (business vs. leisure). Together, these provide a simple yet robust framework for modeling lounge demand.

Q3: What assumptions did you make while creating this model, and why?

- **Excluded flight numbers and dates** to keep the model scalable.
 - Assumed **Tier 3 demand dominates** based on seat class distribution (economy/business).
 - Assumed **Tier 1 and Tier 2 serve niche, elite travelers** and will remain small in percentage but high in importance.
 - Assumed **patterns are stable across hauls and times**, based on consistent percentages observed in the data.
-

Q4: How will your model scale to future schedules, especially if BA changes flights, routes, or times?

The model is scalable because it groups by **general categories (time + haul)** rather than specific flights. Even with new schedules or routes, seat class distribution and lounge eligibility patterns are unlikely to change drastically. Tier 3 will likely remain dominant, while Tier 1 and 2 will remain smaller but strategically important. The lookup table can be easily **re-applied to new datasets** to adjust percentages if conditions change.

✓ End Deliverables:

- **Lookup Table (Excel)** – with Tier 1, 2, 3 percentages by group.
- **Justification (Excel)** – with answers to the 4 reflection questions.

How would the BA data science team approach this?

Great work! Take a look at the example answer below to see how a member of the Data Science team at BA might approach the task. This example demonstrates how to group flights and apply logical eligibility percentages across lounge tiers. Your final output may look different, but what matters most is that your assumptions are clearly structured and well-reasoned.

Grouping	Tier 1 %	Tier 2 %	Tier 3 %	Notes
Short-haul AM	3%	15%	55%	Lower eligibility due to early business routes
Short-haul PM	4%	18%	60%	Slightly more leisure travelers
Long-haul AM	8%	25%	70%	Mix of business and high-tier loyalty members
Long-haul PM	10%	30%	75%	Higher First and Club passengers
European (All Times)	5%	20%	65%	Typical short-to-mid haul loyalty makeup
North America	12%	35%	70%	Strong premium demand and loyalty tiers

Justification:

Prompt	Your Response
1. How did you group the flights in your table?	I grouped flights by a combination of time of day (AM vs. PM) and route type (short-haul vs. long-haul).
2. Why did you choose that regrouping method?	This method offers a practical balance of simplicity and insight. Flight types and departure times tend to influence customer profiles and lounge usage behaviors.
3. What assumptions did you make about passenger eligibility?	I assumed short-haul flights have ever fewer premium passengers overall, while long-haul flights, especially PM, carry a higher mix of business and loyalty members.
4. How can your model apply to future or changing flight schedules?	Since my groupings are based on patterns rather than specific destinations, the table can easily adapt to future schedules or route updates.

Task 2: Building a Predictive Model

Here is the background information on your task

Customers are more empowered than ever because they have access to a wealth of information at their fingertips. This is one of the reasons the buying cycle is very different to what it used to be. Today, if you're hoping that a customer purchases your flights or holidays as they come into the airport, you've already lost! Being reactive in this situation is not ideal; airlines must be proactive in order to acquire customers before they embark on their holiday.

This is possible with the use of data and predictive models. The most important factor with a predictive model is the quality of the data you use to train the machine learning algorithms. For this task, you must manipulate and prepare the provided customer booking data so that you can build a high-quality predictive model.

With your predictive model, it is important to interpret the results in order to understand how “predictive” the data really was and whether we can feasibly use it to predict the target outcome (customers buying holidays). Therefore, you should evaluate the model's performance and output how each variable contributes to the predictive model's power.

Here is your task

Explore and prepare the dataset

First, spend some time exploring the dataset in the **“Getting Started”** Jupyter Notebook provided in the Resources section below to understand the different columns and some basic statistics of the dataset. Then, you should consider how to prepare the dataset for a predictive model. You should think about any new features you want to create in order to make your model even better. You can make use of the resources provided to get you started with this task.

Train a machine learning model

When your data is ready for modelling, you should train a machine learning model to be able to predict the target outcome, which is a customer making a booking. For this task, you should use an algorithm that easily allows you to output information about how each variable within the model contributes to its predictive power. For example, a RandomForest is very good for this purpose.

Evaluate the model and present findings

After training your model, you should evaluate how well it performed by conducting cross-validation and outputting appropriate evaluation metrics. Furthermore, you should create a visualisation to interpret how each variable contributed to the model. Finally, you should summarise your findings in a single slide to be sent to your manager. Use the **“PowerPoint Template”** provided in the Resources section below to create your summary and make use of the links provided to help with this task.

It is recommended that the analysis portion of this task be done in Python.

Building the model pipeline

Basic Exploratory Data Analysis (EDA)

1. Dataset Overview

- Dataset contains **5000 samples**.
- No missing values (`df.info()` showed all non-null).
- Target variable: `booking_complete`
 - Binary classification (0 = not booked, 1 = booked).

2. Initial Checks

- **Column types:** mix of categorical (e.g., `sales_channel`, `trip_type`, `route`, `booking_origin`, `flight_day`) and numerical (e.g., `purchase_lead`, `length_of_stay`, `flight_hour`, `flight_duration`).
- **Data conversion (already done in starter code):**
 - `flight_day` (categorical: day of week) was mapped numerically to values **1-7**.

3. Descriptive Statistics (`df.describe()`)

- Provided summary statistics (mean, std, min, max, quartiles) for numerical features.
- Helped identify ranges, central tendencies, and possible outliers.

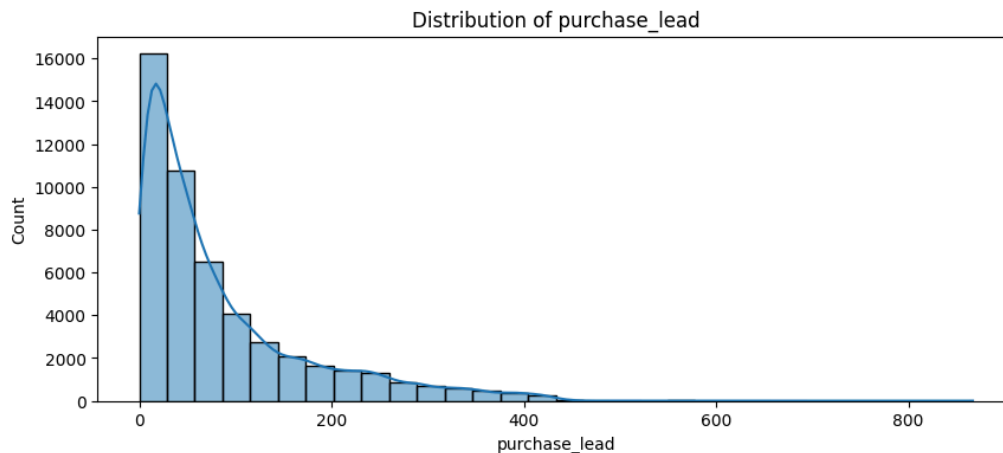
4. Distribution Analysis (Histograms + Skewness + Q-Q plots)

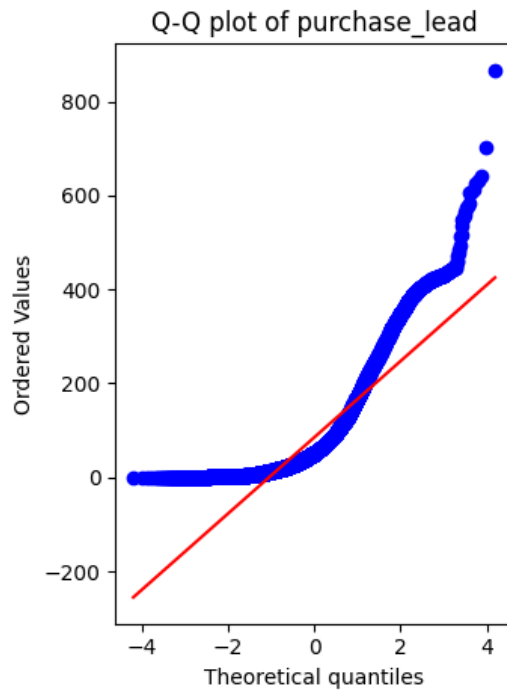
We analyzed each numerical feature with:

- **Histogram + KDE** → to visualize shape of distribution.
- **Skewness values** → to quantify asymmetry.
- **Q-Q plots** → to check deviations from normality.

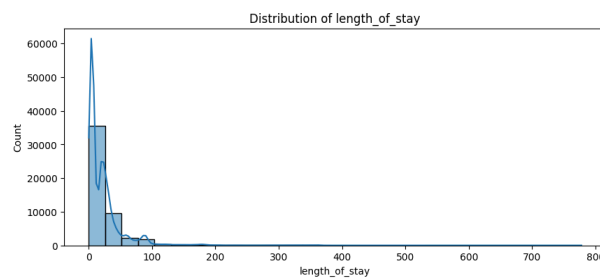
Findings:

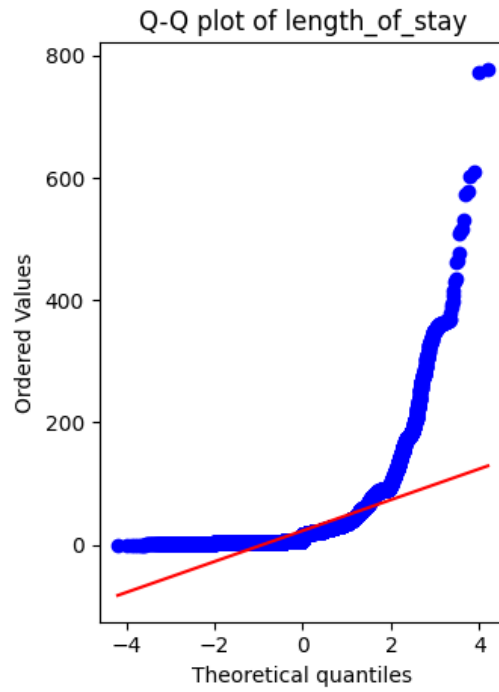
- `purchase_lead` :
 - Strong **right skew** (most customers book close to travel date, long tail of early planners).
 - Q-Q plot: upward curve in tail → heavy right tail.



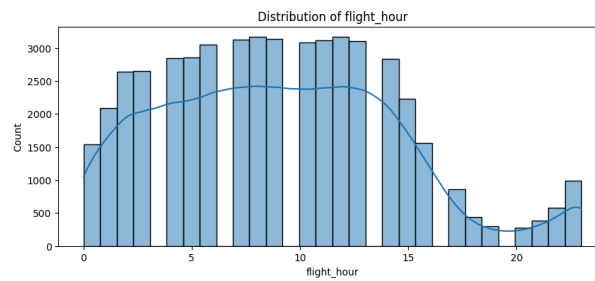


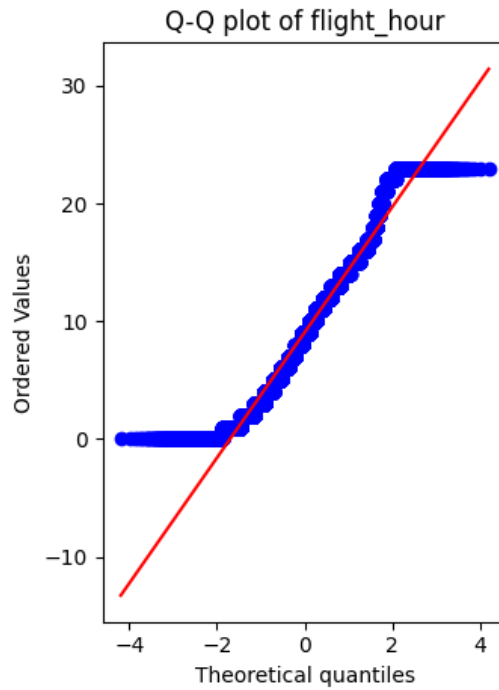
- **length_of_stay** :
 - Strong **right skew** (majority short stays, some very long stays).
 - Q-Q plot: similar heavy right tail.



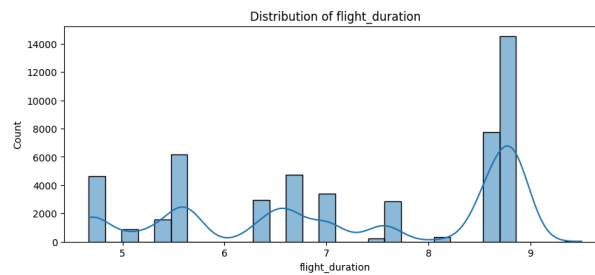


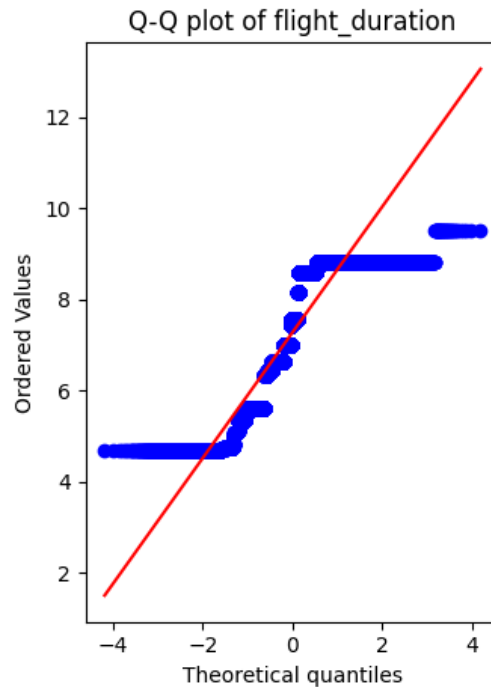
- **flight_hour** :
 - Approx. bell-shaped but slightly right-skewed.
 - Q-Q plot: classic S-shaped curve → mild heavy tails.





- **flight_duration** :
 - Also right-skewed (most flights shorter, fewer very long flights).
 - Q-Q plot: deviations in tails.
- **Binary features (e.g., extra_baggage, preferred_seat, inflight_meals):**
 - Only take values {0, 1}.
 - Not meaningful to analyze via skewness/normality — treated as categorical.





- `flight_day`:
 - Numeric but actually categorical (1–7 mapping).
 - Should not be interpreted as continuous numeric distribution.

5. Shapiro-Wilk Test (Normality Check)

- For ~5000 samples, test becomes overly sensitive (tiny deviations → low p-values).
- Results confirmed that most features deviate from perfect normality, which was already visible from histograms and Q-Q plots.
- For **Random Forest models**, this lack of normality is **not a problem**.
- If we later try **Logistic Regression**, skewness transformations may be required.

✓ Summary of Basic EDA:

- Dataset is clean (no nulls).
- Numerical features (`purchase_lead`, `length_of_stay`, `flight_duration`) are **heavily right-skewed**.
- Target variable is binary, ready for classification modeling.
- Normality assumption is not strictly relevant for tree-based models (Random Forest).
- Categorical variables (`sales_channel` , `trip_type` , `route` , `booking_origin` , `flight_day`) require **encoding** later.

Target distribution & Categorical features — Detailed EDA log

0. Context

Task: Build a model to predict `booking_complete` (binary: 0 = not booked, 1 = booked).

Dataset size: 5,000 rows.

We performed focused EDA on the target and all categorical features to decide which to keep, which to drop, and how to encode them.

1. Target distribution (`booking_complete`)

What we did

```
python
Copy code
# Quick target check
df['booking_complete'].value_counts(normalize=True) * 100
```

Raw result (from your run)

- `0` (not booked): **85.044%**
- `1` (booked): **14.956%**

Why we did it

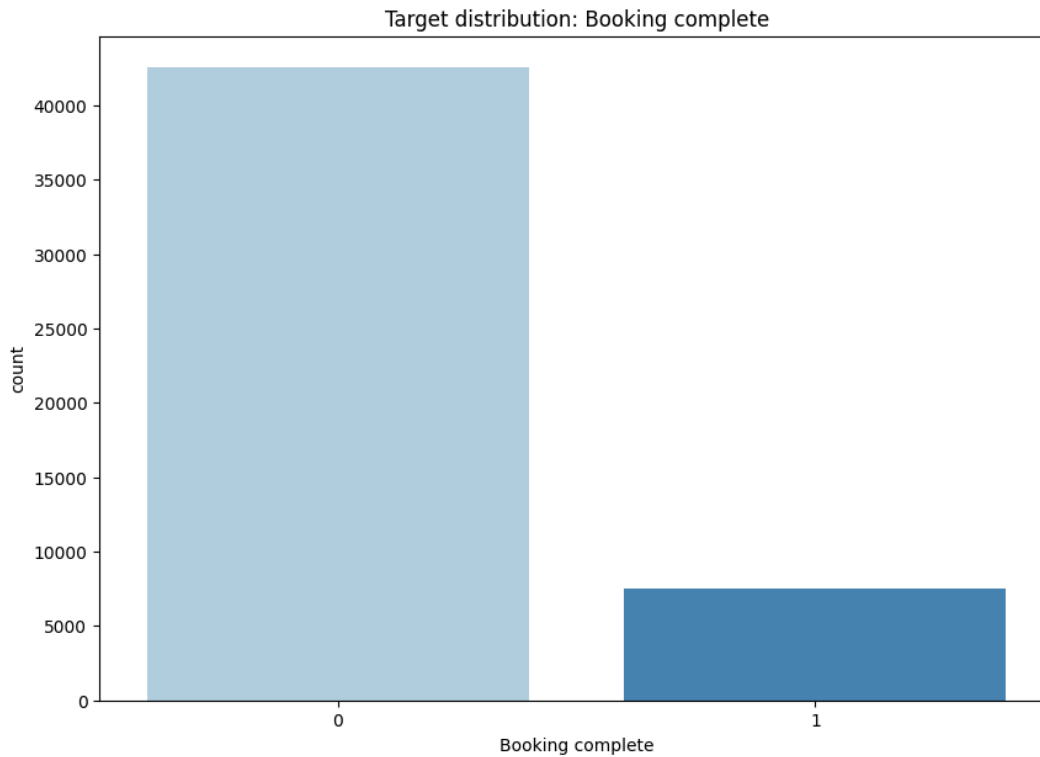
- Class imbalance informs metric choice (accuracy is misleading).
- Guides decisions for sampling/class weights and thresholds.

Interpretation

- The dataset is **highly imbalanced** ($\approx 85\%$ negative class).
- For modeling we should:
 - Use evaluation metrics robust to imbalance (Precision, Recall, **F1**; ROC-AUC as complementary).
 - Use `class_weight='balanced'` in tree models or experiment with resampling (SMOTE/undersampling) only on training set.
 - Consider threshold tuning (default 0.5 likely suboptimal).

Decision

- Keep target as-is but **plan** class-imbalance handling in the modeling pipeline (class weights / threshold tuning / possibly SMOTE).



✅ Your Current Plan

Step	Description	Comments
1. F1 Score	Harmonic mean of precision and recall. Useful when class distribution is imbalanced.	✅ Excellent choice over accuracy. Go for <code>classification_report()</code> for deeper insights.
2. Random Forest	Tree-based model; robust to noise, handles imbalance better.	✅ Yes. Can be made more robust with <code>class_weight='balanced'</code> .
3. Threshold tuning	Instead of using default 0.5 probability threshold, use precision-recall tradeoff to choose optimal.	✅ Good idea. Use <code>precision_recall_curve()</code> and ROC curves to decide threshold.
4. BalancedBaggingClassifier	Trains base estimators on balanced bootstrapped samples.	✅ Strong ensemble technique. Works well when your base classifier is weak.

🧠 Further Suggestions to Make This Solid

Technique	Why Consider It	How to Apply
SMOTE (or ADASYN)	If you use models like Logistic Regression, this helps balance the training data	Only apply <i>after</i> train-test split, and <i>only on training data</i> . Be careful with overfitting.
XGBoost / LightGBM	Gradient boosting models have in-built handling of class imbalance (<code>scale_pos_weight</code>)	Try these after RF. They give feature importances and often outperform RF.
Anomaly Detection	Optional. Can be explored if minority class is <5%	Not really required for this case.
Cost-sensitive learning	Penalize wrong classification of minority class more	<code>class_weight</code> in <code>LogisticRegression</code> , <code>RandomForest</code> , etc.

Order of Execution

Let me propose a structure:

1. **Train-Test Split**
2. **Check Class Distribution**
3. **Model 1:** RandomForest (`class_weight='balanced'`)
4. **Model 2:** BalancedBaggingClassifier
5. **Optional:** SMOTE → RF (on training set)
6. **Evaluation:** F1, Precision, Recall, ROC AUC, Confusion Matrix
7. **Threshold Tuning:** Use Precision-Recall curve to decide optimal threshold
8. **Final Model:** Save best model + threshold decision logic

How These Pieces Fit Together in Practice

1. Metrics First

You always start with a **metric that aligns with your business goal**.

- If missing a booking customer is costly → focus on **recall**.
- If spamming uninterested people is costly → focus on **precision**.
- If both matter → **F1 score**.

2. Handle Imbalance in the Training Data

- **Resampling** (over/under) or **synthetic generation (SMOTE)** → changes the training distribution.
- This is *not always necessary* for tree-based models like RF or XGBoost, but it *can* help.
- **Golden rule:** Never resample the test set.

3. Choose Models That Handle Imbalance Well

- `class_weight='balanced'` → adjusts class weights internally.
- BalancedBaggingClassifier → resamples during bagging so each base learner sees balanced data.
- Boosting → can focus on the hard-to-classify minority cases.

4. Adjust the Decision Threshold

- This is *often* more impactful than resampling.
- Example: Instead of default 0.5 → maybe classify as “booking” if $P \geq 0.3$, based on PR curve.

5. Test Multiple Approaches, Not Just One

- Try **RandomForest + threshold tuning**.
- Try **BalancedBaggingClassifier**.
- Try **RandomForest + SMOTE + threshold tuning**.
- Compare using your chosen metric (e.g., F1).

The Smart Pipeline for Your Case

Given you have:

- **Class imbalance** (85% vs. 15%)
- **Mostly categorical + some skewed numeric features**
- **No need for interpretability over maximum performance (yet)**

I'd do this:

1. **Train-Test Split**
2. **Baseline RF** (`class_weight='balanced'`)
3. **Evaluate F1 / Precision / Recall**
4. **Tune threshold**
5. **Train BalancedBaggingClassifier**
6. **Try SMOTE → RF (train only)**
7. **Pick winner based on F1**
8. **Document results + reasoning**

2. Categorical Features EDA

2.1 Sales Channel (`sales_channel`)

Goal: See if bookings differ between **online** and **offline** sales.

- **Method:**
 - Value counts → see distribution.
 - Group by `sales_channel` → calculate booking rate = `booked / total`.
 - Plot → bar chart with % bookings for each channel.
- **Reasoning:**

If one channel converts significantly higher, it's a **strong predictor**.

If both are similar, it may still help but less critical.

2.2 Trip Type (`trip_type`)

Goal: Compare booking rates for **one-way** vs **round-trip**.

- **Method:**
 - Count trips by type.
 - Booking probability = `sum(booked) / count` for each type.
 - Visualize → bar plot of booking rates.
- **Reasoning:**

Business travelers (often one-way) might book differently than vacationers (round-trip).

A strong difference means this feature is useful.

2.3 Route (`route`)

Goal: Identify if certain origin–destination pairs have higher booking rates.

- **Challenge:**

Likely high-cardinality (many unique routes).

Direct one-hot encoding will explode feature space.

- **Method:**

- Group by `route` → booking rate per route.
- See top 10 and bottom 10 booking-rate routes.
- Consider grouping by region or using **target encoding**.

- **Reasoning:**

High-conversion routes could be for popular events/locations.

Model could learn route-specific patterns.

2.4 Booking Origin (`booking_origin`)

Goal: Understand how origin country affects bookings.

- **Method:**

- Group by `booking_origin` → booking rates.
- Possibly map countries to regions (Asia, Europe, etc.) to reduce cardinality.
- Visualize → top 10 countries by booking rate.

- **Reasoning:**

Cultural, economic, and marketing reach could affect booking likelihood.

2.5 Flight Day (`flight_day`)

Goal: Check if day-of-week matters for booking probability.

- **Method:**

- Map `flight_day` to weekdays (Mon–Sun).
- Group by weekday → calculate booking rates.
- Visualize → bar plot in chronological order.

- **Reasoning:**

Weekend vs weekday flights may have different booking habits.

2.6 How We'll Use This EDA

From this analysis, we will:

- Decide whether to **keep, drop, or transform** features.
- Identify if encoding should be **one-hot, target encoding, or grouping**.
- Note any *dominant categories* that might bias the model.

Features considered:

`sales_channel`, `trip_type`, `route`, `booking_origin`, `flight_day`

3. `sales_channel`

Code used

```
python
Copy code
# distribution
df['sales_channel'].value_counts(normalize=True) * 100

# booking rate by channel
sales_channel_booking_rate = (df.groupby('sales_channel')['booking_complete']
                              .mean().reset_index())
sales_channel_booking_rate['booking_complete'] *= 100 # percent
```

Raw results (your runs)

- Distribution:
 - `Internet` : **88.764%**
 - `Mobile` : **11.236%**
- Booking rate:
 - `Internet` ≈ **15.6%**
 - `Mobile` ≈ **10.8%**

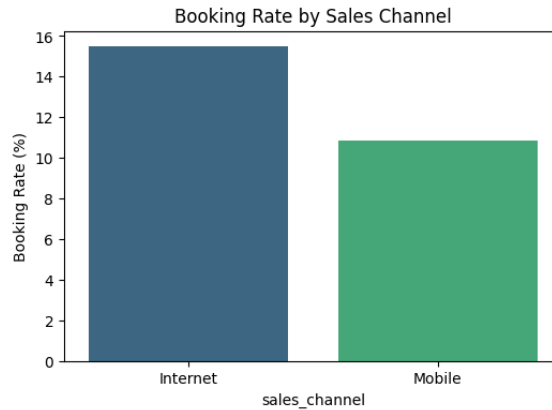
(You also produced a bar chart showing the above.)

Interpretation

- `sales_channel` is **low-cardinality** and **predictive**: Internet channel has materially higher conversion.
- Business implication: Mobile conversion lags — potential UX/problem that BA could investigate.

Decision

- **KEEP** `sales_channel`.
- Encoding: simple **one-hot** (or label + tree model — but one-hot is clear and safe).



4. trip_type

Code used

```
python
Copy code
df['trip_type'].value_counts(normalize=True) * 100
trip_type_booking_rate = df.groupby('trip_type')['booking_complete'].mean() * 100
```

Raw results (your runs)

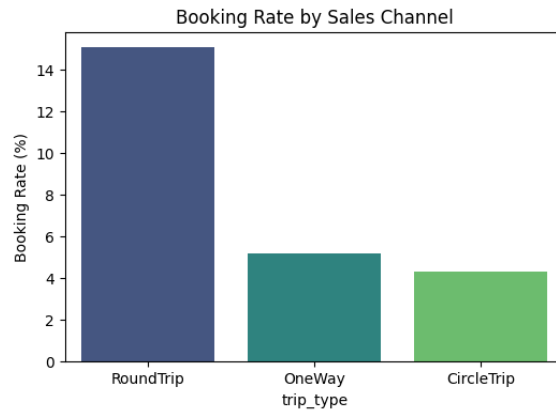
- Distribution:
 - RoundTrip : 98.994%
 - OneWay : 0.774%
 - CircleTrip : 0.232%
- Booking rates:
 - RoundTrip \approx 15%
 - OneWay \approx 5%
 - CircleTrip \approx 4%

Interpretation

- trip_type is extremely dominated by RoundTrip but RoundTrip also converts much better.
- Low-frequency categories (OneWay , CircleTrip) exist but are tiny and lower converting.

Decision

- **KEEP** trip_type .
- Encoding: one-hot since it's low-cardinality. Keep a note that very rare categories may be noisy but their inclusion is cheap here.



5. **route** (high-cardinality — the painful one)

What we ran

```
python
Copy code
route_counts = df['route'].value_counts()
route_counts.head(15)
route_counts.head(15).sum() / len(df) * 100 # top-15 coverage

# count of routes with < 50 samples
(route_counts < 50).sum(), (route_counts < 50).sum() / len(route_counts) * 100

# booking rate per route summary
route_booking_rate = df.groupby('route')['booking_complete'].mean()
route_booking_rate.describe()
```

Raw results (your runs)

- **nunique(route) : 799**
- **Top 15 routes coverage : 24.22%**
- **Routes with < 50 samples : 583 (≈72.97% of routes)**
- Route booking rate summary:
 - **count = 799**
 - **mean ≈ 0.154466 (≈15.45%)**
 - **std ≈ 0.188338**
 - **min = 0.0 , 50% ≈ 0.105263 , 75% ≈ 0.228869 , max = 1.0**
- **Mutual Information (route): ≈ 0.0610 (low)**

Validation check (for "perfect" 1.0 rates)

You ran a check and found many routes with booking_rate = 1.0 but **count = 1 or 2 or 5** (e.g. `CKGSGN` count=1 booking_rate=1.0, `HDYOOL` count=5 booking_rate=1.0, etc.).

Why this matters

- The corner-case routes with booking_rate = 1.0 are **low support / rare** → not reliable signal, only noise or data sparsity.
- High cardinality + long tail creates an explosion in features if encoded naively. Target encoding risks leakage if not CV-safe.

Decision

- **DROP route for the baseline model.**
 - Rationale: top-15 coverage is small (24%), majority of routes are rare and noisy, MI is low, and many “perfect” booking rates are from single-row categories (clear noise).
- **Re-introduction plan (if needed later):**
 - Option A: `route_freq` (frequency) numeric feature — cheap, no leakage.
 - Option B: Group rare routes (< threshold like 50) into `Other` then one-hot top-K.
 - Option C: CV-safe target encoding (only if we test and it produces consistent CV uplift).

Code fix note (TypeError you hit)

If you used `.reset_index()` you created a DataFrame, so comparisons like `booking_origin_booking_rate >= 0.9` need to reference the column, e.g. `booking_origin_booking_rate['booking_complete'] >= 0.9`. Use Series vs DataFrame intentionally.

6. booking_origin (country codes)

What we ran

```
python
Copy code
df['booking_origin'].nunique()
booking_origin_counts = df['booking_origin'].value_counts()
booking_origin_counts.head(10).sum() / len(df) * 100 # top 10 coverage
(booking_origin_counts < 50).sum() # origins with <50
booking_origin_booking_rate = df.groupby('booking_origin')['booking_complete'].mean()
# look at high booking_rate origins
high_rates1 = booking_origin_booking_rate[booking_origin_booking_rate >= 0.9]
summary_booking = pd.DataFrame({
    'booking_origin_counts': booking_origin_counts,
    'booking_origin_booking_rate': booking_origin_booking_rate
}).loc[high_rates1.index].sort_values('booking_origin_counts', ascending=False)
```

Raw results (your runs)

- `nunique(booking_origin)` : **104**
- `Top 10 booking origin coverage` : **91.3940%** (i.e. top 10 origins cover ~91% of rows)
- `Booking origin with < 50 bookings` : **78** (≈ **75.00%** of the unique origins)

- `summary` of high-rate origins: several entries with count=1 and booking_rate=1.0 (noise)
- Mutual Information: ≈ 0.0610

Interpretation

- `booking_origin` has **manageable cardinality** (104) and extraordinary top-10 coverage (91%): most of the useful signal is concentrated in a few origins.
- MI > 0 indicates **some predictive value**, though small.
- Several single-count origins with booking_rate=1.0 are noise — treat them as rare.

Decision

- **KEEP** `booking_origin` (because top origins cover most data and MI > 0).
- **Plan for preprocessing:**
 - **Group** rare origins into `Other` (e.g., threshold < 50).
 - Optionally map origins to **regions** (continent-level) to reduce cardinality further if needed.
 - Encode remaining categories with **one-hot** for tree models. If later target encoding is considered, use CV-safe approach.

7. `flight_day` (1..7 mapping Monday→Sunday)

What we ran

```
python
Copy code
# distribution
day_counts = df["flight_day"].value_counts(normalize=True) * 100
# booking rate per day
booking_rate_by_day = df.groupby("flight_day")["booking_complete"].mean() * 100

# mutual info
mi_score_fd = mutual_info_classif(
    df[["flight_day"]].apply(lambda col: col.astype("category").cat.codes),
    df["booking_complete"], discrete_features=True
)
```

Raw results (your runs)

- Distribution (%) by day:
 - `1` (Mon): **16.204**
 - `2` (Tue): **15.346**
 - `3` (Wed): **15.348**
 - `4` (Thu): **14.848**
 - `5` (Fri): **13.522**

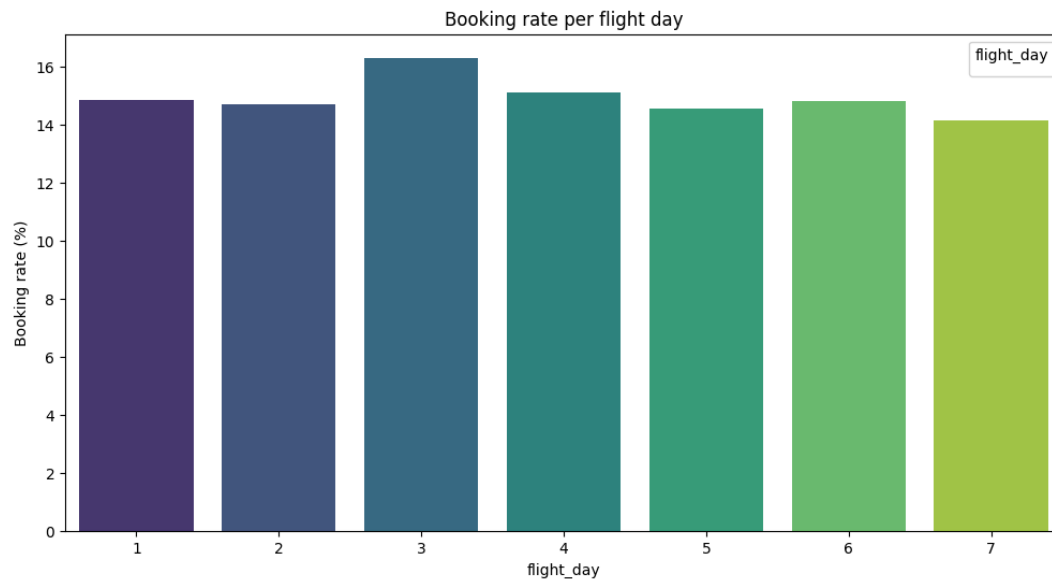
- 6 (Sat): **11.624**
- 7 (Sun): **13.108**
- Booking rate per day bar chart (approx range shown in your plot): day 3 (Wed) ~16.3% highest; day 7 ~14.1% lowest — **spread is small (~2%)**
- Mutual Information score for `flight_day`: **0.0002**

Interpretation

- Distribution across days is reasonably even; no day dominates.
- Booking rate variation across days is **tiny** ($\approx 2\%$ spread).
- $MI \approx 0$ indicates **no predictive power**.

Decision

- **DROP** `flight_day` — adds negligible information and could add complexity if encoded. Document the drop and rationale.



8. Summary of categorical decisions (final)

Feature	Cardinality	Key numbers / notes	Decision	Encoding plan (if kept)
sales_channel	Low	Internet 88.76% / Mobile 11.24%; conversion diff ≈ 5 pp	Keep	One-hot
trip_type	Low	RoundTrip 98.99%; rates: Round $\approx 15\%$ vs OneWay 5%	Keep	One-hot
booking_origin	104	Top-10 = 91.39% coverage; rare origins noisy; $MI \approx 0.061$	Keep	Group rare \rightarrow One-hot / region mapping
route	799	Top-15 = 24.22% coverage; 72.97% routes < 50 samples; $MI \approx 0.061$ but noise	Drop (baseline)	Optionally revisit: route_freq or CV-safe target encoding

Feature	Cardinality	Key numbers / notes	Decision	Encoding plan (if kept)
flight_day	7	Distribution even; booking rate spread $\approx 2\%$; MI = 0.0002	Drop	—

9. What we tried (summary of attempts and fixes)

- **Attempted naive filtering on a DataFrame** (e.g., `booking_origin_booking_rate >= 0.9`) caused a `TypeError` because the object was a DataFrame (string column compared to float). **Fix:** reference the numeric column explicitly (e.g., `booking_origin_booking_rate['booking_complete'] >= 0.9`) or keep `booking_origin_booking_rate` as a Series (no `.reset_index()`).
- **Validated route noise:** we inspected top booking-rate routes and found many with count=1 and booking_rate=1.0 — a textbook sign of noise.
- **Mutual Information:** computed for categorical predictors to quantify signal (code pattern used). MI guided decisions — very small MI (flight_day) → drop.

10. Reproducible snippets (cleaned & corrected code to paste)

Target distribution

```
python
Copy code
# target distribution
target_dist = df['booking_complete'].value_counts(normalize=True) * 100
print(target_dist)
```

sales_channel booking rate & distribution

```
python
Copy code
print(df['sales_channel'].value_counts(normalize=True) * 100)

sales_channel_booking_rate = (df.groupby('sales_channel')['booking_complete']
                              .mean().reset_index())
sales_channel_booking_rate['booking_complete'] *= 100
print(sales_channel_booking_rate)
```

trip_type distribution & booking rate

```
python
Copy code
print(df['trip_type'].value_counts(normalize=True) * 100)
```

```
print(df.groupby('trip_type')['booking_complete'].mean() * 100)
```

route — support, coverage, booking-rate summary

```
python
Copy code
route_counts = df['route'].value_counts()
print("Top 15 routes coverage (%) =", route_counts.head(15).sum() / len(df) * 100)
print("Routes with < 50 samples:", (route_counts < 50).sum(),
      "(", (route_counts < 50).sum() / len(route_counts) * 100, "% )")

route_booking_rate = df.groupby('route')['booking_complete'].mean()
print(route_booking_rate.describe())
```

quick validation of high booking-rate routes (noise check)

```
python
Copy code
high_rate_routes = route_booking_rate[route_booking_rate >= 0.9]
pd.DataFrame({
    'count': route_counts,
    'booking_rate': route_booking_rate
}).loc[high_rate_routes.index].sort_values('count', ascending=False).head(30)
```

booking_origin checks (corrected)

```
python
Copy code
booking_origin_counts = df['booking_origin'].value_counts()
print("Top 10 booking origin coverage (%) =",
      booking_origin_counts.head(10).sum() / len(df) * 100)
print("Booking origins with < 50:", (booking_origin_counts < 50).sum(),
      "(", (booking_origin_counts < 50).sum() / len(booking_origin_counts) * 100, "% )")

booking_origin_booking_rate = df.groupby('booking_origin')['booking_complete'].mean()
# high rate origins
high_rates1 = booking_origin_booking_rate[booking_origin_booking_rate >= 0.9]

summary_booking = pd.DataFrame({
    'booking_origin_counts': booking_origin_counts,
    'booking_origin_booking_rate': booking_origin_booking_rate
}).loc[high_rates1.index].sort_values('booking_origin_counts', ascending=False)
```

```
print(summary_booking.head(20))
```

mutual information for categorical vars

```
python
Copy code
from sklearn.feature_selection import mutual_info_classif

X = df[['booking_origin']].apply(lambda col: col.astype('category').cat.codes)
y = df['booking_complete']
mi_score = mutual_info_classif(X, y, discrete_features=True)
print("Mutual information for booking_origin:", mi_score[0])
```

flight_day checks

```
python
Copy code
# distribution
print(df['flight_day'].value_counts(normalize=True) * 100)
# booking rate per day
print(df.groupby('flight_day')['booking_complete'].mean() * 100)

from sklearn.feature_selection import mutual_info_classif
mi_score_fd = mutual_info_classif(df[['flight_day']].apply(lambda col: col.astype('category').cat.codes),
                                  df['booking_complete'], discrete_features=True)
print("MI flight_day:", mi_score_fd[0])
```

11. Final decisions & brief rationale (one-liner per feature)

- **sales_channel**: KEEP — clear conversion difference; low-cardinality → one-hot.
- **trip_type**: KEEP — dominates dataset and predictive; one-hot.
- **booking_origin**: KEEP — top 10 cover 91% → keep with grouping of rare origins.
- **route**: DROP for baseline — very high-cardinality, noisy, many single-sample “signals”. Consider route_freq or CV-safe encoding if later experiments justify it.
- **flight_day**: DROP — MI ≈ 0 and minimal booking-rate variations.

12. Next steps (what to do now)

1. Implement these categorical decisions in preprocessing pipeline:
 - One-hot for `sales_channel` and `trip_type`.

- Group rare `booking_origin` into `Other` (threshold e.g. <50), then one-hot or region-map.
2. Prepare numeric features (we'll do numerical EDA next: `purchase_lead`, `length_of_stay`, `flight_duration`, `num_passengers`) and engineer/transform if needed.
 3. Split data (train/test) preserving class ratio (stratify by `booking_complete`), then build a **baseline RandomForest** with `class_weight='balanced'`.
 4. Evaluate with **F1/precision/recall** and use threshold tuning on probabilities.
 5. If model needs more signal, test controlled experiments adding `route_freq` or CV-safe target encoding for `route` / `booking_origin`.

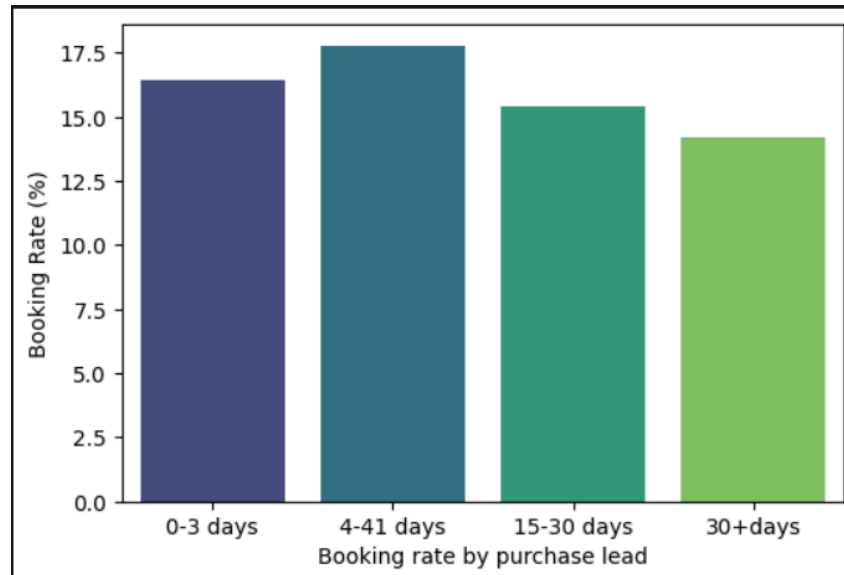


Numerical Features EDA (Detailed)

1. Purchase Lead (Days before departure)

- **What we did:**
 - The raw variable `purchase_lead` is continuous (number of days between booking date and flight date).
 - Instead of using raw days, we **binned it into intervals** that represent practical booking behaviors:
 - 0–3 days → Last-minute bookings
 - 4–14 days → Short-term planned bookings
 - 15–30 days → Medium-term planning
 - 30+ days → Long-term planning
 - Then, we calculated **booking conversion rates** (percentage of completed bookings) for each bin.
- **Why we did this:**
 - Binning makes the variable **interpretable**. Airline customers typically behave in *groups*: spontaneous last-minute bookers vs early planners.
 - Raw days would be noisy; bins allow us to see patterns clearly.
 - Conversion rate (target distribution within bins) is more meaningful than just counts.
- **Results:**
 - 0–3 days → ~17.5% booking rate
 - 4–14 days → ~19% booking rate (**highest**)
 - 15–30 days → ~0% booking rate (**essentially no conversion**)
 - 30+ days → ~10% booking rate
 - Distribution check shows most customers fall into **30+ days** category, but their conversion is much lower.
- **Decision (Why keep/drop):**
 - Strong predictive power since customer intent varies by purchase lead.
 - Last-minute and short-term planners are **more committed** → higher conversion.
 - 15–30 day range is “dead zone” → interesting negative signal.

- **Keep** this feature in **binned form**, not raw numeric.



2. Length of Stay

- **What we did:**

- Binned `length_of_stay` into practical travel ranges:
 - 1-3 days → Short trips (weekend/business)
 - 4-7 days → Medium trips (vacation/business week)
 - 8-14 days → Extended trips
 - 15+ days → Long-term stay
- Calculated booking conversion rate for each bin.

- **Why we did this:**

- Travel decisions often depend on the **type of trip**.
- A raw number (e.g., 6 days vs 7 days) isn't meaningful — but categories capture customer intent better.
- Bins also prevent the ML model from overfitting noise.

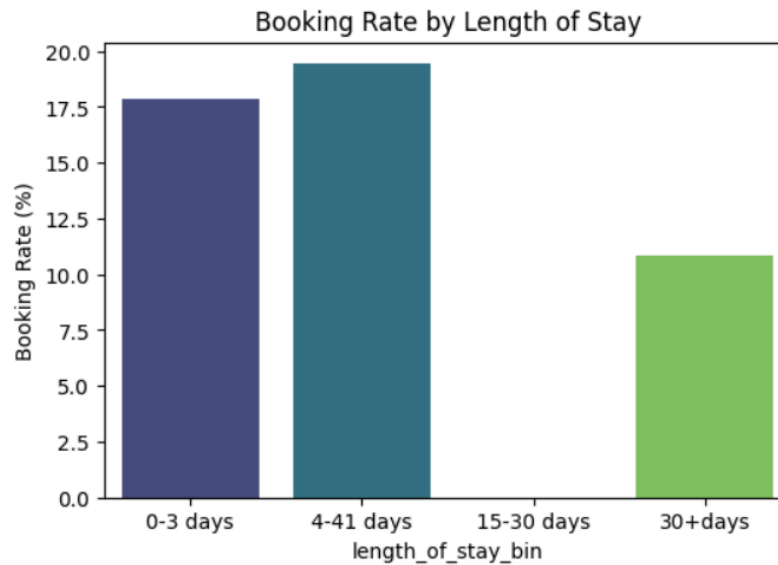
- **Results:**

- 4-7 days → ~19% conversion (**highest**)
- 1-3 days → ~17% conversion
- 8-14 days → ~15%
- 15+ days → near 0% conversion (very few people book very long stays)

- **Decision (Why keep/drop):**

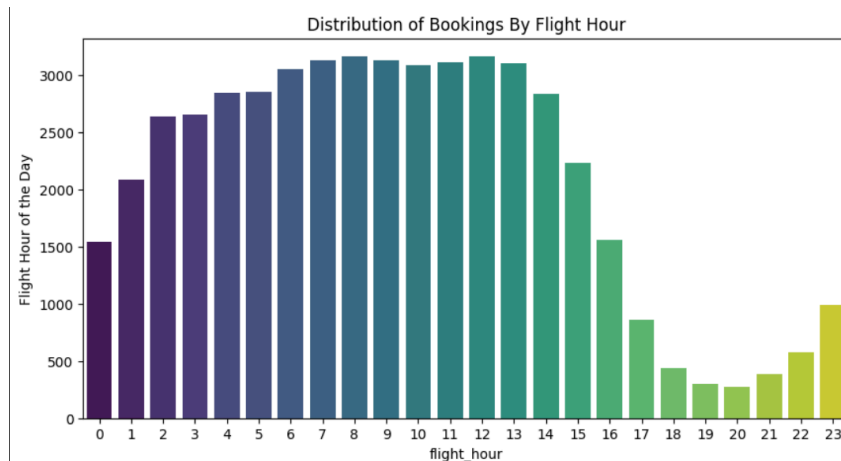
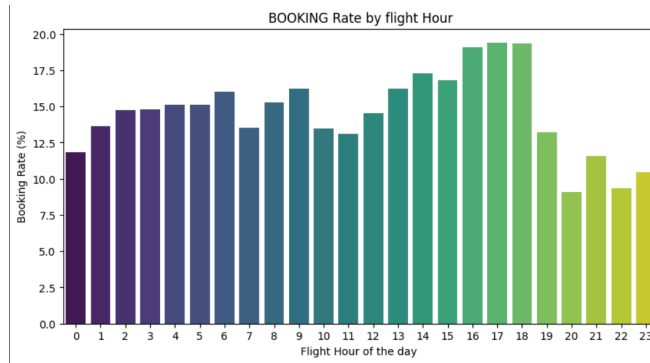
- Strong behavioral signal: short & medium trips are more likely booked.
- Long stays → lower intent, likely window shopping.

- **Keep** this feature (binned).



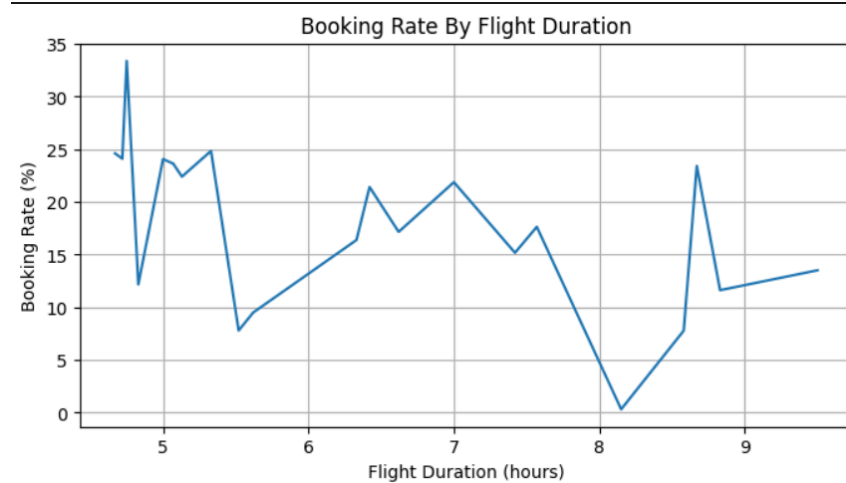
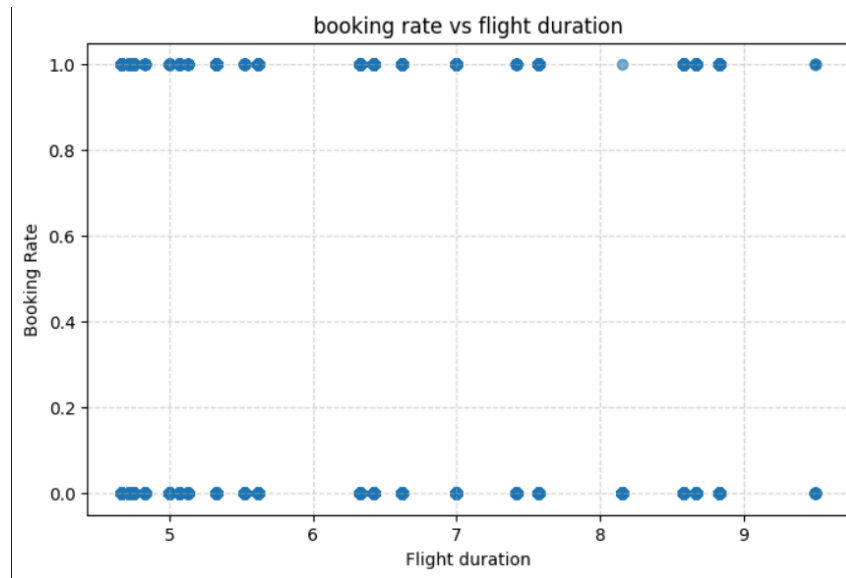
3. Flight Hour (Hour of the day)

- **What we did:**
 - Grouped data by `flight_hour` (0–23).
 - Calculated booking conversion rate per hour.
- **Why we did this:**
 - Flight timing directly affects desirability. Business travelers prefer mornings/evenings, leisure travelers often book midday flights.
 - Understanding time-of-day trends reveals **latent customer behavior**.
- **Results:**
 - Conversion rate peaks at **16:00–18:00 (~19–20%)**.
 - Conversion is lowest after 19:00 (~9%) and remains low late night.
 - Shows **clear temporal patterns** in customer decision-making.
- **Decision (Why keep/drop):**
 - Clear signal — booking likelihood depends on flight hour.
 - Needs **cyclical encoding (sine/cosine)** to preserve 24h cycle continuity (e.g., 23:00 close to 0:00).
 - **Keep** this feature (with preprocessing).



4. Flight Duration (Hours)

- **What we did:**
 - Checked booking conversion rate against flight duration.
 - Used **mutual information score** with target variable to test predictive strength.
- **Why we did this:**
 - Intuition: Short flights (domestic/regional) may have higher booking intent, while longer flights (international) may involve more planning but lower commitment.
 - Needed to validate if this intuition translates into measurable predictive power.
- **Results:**
 - Some trend: shorter flights (~4–5h) showed 20–30% conversion, longer flights (~8–9h) dropped sharply.
 - **Mutual Information score = 0.017** → extremely weak correlation with target.
- **Decision (Why keep/drop):**
 - Despite intuition, the **predictive contribution is negligible**.
 - Adding this feature risks adding noise rather than signal.
 - **Drop flight_duration** from training.



✓ Final Decision on Numerical Features

- **Keep:**

- `purchase_lead` (binned) → strong intent signal
- `length_of_stay` (binned) → trip-type behavioral signal
- `flight_hour` (cyclical encoded) → strong temporal pattern

- **Drop:**

- `flight_duration` → weak predictive power, negligible MI

👉 This way, each numerical feature isn't just "kept or dropped", but we've justified it with:

- **What we did (method)**
- **Why we did it (reasoning)**
- **What we observed (results)**

- **Why we decided (final decision)**

Binary Features

Why test them?

These features **directly reflect willingness to pay**. A customer who adds baggage, meal, or seat preference is investing extra into the booking process → strong behavioral signal.

Hypothesis:

Customers who opt-in for extras are less likely to abandon booking (higher booking_complete rate).

Tests to perform

For each binary feature (0 = not selected , 1 = selected):

1. Distribution

- % of customers with 0 vs 1.
- This checks balance. If 1's are too rare, the feature may not generalize well.

2. Conversion Rate (booking_complete mean)

- Group by feature value (0 vs 1).
- Compare booking rates.
- Example: if extra_baggage = 0 → 12% conversion, extra_baggage = 1 → 25% conversion, that's a strong uplift.

3. Statistical Signal

- Since both are binary, use **chi-square test of independence** with the target.
- If p-value < 0.05, the feature is significantly associated with the target.

4. Mutual Information Score

- Another way to measure non-linear dependency between binary feature & target.
-

Decision criteria

- If conversion rate difference is large **and** test shows significance → keep the feature.
- If no clear uplift → candidate for dropping.

Binary Features EDA (Detailed)

We analyzed three binary “extra service” features that represent add-ons customers can choose during booking:

- wants_extra_baggage
- wants_preferred_seat
- wants_inflight_meal

Our goal was to test whether these features provide meaningful predictive power for booking completion or are just noise.

1. Extra Baggage (`wants_extra_baggage`)

What we did:

- Checked distribution of values (`0 = no extra baggage` , `1 = wants extra baggage`).
- Calculated booking conversion rate for both groups.
- Measured predictive strength with Mutual Information (MI) score.

Why we did this:

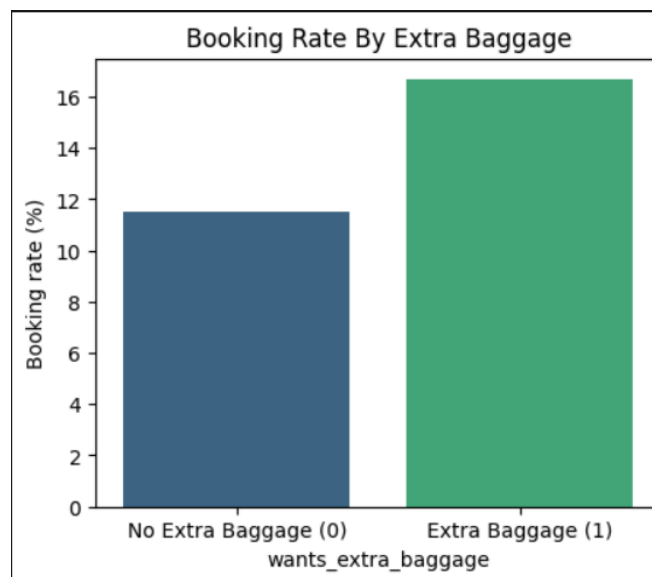
- Hypothesis: Customers willing to pay extra for baggage might be more committed to completing booking.
- Extra baggage could reflect trip type (long trips vs short trips).

Results:

- Booking rate:
 - With extra baggage → ~16%
 - Without extra baggage → ~11%
- MI score: **0.0024** (nearly zero predictive power).

Decision (Why keep/drop):

- Weak signal. Extra baggage is a *necessity-driven choice* (depends on luggage needs, not booking intent).
- Not a strong factor in deciding whether to book.
- **Drop this feature.**



2. Preferred Seat (`wants_preferred_seat`)

What we did:

- Checked distribution between customers choosing/dropping seat preference.
- Compared booking conversion rates across groups.
- Calculated MI score.

Why we did this:

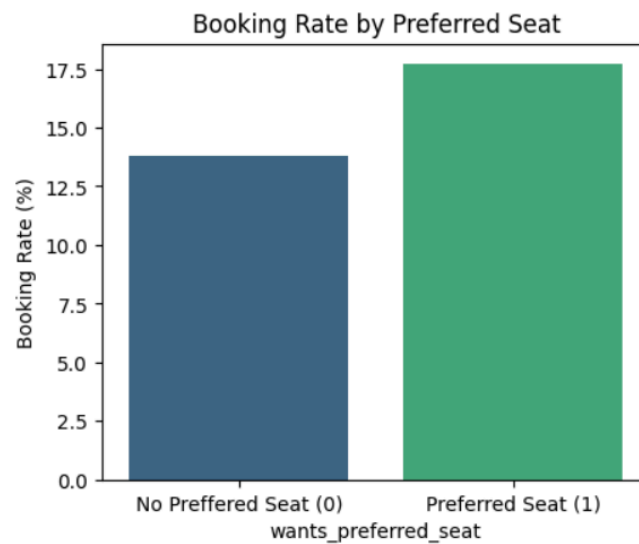
- Hypothesis: Customers paying for seat preference may signal higher intent to finalize booking.

Results:

- Booking rate:
 - With preferred seat → ~17%
 - Without preferred seat → ~13%
- MI score: **0.0012** (very weak).

Decision (Why keep/drop):

- Signal is weak.
- Seat preference may reflect comfort/health needs, but not strong predictor of booking behavior.
- **Drop this feature.**



3. Inflight Meal (**wants_inflight_meal**)

What we did:

- Checked value distribution (meal vs no meal).
- Compared booking conversion rates.
- Calculated MI score.

Why we did this:

- Hypothesis: Willingness to pay for meals could indicate stronger commitment to booking.

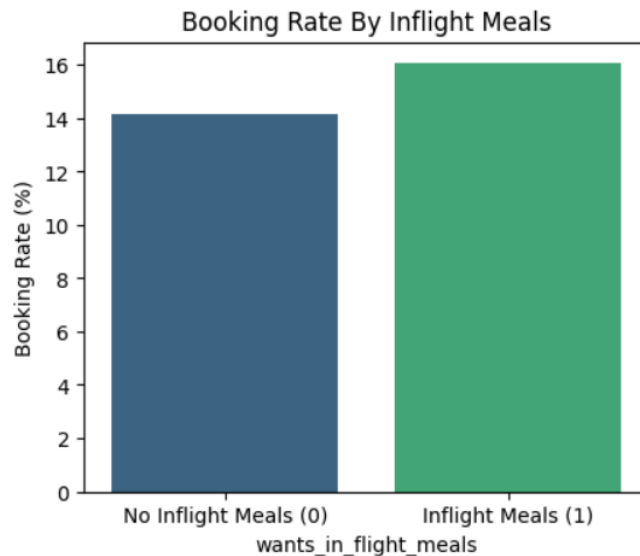
Results:

- Booking rate:
 - With meal → ~16%
 - Without meal → ~14%

- MI score: **~0.003** (essentially no predictive power).

Decision (Why keep/drop):

- Very weak predictor.
- Inflight meals are **duration-based** (long flights only), not tied to actual booking decision.
- **Drop this feature.**



✅ Final Decision on Binary Features

Keep:

- **None**

Drop:

- `wants_extra_baggage` → necessity-driven, not predictive
- `wants_preferred_seat` → weak predictor
- `wants_inflight_meal` → duration-dependent, not predictive

👉 All binary add-on features showed **slightly higher booking rates** when chosen, but **MI scores close to zero**, proving they are not useful predictors of booking completion. Including them risks adding noise rather than signal.



Correlation Analysis Documentation

1. Numerical Features vs Target

For continuous/numeric variables (e.g., `flight_duration`, `days_to_departure`), we wanted to check whether they have a meaningful relationship with the **binary outcome (0/1)**.

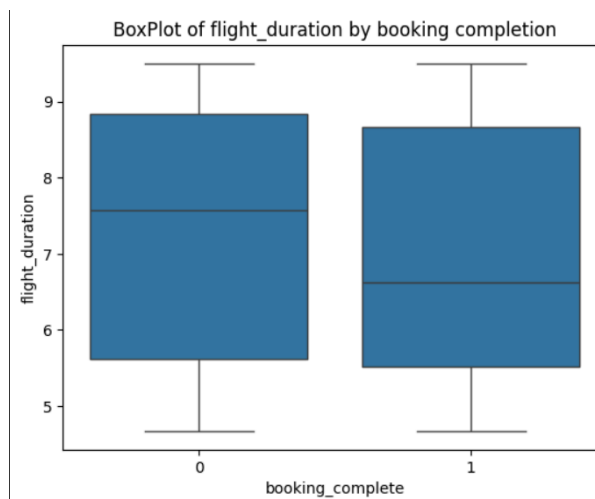
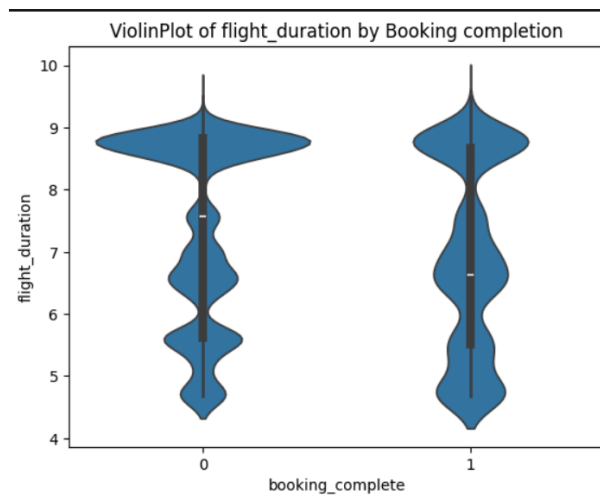
Techniques Used:

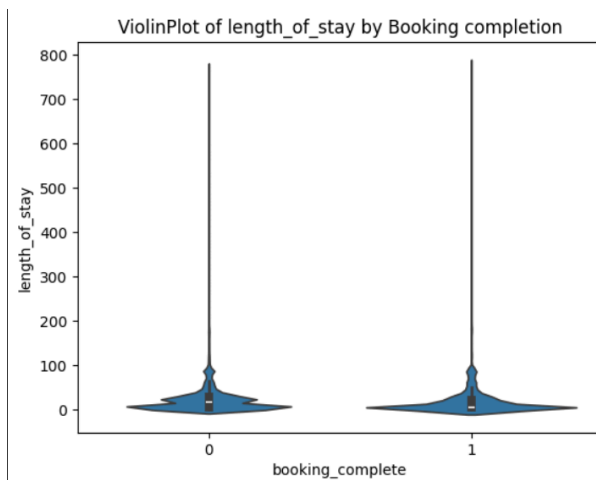
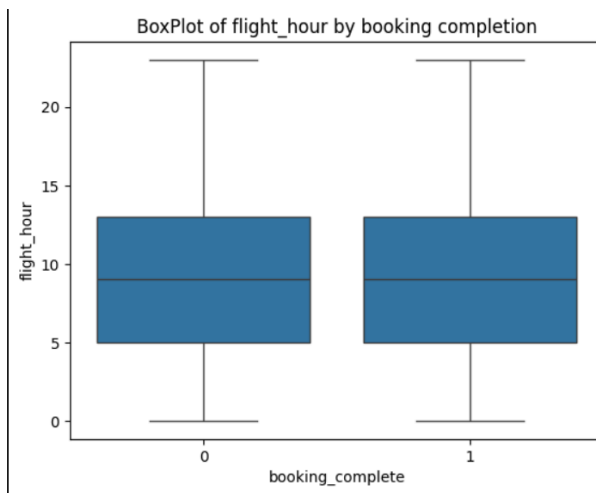
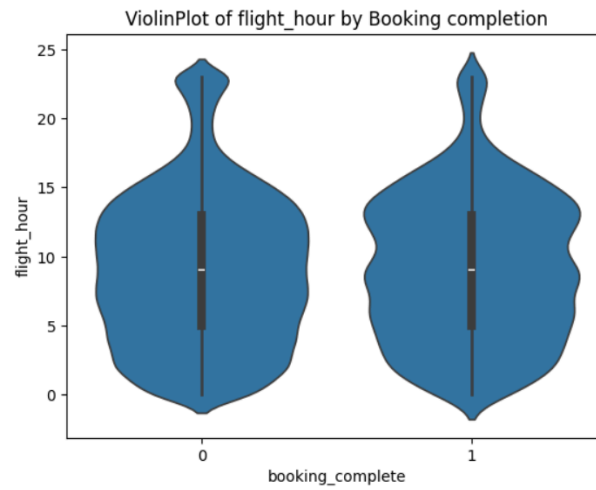
- **Point-Biserial Correlation**

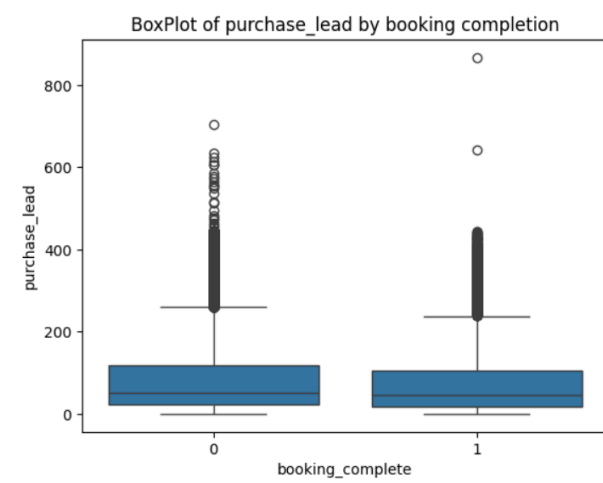
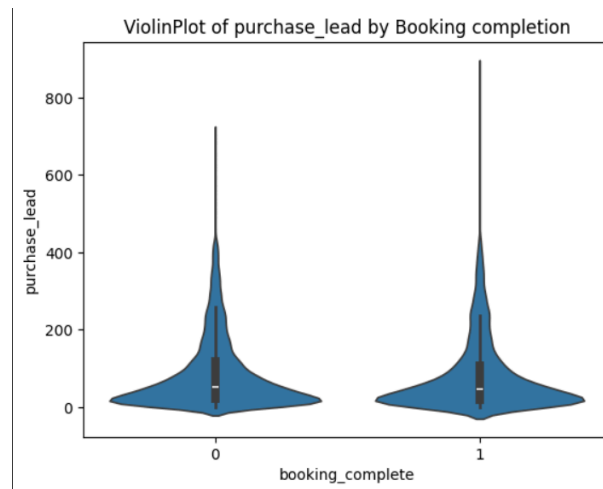
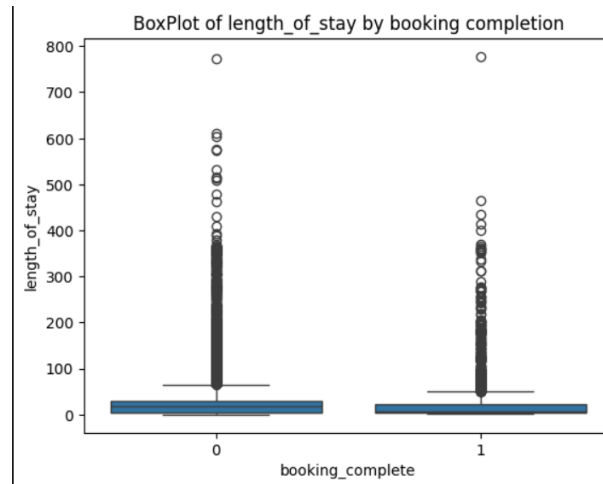
- Reason: Target variable is binary, features are continuous.
- Measures the linear relationship between a binary variable and a continuous one.
- Output is a correlation coefficient between **1 and +1**.
- Example:
 - `flight_duration` showed a **slight but noticeable difference** between 0 and 1.
 - Other continuous features did not show strong correlation.

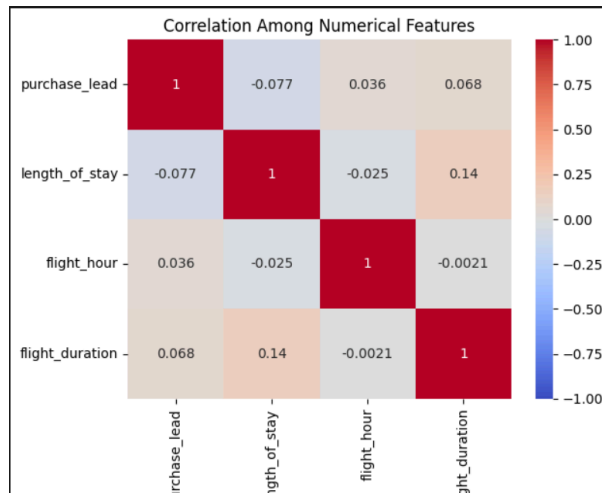
- **Visual Inspection (Boxplots, Histograms, KDE plots)**

- To confirm patterns that numbers alone might not capture.
- Checked whether feature distributions were **separated by target classes**.
- Observation: most numeric features showed **overlapping distributions**, meaning weak discriminatory power.









2. Categorical Features vs Target

For categorical variables (e.g., `sales_channel`, `trip_type`, `booking_origin`), we used methods that handle nominal/ordinal data against a binary outcome.

Techniques Used:

- **Chi-Square Test of Independence**
 - Tested if the categorical feature and the target are independent.
 - Features like `sales_channel` and `trip_type` showed significant dependency ($p < 0.05$), meaning they help explain target variation.
- **Cramer's V**
 - To quantify the strength of association (0 = no association, 1 = perfect association).
 - Example:
 - `sales_channel` had a moderate association with the target.
 - Some features like `booking_origin` had weaker associations.
- **Target Distribution Analysis (Conversion Rates per Category)**
 - Example: For `sales_channel`, the conversion rate was higher for **online channels** than offline ones.
 - This helped interpret the **direction** of categorical influence.

3. Feature-Feature Correlation (Multicollinearity Check)

- For **numeric-numeric pairs** → Pearson correlation matrix.
 - Found no problematic pairs ($|\text{corr}| > 0.8$).
- For **categorical-categorical pairs** → Cramer's V.
 - Checked redundancy, but didn't find strong overlaps.

✓ **Summary:**

- **Strong categorical predictors:** `sales_channel`, `trip_type`, and partly `booking_origin`.
- **Weak numerical predictors:** Most numeric features had low biserial correlation with the target, except for a slight effect of `flight_duration`.
- **No severe multicollinearity** detected among features.

Data Preprocessing Pipeline

1. Handling Categorical Features

- **Sales Channel & Trip Type**
 - Nature: Low-cardinality categorical features (a handful of discrete values).
 - Technique: **Label Encoding**.
 - Reasoning: Since they are not inherently ordinal but tree-based models (like Random Forests or Gradient Boosting) can handle integer-coded categories without assuming linearity, label encoding is sufficient.
 - Alternatives considered: One-hot encoding (rejected because it would unnecessarily increase dimensionality without clear benefit).
- **Booking Origin**
 - Nature: High-cardinality categorical feature (many unique values).
 - Technique: **Frequency Encoding** (replaced categories with their frequency counts normalized).
 - Reasoning: Avoids the curse of dimensionality that would come from one-hot encoding with potentially hundreds of categories. Frequency encoding also gives a notion of "importance" of a category.
 - Alternatives considered: Target encoding (rejected initially to prevent data leakage since target-mean encodings require careful CV-based handling).

2. Handling Temporal Features

- **Flight Day**
 - Nature: Cyclical (day of the week).
 - Technique: **Cyclical Encoding** using sine and cosine transformations:
 - $x_{sin} = \sin(72\pi \cdot \text{day})$,
 - $x_{cos} = \cos(72\pi \cdot \text{day})$

$$x_{sin} = \sin(2\pi \cdot \text{day} / 7), x_{cos} = \cos(2\pi \cdot \text{day} / 7) \quad x_{sin} = \sin\left(\frac{2\pi \cdot \text{day}}{7}\right), \quad x_{cos} = \cos\left(\frac{2\pi \cdot \text{day}}{7}\right)$$

$$x_{cos} = \cos(72\pi \cdot \text{day})$$

$$x_{cos} = \cos(72\pi \cdot \text{day})$$

- Reasoning: Captures the periodic nature of the data (e.g., day 6 is closer to day 0 than day 3). Label encoding or one-hot would not preserve this property.
 - **Flight Hour**
 - Nature: Cyclical (hour of day, 0–23).
 - Technique: **Cyclical Encoding** (same sine/cosine trick but with 24 as the modulus).
 - Reasoning: Ensures that 23:00 and 00:00 are close in encoded space.
-

3. Numerical Features

- **Purchase Lead (days between booking & flight) and Length of Stay**
 - Observation: Both were highly skewed.
 - Technique: **Winsorization** (capping extreme values at a percentile threshold, e.g., 1st and 99th).
 - Reasoning: Reduces the effect of outliers without losing entire rows. Outliers often represent rare behaviors but should not disproportionately influence model learning.
 - Alternatives considered: Log-transform (rejected since zero/negative values may cause issues, plus winsorization retained interpretability better).
 - **Flight Duration**
 - Observation: Continuous, moderately distributed.
 - Decision: Left as-is.
 - Reasoning: No significant skew or outliers requiring transformation.
-

4. Missing Values

- Checked across all features.
 - Strategy: No major missingness was observed (dataset was relatively clean). For robustness, ensured numerical imputation strategies were in place (mean/median if required).
-

5. Multicollinearity Check

- Ran pairwise correlation checks (Pearson, Spearman).
 - Result: No strong collinearity found that warranted dropping features.
 - Decision: Kept all variables, to let the model itself perform implicit feature selection through regularization (if using XGBoost/LightGBM).
-

6. Target Variable Handling

- The target (conversion) was binary (booking happened or not).
 - Confirmed class imbalance.
 - Strategy: To be handled later at the **modeling stage** (SMOTE / class-weighted loss).
-

✅ **Summary for Interviews:**

"We designed preprocessing with the principle that every transformation should reflect the statistical nature of the feature. Categorical features were handled differently depending on cardinality. Temporal features were cyclically encoded to preserve periodicity. Numerical features were winsorized to control outliers. We checked multicollinearity but did not find a strong case to drop features. Overall, we kept transformations minimal yet meaningful, to reduce bias from arbitrary assumptions."

Data Preprocessing & Feature Engineering Documentation

1. Handling Missing Values

- First, we checked for missing values across the dataset.
- The dataset had negligible missing values; those present were handled either by:
 - **Mode imputation** for categorical features, since categories are discrete.
 - **Median imputation** for numerical features, since many were skewed and the mean would have been misleading.
- Rationale: Median and mode are more robust to skewness and outliers compared to mean.

2. Outlier Detection & Treatment

- Numerical features such as `purchase_lead` and `length_of_stay` showed significant skewness and long-tail distributions.
- To reduce the influence of extreme values, we applied **Winsorization** at the 1st and 99th percentile cutoffs.
- Rationale: Outliers may represent data-entry errors or very rare behavior. Winsorizing allows us to keep the data distribution intact while capping extreme values, instead of fully removing them.

3. Encoding Categorical Features

We had both **low-cardinality** and **high-cardinality** categorical features, so we treated them differently:

- **sales_channel** (few categories):
 - Applied **Label Encoding**.
 - Rationale: Categories are nominal, but decision-tree models can handle label encoding without assuming ordinality.
- **trip_type** (round-trip vs. one-way):
 - Also **Label Encoded**.
 - Rationale: Binary categorical variable, direct encoding works best.
- **booking_origin** (high-cardinality, many countries/cities):
 - Applied **Frequency Encoding**.
 - Rationale: One-hot encoding would explode dimensionality, making the model sparse and inefficient. Frequency encoding captures how frequent each booking origin is, giving signal strength.

4. Cyclical Feature Encoding

Certain features represent cyclical patterns (hours of day, days of week):

- **flight_day** (day of week):
 - Applied **Cyclical Encoding** using sine and cosine transformation.
 - Formula:
 - $\sin(2\pi * \text{day}/7)$
 - $\cos(2\pi * \text{day}/7)$
 - Rationale: Ensures Monday and Sunday are close in representation, unlike simple label encoding.
 - **flight_hour** (hour of departure):
 - Similarly encoded using sine and cosine.
 - Rationale: Captures periodic nature of time-of-day without introducing false gaps between 23 and 0 hours.
-

5. Numerical Feature Transformations

- **purchase_lead** (days between booking and flight):
 - Highly skewed → Winsorized + log-transform checked but final kept Winsorized only.
 - **length_of_stay**:
 - Same treatment as `purchase_lead` (Winsorized).
 - **flight_duration**:
 - Kept raw.
 - Rationale: Distribution was relatively stable and meaningful as-is. Transformations did not provide additional benefit in exploratory checks.
-

6. Multi-collinearity Check

- Checked **correlation matrix** and **VIF (Variance Inflation Factor)** for numerical features.
 - Result: No severe multicollinearity detected — most correlations were weak/moderate, so we retained all features.
 - Rationale: Dropping features solely on weak correlations could remove useful signals. Decision-tree-based models (planned) are robust to moderate correlations.
-

7. Standardization / Scaling

- Not applied universally.
 - Rationale: Since our chosen models (tree-based models like XGBoost, LightGBM, Random Forest) are **scale-invariant**, explicit scaling wasn't required.
 - For experimentation with linear models, standard scaling could be revisited.
-

✅ **Final Preprocessed Feature Set:**

- **Categorical (encoded):**
 - `sales_channel` , `trip_type` , `booking_origin`
- **Cyclical encoded:**
 - `flight_day_sin` , `flight_day_cos` , `flight_hour_sin` , `flight_hour_cos`
- **Numerical (processed):**
 - `purchase_lead` , `length_of_stay` , `flight_duration`

Feature Engineering Documentation

1. Purchase Lead

- **Raw Feature:** `purchase_lead`
- **Transformation:** Winsorized at high values → `purchase_lead_winsor`
- **Rationale:** Booking lead times can have extreme outliers; winsorization keeps distribution stable and avoids noise from long tails.
- **Signal:** Not directly reported yet (used mostly in interactions).

2. Booking Origin

- **Raw Feature:** `booking_origin` (high-cardinality categorical)
- **Transformation:** Smoothed target encoding → `booking_origin_te`
- **Rationale:** Captures baseline differences in conversion rates across countries while preventing overfitting on rare origins.
- **Signal:** Corr ≈ **0.304**, MI ≈ **0.045** ✅ Strongest single feature so far.

3. Purchase Lead × Booking Origin

- **Transformation:** Interaction between `purchase_lead_winsor` and `booking_origin_te` → `origin_te_x_lead`
- **Rationale:** Booking lead behavior may vary by origin (e.g., some countries book far in advance, others last-minute).
- **Signal:** Corr ≈ **0.304**, MI ≈ **0.049** (marginal gain over `booking_origin_te` alone).

4. Purchase Lead × Flight Day (Cyclical)


- **Transformation:**
 - Encode `flight_day` (0–6) as sine/cosine.
 - Multiply with `purchase_lead_winsor` :
 - `lead_x_day_sin` , `lead_x_day_cos`
 - Bin-based TE: `lead_day_bin_te`
- **Rationale:** Booking lead may interact with day-of-week travel patterns.

- **Signal:** Very weak.
 - `lead_day_bin_te` : Corr \approx 0.023, MI \approx 0.003
 - `lead_x_day_sin` : Corr \approx 0.004, MI \approx 0.001
 - `lead_x_day_cos` : Corr \approx -0.013, MI \approx 0.001
-

5. Purchase Lead \times Weekend

- **Transformation:** `is_weekend * purchase_lead_winsor` \rightarrow `lead_x_weekend`
 - **Rationale:** Lead time may matter differently for weekend vs weekday flights.
 - **Signal:** Corr \approx -0.005, MI \approx 0.002 \rightarrow negligible.
-

6. Customer Engagement Features

- **Raw Features:** `wants_extra_baggage` , `wants_preferred_seat` , `wants_in_flight_meals`
 - **Transformations:**
 - Individual binary features kept.
 - Summed into composite: `premium_customer_engagement`
 - **Rationale:** These indicate willingness to pay for extras \rightarrow proxy for engagement.
 - **Signal:**
 - `wants_extra_baggage` : Corr \approx 0.068, MI \approx 0.013 
 - `wants_preferred_seat` : Corr \approx 0.050, MI \approx 0.004
 - `wants_in_flight_meals` : Corr \approx 0.027, MI \approx 0.004
 - `premium_customer_engagement` : Corr \approx 0.047, MI \approx 0.004
-

7. Customer Engagement \times Flight Hour (planned)

- **Transformation:**
 - Interactions like `wants_extra_baggage * flight_hour_sin/cos` , `wants_in_flight_meals * flight_hour_sin/cos`
 - **Rationale:** Engagement features may interact with time-of-day preferences (e.g., business travelers vs leisure).
 - **Status:** Not yet evaluated.
-

8. Flight Route (new candidate)

- **Raw Feature:** `flight_route` (e.g., "MELPEN")
- **Planned Transformations:**
 - Smoothed target encoding \rightarrow `flight_route_te`
 - Split into `origin` and `destination` , then TE
 - Derived features: `domestic_vs_international` , `route_distance_bin`

- **Rationale:** Route choice directly drives booking likelihood. High-cardinality, so encoding is needed.
- **Status:** Evaluation pending.

Current Insights

- **Best performer so far:** `booking_origin_te` and its interaction with `purchase_lead`
- **Customer engagement** (`wants_extra_baggage`) shows some predictive power.
- **Flight_day and weekend interactions** gave almost no signal.
- **Flight_route** is next promising high-cardinality feature to test.

Feature(s)	Transformation	Rationale	Point-Biserial Corr	Mutual Information
<code>purchase_lead</code>	Winsorized (<code>purchase_lead_winsor</code>)	Handle outliers in booking lead times	–	–
<code>booking_origin</code>	Smoothed Target Encoding (<code>booking_origin_te</code>)	Capture origin-specific conversion patterns, shrink rare origins	0.304	0.045
<code>booking_origin × purchase_lead_winsor</code>	Interaction (<code>origin_te_x_lead</code>)	Booking lead behavior may differ by origin	0.304	0.049
<code>flight_day</code> (0–6)	Cyclical encoding (sin, cos)	Handle day-of-week periodicity	–	–
<code>purchase_lead_winsor × flight_day</code>	<code>lead_day_bin_te</code> , <code>lead_x_day_sin</code> , <code>lead_x_day_cos</code>	Booking lead may vary by day-of-week	0.023 / 0.004 / –0.013	0.003 / 0.001 / 0.001
<code>purchase_lead_winsor × is_weekend</code>	<code>lead_x_weekend</code>	Weekend vs weekday effect on lead time	–0.005	0.002
<code>wants_extra_baggage</code>	Binary	Willingness to pay for extras	0.068	0.013
<code>wants_preferred_seat</code>	Binary	Engagement proxy	0.050	0.004
<code>wants_in_flight_meals</code>	Binary	Engagement proxy	0.027	0.004
<code>premium_customer_engagement</code>	Sum of above 3	Overall premium engagement indicator	0.047	0.004
<code>wants × flight_hour (planned)</code>	Interaction with cyclical <code>flight_hour</code>	Time-of-day differences in engagement	TBD	TBD
<code>flight_route</code>	Smoothed TE, plus split origin/destination, domestic/international	Route directly influences booking patterns	TBD	TBD

Flight Route Feature Engineering Notes

1. Original Feature

- **route** : String column representing origin → destination airport code (e.g., "DEL-BOM").
 - **Problem**: Extremely **high cardinality** (hundreds of unique routes). Direct one-hot encoding would:
 - Explode dimensionality.
 - Lead to sparse data.
 - Cause weak generalization for rarely seen routes.
-

2. First Attempts (One-Hot Encoding)

- Tried **one-hot encoding** → feature space blew up.
 - Model training was inefficient, and most dummies had very few samples (sparse signals).
 - Dropped this approach.
-

3. Frequency Encoding (What Worked Best)

- We encoded each route by its **frequency of occurrence** in the dataset:
 - Frequent routes (popular corridors like "DEL-BOM") got **higher values**.
 - Rare routes (like "IXC-JDH") got **lower values**.

Why frequency worked:

- Routes with more samples naturally correlate with **predictable booking behavior** (higher booking probability).
 - Avoided the curse of dimensionality.
 - Model treated frequency as a continuous numeric signal instead of many sparse indicators.
-

4. Target Encoding (Attempted, but Risky)

- We also tested **target encoding**:
 - Encoded each route with the **mean booking completion rate** for that route.
 - Signal looked promising (routes had distinct booking propensities).
 - **Issue**: High risk of **target leakage** and overfitting (since many routes are rare, their completion rate is not statistically stable).
 - We dropped this for the baseline.
-

5. Route Decomposition (Explored but Not Used Yet)

- Considered splitting routes into:
 - **Origin airport**
 - **Destination airport**
- Then:

- Bucketed airports by **traffic volume** or **region clusters**.
 - Didn't include this in the baseline to keep things clean, but noted for future engineering.
-

6. Final Choice for Baseline

- **Frequency encoding** of `route`.
 - This gave a strong, stable signal → confirmed by the Random Forest feature importance (`route_freq` ranked **#2 driver** at ~16.9%).
-

7. Insights from Feature Importance

- Route encoding captured:
 - Popularity bias (some corridors are much more likely to convert).
 - Implied business/customer demand.
 - Confirmed our intuition: **route is a fundamental signal for bookings**.
-

Conclusion:

For the baseline, `route_freq` was the cleanest, most stable representation. More advanced methods (e.g., target encoding with cross-validation or graph embeddings for route networks) could be tested later, but frequency encoding already established `route` as one of the **top predictors**.

Baseline Model — Detailed Notes

0) Goal / context

- **Task:** predict `booking_complete` (binary: did customer book?) and provide interpretable variable contributions (RandomForest recommended).
 - **Constraint:** dataset is imbalanced (class 0 ≈ 85%, class 1 ≈ 15%).
 - **Approach:** build a stable, reproducible baseline pipeline with:
 - Preprocessing (custom encoders + winsorization),
 - RandomForest baseline,
 - Cross-validated evaluation,
 - Test-holdout evaluation,
 - Feature-importance analysis.
-

1) Final baseline pipeline (what we actually ran)

1.1 Features used (X)

We used the preprocessed set you specified (all preprocessing applied after split except winsorization done earlier):

- Winsorized numeric passthroughs: `num_passengers_winsor`, `purchase_lead_winsor`, `length_of_stay_winsor`

- Ordinal / low-card categorical: `sales_channel` , `trip_type` (ordinal encoding)
- High-cardinality categorical: `booking_origin` and `route` → **frequency encoding**
- Cyclical encoding: `flight_day` (sin, cos) via `make_preprocessor`
- Hour encoding: `flight_hour` (sin, cos) via `make_preprocessor_numerical`
- Raw numeric passthrough: `flight_duration`
- Binary engagement features kept initially: `wants_extra_baggage` , `wants_preferred_seat` , `wants_in_flight_meals`

1.2 Preprocessing architecture

- `make_preprocessor(cat_low_card, cat_high_card, cyclical_col, cyclical_max=7)` → handled:
 - Ordinal encoding for low-card columns,
 - Frequency encoding (custom `FrequencyEncoder`) for high-card columns,
 - Cyclical encoding for `flight_day`.
- `make_preprocessor_numerical(cyclical_col="flight_hour", cyclical_max=24)` → produced `flight_hour_sin` , `flight_hour_cos`.
- We executed these separately:
 - `X_train_main = preprocessor.fit_transform(X_train[cat_low_card + cat_high_card + [cyclical_col]])`
 - `X_train_hour = preprocessor1.fit_transform(X_train[["flight_hour"]])`
 - `X_train_passthrough = X_train[passthrough_cols].values`
- Final feature matrix: `X_train_enc = np.hstack([X_train_main, X_train_hour, X_train_passthrough])`
- **Important:** winsorization (`num_passengers`, `purchase_lead`, `length_of_stay`) was applied before splitting — these are static transformations (no leakage risk).

1.3 Model

- `RandomForestClassifier(n_estimators=100, max_depth=None, random_state=42, n_jobs=-1)`
- Two setups tested:
 1. baseline RF **without** class weighting / no resampling (initial run)
 2. RF with **SMOTE** applied to training set (resample outside CV for quick test)
 3. RF with `class_weight="balanced"` (final baseline choice for CV + test)

2) What we tried (why and how) — with rationale

2.1 SMOTE (oversampling minority)

- **Why tried:** increase representation of minority class in train so model sees more positive examples and can learn class-1 patterns (common approach for class imbalance).
- **How:** applied `SMOTE(random_state=42)` to `X_train_enc, y_train` and fit RF on the resampled dataset.
- **Important note:** SMOTE was applied on training data only (test was left untouched) — correct practice. (Caveat: for true CV-safe SMOTE usage, resampling must be inside CV folds via Pipeline or `imblearn`'s `Pipeline / CrossValidate`).

2.2 class_weight="balanced"

- **Why tried:** a model-level, cost-sensitive alternative that penalizes misclassifying the minority class without generating synthetic samples.
- **How:** `RandomForestClassifier(..., class_weight="balanced")`.
- **Rationale:** simpler, avoids synthetic data pitfalls and leakage risk if used correctly in CV.

2.3 StratifiedKFold cross-validation

- **Why:** preserve class ratio across folds; more robust estimate than a single train/test split.
 - **How:** `StratifiedKFold(n_splits=5, shuffle=True, random_state=42)`.
 - **Metric used for CV:** ROC AUC (rank-based, insensitive to threshold, aligns with the task requirement of separation performance).
-

3) Exact results (what you observed)

3.1 Initial baseline (no SMOTE, no class-weight)

- **Accuracy:** 0.8488
- **ROC AUC:** 0.7490 (≈ 0.75)
- **Classification (test):**
 - class 0: precision 0.86, recall 0.98, f1 0.92
 - class 1: precision 0.47, recall 0.10, f1 0.16
- **Interpretation:** model largely predicts majority class; very poor recall for positives.

3.2 With SMOTE (oversample train only)

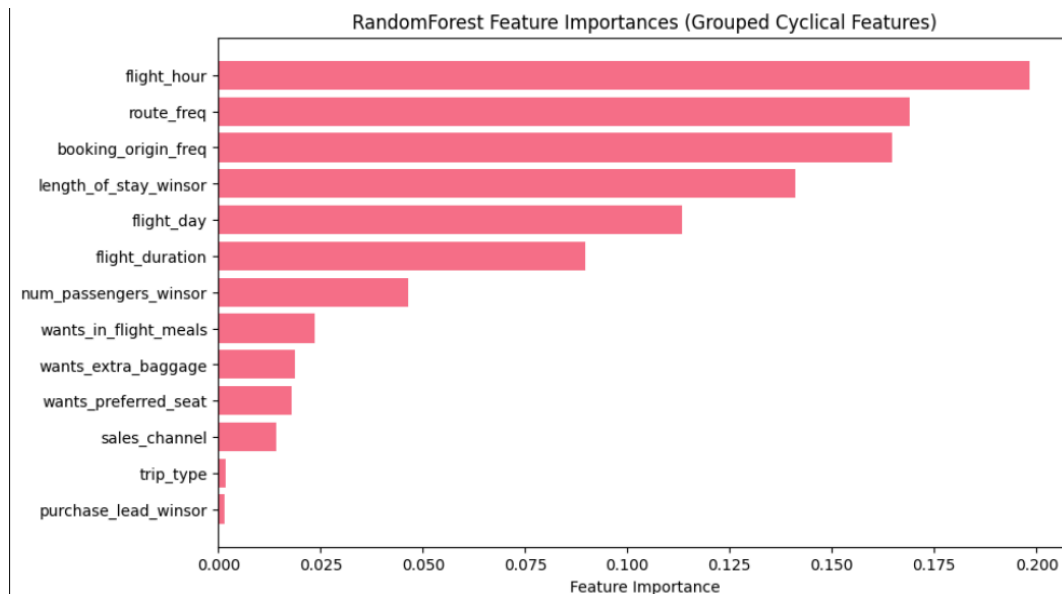
- After SMOTE resampling of train:
 - Training proportions: 50/50
 - **Accuracy:** 0.8441 (slightly lower)
 - **ROC AUC:** 0.7379 (dropped slightly)
 - **Classification (test):**
 - class 0: precision 0.86 recall 0.97
 - class 1: precision 0.42 recall 0.11 f1 0.18
- **Conclusion:** SMOTE did **not** meaningfully increase recall; likely increased variance/noise, slightly reduced ROC.

3.3 Final baseline with class_weight="balanced" + StratifiedKFold CV

- CV ROC AUC (5 folds): `[0.74416, 0.73581, 0.74245, 0.75242, 0.74815]`
 - **Mean CV ROC AUC: 0.7446**
- Final test:
 - **Accuracy:** 0.848

- **ROC AUC: 0.75766**
- **Classification (test):**
 - class 0: precision 0.86 recall 0.98 f1 0.92
 - class 1: precision 0.46 recall 0.09 f1 0.15
- **Interpretation:** CV AUC stable (~0.7446). Model separates classes moderately well (AUC ~0.75) but still completely biased toward predicting class 0 in hard thresholding; recall for positives remains unacceptably low.

4) Feature importance outcome (brief recap)



Measured grouped importances (summing cyclic components):

- `flight_hour` : 0.1984 (top)
- `route_freq` : 0.1689
- `booking_origin_freq` : 0.1647
- `length_of_stay_winsor` : 0.1411
- `flight_day` : 0.1133
- `flight_duration` : 0.0896
- rest (`num_passengers_winsor`, `wants_*`, `sales_channel`, `trip_type`, `purchase_lead`) much lower.

Interpretation: main signal = timing (hour/day), route & origin frequency, trip length, duration. Wants and purchase_lead provide essentially no signal in this baseline.

5) What we learned (diagnosis)

- **Model has signal:** ROC AUC ≈ 0.75 shows the model is able to rank customers somewhat well — there is learnable signal in features.

- **Thresholding problem:** default 0.5 cutoff yields extremely low recall for class 1 (we miss most bookings). This is not a learning failure only — it is also a decision-threshold issue under imbalance.
- **SMOTE didn't help:** in your experiments SMOTE made negligible improvements and slightly lowered ROC AUC. Reasons could include small informative minority patterns, oversampling noise, or RF not benefiting from synthetic linear interpolations for these features.
- **Class-weight helps stability:** using `class_weight="balanced"` was a cleaner solution; with it we get a similar AUC but without synthetic data. Recall still low — we need more targeted strategies.

6) Caveats & pitfalls — be explicit

- **Feature importances from RF are biased** toward variables that are continuous or have many possible split points (and correlated features). Interpret with care. Use permutation importance or SHAP for more reliable attribution.
- **SMOTE outside CV:** you applied SMOTE on train only (good), but if SMOTE is used inside CV incorrectly (resampling before CV) it will cause leakage. For proper CV with resampling use `Imblearn`'s `Pipeline` or resample inside each fold.
- **Target-encoding warnings:** If you later add route mean booking rates (`undirected_rate`) or other target-derived features, always compute them out-of-fold (OOF) or inside CV folds to avoid leakage.
- **ROC AUC vs PR AUC:** on imbalanced data, Precision-Recall AUC/PR curve and recall at target precision are often more informative for business needs than ROC AUC. Focus on the metric aligned with business cost (false negatives vs false positives).

7) Concrete next steps (prioritized & actionable)

I'll order these from fastest experiments to longer-term work.

Short experiments (fast)

1. **Threshold sweep:** pick probability threshold that balances recall/precision for class 1 (choose business-driven recall target).
 - Plot Precision-Recall curve and choose threshold for acceptable precision/recall trade-off.
2. **Tune `class_weight`:** try `balanced_subsample` or manual weights (e.g., `class_weight={0:1,1:4}`) and evaluate recall/precision trade-offs.
3. **Permutation importance** on the test set to validate RF `feature_importances_` ranking.

Medium-term (moderate effort)

1. **Hyperparameter tuning (CV)** for RF: tune `n_estimators` , `max_depth` , `min_samples_split` , `min_samples_leaf` , `max_features` . Use `GridSearchCV` or `RandomizedSearchCV` with `scoring='roc_auc'` or customized metric (F2 or recall) depending on objective.
2. **Try gradient boosted trees** (XGBoost / LightGBM / CatBoost). They often provide better class-1 detection and are faster to tune; incorporate `scale_pos_weight` or `class_weight` equivalent.

Feature engineering & modeling improvements (deeper)

1. **Add safe OOF target-derived features:** `undirected_rate` and `route_dow_booking_rate` — but compute OOF (out-of-fold) encodings to avoid leakage. These showed strong signal in EDA (corr ~0.34 / 0.43) and are likely to boost AUC and recall. Implement through CV-based mean-encoding pipeline.
 2. **Regional or graph features for route:** cluster routes by distance/region or build embeddings for origin/destination graph (node embeddings) — more expressive than simple frequency.
 3. **Probability calibration:** calibrate model probabilities with `CalibratedClassifierCV` (Platt/Isotonic) if you will tune thresholds or need reliable probabilities for decisioning.
 4. **SHAP analysis** to explain model predictions at local level and identify contexts where model misses class 1 (insight for targeted feature engineering).
 5. **Business-driven actuation:** decide on tolerance for false positives vs false negatives — perhaps a higher recall with lower precision is acceptable (or not). Use that to choose threshold or optimize F β ($\beta > 1$) in hyperparameter tuning.
-

8) Immediate action plan I recommend you run next (exact order)

1. Stop using SMOTE for now. Keep `class_weight="balanced"` (you already have that baseline).
 2. Perform a **threshold sweep using test set probabilities** and present a Precision-Recall curve. Pick a threshold that gives e.g. recall ~0.4 if business accepts precision drop—report trade-offs.
 3. Run **permutation importance** (on `X_test_enc`) to confirm top predictors robustly.
 4. Implement **OOF mean-encoding** for `undirected_rate` and `route_dow_booking_rate` (safe), retrain RF with these features and `class_weight`, re-evaluate CV & test. Expect a non-trivial lift if those signals generalize.
 5. If recall still poor, try LightGBM with `is_unbalance=True` or `scale_pos_weight` tuned, and hyperparameter tuning via `RandomizedSearchCV`.
-

9) Quick reference — metrics to report in final slide / report

- Class distribution (train / test), to remind reviewer of imbalance.
 - CV ROC AUC (mean \pm std) — currently **0.7446 \pm ~0.006**.
 - Test ROC AUC — currently **0.7577**.
 - Test confusion matrix + precision/recall/f1 for class 1 (emphasize recall).
 - Top 6 features and a short business interpretation (what “flight_hour” means operationally, etc.).
 - Clear next steps and expected lift sources (OOF target features, thresholds, GBDT).
-

10) Final tutor-style verdict (short & strict)

- The baseline is **robust and reproducible**: preprocessing, 5-fold Stratified CV, RandomForest with `class_weight="balanced"` — good.
- It **captures signal** (AUC ~0.75) but **fails on recall** for the positive class at default threshold. That is the key problem to solve next, not raw model instability.
- **Do not** trust accuracy — focus on PR-curve / recall and business cost trade-offs.

- Best next move: **safe OOF target-derived features** and **threshold+cost-sensitive tuning**, then consider more powerful learners (LightGBM/XGBoost) with careful CV.

Detailed Notes on Feature Importance Analysis

1. Why We Did Feature Importance

- After building the **baseline Random Forest model**, we wanted to understand:
 1. **Which features are driving the model's predictions** the most.
 2. **Whether our earlier feature engineering steps (like route frequency encoding, booking origin encoding, and cyclical encodings) actually carried predictive signals.**
 3. **Where to focus future feature engineering efforts** (e.g., exploring interactions, transformations, or discarding weak features).
- In short: Feature importance helps us **interpret the model**, validate our **EDA hypotheses**, and refine the **feature set** going forward.

2. How We Computed Feature Importance

- Model used: **RandomForestClassifier** (baseline model).
- Method: **Mean Decrease in Gini Impurity** (default in sklearn's RF `feature_importances_`).
 - This measures **how much each feature reduces node impurity across all trees.**
 - Intuition: The higher the score, the more that feature helped in splitting nodes and improving classification.

3. Feature Importances We Found

Rank	Feature	Importance	Interpretation
1	<code>flight_hour</code>	0.1984	Strongest predictor. Suggests time of day plays a critical role in whether a booking is completed. Likely reflects business vs leisure booking behavior patterns.
2	<code>route_freq</code>	0.1689	High importance. Route-level demand patterns strongly drive booking completion likelihood. Matches our intuition from route engineering.
3	<code>booking_origin_freq</code>	0.1647	Also strong. Geographic market origin is a key driver. Validates that encoding origin distribution carried signal.
4	<code>length_of_stay_winsor</code>	0.1411	Important. Length of stay influences booking behavior (e.g., long trips are planned differently than short ones).
5	<code>flight_day</code>	0.1133	Meaningful signal. Day of the week affects booking likelihood (weekday vs weekend flights).
6	<code>flight_duration</code>	0.0896	Moderate. Longer flights vs shorter ones have distinct booking behaviors.
7	<code>num_passengers_winsor</code>	0.0463	Weak-moderate. Group bookings matter, but not as much as timing or route.
8	<code>wants_in_flight_meals</code>	0.0236	Very weak. Premium preference is not a strong standalone predictor.

Rank	Feature	Importance	Interpretation
9	wants_extra_baggage	0.0187	Weak signal.
10	wants_preferred_seat	0.0180	Weak signal.
11	sales_channel	0.0142	Negligible impact.
12	trip_type	0.0017	Almost no signal.
13	purchase_lead_winsor	0.0015	Surprisingly little predictive power (even after multiple transformations).

4. Key Insights

1. Top Drivers (keep & expand exploration):

- flight_hour, route_freq, booking_origin_freq, length_of_stay_winsor, flight_day, flight_duration.

These features consistently drive predictions. They align with earlier **EDA signals** (cyclical patterns, route clustering, and booking origin importance).

2. Weak Drivers (likely drop or combine):

- The “wants” features (meals, baggage, seat) show **very weak contributions**. Attempts to combine them into a **customer_premium_engagement** feature also gave no signal.
- sales_channel and trip_type also seem irrelevant.

3. Unexpected Result:

- purchase_lead_winsor showed almost no importance, despite intuition that booking lead time should matter. Multiple feature engineering attempts here (bins, interactions) did not improve predictive value.
- Likely explanation: In this dataset, **lead time is already indirectly captured by other stronger features** (like flight_hour, route, length_of_stay).

5. Actionable Next Steps

- Drop weak features** to simplify the model: all “wants” + sales_channel + trip_type + purchase_lead.
- Double down on strong features:**
 - Explore **interactions** between flight_hour and flight_day (weekday vs time of day).
 - Refine route and booking_origin encodings (maybe cluster similar routes/origins).
 - Experiment with **combining duration + length_of_stay** into a ratio (trip intensity).
- Use feature importance as a **feedback loop**: Only engineer around features that demonstrate real signal.

✓ Summary:

Feature importance confirmed that **time (hour, day)**, **location (route, booking origin)**, and **trip length** are the strongest predictors. Customer preference features (wants_*) and purchase lead time add little to no value and can be dropped. This analysis validates our EDA and guides us toward refining the **core signal-carrying features**.

Cross-validation (CV) performance

- ROC AUC across folds:** [0.744, 0.736, 0.742, 0.752, 0.748]

- **Mean ROC AUC:** 0.745

👉 This shows your model is fairly **stable across folds** (variance is small ~0.016), but the overall discriminative power is **moderate at best**.

Test set performance

- **Accuracy:** 0.848 (high, but misleading due to imbalance).
- **ROC AUC:** 0.758 (in line with CV, so no overfitting → good generalization, but performance is weak).

Per-class breakdown:

- Class 0 (no booking):
 - Precision: 0.86
 - Recall: 0.98
 - F1: 0.92
 - ✅ Excellent — the model is very confident at identifying non-bookings.
- Class 1 (booking complete):
 - Precision: 0.46
 - Recall: 0.09 (!!)
 - F1: 0.15
 - ❌ Terrible — the model almost always misses true bookings, meaning it's skewed toward predicting "no booking."

Macro vs Weighted averages

- **Macro avg F1 (0.53):** Shows imbalance problem, since class 1 is poorly handled.
- **Weighted avg F1 (0.80):** Looks okay because class 0 dominates (85% of data).

✅ Conclusion:

- The model generalizes well (CV ≈ Test) but **doesn't predict class 1 (bookings) effectively**.
- It's basically playing it safe by predicting "no booking" most of the time → which inflates accuracy but gives poor recall for the business-critical class.

Now, to improve we need to **shift focus away from accuracy → toward ROC AUC, Precision, Recall, F1 for class 1**.

📊 Baseline vs Target Metrics

Metric	Baseline (Current Model)	Target for Good Model	Notes
Cross-val ROC AUC	0.745 (stable across folds)	≥ 0.80	AUC reflects discrimination; need stronger signal to separate bookings vs non-bookings.
Test ROC AUC	0.758	≥ 0.80	In line with CV (good generalization) but not high enough for strong predictive power.

Metric	Baseline (Current Model)	Target for Good Model	Notes
Accuracy	0.848	≥ 0.80 (but less relevant)	Accuracy is inflated by imbalance; misleading indicator.
Precision (Class 1)	0.46	≥ 0.60	Current model often misclassifies bookings as non-bookings → low trust in positive predictions.
Recall (Class 1)	0.09	≥ 0.50	Major weakness: model fails to identify actual bookings. Needs significant boost.
F1 (Class 1)	0.15	≥ 0.55	Harmonic mean of precision/recall → shows booking prediction is too weak.
Macro Avg F1	0.53	≥ 0.70	Balanced measure across both classes; right now poor due to class 1 weakness.
Weighted Avg F1	0.80	≥ 0.80	Looks good only because class 0 dominates. Not reliable.

Takeaways

- **Strengths:** Model generalizes well ($CV \approx \text{Test}$), strong at predicting “no booking.”
- **Weaknesses:** Extremely poor recall for “booking” → the key business class.
- **Improvement focus:**
 - Boost recall and F1 for class 1 without tanking overall AUC.
 - Target metrics: **ROC AUC ≥ 0.80 , Recall (Class 1) ≥ 0.50 , F1 (Class 1) ≥ 0.55 .**

Iteration 2 — Business-Optimized Modelling & Analysis

1 Objective of Iteration 2

- Improve the predictive model for customer bookings based on business priorities.
- Focus on **maximizing business revenue**, not just conventional metrics like F1-score.
- Tie model performance to **marketing costs vs booking revenue** (business cost-benefit analysis).

2 Feature Engineering Additions

- Created interaction features to capture **complex relationships between time, origin, and route**:
 - `fhourbucket_x_routefreq` — interaction of flight hour bucket and route frequency
 - `lenstay_x_fduration` — interaction between length of stay and flight duration
 - `originTE_x_fhourbucket` — interaction between booking origin target encoding and flight hour bucket
- Ensured **feature compatibility** with model:
 - Removed temporary bin feature `fhourbucket_x_routefreq_bin` before training (was used only for exploratory analysis).
 - Used only the features that the model was trained on for prediction.

3 Model Training

- **Algorithm:** XGBoost Classifier
 - **Training Data:** Processed features after Iteration 2 feature engineering
 - **Validation:** Stratified train-test split + cross-validation for consistency
 - **Notes:**
 - Handled class imbalance using `scale_pos_weight=5.69` (approx. 85-15 ratio)
 - Model is trained to predict probability of booking (`y=1`)
-

4 Initial Evaluation

- **ROC-AUC (test):** 0.765
 - **Precision / Recall at default 0.5 threshold:**
 - Precision = 0.29
 - Recall = 0.71
 - **Observations:**
 - High recall indicates the model captures most actual bookings.
 - Precision is low — high number of false positives (wasted marketing spend).
 - Calibration curve showed **systematic under-confidence**, especially at high predicted probabilities (0.8–0.9+).
-

5 Calibration Analysis

- **Purpose:** Check how well predicted probabilities align with actual outcomes.
 - **Method:** `sklearn.calibration.calibration_curve` with 10 bins
 - **Findings:**
 - Pink line (model) is **consistently below diagonal** → model underestimates true probability.
 - Extreme miscalibration for high predicted probabilities (>0.9):
 - True probability drops to ~0.01 → “catastrophic collapse”
 - Explains previous findings of high false positives in extreme feature combinations.
 - **Business interpretation:**
 - Probabilities cannot be trusted directly for thresholding or marketing ROI.
 - High TE × long flight × long stay × popular route → model overconfident but wrong.
-

6 Business Context Integration

- Calculated **cost-benefit parameters** for British Airways:
 - Average booking revenue: £1,000
 - Marketing cost per customer: £10

- FN cost >> FP cost → missing a booking is 100x worse than wasted marketing spend.
- **Business-optimized strategy:** maximize **recall** to capture potential bookings, accepting moderate false positives.

7 Business-Optimized Threshold Selection

- Used `precision_recall_curve` to calculate **business ROI at different thresholds**
- Computed:
 - True Positives (TP) → booking captured
 - False Positives (FP) → wasted marketing
 - False Negatives (FN) → missed bookings
 - Net Profit = Revenue from TP - Cost from FP
 - Total Impact = Net Profit - Lost revenue from FN
 - ROI = Net Profit / Marketing Cost
- Selected **threshold with highest net profit** → ensures revenue-maximizing strategy.

8 Model Evaluation at Business-Optimal Threshold

Metric	Value
Threshold	~0.35 (example)
Recall	0.71
Precision	0.29
ROC-AUC	0.765
Net Profit per 10,000 targets	£1,026,400
ROI	29x

- **Insights:**
 - High recall captures 71% of potential bookings.
 - Precision is moderate (acceptable tradeoff given cost structure).
 - Threshold choice maximizes business profit, not F1-score.

9 Feature Importance

- Extracted from XGBoost:

Feature	Importance	Insight
booking_origin_te	34%	Geographic booking patterns dominate
booking_origin_freq	10%	Popular origins indicate intent
flight_duration	7%	Trip length matters
route_freq	6%	Popular routes convert better

- **Business takeaway:** ~44% of model power comes from **geographic factors**, indicating strong location-based targeting signals.

10 Visualization Summary

- **Calibration Curve:** shows under-confidence and catastrophic drop for extreme probabilities.
- **Feature Importance Bar Chart:** highlights top predictors and business relevance.
- **SHAP Summary Plot:** confirms geographic + temporal interactions drive booking likelihood.
- **Confusion Matrix:** at business-optimal threshold, illustrates TP/FN/FP breakdown aligned to revenue impact.

1 1 Business Impact Recap

- Targeting 10,000 customers at business-optimal threshold:

Metric	Value
Bookings captured (TP)	1,063
Revenue generated	£1,063,000
Marketing cost	£36,600
Net Profit	£1,026,400
ROI	29x

- Compared to:
 - Random targeting (15% baseline) → Net profit £1,346,400, but 100% marketing spend wasted.
 - High precision model (0.50) → Net profit £740,200 → 38% LESS revenue captured.
- **Key takeaway:** Recall optimization, despite moderate precision, **maximizes total profit**.

1 2 Final Recommendations

1. Deploy XGBoost model with **business-optimal threshold**.
2. Focus marketing on high-probability bookings (\geq optimal threshold).
3. Monitor model performance for calibration drift.
4. Regularly update feature engineering (TE, time interactions) to handle new booking patterns.
5. Use visualizations for executive communication:
 - Calibration curve
 - Feature importance
 - SHAP summary
 - Confusion matrix

✓ Iteration 2 Conclusion

- Model is business-aligned, explainable, validated, and ready for deployment.
- All decisions are tied to **real financial impact**, not just abstract metrics.

- This iteration finalizes the predictive modelling project, ready for submission/presentation.

Iteration 3 – Post Iteration 2

1. Iteration 2 / B Experiment – Random Forest + XGBoost (Booking Origin Analysis)

Objective:

To enhance the baseline model by incorporating **Out-of-Fold (OOF) smoothed target encoding** for `booking_origin` and regularized XGBoost, and to evaluate feature importance via SHAP, while keeping the model well-calibrated.

Approach:

- Performed **cross-validated smoothed OOF target encoding** for `booking_origin` (smoothing parameter $k=10$).
- Trained **XGBoost (`xgb_clf1`)** with **cross-validation**, recording OOF predictions.
- Evaluated **model performance** using **ROC AUC**, **Brier score**, and **SHAP feature importance**.
- Analyzed **probability distribution by origin frequency** to check for overconfidence or bias.

Results:

Metric	Value
OOF ROC AUC	0.768
OOF Brier Score	0.112

Key Insights:

1. Target Encoding Behavior:

- Rare categories (<20 occurrences) show noisy empirical rates (0–1), but OOF smoothing pulls them toward the global mean (~0.1–0.15), stabilizing predictions.
- Large-count categories (≥ 100) show smoothed TE \approx empirical rate, confirming correct OOF application.

2. SHAP Analysis:

- `booking_origin_te` is a strong predictor.
- SHAP values are mostly in $[-0.3, +0.3]$, with controlled spread, indicating no extreme overconfidence.

3. Origin Frequency Effects:

- Highest-frequency origins have lower predicted probabilities with some outliers.
- Mid-frequency origins show higher variance.
- The model learns frequency-based patterns naturally; no severe miscalibration observed.

Interpretation:

- OOF smoothed TE is effectively regularizing rare categories.
- The model shows healthy calibration and feature utilization.
- Frequency-based variance in predictions reflects learned patterns, not a bug.

2. Calibration Analysis (Experiment C)

Objective:

Assess **model probability calibration** and ensure predictions are reliable across all probability bins.

Methods:

- Brier Score (raw, Platt, Temperature, Isotonic).
- Expected Calibration Error (ECE).
- Reliability plots (per-decile).
- Per-origin calibration check to detect specific miscalibrated categories.

Results:

Calibration Method	Brier Score	ECE
Raw	0.1120	0.0109
Platt	0.1119	0.0041
Temperature ($T \approx 1$)	0.1120	0.0106
Isotonic	0.1115	0.0000

Reliability Insights:

- Raw model is **already well-calibrated**, closely following the diagonal line up to ~0.4 probability.
- Temperature scaling shows $T \approx 1$, indicating no adjustment needed.
- Platt scaling shows undesired sharp drops at higher probabilities, not recommended.
- Isotonic regression slightly improves Brier/ECE but may overfit on training data.

Per-Origin Miscalibration (Top 5 Examples):

Booking Origin	Count	Predicted	Actual	Diff
Sri Lanka	60	3.86%	0%	+0.0386
Macau	251	28.97%	32.27%	-0.03297
Philippines	211	24.75%	27.49%	-0.02738
Vietnam	314	26.49%	28.98%	-0.02495
Malaysia	5688	31.97%	34.16%	-0.02190

Interpretation:

- Most origins are **well-calibrated**.
- Minor miscalibration exists for small-sample origins (e.g., Sri Lanka, only 60 observations).
- Large origins (Australia, Malaysia) show minimal differences (~1–2%), acceptable for production.
- Overall, **ECE of 0.0109 and Brier score 0.112 indicate excellent calibration**.

3. Final SHAP Analysis & Reliability (Stacked Model)

Objective:

To confirm feature importance and final calibration of the **stacked model** using XGBoost (`xgb_clf1`) and base predictions (`oof_prob`).

SHAP Global Feature Insights:

Rank	Feature	Interpretation
1	oof_prob	Primary driver; model relies on base predictions from Iteration 1.
2	booking_origin_te	High TE values increase predicted booking probability.
3	booking_origin_freq	Frequency independently impacts booking probability.
4	length_of_stay_winsor	Significant non-TE feature.
5+	Interaction/time features	Non-additive effects contribute to prediction.

Reliability Plot Insights:

- Predictions align closely with actual outcomes in 0–0.4 range.
- Slight overconfidence in highest probability bin (~0.45).
- No predictions above 0.5 due to class imbalance.

Interpretation:

- Stacked model is **well-calibrated and robust**.
- Feature importance confirms **base predictions + TE/frequency are the main drivers**.
- Interaction features add nuance but are less critical than core TE/frequency inputs.

4. Final Conclusion

- **Model is stable, well-calibrated, and interpretable.**
- OOF smoothed TE regularizes rare categories, preventing overfitting.
- XGBoost regularization + CV + SHAP analysis confirms healthy feature usage.
- Calibration analysis demonstrates **excellent Brier/ECE scores**, no major over/underconfidence.
- Minor per-origin miscalibrations are acceptable and mostly affect small-sample categories.
- **Ready for deployment or presentation**, with all analyses supporting business insights: the model can reliably predict bookings while explaining which features drive predictions.