Exploratory Analysis for TravelTide Marketing Project

Goal is to segment customer behavior to target specific perks for the new rewards program to improve customer retention

Perks

- Free Hotel Meal
- Free Checked Bag
- No Cancellation Fees
- Exclusive Discounts
- 1 Night Free Hotel With Flight
- Complementary Lounge Access

Initial exploration was done in the Travel Tide Exploration Viz notebook but was starting to get sidetracked. I am moving forward with exploring potential segments in this notebook.

In [3]:
```
%run vincenty.ipynb
```

First, we will establish the Database connection and create the cohort filter

In [4]:
```
from sqlalchemy import create_engine

# Create a connection using SQLAlchemy
DATABASE_URL = "postgresql+psycopg2://Test:bQNxVzJL4g6u@ep-noisy-flower-846766.us-east
engine = create_engine(DATABASE_URL)

# Cohort filter definition
cohort_filter = """
WITH CohortUsers AS (
    SELECT user_id
    FROM sessions
    WHERE session_start > '2023-01-04'
    GROUP BY user_id
    HAVING COUNT(session_id) > 7
)
"""
```

First we need to get the total number of unique users in this cohort.

In [5]:
```
cohort_size_query = f"""
{cohort_filter}
```

```
SELECT
    COUNT(*) AS total_sessions,
    COUNT(DISTINCT user_id) AS users
FROM sessions
WHERE user_id IN (SELECT user_id FROM CohortUsers);
"""

with engine.connect() as connection:
    result = connection.execute(cohort_size_query).fetchall()

total_sessions = result[0][0]
total_users = result[0][1]

# Print the results
print("Total Sessions:", total_sessions)
print("Total Users:", total_users)
```

```
Total Sessions: 50547
Total Users: 5998
```

Potential customers for the **Free Hotel Meal** perk

Couples on a Weekend Getaway

User must be married. Booking for hotel or flight and hotel must start on a Friday and end on a Sunday. If there is a flight, there must be only 2 seats and have a return flight. If it is a hotel only booking, there must be only 1 room booked.

In [6]:
```
import pandas as pd

# Create the query
weekend_getaway_query = f"""
{cohort_filter},
WeekendGetaways AS (
    SELECT hb.trip_id, s.user_id, fb.trip_id AS flight_trip_id
    FROM hotels AS hb
    JOIN sessions AS s ON hb.trip_id = s.trip_id
    JOIN users AS u ON s.user_id = u.user_id
    LEFT JOIN flights AS fb ON hb.trip_id = fb.trip_id
    WHERE u.married = TRUE
      AND EXTRACT(DOW FROM hb.check_in_time) = 5  -- Friday
      AND EXTRACT(DOW FROM hb.check_out_time) = 0  -- Sunday
      AND hb.rooms = 1  -- Exactly 1 room
),
TwoSeatFlights AS (
    SELECT trip_id
    FROM flights
    WHERE seats = 2 AND return_flight_booked = TRUE  -- Exactly 2 seats and return fli
)
SELECT wg.user_id, COUNT(wg.trip_id) AS num_bookings
FROM WeekendGetaways AS wg
JOIN CohortUsers AS cu ON wg.user_id = cu.user_id
LEFT JOIN TwoSeatFlights AS tsf ON wg.flight_trip_id = tsf.trip_id
GROUP BY wg.user_id
ORDER BY num_bookings DESC;
"""
```

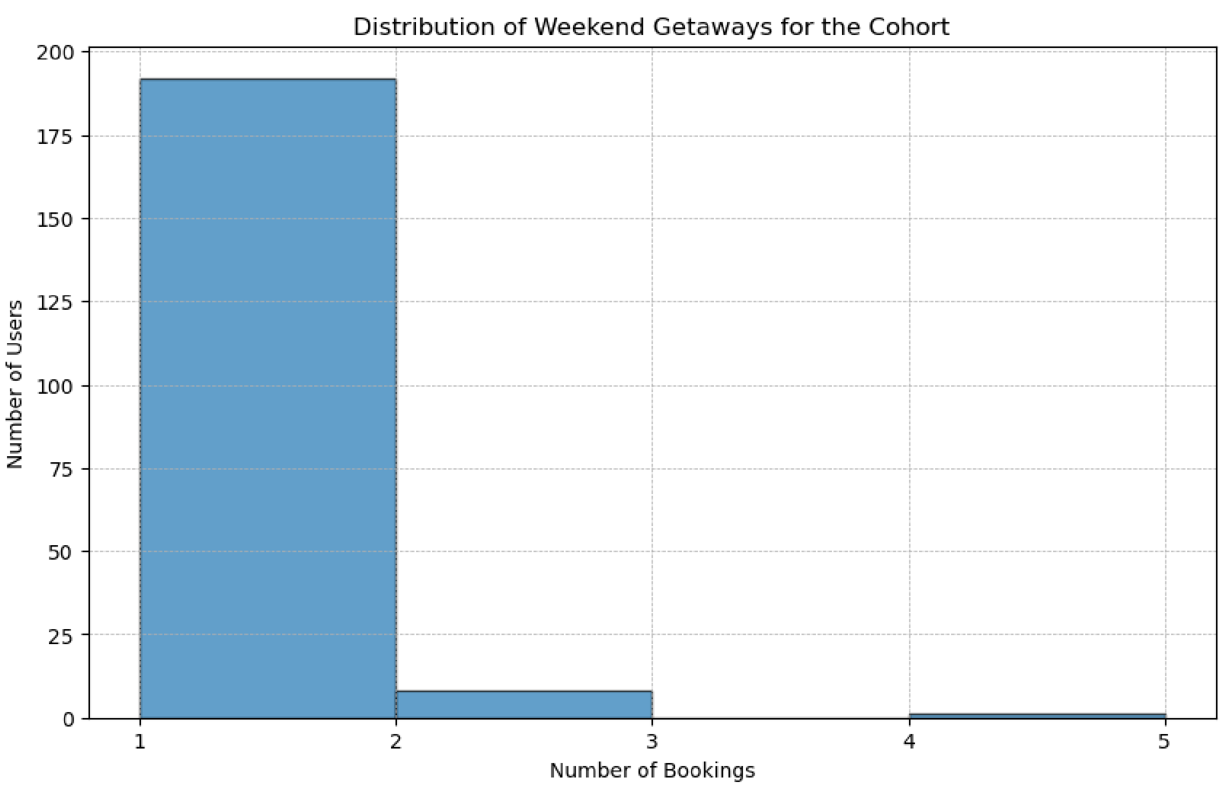
```
#Run the query and store in a DF
df_weekend_getaway = pd.read_sql(weekend_getaway_query, engine)
```

We will now look at what this group contains

In [7]:
```
print(df_weekend_getaway['num_bookings'].describe())
```

```
count    201.000000
mean       1.054726
std        0.286339
min        1.000000
25%        1.000000
50%        1.000000
75%        1.000000
max        4.000000
Name: num_bookings, dtype: float64
```

In [8]:
```
import matplotlib.pyplot as plt

# Histogram for Weekend Getaway

plt.figure(figsize=(10, 6))
plt.hist(df_weekend_getaway['num_bookings'], bins=range(1, 5+1), edgecolor='k', alpha=
plt.title('Distribution of Weekend Getaways for the Cohort')
plt.xlabel('Number of Bookings')
plt.ylabel('Number of Users')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.xticks(range(1, 5+1))
plt.show()
```



In [9]:
```
users_with_multiple_weekend_bookings = df_weekend_getaway[df_weekend_getaway['num_book
num_users_with_multiple_weekend_bookings = len(users_with_multiple_weekend_bookings)
print("Number of users who booked weekend getaway at least twice:", num_users_with_mul
```

```
Number of users who booked weekend getaway at least twice: 9
```

Couples on a Weekend Getaway Analysis

This is a very small group - 201 users have booked this type of trip. Most of them have booked it once. We only have 9 that have booked at least twice.

Budget-Conscious Travelers

We are going to look at the following critera to consider a user a budget-conscious traveler:

- Advance Bookings
- Low-Cost Flights
- Off Peak Bookings
- Uses discounts consistently

Low Cost Metric

In [10]:
```
#Build the flight query

flight_query = f"""
{cohort_filter}
SELECT
    u.user_id AS user_id,
    f.base_fare_usd AS flight_cost,
    s.trip_id AS trip_id,
    s.flight_discount AS flight_discount,
    s.flight_discount_amount AS flight_discount_percent,
    f.seats AS seats_booked,
    s.hotel_discount AS hotel_discount,
    s.hotel_discount_amount AS hotel_discount_percent,
    h.rooms AS rooms_booked,
    h.hotel_per_room_usd AS hotel_cost_per_room,
    s.session_end AS session_end,
    f.departure_time AS departure_time,
    f.destination_airport AS destination_airport,
    u.home_airport AS origin_airport,
    f.destination_airport_lat AS destination_lat,
    f.destination_airport_lon AS destination_lon,
    u.home_airport_lat AS origin_lat,
    u.home_airport_lon AS origin_lon
FROM
    users AS u
JOIN
    sessions AS s ON u.user_id = s.user_id
JOIN
    flights AS f ON s.trip_id = f.trip_id
LEFT JOIN
    hotels AS h ON f.trip_id = h.trip_id
WHERE
    u.user_id IN (SELECT user_id FROM CohortUsers);
"""

# Fetch the data into a DataFrame
```

```
df_flight_data = pd.read_sql(flight_query, engine)

print(df_flight_data.head)
```

```
<bound method NDFrame.head of        user_id  flight_cost                                  
trip_id  \
0        23557        98.46   23557-127abde78c8f4352b7a84483d2576c25
1       433080       436.10  433080-ae391e65085b425d93c663f33c9eb552
2       447737       299.93  447737-2db2e8c63d5f4390889eae142362591c
3       407434       143.18  407434-94fb8965a20e4cf0bd03c39f87135991
4       465568       577.43  465568-ce24a24881bc47eea5cb86a1bb87d027
...        ...          ...                                      ...
14914   438551     13645.96  438551-1a9fcfc16bd8487ea23e820957d23279
14915   423042        61.63  423042-d80c9d5c1b5f4ee1ac40567b34c7242b
14916   614311      4912.79  614311-d21df59cc4ab4640add93d76b450f37c
14917   520947       409.10  520947-83843cd2814a4b48b62399969dfa5c60
14918   622343       214.03  622343-2061dc37f60d410f9af97a51a7c8b676

       flight_discount  flight_discount_percent  seats_booked  hotel_discount  \
0                 True                     0.15             1           False
1                False                      NaN             1            True
2                False                      NaN             1           False
3                False                      NaN             1           False
4                 True                     0.05             1           False
...                ...                      ...           ...             ...
14914             True                      NaN             6            True
14915             True                      NaN             1            True
14916             True                      NaN             2            True
14917             True                      NaN             1            True
14918             True                      NaN             1            True

       hotel_discount_percent  rooms_booked  hotel_cost_per_room  \
0                         NaN           1.0                118.0
1                        0.15           1.0               1190.0
2                         NaN           1.0                187.0
3                         NaN           NaN                  NaN
4                         NaN           1.0                118.0
...                       ...           ...                  ...
14914                     NaN           3.0                143.0
14915                     NaN           1.0                152.0
14916                     NaN           NaN                  NaN
14917                     NaN           1.0                181.0
14918                     NaN           NaN                  NaN

                      session_end      departure_time destination_airport  \
0     2021-07-24 09:17:51.000000 2021-07-30 10:00:00                 YKZ
1     2022-11-26 12:46:18.000000 2022-12-05 11:00:00                 HLZ
2     2022-12-07 15:34:17.000000 2022-12-15 07:00:00                 LAS
3     2022-12-13 18:46:29.000000 2022-12-24 15:00:00                 YMX
4     2022-12-13 11:22:56.000000 2022-12-17 12:00:00                 LGA
...                          ...                 ...                 ...
14914 2023-05-31 23:03:37.000000 2023-10-09 07:00:00                 AGR
14915 2023-05-22 11:48:08.220450 2023-05-26 07:00:00                 YTZ
14916 2023-07-23 16:21:46.323900 2024-02-15 07:00:00                 HAN
14917 2023-07-10 22:44:16.677255 2024-01-04 07:00:00                 DAL
14918 2023-07-20 19:51:01.770539 2023-07-21 16:00:00                 LGA

      origin_airport  destination_lat  destination_lon  origin_lat  origin_lon
0                LGA           43.862          -79.370      40.777     -73.872
1                COS           43.173          -79.935      38.806    -104.700
2                YYC           36.080         -115.152      51.114    -114.020
3                YAW           45.517          -73.417      44.640     -63.499
4                BIF           40.640          -73.779      31.849    -106.380
...              ...              ...              ...         ...         ...
```

```
14914           CLT         27.156        77.961     35.214     -80.943
14915           YOW         43.862       -79.370     45.323     -75.669
14916           MEM         21.222       105.806     35.042     -89.977
14917           LGA         32.847       -96.852     40.777     -73.872
14918           ORD         40.640       -73.779     41.979     -87.904

[14919 rows x 18 columns]>
```

We are going to need to clean the City/Country data

In exploration it was found there were city/country mismatches. We will use Ninja API to first match the IATA Code, then Google Maps for Lat/Long. Finally, we will review the data and make any final manual matches.

In [11]:
```
!pip install pyyaml

import yaml

with open('key.yaml', 'r') as f:
    api_keys = yaml.safe_load(f)

ninja_key = api_keys['ninja_key']
gmaps_key = api_keys['gmaps_key']
```

```
Requirement already satisfied: pyyaml in c:\users\bhaze\anaconda3\lib\site-packages
(6.0)
```

In [12]:
```
import requests

# Step 1: Create a list of unique airport codes
unique_origin_airports = df_flight_data['origin_airport'].unique().tolist()
unique_destination_airports = df_flight_data['destination_airport'].unique().tolist()

# Combine and remove duplicates
unique_airports = list(set(unique_origin_airports + unique_destination_airports))

# Step 2: Fetch city and country info using Airports API
airport_info = {}
for iata_code in unique_airports:
    api_url = f'https://api.api-ninjas.com/v1/airports?iata={iata_code}'
    response = requests.get(api_url, headers={'X-Api-Key': ninja_key})
    if response.status_code == requests.codes.ok:
        data = response.json()
        if data:  # Check if the response is empty
            airport_info[iata_code] = {'city': data[0].get('city', None), 'country': d
        else:
            airport_info[iata_code] = {'city': None, 'country': None}


# Step 3: Populate DataFrame with city and country info
def get_airport_info(row, col_name, info_type):
    iata_code = row[col_name]
    return airport_info.get(iata_code, {}).get(info_type, None)

df_flight_data['origin_city'] = df_flight_data.apply(get_airport_info, args=('origin_a
df_flight_data['origin_country'] = df_flight_data.apply(get_airport_info, args=('origi
df_flight_data['destination_city'] = df_flight_data.apply(get_airport_info, args=('des
df_flight_data['destination_country'] = df_flight_data.apply(get_airport_info, args=('
```

```python
# Check for missing data
missing_origin_city = df_flight_data['origin_city'].isna().sum()
missing_origin_country = df_flight_data['origin_country'].isna().sum()
missing_destination_city = df_flight_data['destination_city'].isna().sum()
missing_destination_country = df_flight_data['destination_country'].isna().sum()

print(f"Missing origin cities: {missing_origin_city}")
print(f"Missing origin countries: {missing_origin_country}")
print(f"Missing destination cities: {missing_destination_city}")
print(f"Missing destination countries: {missing_destination_country}")

print(df_flight_data.head())
```

```
Missing origin cities: 2477
Missing origin countries: 2477
Missing destination cities: 2367
Missing destination countries: 2367
   user_id  flight_cost                                trip_id  \
0    23557        98.46   23557-127abde78c8f4352b7a84483d2576c25
1   433080       436.10  433080-ae391e65085b425d93c663f33c9eb552
2   447737       299.93  447737-2db2e8c63d5f4390889eae142362591c
3   407434       143.18  407434-94fb8965a20e4cf0bd03c39f87135991
4   465568       577.43  465568-ce24a24881bc47eea5cb86a1bb87d027

   flight_discount  flight_discount_percent  seats_booked  hotel_discount  \
0             True                     0.15             1           False
1            False                      NaN             1            True
2            False                      NaN             1           False
3            False                      NaN             1           False
4             True                     0.05             1           False

   hotel_discount_percent  rooms_booked  hotel_cost_per_room  ...  \
0                     NaN           1.0                118.0  ...
1                    0.15           1.0               1190.0  ...
2                     NaN           1.0                187.0  ...
3                     NaN           NaN                  NaN  ...
4                     NaN           1.0                118.0  ...

  destination_airport origin_airport destination_lat destination_lon  \
0                 YKZ            LGA          43.862         -79.370
1                 HLZ            COS          43.173         -79.935
2                 LAS            YYC          36.080        -115.152
3                 YMX            YAW          45.517         -73.417
4                 LGA            BIF          40.640         -73.779

   origin_lat  origin_lon         origin_city  origin_country  \
0      40.777     -73.872            New York              US
1      38.806    -104.700    Colorado Springs              US
2      51.114    -114.020             Calgary              CA
3      44.640     -63.499                None            None
4      31.849    -106.380  Fort Bliss/El Paso              US

  destination_city destination_country
0          Toronto                  CA
1         Hamilton                  NZ
2        Las Vegas                  US
3         Montreal                  CA
4         New York                  US

[5 rows x 22 columns]
```

In [13]:

```python
# Create sets to store unique lat/long pairs for origin and destination
unique_origin_lat_long = set(df_flight_data[df_flight_data['origin_city'].isna()][['or
unique_destination_lat_long = set(df_flight_data[df_flight_data['destination_city'].is

# Combine and remove duplicates
unique_lat_long_pairs = unique_origin_lat_long.union(unique_destination_lat_long)

import googlemaps

# Initialize Google Maps API client
gmaps = googlemaps.Client(key=gmaps_key)
```

```
# Dictionary to store city and country info for each unique lat/long pair
lat_long_info = {}

# Fetch city and country info using Google Maps API
for lat, lon in unique_lat_long_pairs:
    geocode_result = gmaps.reverse_geocode((lat, lon))
    city = None
    country = None
    for component in geocode_result[0]['address_components']:
        if 'locality' in component['types']:
            city = component['long_name']
        if 'country' in component['types']:
            country = component['short_name']
    lat_long_info[(lat, lon)] = {'city': city, 'country': country}

# Function to populate DataFrame with missing city and country info
def get_missing_info(row, lat_col, lon_col, info_type):
    lat = row[lat_col]
    lon = row[lon_col]
    return lat_long_info.get((lat, lon), {}).get(info_type, None)

# Populate DataFrame
df_flight_data.loc[df_flight_data['origin_city'].isna(), 'origin_city'] = df_flight_da
df_flight_data.loc[df_flight_data['origin_country'].isna(), 'origin_country'] = df_fli
df_flight_data.loc[df_flight_data['destination_city'].isna(), 'destination_city'] = df
df_flight_data.loc[df_flight_data['destination_country'].isna(), 'destination_country'

# Check for missing data again
missing_origin_city = df_flight_data['origin_city'].isna().sum()
missing_origin_country = df_flight_data['origin_country'].isna().sum()
missing_destination_city = df_flight_data['destination_city'].isna().sum()
missing_destination_country = df_flight_data['destination_country'].isna().sum()

print(f"Missing origin cities: {missing_origin_city}")
print(f"Missing origin countries: {missing_origin_country}")
print(f"Missing destination cities: {missing_destination_city}")
print(f"Missing destination countries: {missing_destination_country}")

print(df_flight_data.head())
```

```
Missing origin cities: 37
Missing origin countries: 37
Missing destination cities: 1
Missing destination countries: 0
   user_id  flight_cost                                  trip_id  \
0    23557        98.46   23557-127abde78c8f4352b7a84483d2576c25
1   433080       436.10  433080-ae391e65085b425d93c663f33c9eb552
2   447737       299.93  447737-2db2e8c63d5f4390889eae142362591c
3   407434       143.18  407434-94fb8965a20e4cf0bd03c39f87135991
4   465568       577.43  465568-ce24a24881bc47eea5cb86a1bb87d027

   flight_discount  flight_discount_percent  seats_booked  hotel_discount  \
0             True                     0.15             1           False
1            False                      NaN             1            True
2            False                      NaN             1           False
3            False                      NaN             1           False
4             True                     0.05             1           False

   hotel_discount_percent  rooms_booked  hotel_cost_per_room  ...  \
0                     NaN           1.0                118.0  ...
1                    0.15           1.0               1190.0  ...
2                     NaN           1.0                187.0  ...
3                     NaN           NaN                  NaN  ...
4                     NaN           1.0                118.0  ...

  destination_airport origin_airport destination_lat destination_lon  \
0                 YKZ            LGA          43.862         -79.370
1                 HLZ            COS          43.173         -79.935
2                 LAS            YYC          36.080        -115.152
3                 YMX            YAW          45.517         -73.417
4                 LGA            BIF          40.640         -73.779

   origin_lat  origin_lon         origin_city  origin_country  \
0      40.777     -73.872            New York              US
1      38.806    -104.700   Colorado Springs              US
2      51.114    -114.020             Calgary              CA
3      44.640     -63.499          Shearwater              CA
4      31.849    -106.380  Fort Bliss/El Paso              US

  destination_city destination_country
0          Toronto                  CA
1         Hamilton                  NZ
2        Las Vegas                  US
3         Montreal                  CA
4         New York                  US

[5 rows x 22 columns]
```

In [14]:

```
# Manually update rows with IATA code 'TNT', they are not being added with either API.
df_flight_data.loc[df_flight_data['origin_airport'] == 'TNT', 'origin_city'] = 'Miami'
df_flight_data.loc[df_flight_data['origin_airport'] == 'TNT', 'origin_country'] = 'US'
df_flight_data.loc[df_flight_data['destination_airport'] == 'TNT', 'destination_city']
df_flight_data.loc[df_flight_data['destination_airport'] == 'TNT', 'destination_countr

# Manually update rows with IATA code 'STU', there is one row and it is not being upda
df_flight_data.loc[df_flight_data['destination_airport'] == 'STU', 'destination_city']
df_flight_data.loc[df_flight_data['destination_airport'] == 'STU', 'destination_countr

# Manually correct YAV as it was giving Parkdale, US not Winnipeg, CA
df_flight_data.loc[df_flight_data['origin_airport'] == 'YAV', 'origin_city'] = 'Winnip
```

```
df_flight_data.loc[df_flight_data['origin_airport'] == 'YAV', 'origin_country'] = 'CA'
df_flight_data.loc[df_flight_data['destination_airport'] == 'YAV', 'destination_city']
df_flight_data.loc[df_flight_data['destination_airport'] == 'YAV', 'destination_countr


# Check for missing data again
missing_origin_city = df_flight_data['origin_city'].isna().sum()
missing_origin_country = df_flight_data['origin_country'].isna().sum()
missing_destination_city = df_flight_data['destination_city'].isna().sum()
missing_destination_country = df_flight_data['destination_country'].isna().sum()

print(f"Missing origin cities: {missing_origin_city}")
print(f"Missing origin countries: {missing_origin_country}")
print(f"Missing destination cities: {missing_destination_city}")
print(f"Missing destination countries: {missing_destination_country}")



print(df_flight_data.head())
```

```
Missing origin cities: 0
Missing origin countries: 0
Missing destination cities: 0
Missing destination countries: 0
   user_id  flight_cost                                  trip_id  \
0    23557        98.46   23557-127abde78c8f4352b7a84483d2576c25
1   433080       436.10  433080-ae391e65085b425d93c663f33c9eb552
2   447737       299.93  447737-2db2e8c63d5f4390889eae142362591c
3   407434       143.18  407434-94fb8965a20e4cf0bd03c39f87135991
4   465568       577.43  465568-ce24a24881bc47eea5cb86a1bb87d027

   flight_discount  flight_discount_percent  seats_booked  hotel_discount  \
0             True                     0.15             1           False
1            False                      NaN             1            True
2            False                      NaN             1           False
3            False                      NaN             1           False
4             True                     0.05             1           False

   hotel_discount_percent  rooms_booked  hotel_cost_per_room  ...  \
0                     NaN           1.0                118.0  ...
1                    0.15           1.0               1190.0  ...
2                     NaN           1.0                187.0  ...
3                     NaN           NaN                  NaN  ...
4                     NaN           1.0                118.0  ...

  destination_airport origin_airport destination_lat destination_lon  \
0                 YKZ            LGA          43.862         -79.370
1                 HLZ            COS          43.173         -79.935
2                 LAS            YYC          36.080        -115.152
3                 YMX            YAW          45.517         -73.417
4                 LGA            BIF          40.640         -73.779

   origin_lat  origin_lon         origin_city  origin_country  \
0      40.777     -73.872            New York              US
1      38.806    -104.700    Colorado Springs              US
2      51.114    -114.020             Calgary              CA
3      44.640     -63.499          Shearwater              CA
4      31.849    -106.380  Fort Bliss/El Paso              US

  destination_city destination_country
0          Toronto                  CA
1         Hamilton                  NZ
2        Las Vegas                  US
3         Montreal                  CA
4         New York                  US

[5 rows x 22 columns]
```

In [15]:

```python
# Create a set to store unique origin and destination pairs
city_country_pairs = set()

# Function to add a city/country pair to the set
def add_city_country_pair(city, country, airport):
    if city and country and airport:
        city_country_pairs.add((city, country, airport))

# Iterate through the DataFrame to add origin and destination pairs
for index, row in df_flight_data.iterrows():
    origin_city = row["origin_city"]
    origin_country = row["origin_country"]
```

```
    origin_airport = row["origin_airport"]
    destination_city = row["destination_city"]
    destination_country = row["destination_country"]
    destination_airport = row["destination_airport"]

    add_city_country_pair(origin_city, origin_country, origin_airport)
    add_city_country_pair(destination_city, destination_country, destination_airport)

# Convert the set to a list for easier manipulation
city_country_pairs_list = list(city_country_pairs)

# Sort the list by city name (the first element in each tuple)
city_country_pairs_list = sorted(city_country_pairs_list, key=lambda x: x[0])

# Print the sorted list of city/country pairs
for pair in city_country_pairs_list:
    print(pair)
```

```
('Abu Dhabi', 'AE', 'AUH')
('Akure', 'NG', 'AKR')
('Amarillo', 'US', 'AMA')
('Amman', 'JO', 'AMM')
('Amsterdam', 'NL', 'AMS')
('Anchorage', 'US', 'MRI')
('Anchorage', 'US', 'ANC')
('Anchorage', 'US', 'EDF')
('Antalya', 'TR', 'AYT')
('Atlanta', 'US', 'ATL')
('Auckland', 'NZ', 'AKL')
('Austin', 'US', 'AUS')
('Bakersfield', 'US', 'BFL')
('Baltimore', 'US', 'BWI')
('Bangalore', 'IN', 'BLR')
('Bangkok', 'TH', 'BKK')
('Barcelona', 'ES', 'BCN')
('Baton Rouge', 'US', 'BTR')
('Beijing', 'CN', 'PEK')
('Belle Chasse', 'US', 'NBG')
('Berlin', 'DE', 'TXL')
('Berlin', 'DE', 'THF')
('Birmingham', 'US', 'BHM')
('Bossier City', 'US', 'BAD')
('Boston', 'US', 'BOS')
('Brownsville', 'US', 'BRO')
('Brussels', 'BE', 'BRU')
('Budapest', 'HU', 'BUD')
('Buenos Aires', 'AR', 'AEP')
('Buffalo', 'US', 'BUF')
('Burlington', 'US', 'BTV')
('Cairo', 'EG', 'CAI')
('Calgary', 'CA', 'YYC')
('Cape Town', 'ZA', 'CPT')
('Casablanca', 'MA', 'CMN')
('Charlotte', 'US', 'CLT')
('Chengdu', 'CN', 'CTU')
('Chicago', 'US', 'ORD')
('Chicago', 'US', 'MDW')
('Chicago', 'US', 'UGN')
('Cincinnati', 'US', 'LUK')
('Cleveland', 'US', 'CLE')
('Colombo', 'LK', 'CMB')
('Colorado Springs', 'US', 'COS')
('Columbus', 'US', 'LCK')
('Columbus', 'US', 'CMH')
('Copenhagen', 'DK', 'RKE')
('Coronado', 'US', 'NZY')
('Corpus Christi', 'US', 'CRP')
('Dalian', 'CN', 'DLC')
('Dallas', 'US', 'DAL')
('Denpasar-Bali Island', 'ID', 'DPS')
('Denver', 'US', 'DEN')
('Des Moines', 'US', 'DSM')
('Detroit', 'US', 'YIP')
('Detroit', 'US', 'DTW')
('Detroit', 'US', 'DET')
('Dubai', 'AE', 'DXB')
('Dublin', 'IE', 'DUB')
('Durban', 'ZA', 'DUR')
```

```
('Durban', 'ZA', 'VIR')
('Edinburgh', 'GB', 'EDI')
('Edmonton', 'CA', 'YEG')
('Edmonton', 'CA', 'YED')
('Edmonton', 'CA', 'YXD')
('El Paso', 'US', 'ELP')
('Fairchild Air Force Base', 'US', 'SKA')
('Fayetteville', 'US', 'FYV')
('Florence', 'US', 'FLO')
('Fort Bliss/El Paso', 'US', 'BIF')
('Fort Worth', 'US', 'FTW')
('Fresno', 'US', 'FAT')
('Fukuoka', 'JP', 'FUK')
('Gatineau', 'CA', 'YND')
('Geneva', 'CH', 'GVA')
('Glendale', 'US', 'LUF')
('Grand Rapids', 'US', 'GRR')
('Guangzhou', 'CN', 'CAN')
('Guilin City', 'CN', 'KWL')
('Halifax', 'CA', 'YHZ')
('Hamburg', 'DE', 'HAM')
('Hamburg', 'DE', 'XFW')
('Hamilton', 'NZ', 'HLZ')
('Hamilton', 'CA', 'YHM')
('Hanoi', 'VN', 'HAN')
('Hebron', 'US', 'CVG')
('Heraklion', 'GR', 'HER')
('Ho Chi Minh City', 'VN', 'SGN')
('Hong Kong', 'HK', 'HKG')
('Honolulu', 'US', 'HNL')
('Houston', 'US', 'IAH')
('Houston', 'US', 'EFD')
('Houston', 'US', 'HOU')
('Hurghada', 'EG', 'HRG')
('Indianapolis', 'US', 'IND')
('Istanbul', 'TR', 'IST')
('JBSA Randolph', 'US', 'RND')
('JBSA Randolph', 'US', 'SKF')
('Jacksonville', 'US', 'LRF')
('Jacksonville', 'US', 'JAX')
('Jacksonville', 'US', 'NZC')
('Jacksonville', 'US', 'NIP')
('Jaipur', 'IN', 'JAI')
('Jakarta', 'ID', 'PCB')
('Jakarta', 'ID', 'HLP')
('Jerusalem', 'IL', 'JRS')
('Johannesburg', 'ZA', 'HLA')
('Johannesburg', 'ZA', 'JNB')
('Joint Base Lewis-McChord', 'US', 'TCM')
('Kansas City', 'US', 'MCI')
('Knoxville', 'US', 'TYS')
('Kuala Lumpur', 'MY', 'KUL')
('Lagos', 'NG', 'LOS')
('Laredo', 'US', 'LRD')
('Las Vegas', 'US', 'LAS')
('Lincoln', 'US', 'LNK')
('Lisbon', 'PT', 'LIS')
('Little Rock', 'US', 'LIT')
('London', 'GB', 'LTN')
('London', 'GB', 'LCY')
```

```
('London', 'GB', 'STN')
('London', 'GB', 'LGW')
('London', 'CA', 'YXU')
('Long Beach', 'US', 'LGB')
('Los Angeles', 'US', 'LAX')
('Los Angeles', 'CL', 'LSQ')
('Louisville', 'US', 'LOU')
('Lubbock', 'US', 'LBB')
('Madison', 'US', 'MSN')
('Madrid', 'ES', 'MAD')
('Madrid', 'ES', 'TOJ')
('Manila', 'PH', 'MNL')
('March Air Reserve Base', 'US', 'RIV')
('Markham', 'CA', 'YZD')
('McClellan Park', 'US', 'MCC')
('Melbourne', 'AU', 'MEL')
('Memphis', 'US', 'MEM')
('Mexico City', 'MX', 'MEX')
('Miami', 'US', 'OPF')
('Miami', 'US', 'MIA')
('Miami', 'US', 'TNT')
('Milan', 'IT', 'LIN')
('Milwaukee', 'US', 'MKE')
('Minneapolis', 'US', 'MSP')
('Mobile', 'US', 'BFM')
('Mobile', 'US', 'MOB')
('Modesto', 'US', 'MOD')
('Montgomery', 'US', 'MXF')
('Montreal', 'CA', 'YHU')
('Montreal', 'CA', 'YMX')
('Montreal', 'CA', 'YUL')
('Moscow', 'RU', 'VKO')
('Moscow', 'RU', 'SVO')
('Munich', 'DE', 'MUC')
('Napoli', 'IT', 'NAP')
('Nashville', 'US', 'BNA')
('New Delhi', 'IN', 'DEL')
('New Orleans', 'US', 'MSY')
('New York', 'US', 'JFK')
('New York', 'US', 'LGA')
('Newark', 'US', 'EWR')
('Newport News', 'US', 'PHF')
('Nice', 'FR', 'NCE')
('Norfolk', 'US', 'NGU')
('Norfolk', 'US', 'ORF')
('Oakland', 'US', 'OAK')
('Offutt Air Force Base', 'US', 'OFF')
('Oklahoma City', 'US', 'OKC')
('Oklahoma City', 'US', 'TIK')
('Omaha', 'US', 'OMA')
('Opa-locka', 'US', 'OPF')
('Orlando', 'US', 'MCO')
('Orlando', 'US', 'ORL')
('Osaka', 'JP', 'ITM')
('Ottawa', 'CA', 'YOW')
('Paradise', 'US', 'LSV')
('Paris', 'FR', 'CDG')
('Paris', 'FR', 'ORY')
('Paris', 'FR', 'LBG')
('Philadelphia', 'US', 'PHL')
```

```
('Philadelphia', 'US', 'PNE')
('Phoenix', 'US', 'PHX')
('Phuket', 'TH', 'HKT')
('Pope Field', 'US', 'POB')
('Portland', 'US', 'PDX')
('Portland', 'US', 'PWM')
('Prague', 'CZ', 'PRG')
('Providence', 'US', 'PVD')
('Pune', 'IN', 'PNQ')
('Punta Cana', 'DO', 'PUJ')
('Qingdao', 'CN', 'TAO')
('Quebec', 'CA', 'YQB')
('Rancho Cordova', 'US', 'MHR')
('Reno', 'US', 'RNO')
('Richmond', 'US', 'RIC')
('Rio De Janeiro', 'BR', 'GIG')
('Riverside', 'US', 'RAL')
('Riyadh', 'SA', 'RUH')
('Rochester', 'US', 'ROC')
('Roma', 'IT', 'CIA')
('Rome', 'IT', 'FCO')
('Rome', 'US', 'RME')
('Sacramento', 'US', 'SMF')
('Sacramento', 'US', 'SAC')
('Salt Lake City', 'US', 'SLC')
('San Antonio', 'US', 'SAT')
('San Antonio', 'US', 'SKF')
('San Diego', 'US', 'SAN')
('San Francisco', 'US', 'SFO')
('San Jose', 'PH', 'SJI')
('San Jose', 'CR', 'SJO')
('San Jose', 'US', 'SJC')
('Santa Ana', 'US', 'SNA')
('Santa Cruz', 'BZ', 'STU')
('Saskatoon', 'CA', 'YXE')
('Schönefeld', 'DE', 'SXF')
('Seattle', 'US', 'SEA')
('Seattle', 'US', 'BFI')
('Seletar', 'SG', 'XSP')
('Senai', 'MY', 'JHB')
('Seoul', 'KR', 'GMP')
('Shanghai', 'CN', 'SHA')
('Shearwater', 'CA', 'YAW')
('Shenzhen', 'CN', 'SZX')
('Shreveport', 'US', 'SHV')
('Singapore', 'SG', 'SIN')
('Spokane', 'US', 'SFF')
('Spokane', 'US', 'GEG')
('St Louis', 'US', 'STL')
('St Petersburg-Clearwater', 'US', 'PIE')
('St. Petersburg', 'US', 'SPG')
('Stockton', 'US', 'SCK')
('Sunrise Manor', 'US', 'LSV')
('Sydney', 'AU', 'BWU')
('Taipa', 'MO', 'MFM')
('Taipei', 'TW', 'TPE')
('Taipei City', 'TW', 'TSA')
('Tallahassee', 'US', 'TLH')
('Tampa', 'US', 'MCF')
('Tampa', 'US', 'TPA')
```

```
('Tinicum Township', 'US', 'PNE')
('Tokyo', 'JP', 'NRT')
('Tokyo', 'JP', 'HND')
('Toronto', 'CA', 'YTZ')
('Toronto', 'CA', 'YYZ')
('Toronto', 'CA', 'YKZ')
('Toronto', 'CA', 'YZD')
('Tucson', 'US', 'TUS')
('Tucson', 'US', 'DMA')
('Tulsa', 'US', 'TUL')
('Vancouver', 'CA', 'YVR')
('Venezia', 'IT', 'VCE')
('Victoria', 'CA', 'YYJ')
('Vienna', 'AT', 'VIE')
('Warsaw', 'PL', 'WAW')
('Washington', 'US', 'DCA')
('Washington', 'US', 'IAD')
('Waukegan', 'US', 'UGN')
('Wichita', 'US', 'IAB')
('Wichita', 'US', 'ICT')
('Windsor', 'CA', 'YQG')
('Winnipeg', 'CA', 'YAV')
('Winnipeg', 'CA', 'YWG')
('Winston Salem', 'US', 'INT')
('Xiamen', 'CN', 'XMN')
('Xianyang', 'CN', 'XIY')
```

In [16]:

```python
import numpy as np

# Calculate flight cost after discount
def calculate_flight_cost(row):
    if row['flight_discount']:
        return round(row['flight_cost'] * (1 - row['flight_discount_percent']), 2)
    return round(row['flight_cost'], 2)

def calculate_total_flight_cost(row):
    if row['flight_cost']:
        return round(row['flight_cost'] * row['seats_booked'], 2)
    return row['flight_cost']

# Calculate total hotel cost
def calculate_hotel_cost(row):
    if pd.isna(row['rooms_booked']) or pd.isna(row['hotel_cost_per_room']):
        return np.nan
    return round(row['rooms_booked'] * row['hotel_cost_per_room'], 2)

# Calculate hotel cost after discount
def calculate_hotel_cost_after_discount(row):
    if row['hotel_discount']:
        return round(row['total_hotel_cost'] * (1 - row['hotel_discount_percent']), 2
    return row['total_hotel_cost']

# Calculate total trip cost
def calculate_trip_cost(row):
    flight_cost = row['total_flight_cost']
    hotel_cost = row['hotel_cost_after_discount']

    if pd.isna(flight_cost) and pd.isna(hotel_cost):
        return np.nan
    elif pd.isna(flight_cost):
```

```
        return hotel_cost
    elif pd.isna(hotel_cost):
        return flight_cost
    else:
        return round(flight_cost + hotel_cost, 2)

# Apply the calculations
df_flight_data['flight_cost_after_discount'] = df_flight_data.apply(calculate_flight_c
df_flight_data['total_flight_cost'] = df_flight_data.apply(calculate_total_flight_cost
df_flight_data['total_hotel_cost'] = df_flight_data.apply(calculate_hotel_cost, axis=1
df_flight_data['hotel_cost_after_discount'] = df_flight_data.apply(calculate_hotel_cos
df_flight_data['trip_cost'] = df_flight_data.apply(calculate_trip_cost, axis=1)
```

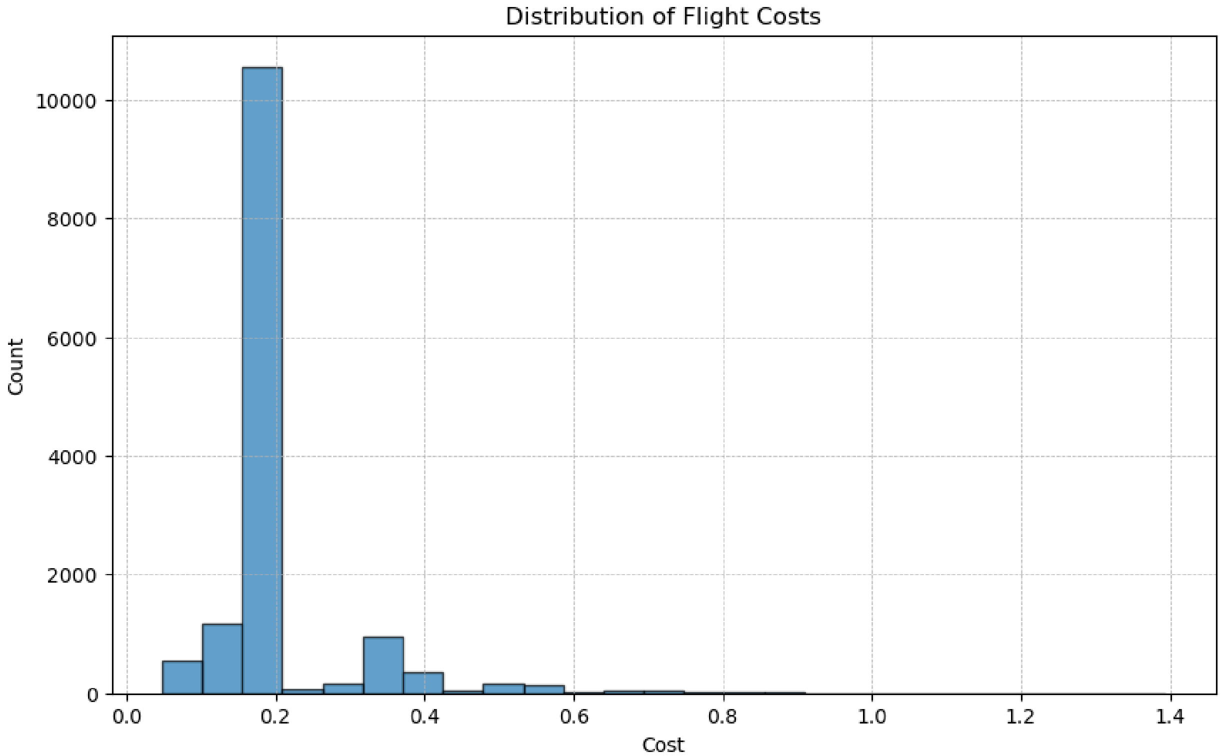
In [17]:
```
# Calculate the distance using the vincenty_distance function and add it as a new colu
df_flight_data['trip_distance_km'] = df_flight_data.apply(
    lambda row: vincenty_distance(
        row['origin_lat'], row['origin_lon'],
        row['destination_lat'], row['destination_lon']
    ), axis=1
)
```

In [18]:
```
# Create a cost per km metric for the flight, after discount

# Calculate the cost per km and add it as a new column
df_flight_data['cost_per_km'] = df_flight_data['flight_cost_after_discount'] / df_flig

# 587 rows do not have a discount amount for flights even though the discount boolean
df_flight_data = df_flight_data[df_flight_data['cost_per_km'].notna()]
```

In [19]:
```
print(df_flight_data['cost_per_km'].describe())
```

```
count    14332.000000
mean         0.205004
std          0.099782
min          0.047749
25%          0.167802
50%          0.178233
75%          0.190132
max          1.394014
Name: cost_per_km, dtype: float64
```

In [20]:
```
# Histogram for Flight Cost Ratio
plt.figure(figsize=(10, 6))
plt.hist(df_flight_data['cost_per_km'], bins=25, edgecolor='k', alpha=0.7)
plt.title('Distribution of Flight Costs')
plt.xlabel('Cost')
plt.ylabel('Count')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.show()
```



In [21]:

```
# Skewness and Kurtosis
from scipy.stats import skew, kurtosis

skewness = skew(df_flight_data['cost_per_km'])
kurt = kurtosis(df_flight_data['cost_per_km'])
print(f"Skewness: {skewness}")
print(f"Kurtosis: {kurt}")
```

```
Skewness: 3.4272418640595848
Kurtosis: 16.960699975848417
```

Far from a normal distribution! So, we are just looking at customers who are frequent bookers, or maybe those who have racked up big discounts.

In [22]:

```
# Group by user_id and count the number of discounted flights
df_frequency_analysis = df_flight_data[df_flight_data['flight_discount'] == True].grou
# Sort by count in descending order
df_frequency_analysis = df_frequency_analysis.sort_values(by='count_discounted_flights

print(df_frequency_analysis['count_discounted_flights'].describe())
skewness = skew(df_frequency_analysis['count_discounted_flights'])
kurt = kurtosis(df_frequency_analysis['count_discounted_flights'])
print(f"Skewness: {skewness}")
print(f"Kurtosis: {kurt}")
```

```
count    1767.000000
mean        1.160724
std         0.415140
min         1.000000
25%         1.000000
50%         1.000000
75%         1.000000
max         4.000000
Name: count_discounted_flights, dtype: float64
Skewness: 2.7385141519469838
Kurtosis: 8.126055068030034
```

In [23]:

```
count_over_3_bookings = df_frequency_analysis[df_frequency_analysis['count_discounted_
print(f'Number of customers with more than 3 discounted flight bookings: {count_over_3
```

```
Number of customers with more than 3 discounted flight bookings: 3
```

3 customers is TOO SMALL of a group!

In [24]:

```
from datetime import datetime

# Assuming df_flight_data is your DataFrame
df_flight_data['session_end'] = pd.to_datetime(df_flight_data['session_end'])
df_flight_data['departure_time'] = pd.to_datetime(df_flight_data['departure_time'])

# Calculate the advance booking period in days
df_flight_data['advance_booking_days'] = (df_flight_data['departure_time'] - df_flight

# Display some statistics
print(df_flight_data['advance_booking_days'].describe())

# Skewness and Kurtosis
from scipy.stats import skew, kurtosis
skewness = skew(df_flight_data['advance_booking_days'].dropna())
kurt = kurtosis(df_flight_data['advance_booking_days'].dropna())
print(f"Skewness: {skewness}")
print(f"Kurtosis: {kurt}")
```

```
count    14332.000000
mean        15.406852
std         44.015516
min          1.000000
25%          5.000000
50%          7.000000
75%          9.000000
max        392.000000
Name: advance_booking_days, dtype: float64
Skewness: 5.4207672801598825
Kurtosis: 29.37439272808127
```

In [25]:

```
# Filter the DataFrame to only include rows where 'advance_booking_days' is 90 or more
bargain_hunters_df = df_flight_data[df_flight_data['advance_booking_days'] >= 90]

# Display some basic statistics about this subset
print(bargain_hunters_df['advance_booking_days'].describe())

# You can also look at other columns to understand more about this subset
# For example, you might want to know the average 'flight_cost' among these bargain hu
print("Average flight cost among bargain hunters:", bargain_hunters_df['flight_cost'].
```

```
count    517.000000
mean     235.338491
std       56.400866
min      112.000000
25%      196.000000
50%      224.000000
75%      280.000000
max      392.000000
Name: advance_booking_days, dtype: float64
Average flight cost among bargain hunters: 2435.2833849129593
```

In [26]:
```python
# Size of this group compared to the cohort
bargain_hunters = bargain_hunters_df['user_id'].nunique()
percent_of_cohort = (bargain_hunters / total_users) * 100

print(f'The Bargain Hunters are {percent_of_cohort}% of the cohort, there are {bargair
```

```
The Bargain Hunters are 8.53617872624208% of the cohort, there are 512 Bargain Hunter
s
```

In [27]:
```python
### This seems to be a viable segment at 8.5% of the cohort.
```

Travelers that are spending 1 night in a hotel, how about this group?

Critera:

- Check in before 6pm
- Check out is the next day
- Flight booking is not required

In [28]:
```python
#Buld the query
overnight_hotel_query = f"""
{cohort_filter},
OneNightHotelUsers AS (
    SELECT s.user_id, h.trip_id
    FROM sessions AS s
    JOIN hotels AS h ON s.trip_id = h.trip_id
    JOIN CohortUsers AS cu ON s.user_id = cu.user_id
    WHERE h.check_in_time::time < '18:00:00'  -- Check-in before 6pm
      AND h.check_out_time::date = (h.check_in_time::date + interval '1 day')  -- Chec
)
SELECT user_id, COUNT(trip_id) AS num_hotel_stays
FROM OneNightHotelUsers
GROUP BY user_id;
"""

df_overnight_hotel_stays = pd.read_sql(overnight_hotel_query, engine)
```

In [29]:
```python
print(df_overnight_hotel_stays['num_hotel_stays'].describe())
```

```
count    1368.000000
mean        1.167398
std         0.417841
min         1.000000
25%         1.000000
50%         1.000000
75%         1.000000
max         4.000000
Name: num_hotel_stays, dtype: float64
```

In [30]:
```
# At least 2 bookings
customers_at_least_twice = (df_overnight_hotel_stays['num_hotel_stays'] >= 2).sum()
print(f"Number of customers who booked at least twice: {customers_at_least_twice}")
print(f"This is {(customers_at_least_twice / total_users) * 100}% of the cohort")
```

```
Number of customers who booked at least twice: 207
This is 3.4511503834611537% of the cohort
```

207 Customers isn't huge, but with the budget-consious group this is reasonable segment.

In [31]:
```
# Get the user_ids from bargain_hunters_df
bargain_hunters_users = bargain_hunters_df['user_id'].tolist()

# Get the user_ids from df_overnight_hotel_stays that have at least 2 hotel stays
at_least_two_hotel_stays_users = df_overnight_hotel_stays[df_overnight_hotel_stays['nu

# Combine the user IDs from both groups
free_hotel_meals_users = bargain_hunters_users + at_least_two_hotel_stays_users

# Remove duplicates to ensure unique user IDs
free_hotel_meals_users = list(set(free_hotel_meals_users))

# Calculate the count and percentage
free_hotel_meals_users_count = len(free_hotel_meals_users)
percentage_of_cohort = (free_hotel_meals_users_count / total_users) * 100

print(f"There are {free_hotel_meals_users_count} users to target for the Free Hotel Me
```

```
There are 707 users to target for the Free Hotel Meal perk, making up 11.79% of the c
ohort
```

Free Hotel Meal Segment

Budget-Consious Travelers and Overnight Travelers

When looking at travelers looking for deals we looked at travelers that are booking more than 90 days in advance to get the best pricing. There are 512 users meeting this criteria. For Overnight Travelers we looked at one night hotel bookings with a check in before 6pm. There are 207 users meeting this criteria.

Combining these groups there are 707 users, or 11.8% of the cohort.

The Budget-Conscious travelers are a group that would always be looking for a *free meal* while the overnight travelers would find having a meal after getting settled into their hotel for the night a great perk.

Free Cancellation

This one is sort of a *low hanging fruit* so to speak. We can look at users who had the only intent to cancel their booking. However, these users might belong to other segments if they have multiple bookings.

In [32]:
```
#define query
cancellation_query = f"""
{cohort_filter}
SELECT s.user_id AS user_id, COUNT(cancellation) as number_of_cancellations
FROM sessions AS s
INNER JOIN CohortUsers AS cu ON s.user_id = cu.user_id
WHERE cancellation = True
GROUP BY s.user_id;
"""

df_cancellation = pd.read_sql(cancellation_query, engine)

print(df_cancellation['number_of_cancellations'].describe())
```

```
count    620.000000
mean       1.029032
std        0.168033
min        1.000000
25%        1.000000
50%        1.000000
75%        1.000000
max        2.000000
Name: number_of_cancellations, dtype: float64
```

let's see if these customers rebooked.

In [33]:
```
# Define the query
rebooked_query = f"""
{cohort_filter}
SELECT s.user_id AS user_id,
       SUM(CASE WHEN cancellation = True THEN 1 ELSE 0 END) AS number_of_cancellations
       SUM(CASE WHEN flight_booked = True OR hotel_booked = True THEN 1 ELSE 0 END) AS
FROM sessions AS s
INNER JOIN CohortUsers AS cu ON s.user_id = cu.user_id
GROUP BY s.user_id
HAVING SUM(CASE WHEN cancellation = True THEN 1 ELSE 0 END) > 0
   AND SUM(CASE WHEN flight_booked = True OR hotel_booked = True THEN 1 ELSE 0 END) >
"""

# Execute the query and store the results in a DataFrame
df_rebooked = pd.read_sql(rebooked_query, engine)

print(df_rebooked['number_of_rebookings'].describe())
```

```
count    620.000000
mean       4.195161
std        1.512140
min        2.000000
25%        3.000000
50%        4.000000
75%        5.000000
max       10.000000
Name: number_of_rebookings, dtype: float64
```

Yes, there are enough to make a segment, but, I will want to create this segment last as every user that cancelled rebooked.

We will use this group, but only after we define all other segments.

Complementary Lounge Access

International Travelers will appreciate this perk

In [34]:
```
# Function to determine if a flight is international
def is_international(row):
    if row["origin_country"] != row["destination_country"]:
        return True
    return False

# Add the "is_international" column to the DataFrame
df_flight_data["is_international"] = df_flight_data.apply(is_international, axis=1)

# Print the updated DataFrame
print(df_flight_data)
```

```
       user_id  flight_cost                                trip_id  \
0        23557        98.46   23557-127abde78c8f4352b7a84483d2576c25
1       433080       436.10  433080-ae391e65085b425d93c663f33c9eb552
2       447737       299.93  447737-2db2e8c63d5f4390889eae142362591c
3       407434       143.18  407434-94fb8965a20e4cf0bd03c39f87135991
4       465568       577.43  465568-ce24a24881bc47eea5cb86a1bb87d027
...        ...          ...                                      ...
14908   406210       536.61  406210-6c29806425804fe2b4da095371e25cfb
14909   517403        98.30  517403-df27118ae54948589d761eb88bd25548
14910   528735       560.01  528735-40ab1cc1756745c1814b683d2af2e07c
14911   653640      3060.05  653640-0f7354c3dc7341899c2097b2cb38049d
14912   676762       735.98  676762-fab74cbfff574a19ac12771344a29aeb

       flight_discount  flight_discount_percent  seats_booked  hotel_discount  \
0                 True                     0.15             1           False
1                False                      NaN             1            True
2                False                      NaN             1           False
3                False                      NaN             1           False
4                 True                     0.05             1           False
...                ...                      ...           ...             ...
14908            False                      NaN             1           False
14909            False                      NaN             1            True
14910            False                      NaN             2           False
14911             True                     0.15             2           False
14912            False                      NaN             1           False

       hotel_discount_percent  rooms_booked  hotel_cost_per_room  ...  \
0                         NaN           1.0                118.0  ...
1                        0.15           1.0               1190.0  ...
2                         NaN           1.0                187.0  ...
3                         NaN           NaN                  NaN  ...
4                         NaN           1.0                118.0  ...
...                       ...           ...                  ...  ...
14908                     NaN           1.0                102.0  ...
14909                    0.15           1.0                329.0  ...
14910                     NaN           2.0                224.0  ...
14911                     NaN           1.0                184.0  ...
14912                     NaN           1.0                171.0  ...

      destination_country flight_cost_after_discount total_flight_cost  \
0                      CA                      83.69             98.46
1                      NZ                     436.10            436.10
2                      US                     299.93            299.93
3                      CA                     143.18            143.18
4                      US                     548.56            577.43
...                   ...                        ...               ...
14908                  JP                     536.61            536.61
14909                  US                      98.30             98.30
14910                  CA                     560.01           1120.02
14911                  FR                    2601.04           6120.10
14912                  US                     735.98            735.98

      total_hotel_cost  hotel_cost_after_discount  trip_cost  \
0                118.0                     118.00     216.46
1               1190.0                    1011.50    1447.60
2                187.0                     187.00     486.93
3                  NaN                        NaN     143.18
4                118.0                     118.00     695.43
...                ...                        ...        ...
14908            102.0                     102.00     638.61
```

```
14909          329.0                      279.65     377.95
14910          448.0                      448.00    1568.02
14911          184.0                      184.00    6304.10
14912          171.0                      171.00     906.98

       trip_distance_km  cost_per_km advance_booking_days is_international
0                   568     0.147342                    6             True
1                  2131     0.204646                    8             True
2                  1673     0.179277                    7             True
3                   786     0.182163                   10            False
4                  3068     0.178801                    4            False
...                 ...          ...                  ...              ...
14908              5879     0.091276                  309             True
14909               584     0.168322                    7             True
14910              1642     0.341054                    9             True
14911              8986     0.289455                  224             True
14912              4162     0.176833                    7            False

[14332 rows x 31 columns]
```

In [35]:

```python
# Count the number of international flights
international_flights = df_flight_data["is_international"].sum()

# Print the result
print(f"Number of Intenational flights: {international_flights}")
```

```
Number of Intenational flights: 5450
```

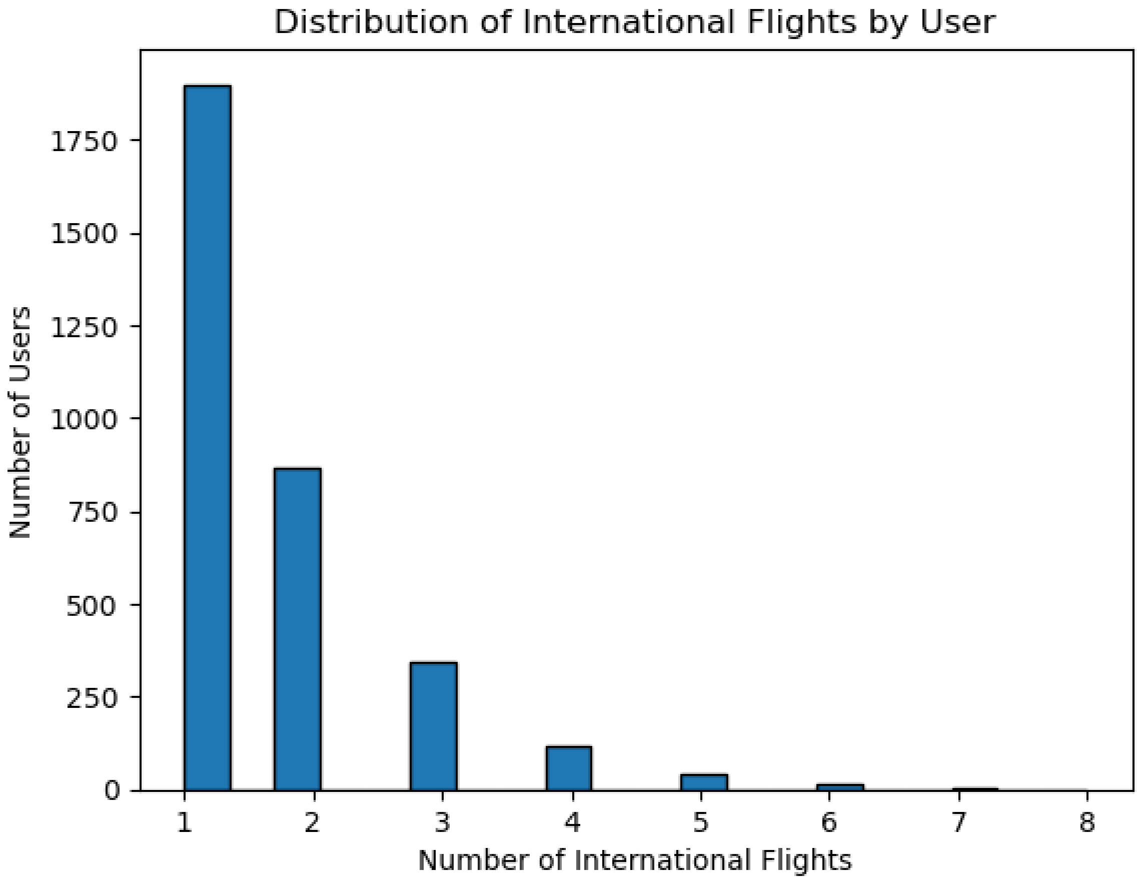
In [36]:

```python
international_flights_by_user = df_flight_data[df_flight_data['is_international'] == T
print(international_flights_by_user)
```

```
user_id
23557     2
94883     2
101486    3
101961    3
106907    1
         ..
780167    1
785107    2
792549    2
796032    2
801660    3
Name: is_international, Length: 3287, dtype: int64
```

In [37]:

```python
# Create a histogram of international flights by user
plt.hist(international_flights_by_user, bins=20, edgecolor='black')
plt.xlabel('Number of International Flights')
plt.ylabel('Number of Users')
plt.title('Distribution of International Flights by User')
plt.show()

# Get descriptive statistics
stats = international_flights_by_user.describe()
print(stats)
```




```
count    3287.000000
mean        1.658047
std         0.957344
min         1.000000
25%         1.000000
50%         1.000000
75%         2.000000
max         8.000000
Name: is_international, dtype: float64
```

In [38]:
```
# Filter users with 3 or more international flights
users_with_3_or_more_flights = international_flights_by_user[international_flights_by_

# Print the users
print(users_with_3_or_more_flights)
```

```
user_id
101486    3
101961    3
149058    3
153982    4
190866    3
         ..
679611    4
689624    3
705215    3
714565    3
801660    3
Name: is_international, Length: 520, dtype: int64
```

We have 520 frequent international travelers. These would be a good target for Complementary Lounge Access

Exclusive Discounts

Here I am thinking we target our high-value customers. Those that book a lot or book expensive trips.

In [39]:
```
# Group the DataFrame by user ID and calculate the sum of trip_cost for each user
total_booking_amount_by_user = df_flight_data.groupby("user_id")["trip_cost"].sum()

# Print the result
print(total_booking_amount_by_user.describe())
```

```
count      5206.000000
mean       2687.740857
std        5189.760853
min           8.710000
25%         934.955000
50%        1654.745000
75%        2869.605000
max      173975.030000
Name: trip_cost, dtype: float64
```

In [40]:
```
# Calculate the threshold for the top 10% of high-value customers
top_15_threshold = total_booking_amount_by_user.quantile(0.85)

# Print the threshold
print(f"Top 15% threshold: {top_15_threshold}")
```

```
Top 15% threshold: 4003.785
```

In [41]:
```
# Get a list of high-value customer IDs
high_value_customers = total_booking_amount_by_user[total_booking_amount_by_user >= to

print(len(high_value_customers))
```

```
781
```

781 high value customers.

These are the top 15% based on their total booking cost.

Free 1 Night Hotel Stay With Flight perk

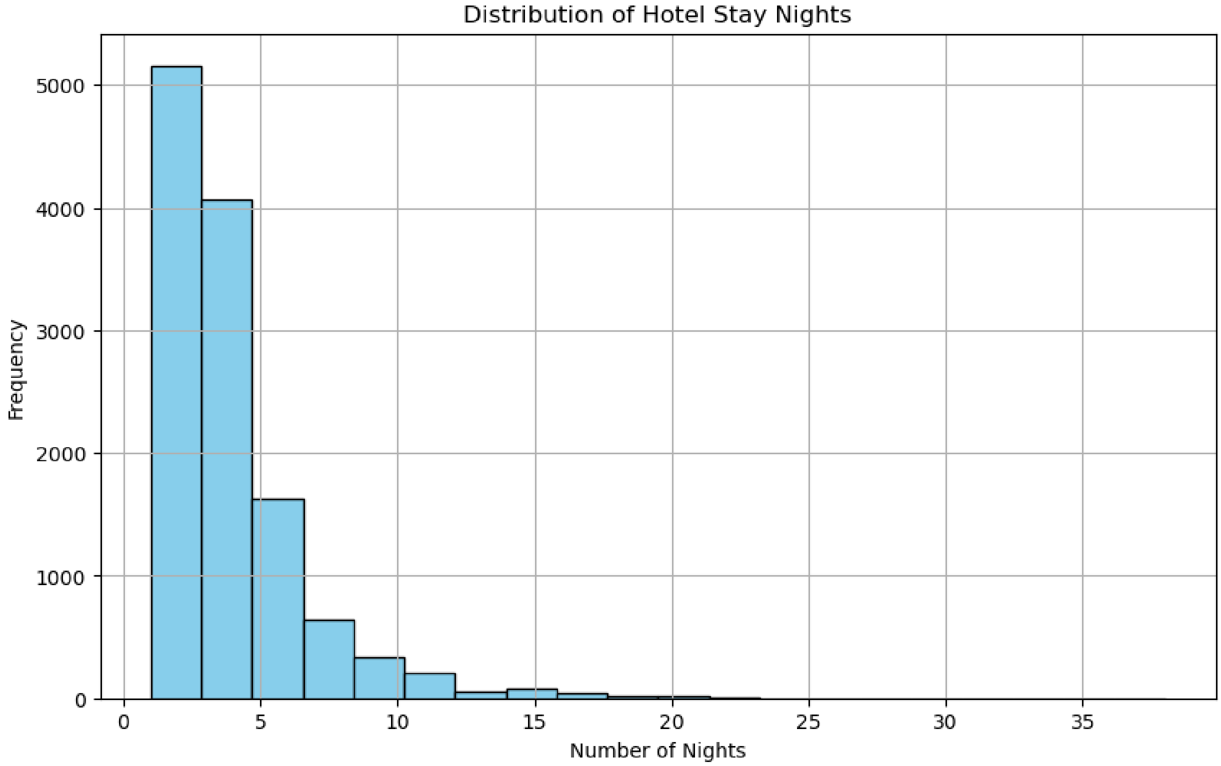
We will need to add the hotel stay data into df_flights_data - we are only missing the check in and check out

While adding this data we will make sure to check for short stays and make them count as 1 night instead of 0. For example, if someone checks in at 1am and then out at 11am it should count as 1 night.

In [42]:
```
hotel_info_query = """
SELECT *
FROM hotels
"""

df_hotel_info = pd.read_sql(hotel_info_query, engine)

# Merge hotel stay data with flight data based on trip_id
df_flight_data = pd.merge(df_flight_data, df_hotel_info, on='trip_id', how='left')

# Calculate hotel stay nights
df_flight_data['check_in_date'] = df_flight_data['check_in_time'].dt.date
df_flight_data['check_out_date'] = df_flight_data['check_out_time'].dt.date

def calculate_hotel_stay_nights(row):
    if row['check_out_date'] < row['check_in_date']:
        return 1
    elif row['check_out_date'] == row['check_in_date']:
        if row['check_out_time'] > row['check_in_time']:
            return 1
    else:
        return (row['check_out_date'] - row['check_in_date']).days

df_flight_data['hotel_stay_nights'] = df_flight_data.apply(calculate_hotel_stay_nights

# Print the stats
print(df_flight_data["hotel_stay_nights"].describe())
```

```
count    12253.000000
mean         3.648494
std          2.872339
min          1.000000
25%          2.000000
50%          3.000000
75%          4.000000
max         38.000000
Name: hotel_stay_nights, dtype: float64
```

In [43]:
```
# Create a histogram of hotel stay nights
plt.figure(figsize=(10, 6))
plt.hist(df_flight_data["hotel_stay_nights"], bins=20, color='skyblue', edgecolor='bla
plt.title("Distribution of Hotel Stay Nights")
plt.xlabel("Number of Nights")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()
```



In [44]:
```
# Define the minimum hotel stay nights for the segment

for x in range (4,10):
    min_stay_nights = x

    # Filter the DataFrame to create the segment
    hotel_stay_segment = df_flight_data[df_flight_data["hotel_stay_nights"] >= min_sta

    # Get the user IDs from the segment
    user_ids_segment = hotel_stay_segment["user_id"].unique()

    # Print the number of users in the segment
    print(f"Number of users with at least {min_stay_nights} nights stay:", len(user_ic
```

```
Number of users with at least 4 nights stay: 3203
Number of users with at least 5 nights stay: 2363
Number of users with at least 6 nights stay: 1714
Number of users with at least 7 nights stay: 1258
Number of users with at least 8 nights stay: 944
Number of users with at least 9 nights stay: 723
```

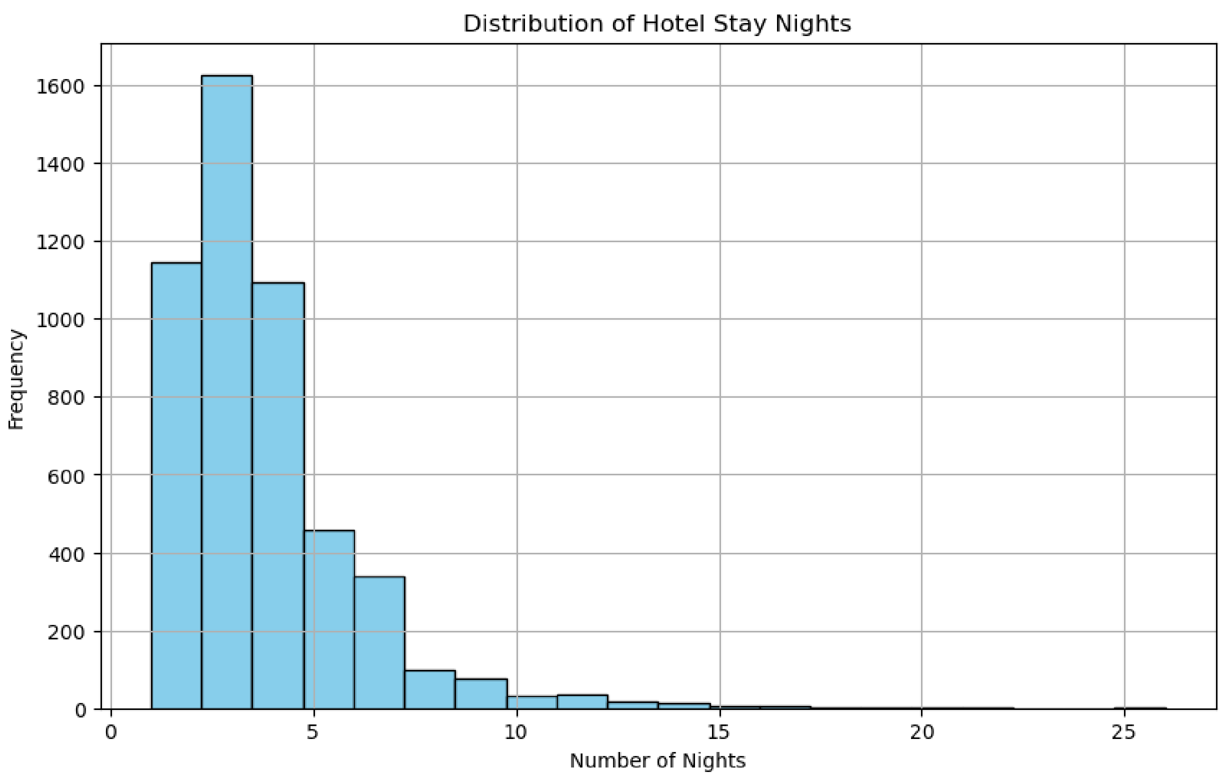
I want to change this a bit. Let's instead look at unique users and their average stay

In [45]:
```
# Group the DataFrame by user ID
user_group = df_flight_data.groupby('user_id')

# Calculate the average hotel stay length and number of unique trips for each user
user_stats = user_group.agg(
    avg_hotel_stay_length=pd.NamedAgg(column='hotel_stay_nights', aggfunc='mean'),
    num_unique_trips=pd.NamedAgg(column='trip_id', aggfunc='nunique')
).reset_index()

# Display the resulting DataFrame
print(user_stats["avg_hotel_stay_length"].describe())
```

```
count    4958.000000
mean        3.718694
std         2.240779
min         1.000000
25%         2.333333
50%         3.000000
75%         4.500000
max        26.000000
Name: avg_hotel_stay_length, dtype: float64
```

In [46]:

```
# Create a histogram of hotel stay nights
plt.figure(figsize=(10, 6))
plt.hist(user_stats["avg_hotel_stay_length"], bins=20, color='skyblue', edgecolor='bla
plt.title("Distribution of Hotel Stay Nights")
plt.xlabel("Number of Nights")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()
```



In [47]:

```
# Define the minimum hotel stay nights for the segment
for x in range(4, 10):
    min_stay_nights = x

    # Filter the DataFrame to create the segment
    hotel_stay_segment = user_stats[user_stats["avg_hotel_stay_length"] >= min_stay_ni

    # Get the user IDs from the segment
    user_ids_segment = hotel_stay_segment["user_id"].unique()

    # Print the number of users in the segment
    print(f"Number of users with at least {min_stay_nights} nights stay:", len(user_ic
```

```
Number of users with at least 4 nights stay: 1824
Number of users with at least 5 nights stay: 1062
Number of users with at least 6 nights stay: 636
Number of users with at least 7 nights stay: 385
Number of users with at least 8 nights stay: 262
Number of users with at least 9 nights stay: 175
```

With this, I would go with users with at least a 4 night stay, but I would take this segment after others as there will be duplication.

Free Checked Bag perk

We have already established that most people check a bag, with the mean for checked bags per flight is 1.053.

```
count    5206.000000
mean        1.053488
std         0.179770
min         1.000000
25%         1.000000
50%         1.000000
75%         1.000000
max         2.000000
Name: avg_checked_bags_per_trip, dtype: float64
```

We shall investigate:

- Families
- Leisure Travelers
- Group Travelers

Familes

- married will be true (I understand non-traditional families may exist but we have limited data)
- has_children will be true
- seats will be greater than 3

After some though, I am going to make this group just married and has childred. We already know they are interested in travel.

Leisure Travelers

We will revisit this if needed. There was no way to get data from an API that could definitively identify a city as a ski or golf destinaton.

Group Travelers

Minimum of 3 hotel rooms booked OR minimum of 6 seats booked.

We are goint to identify places of interest to people traveling for recreation and would be likely to bring their equipment with them - ski resorts and golf resorts

This will be done with the Google Places API. We are looking at Ski Resorts within 200km of the destination and Golf Resorts within 50km. I chose 200km for Ski resorts because Lake Louise is just under 200km from Calgary with Calgary being the closest airport and Banff/Lake Louise is a major destination.

In [48]:
```
unique_destinations = df_flight_data[['destination_city', 'destination_country', 'dest
```

In [49]:
```
# Define Google Places API endpoint
PLACES_API_URL = "https://maps.googleapis.com/maps/api/place/nearbysearch/json"

def get_nearby_places(location, radius, place_type):
    params = {
        "location": location,
        "radius": radius,
        "type": place_type,
        "key": gmaps_key
    }
    response = requests.get(PLACES_API_URL, params=params)
    data = response.json()
    return data.get("results", [])

def identify_resorts(places):
    major_resorts = []
    for place in places:
        if (place.get("user_ratings_total", 0) >= 500) and (place.get("rating", 0) >=
            major_resorts.append(place["name"])
    return major_resorts


# Initialize new DataFrame to store resort information
resorts_data = []

# Loop through unique destinations and retrieve resort information
for index, row in unique_destinations.iterrows():
    city = (row["destination_lat"], row["destination_lon"])
    ski_resorts = get_nearby_places(city, 200000, "point_of_interest")
    golf_resorts = get_nearby_places(city, 50000, "golf_course")
    major_ski_resorts = identify_resorts(ski_resorts)
    major_golf_resorts = identify_resorts(golf_resorts)
    resorts_data.append({
        "destination_city": row["destination_city"],
        "destination_country": row["destination_country"],
        "major_ski_resorts": major_ski_resorts,
        "major_golf_resorts": major_golf_resorts
    })

# Create a DataFrame from the resorts_data list
df_resorts = pd.DataFrame(resorts_data)

# Merge the resort information back into the original DataFrame
```

```
df_flight_data = df_flight_data.merge(df_resorts, on=["destination_city", "destinatior

# Display the updated DataFrame
print(df_flight_data.head())
```

```
   user_id  flight_cost                                 trip_id  \
0    23557        98.46   23557-127abde78c8f4352b7a84483d2576c25
1   433080       436.10  433080-ae391e65085b425d93c663f33c9eb552
2   447737       299.93  447737-2db2e8c63d5f4390889eae142362591c
3   407434       143.18  407434-94fb8965a20e4cf0bd03c39f87135991
4   465568       577.43  465568-ce24a24881bc47eea5cb86a1bb87d027

   flight_discount  flight_discount_percent  seats_booked  hotel_discount  \
0             True                     0.15             1           False
1            False                      NaN             1            True
2            False                      NaN             1           False
3            False                      NaN             1           False
4             True                     0.05             1           False

   hotel_discount_percent  rooms_booked  hotel_cost_per_room  ... nights  \
0                     NaN           1.0                118.0  ...    2.0
1                    0.15           1.0               1190.0  ...    7.0
2                     NaN           1.0                187.0  ...    3.0
3                     NaN           NaN                  NaN  ...    NaN
4                     NaN           1.0                118.0  ...   10.0

  rooms           check_in_time      check_out_time  hotel_per_room_usd  \
0   1.0 2021-07-30 12:42:32.580 2021-08-02 11:00:00               118.0
1   1.0 2022-12-05 15:39:27.540 2022-12-13 11:00:00              1190.0
2   1.0 2022-12-15 11:05:33.585 2022-12-18 11:00:00               187.0
3   NaN                     NaT                 NaT                 NaN
4   1.0 2022-12-17 17:49:39.900 2022-12-28 11:00:00               118.0

   check_in_date  check_out_date  hotel_stay_nights major_ski_resorts  \
0     2021-07-30      2021-08-02                3.0                []
1     2022-12-05      2022-12-13                8.0                []
2     2022-12-15      2022-12-18                3.0                []
3            NaT             NaT                NaN                []
4     2022-12-17      2022-12-28               11.0                []

  major_golf_resorts
0                 []
1                 []
2                 []
3                 []
4                 []

[5 rows x 42 columns]
```

In [50]:

```
# Filter rows with non-empty major_ski_resorts or major_golf_resorts columns
cities_with_ski_resorts = df_flight_data[df_flight_data['major_ski_resorts'].apply(ler
cities_with_golf_resorts = df_flight_data[df_flight_data['major_golf_resorts'].apply(]

# Print the list of cities with major ski resorts
print("Cities with major ski resorts nearby:")
for city, country in zip(cities_with_ski_resorts['destination_city'], cities_with_ski_
    print(f"{city}, {country}")

# Print the list of cities with major golf resorts
print("Cities with major golf resorts nearby:")
```

```
for city, country in zip(cities_with_golf_resorts['destination_city'], cities_with_gol
    print(f"{city}, {country}")
```

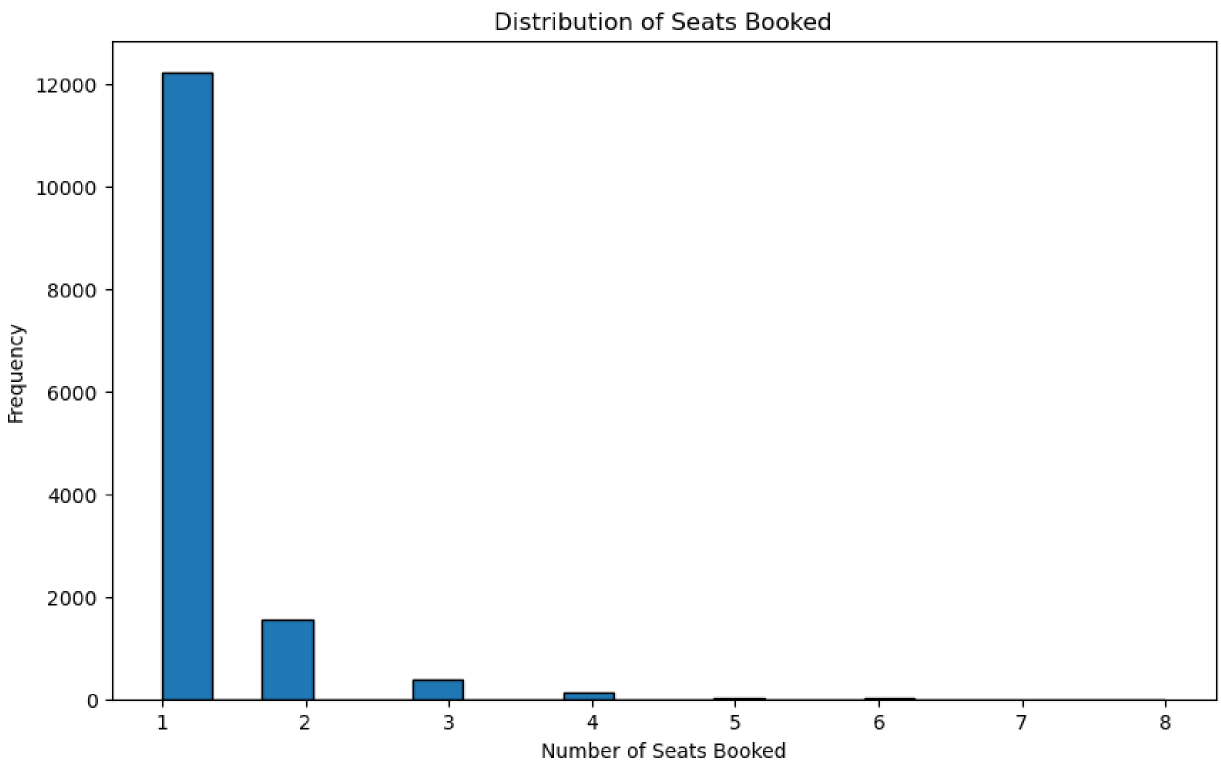
```
Cities with major ski resorts nearby:
Cities with major golf resorts nearby:
```

There is not a simple way to determine ski and golf resorts within the code. I will revisit this if I need and create a list from the destinations we have.

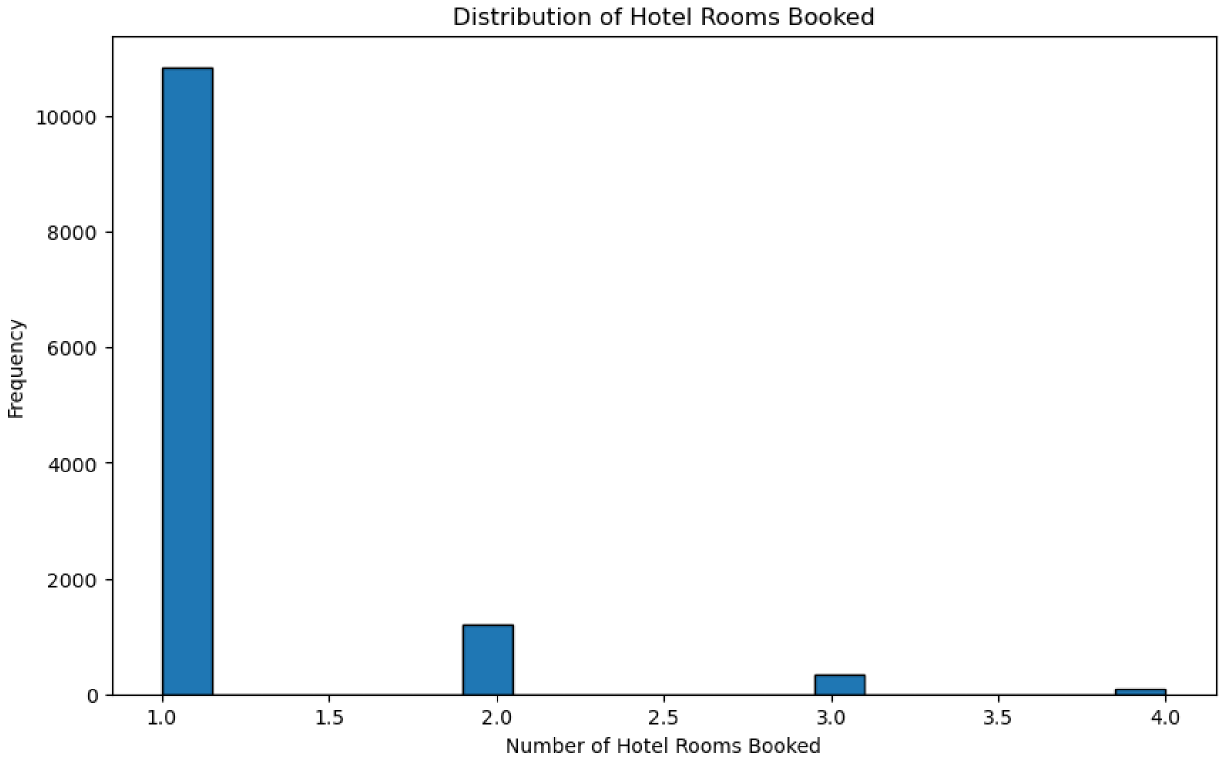
At this point we will continue on the other exploration

In [51]:
```
### We will see what the data looks like for hotel room and flight seats booked to det
```

In [52]:
```
# Create a histogram for the "seats_booked" column
plt.figure(figsize=(10, 6))
plt.hist(df_flight_data['seats_booked'], bins=20, edgecolor='black')
plt.title('Distribution of Seats Booked')
plt.xlabel('Number of Seats Booked')
plt.ylabel('Frequency')
plt.show()

print(df_flight_data['seats_booked'].describe())

# Create a histogram for the "rooms_booked" column
plt.figure(figsize=(10, 6))
plt.hist(df_flight_data['rooms_booked'], bins=20, edgecolor='black')
plt.title('Distribution of Hotel Rooms Booked')
plt.xlabel('Number of Hotel Rooms Booked')
plt.ylabel('Frequency')
plt.show()

print(df_flight_data['rooms_booked'].describe())
```



```
count    14332.000000
mean         1.200740
std          0.556078
min          1.000000
25%          1.000000
50%          1.000000
75%          1.000000
max          8.000000
Name: seats_booked, dtype: float64
```




```
count    12459.000000
mean         1.170399
std          0.487543
min          1.000000
25%          1.000000
50%          1.000000
75%          1.000000
max          4.000000
Name: rooms_booked, dtype: float64
```

In [53]:

```
# Filter the DataFrame for users who booked 6 or more seats OR 3 or more hotel rooms
combined_group_travelers = df_flight_data[(df_flight_data['seats_booked'] >= 6) | (df_

# Create a set of unique user_id values for combined group travelers
combined_group_travelers_user_ids = set(combined_group_travelers['user_id'])

# Display the list of user_id values
print(len(combined_group_travelers_user_ids))
```

413

413 users meating the Group Traveler criteria

In [54]:

```
# Get all users that are married and have children
family_users_query = f"""
```

```
{cohort_filter}
SELECT u.user_id AS user_id
FROM users as U
INNER JOIN CohortUsers AS cu ON u.user_id = cu.user_id
WHERE u.married AND u.has_children
"""

df_family_users = pd.read_sql(family_users_query, engine)

print(df_family_users)
```

```
      user_id
0      513470
1      513402
2      456663
3      513425
4      510733
...       ...
1079   561896
1080   582011
1081   500972
1082   653640
1083   584459

[1084 rows x 1 columns]
```

Segmentation Summary

Free Hotel Meal

Budget-Conscious Travelers

For this group we identified users that booked more than 90 days in advance. Total in Segment: 512

Overnight Hotel Stays

This is anyone who had booked 2 or more 1 night hotel stays. Total in Segment: 207

Total for Free Hotel Meal: 707

Exclusive Discounts

Top 15% Of Customers by Total Spend

There are 781 in this group

1 Night Free Hotel With Flight

Long Stay Guests

Hotel guests with 4 or more nights booked. There are 1824 in this group.

Complementary Lounge Access

International Travelers

There are 520 users who booked 3 or more international trips

No Cancellation Fees

Anyone who cancelled. There are 620 in this group, however, all 620 rebooked

Free Checked Bag

Familes

Any user that is married and has children. There are 1084 in this group. *We can reduce this group size by taking those who have booked multiple flights*

Group Travelers

There are 413 users who booked a trip for 6 or more people or with 3 or more hotel rooms.

Remaining Travelers

As almost every user had checked bags this is a good perk for anyone who doesn't fit in other segments.

In [55]:
```
# Free Hotel Meal Segment List

# Extract user IDs from df_overnight_hotel_stays
single_night_stay_users = df_overnight_hotel_stays.loc[df_overnight_hotel_stays['num_h

# Extract user IDs from bargain_hunters_df
bargin_hunter_ids = bargain_hunters_df['user_id'].tolist()

# Combine the lists
free_hotel_meal_segment_ids = single_night_stay_users + bargin_hunter_ids

# Convert to set to remove duplicates
free_hotel_meal_segment_ids = set(free_hotel_meal_segment_ids)

df_free_hotel_segment = pd.DataFrame({'user_id': list(free_hotel_meal_segment_ids), 's
```

In [56]:
```
# Exclusive Discounts Segment List

# Convert high_value_customers to a set for faster lookup
high_value_customers_set = set(high_value_customers)

df_exclusive_discount_segment = pd.DataFrame({'user_id': list(high_value_customers_set
```

In [57]:
```
# Free Hotel Night with Flight

free_hotel_stay_users = df_flight_data.loc[df_flight_data['hotel_stay_nights'] >= 4, '

# Convert the list to a set to remove duplicates
free_hotel_stay_users = set(free_hotel_stay_users)
```

```
df_free_hotel_night_segment = pd.DataFrame({'user_id': list(free_hotel_stay_users), 's
```

In [58]:
```
# Lounge Access

lounge_access_ids = set(users_with_3_or_more_flights.index)

df_lounge_access_segment = pd.DataFrame({'user_id': list(lounge_access_ids), 'segment'
```

In [59]:
```
# Free Checked Bag - Familes and Groups

group_travelers_ids = set(combined_group_travelers_user_ids)

# Get unique user_ids from the 'user_id' column and convert to a set
family_ids = set(df_family_users['user_id'])

free_checked_bag_set = group_travelers_ids.union(family_ids)


df_free_checked_bag_segment = pd.DataFrame({'user_id': list(free_checked_bag_set), 'se
```

In [60]:
```
# Free Cancellation

free_cancellation_ids = set(df_cancellation['user_id'])

df_free_cancellation_segment = pd.DataFrame({'user_id': list(free_cancellation_ids), '
```

In [103…
```
# Create the Marketing Segments dataframe

df_marketing_segments = pd.DataFrame(columns=['user_id', 'segment'])

# List of DataFrames in the order in which they should be checked for duplicates
df_list = [
    df_exclusive_discount_segment,
    df_free_hotel_segment,
    df_free_hotel_night_segment,
    df_lounge_access_segment,
    df_free_checked_bag_segment,
    df_free_cancellation_segment
]

# Loop through each DataFrame to add its user_ids to the final DataFrame, removing dup
for df in df_list:
    # Remove the user_ids that are already in the final DataFrame
    df = df[~df['user_id'].isin(df_marketing_segments['user_id'])]
    # Concatenate the filtered DataFrame to the final DataFrame
    df_marketing_segments = pd.concat([df_marketing_segments, df], ignore_index=True)

print(df_marketing_segments)
```

```
     user_id             segment
0     563201  Exclusive Discounts
1     579589  Exclusive Discounts
2     651269  Exclusive Discounts
3     538638  Exclusive Discounts
4     483345  Exclusive Discounts
...      ...                  ...
4216  532381    Free Cancellation
4217  534445    Free Cancellation
4218  550830    Free Cancellation
4219  585650    Free Cancellation
4220  518135    Free Cancellation

[4221 rows x 2 columns]
```

In [104…

```
def calculate_user_metrics(df_users, engine):
    user_ids = df_users['user_id'].tolist()
    user_ids_str = ', '.join(map(str, user_ids))

    # Step 1: Fetch data
    query = f"""
    SELECT s.user_id, s.trip_id,
           COALESCE(f.base_fare_usd, 0) as base_fare_usd,
           COALESCE(f.seats, 0) as seats,
           COALESCE(s.flight_discount_amount, 0) as flight_discount_amount,
           COALESCE(h.hotel_per_room_usd, 0) as hotel_per_room_usd,
           COALESCE(h.rooms, 0) as rooms,
           COALESCE(s.hotel_discount_amount, 0) as hotel_discount_amount
    FROM sessions s
    LEFT JOIN flights f ON s.trip_id = f.trip_id
    LEFT JOIN hotels h ON s.trip_id = h.trip_id
    WHERE s.user_id IN ({user_ids_str})
    """
    df_trips = pd.read_sql(query, engine)

    # Step 2: Compute cost for each booking (trip)
    df_trips['flight_cost'] = df_trips['base_fare_usd'] * df_trips['seats'] * (1 - df_
    df_trips['hotel_cost'] = df_trips['hotel_per_room_usd'] * df_trips['rooms'] * (1 -
    df_trips['total_trip_cost'] = round(df_trips['flight_cost'].fillna(0) + df_trips['

    # Step 3: Aggregate Data by User
    total_value_per_user = df_trips.groupby('user_id')['total_trip_cost'].sum()
    number_of_bookings_per_user = df_trips.groupby('user_id')['trip_id'].nunique()
    average_booking_value_per_user = total_value_per_user / number_of_bookings_per_use

    # Combine into a single DataFrame
    df_user_metrics = pd.DataFrame({
        'total_value': total_value_per_user,
        'number_of_bookings': number_of_bookings_per_user,
        'average_booking_value': average_booking_value_per_user
    }).reset_index()

    # Merge metrics back into the provided DataFrame
    df_result = pd.merge(df_users, df_user_metrics, on='user_id', how='left')

    return df_result
```

In [105…

```
df_marketing_segments = calculate_user_metrics(df_marketing_segments, engine)
```

In [106…
```
print(df_marketing_segments['total_value'].describe())
```

```
count      4221.000000
mean       3644.458969
std        8817.791517
min           0.000000
25%        1225.710000
50%        2030.580000
75%        3544.390000
max      208658.110000
Name: total_value, dtype: float64
```

In [107…
```
df_marketing_segments.to_csv('final_marketing_segments.csv', index=False)
```

In [108…
```
# Get all Cohort user ids

cohort_users_query = f"""
{cohort_filter}
SELECT u.user_id AS user_id
FROM users AS u
JOIN CohortUsers AS cu ON u.user_id = cu.user_id;

"""

df_cohort_users = pd.read_sql(cohort_users_query, engine)
```

In [109…
```
# Find the users in the cohort DataFrame who are NOT in the segmented DataFrame
df_unsegmented_users = df_cohort_users.loc[
    ~df_cohort_users['user_id'].isin(df_marketing_segments['user_id'])
].copy()
df_unsegmented_users['segment'] = 'Free Bag - Remaining Users'
```

In [110…
```
# Assuming df_cohort_users and engine are already defined
df_unsegmented_users = calculate_user_metrics(df_unsegmented_users, engine)
print(df_unsegmented_users['total_value'].describe())
```

```
count    1777.000000
mean      842.321880
std       854.804658
min         0.000000
25%       111.890000
50%       633.460000
75%      1268.550000
max      4059.760000
Name: total_value, dtype: float64
```

In [111…
```
df_unsegmented_users.to_csv('final_marketing_remaining_users.csv', index=False)
```

In [112…
```
# Get a summary of each segment's spend

print (df_marketing_segments.groupby('segment')['total_value'].describe())
```

```
                        count          mean           std      min        25%  \
segment
Exclusive Discounts   781.0  11511.337286  18426.652169  4014.66  5021.4400
Free Cancellation      66.0   1783.226667   1191.562367    98.00  1044.3000
Free Checked Bag      473.0   1085.762896   1020.325213     0.00   334.3000
Free Hotel Meal       416.0   2494.670216   1359.660737   107.27  1467.3675
Free Hotel Night     2389.0   1886.448326    935.847381   122.17  1187.0800
Lounge Access          96.0   2261.877292    783.159839   768.55  1628.7375

                          50%         75%        max
segment
Exclusive Discounts  6907.050  10646.6900  208658.11
Free Cancellation    1526.000   2373.9175    6915.29
Free Checked Bag      819.940   1521.4600    4434.45
Free Hotel Meal      2279.705   3329.1975    7520.64
Free Hotel Night     1761.590   2488.8200    8161.38
Lounge Access        2251.315   2707.9100    3925.99
```

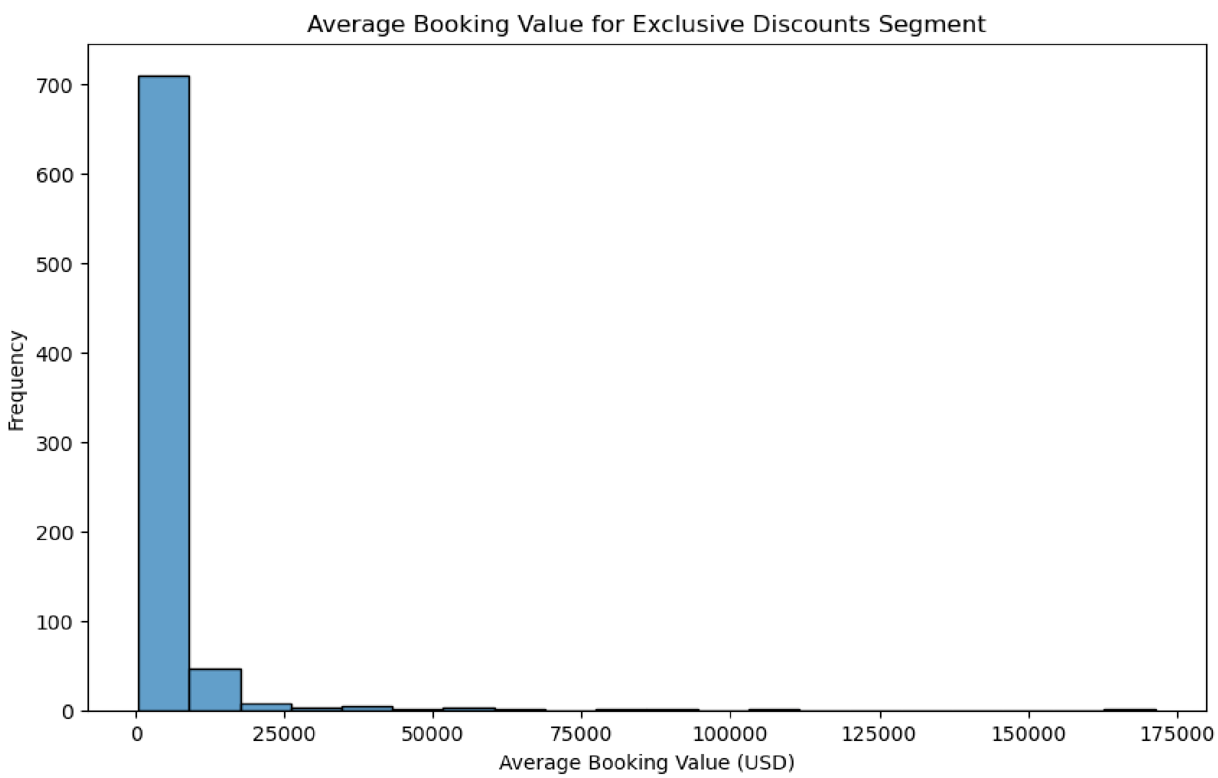
In [148…

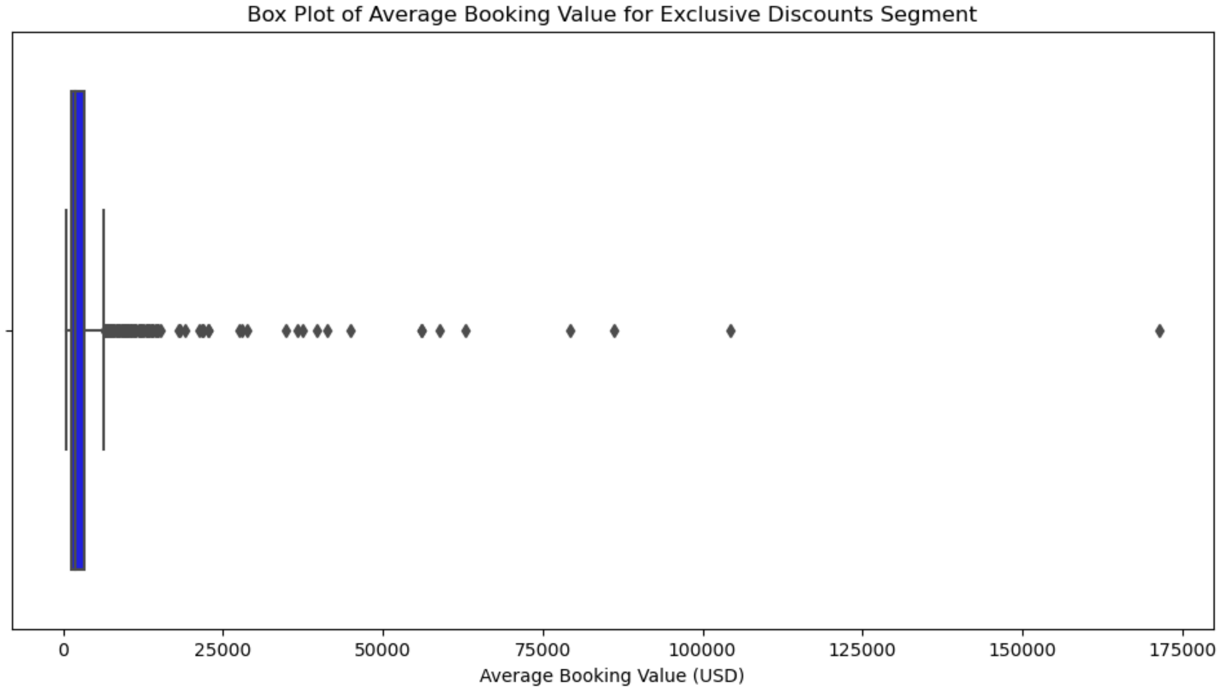
```python
# Functions to create charts
import seaborn as sns

def plot_histogram(df, segment_name, column='average_booking_value'):
    # Filter the DataFrame to only include rows where the segment matches the segment_
    segment_df = df[df['segment'] == segment_name]

    # Create the histogram
    plt.figure(figsize=(10, 6))

    sns.histplot(segment_df[column], bins=20, edgecolor='black', alpha=0.7)
    plt.title(f'{column.replace("_", " ").title()} for {segment_name} Segment')
    plt.xlabel(f'{column.replace("_", " ").title()} (USD)')
    plt.ylabel('Frequency')
    plt.savefig(f'Histogram_{segment_name.replace(" ", "_")}.png', transparent=True, b
    plt.show()

def plot_boxplot(df, segment_name, column='average_booking_value'):
    # Filter the DataFrame to only include rows where the segment matches the segment_
    segment_df = df[df['segment'] == segment_name]

    # Create the box plot
    plt.figure(figsize=(12, 6))
    sns.boxplot(x=segment_df[column], color='blue')
    plt.title(f'Box Plot of {column.replace("_", " ").title()} for {segment_name} Segm
    plt.xlabel(f'{column.replace("_", " ").title()} (USD)')
    plt.savefig(f'BoxPlot_{segment_name.replace(" ", "_")}.png', transparent=True, bbo
    plt.show()

def plot_bar_chart(df, highlight_segment, column='average_booking_value'):
    # Function to assign colors based on segment
    def segment_color(segment):
        if segment == highlight_segment:
            return 'blue'
        else:
            return 'gray'

    # Create a list of unique segments in the same order as they appear in the DataFra
    unique_segments = df['segment'].unique()

    # Create a list of colors based on unique segments
    colors = [segment_color(segment) for segment in unique_segments]
```

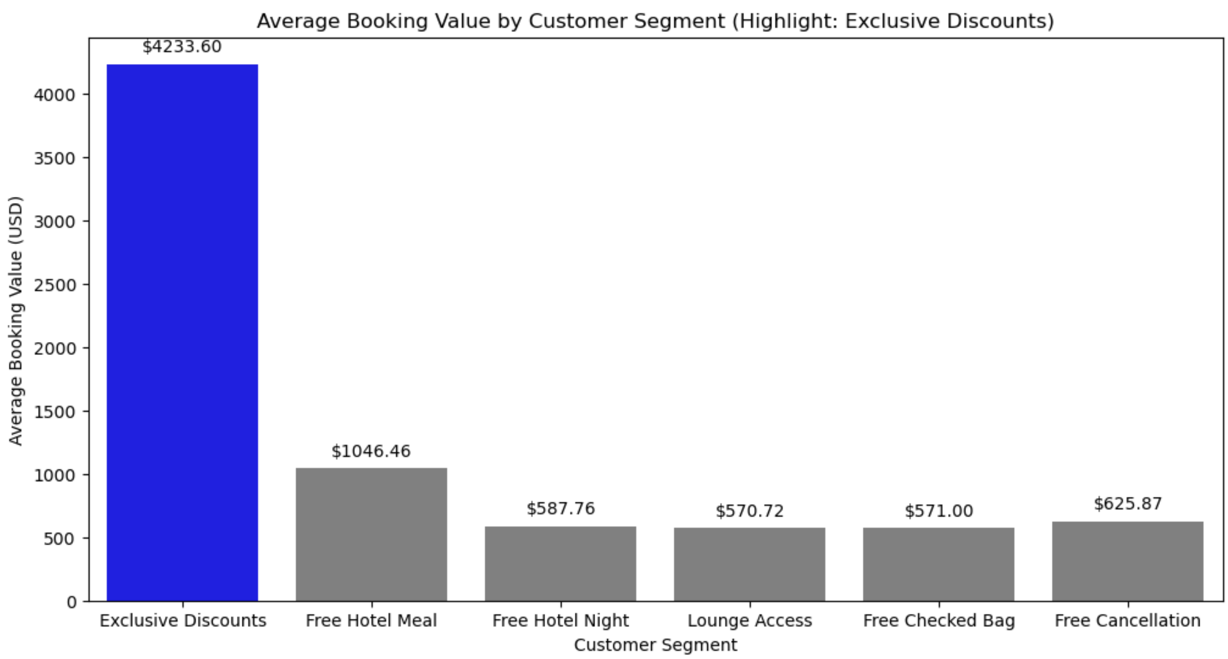
```
    # Create the bar plot
    plt.figure(figsize=(12, 6))
    ax = sns.barplot(x='segment', y=column, data=df, palette=colors, errorbar=None)

    plt.title(f'Average Booking Value by Customer Segment (Highlight: {highlight_segme
    plt.xlabel('Customer Segment')
    plt.ylabel(f'{column.replace("_", " ").title()} (USD)')

    # Annotate each bar with the corresponding value
    for p in ax.patches:
        ax.annotate(f"${p.get_height():.2f}",
                    (p.get_x() + p.get_width() / 2., p.get_height()),
                    ha='center', va='center',
                    xytext=(0, 10),
                    textcoords='offset points')

    plt.savefig(f'BarChart_{highlight_segment.replace(" ", "_")}.png', transparent=Tru
    plt.show()
```

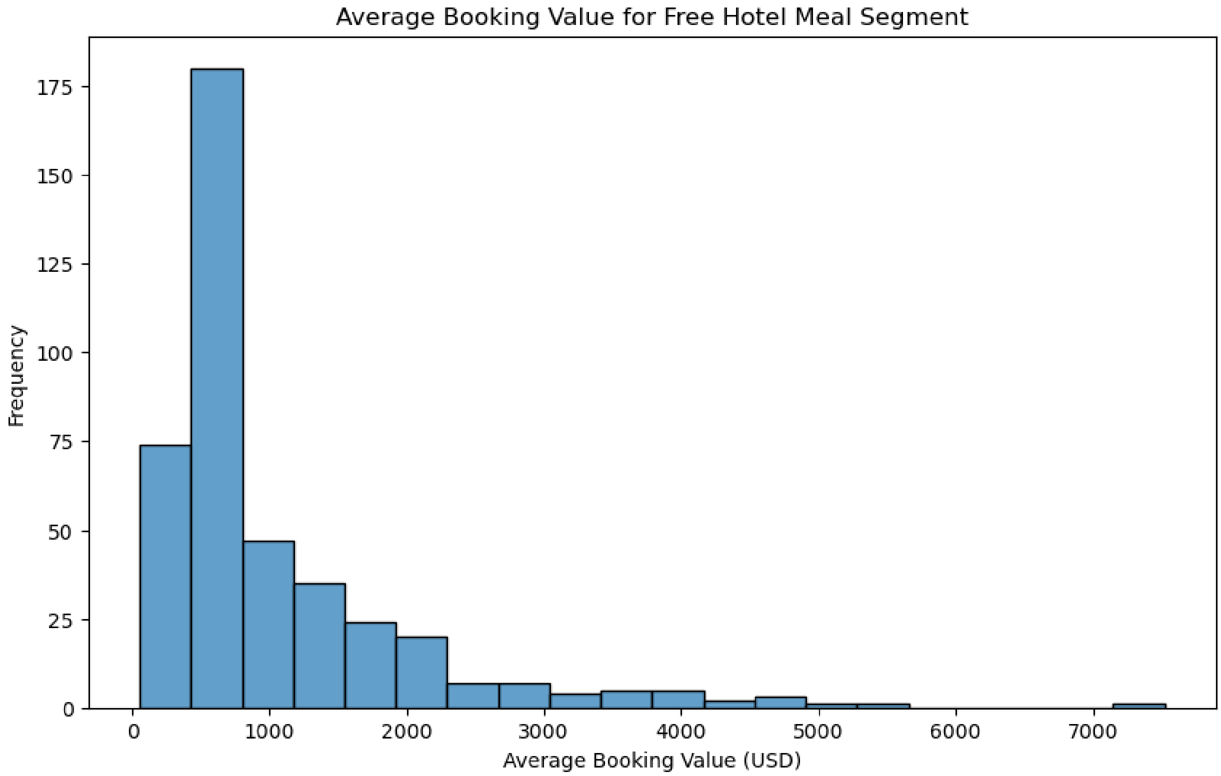
In [149…

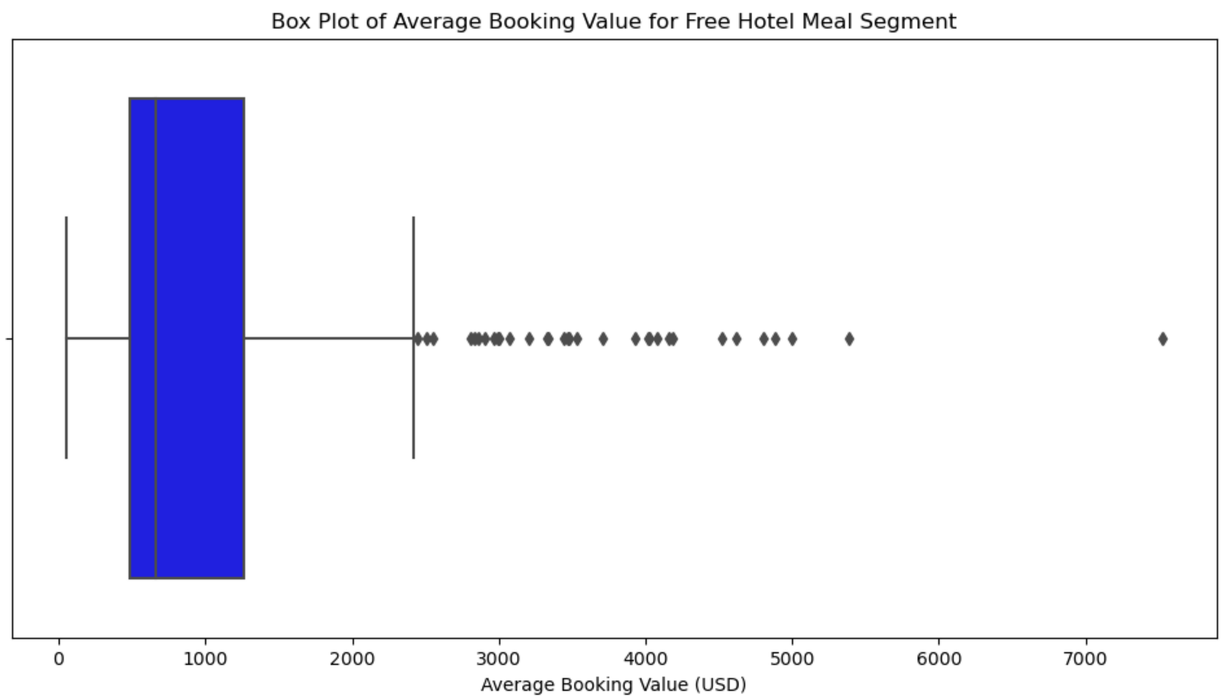
```
# Create the charts for all segments

unique_segments = df_marketing_segments['segment'].unique()

for segment in unique_segments:
    plot_histogram(df_marketing_segments, segment)
    plot_boxplot(df_marketing_segments, segment)
    plot_bar_chart(df_marketing_segments, segment)
```

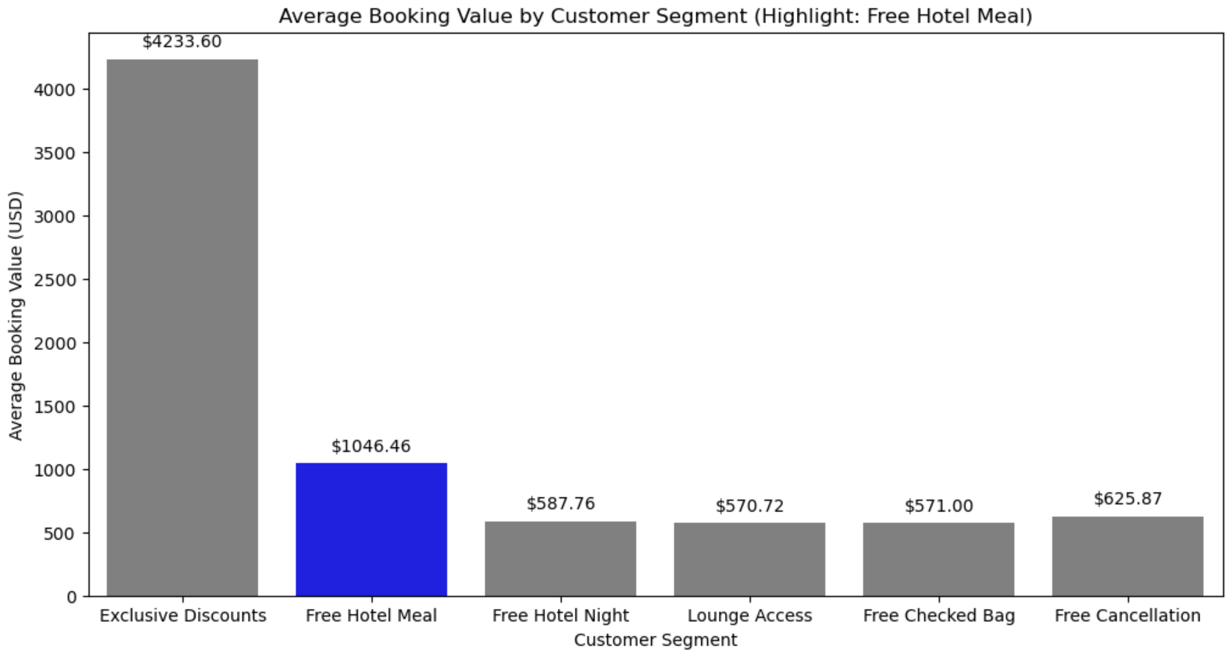


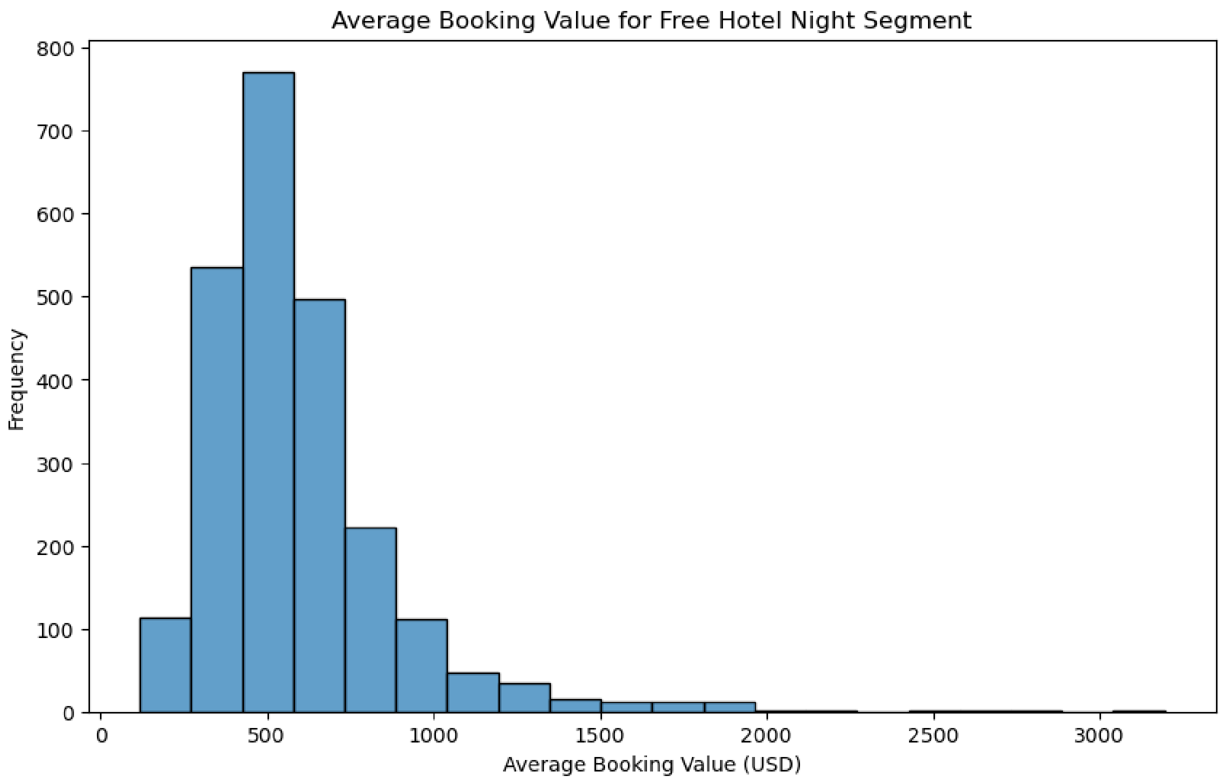

Box Plot of Average Booking Value for Exclusive Discounts Segment
0
25000
50000
75000
100000
125000
150000
175000
Average Booking Value (USD)

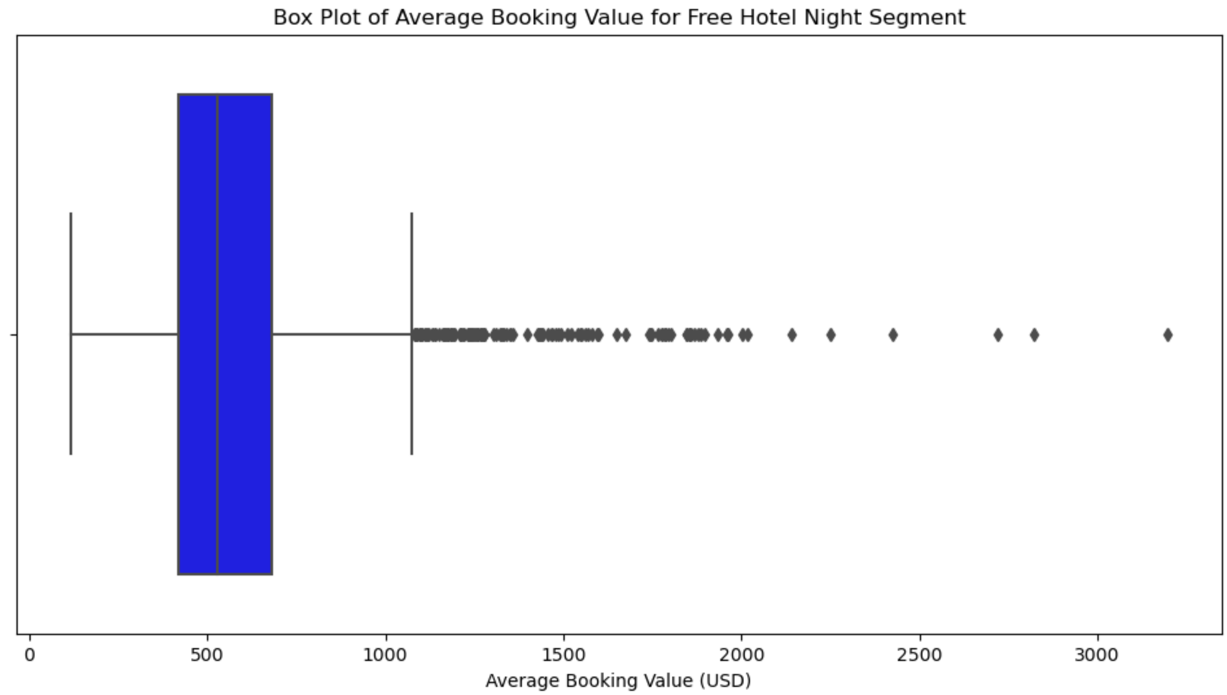

Average Booking Value by Customer Segment (Highlight: Exclusive Discounts)
$4233.60
$1046.46
$587.76
$570.72
$571.00
$625.87
0
500
1000
1500
2000
2500
3000
3500
4000
Average Booking Value (USD)
Exclusive Discounts
Free Hotel Meal
Free Hotel Night
Lounge Access
Free Checked Bag
Free Cancellation
Customer Segment

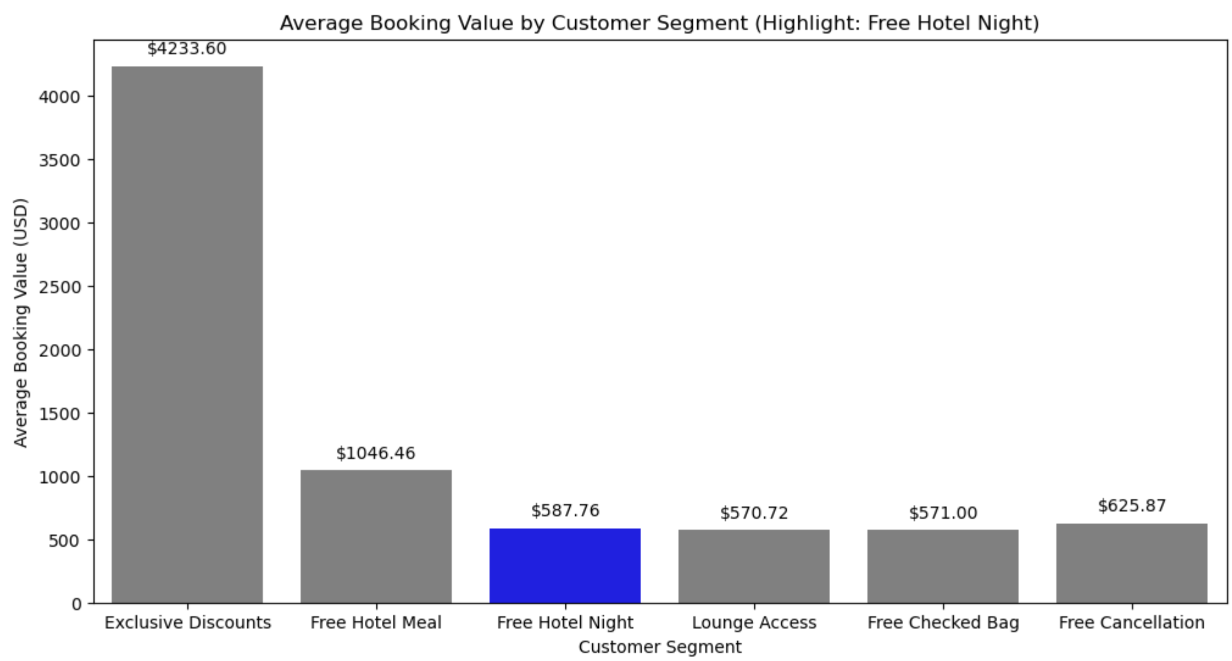

Average Booking Value for Free Hotel Meal Segment
Frequency
0
25
50
75
100
125
150
175
0
1000
2000
3000
4000
5000
6000
7000
Average Booking Value (USD)

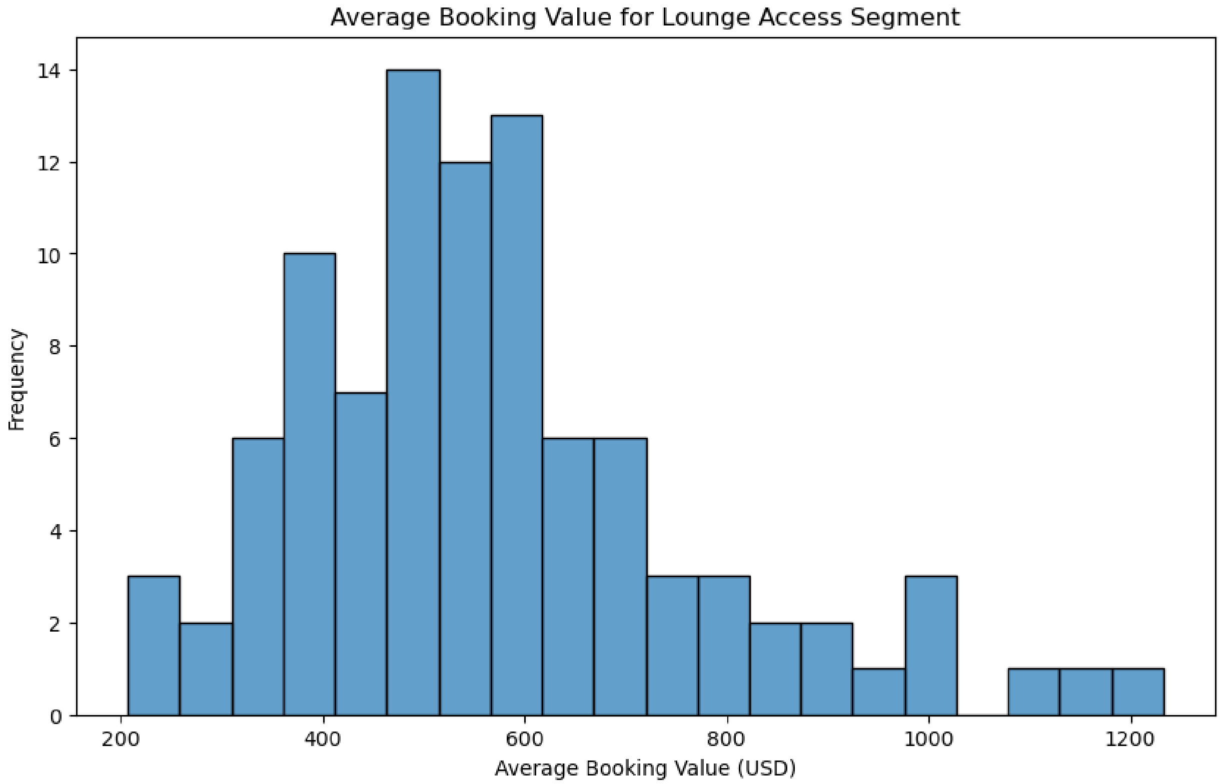

Box Plot of Average Booking Value for Free Hotel Meal Segment
0
1000
2000
3000
4000
5000
6000
7000
Average Booking Value (USD)

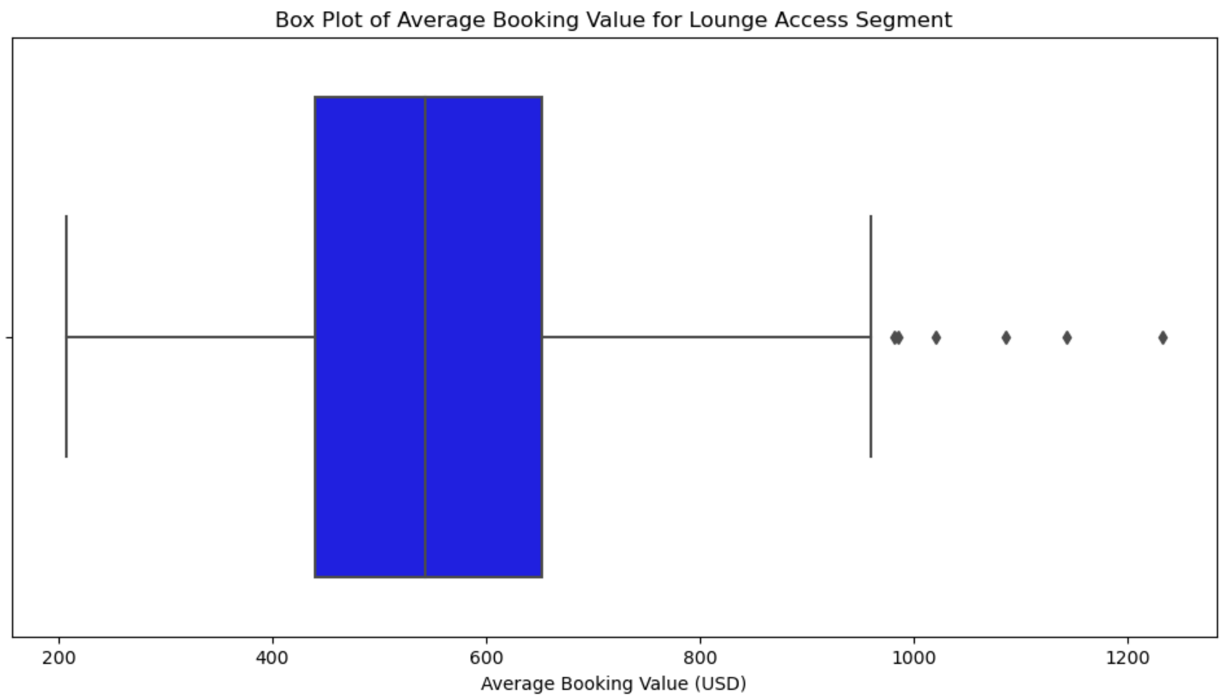

Average Booking Value by Customer Segment (Highlight: Free Hotel Meal)
$4233.60
$1046.46
$587.76
$570.72
$571.00
$625.87
Average Booking Value (USD)
0
500
1000
1500
2000
2500
3000
3500
4000
Exclusive Discounts
Free Hotel Meal
Free Hotel Night
Lounge Access
Free Checked Bag
Free Cancellation
Customer Segment

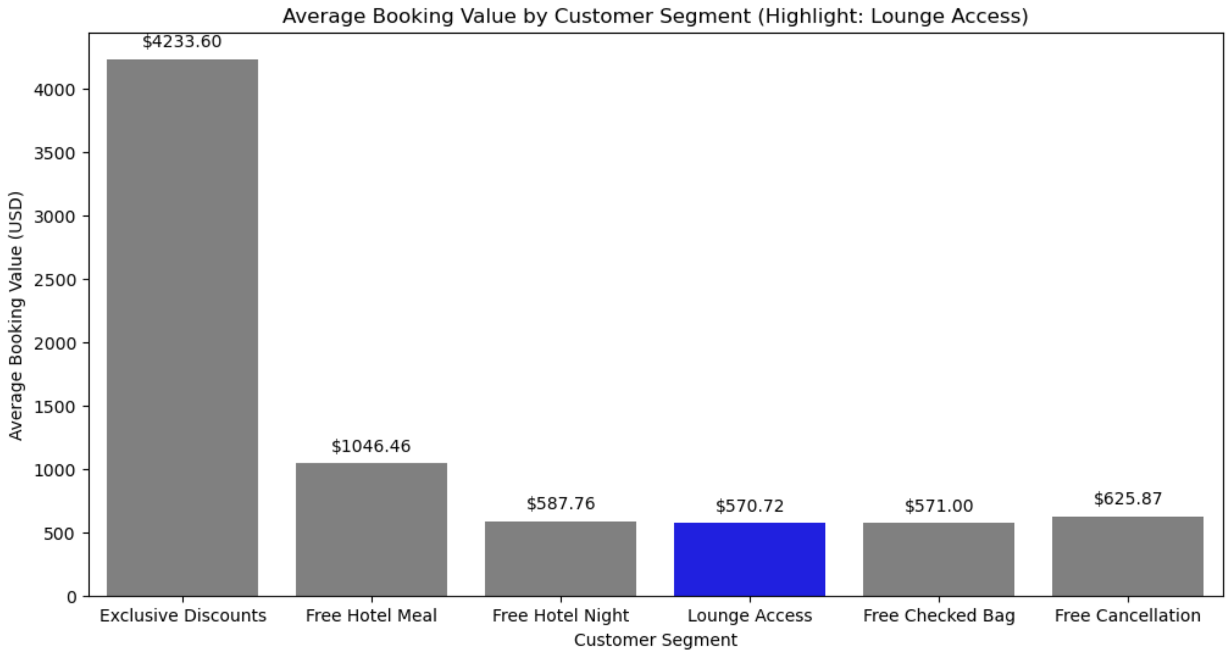

Average Booking Value for Free Hotel Night Segment
Frequency
0
100
200
300
400
500
600
700
800
0
500
1000
1500
2000
2500
3000
Average Booking Value (USD)

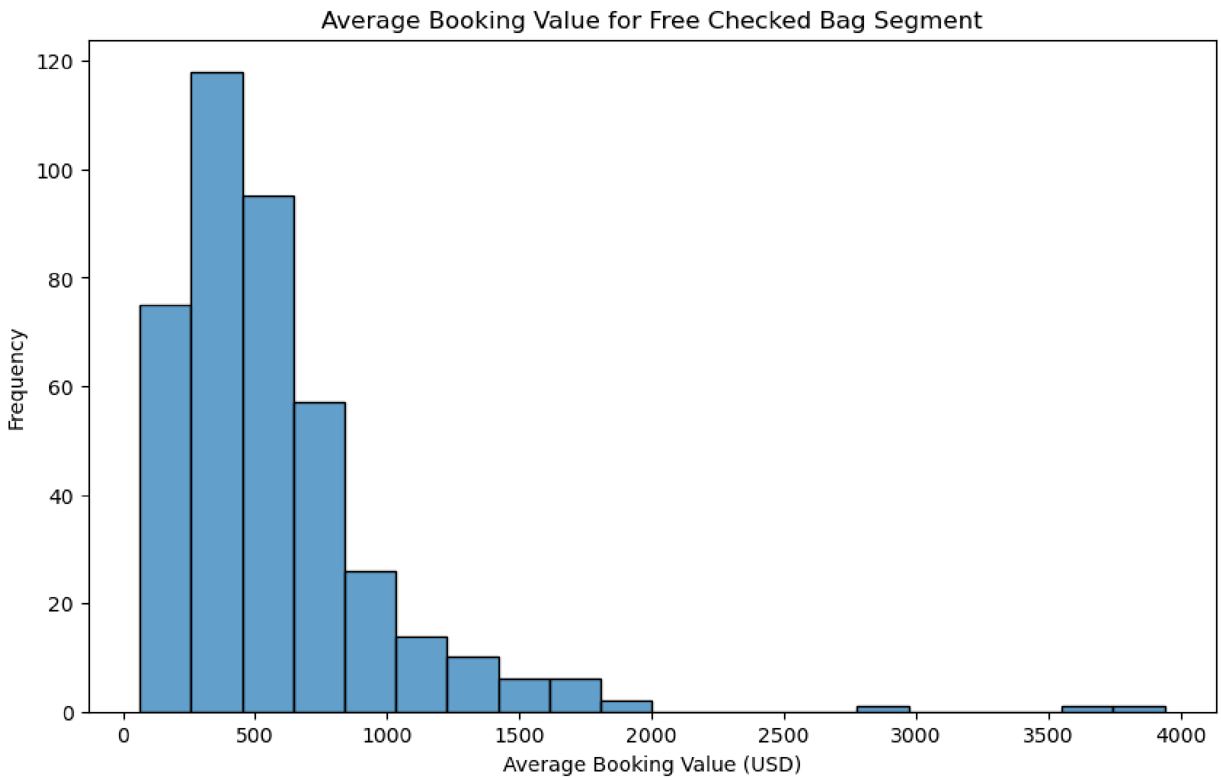

Box Plot of Average Booking Value for Free Hotel Night Segment
0
500
1000
1500
2000
2500
3000
Average Booking Value (USD)

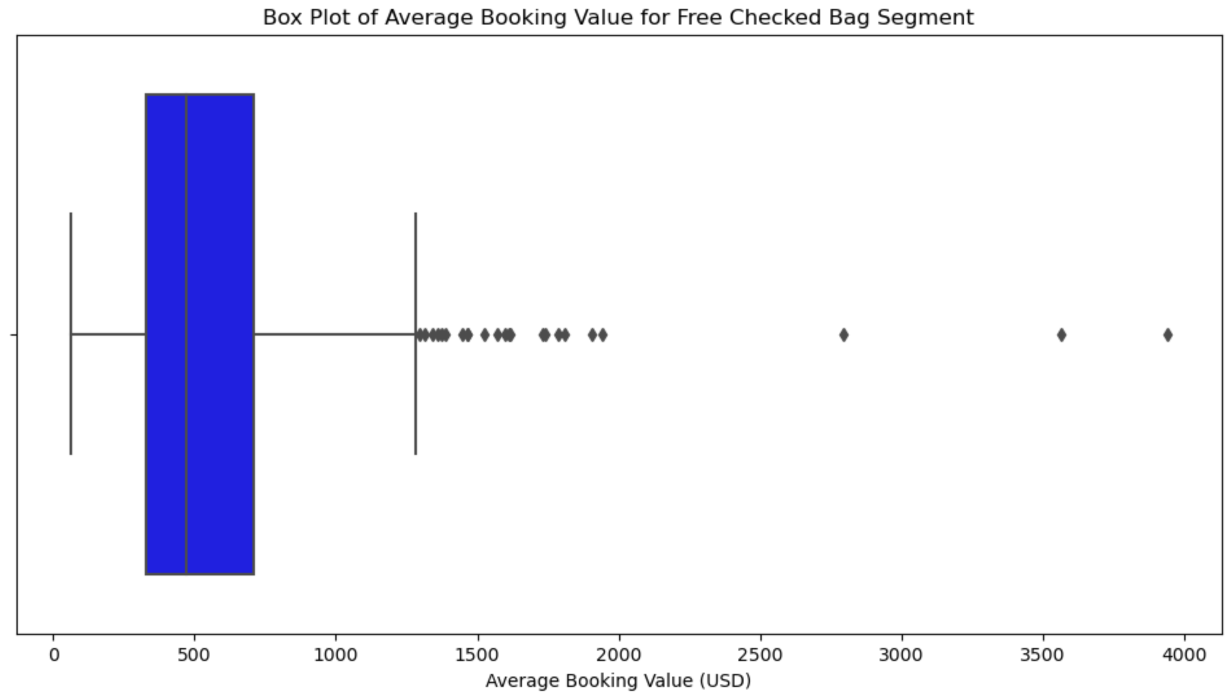

Average Booking Value by Customer Segment (Highlight: Free Hotel Night)
$4233.60
$1046.46
$587.76
$570.72
$571.00
$625.87
Average Booking Value (USD)
0
500
1000
1500
2000
2500
3000
3500
4000
Exclusive Discounts
Free Hotel Meal
Free Hotel Night
Lounge Access
Free Checked Bag
Free Cancellation
Customer Segment

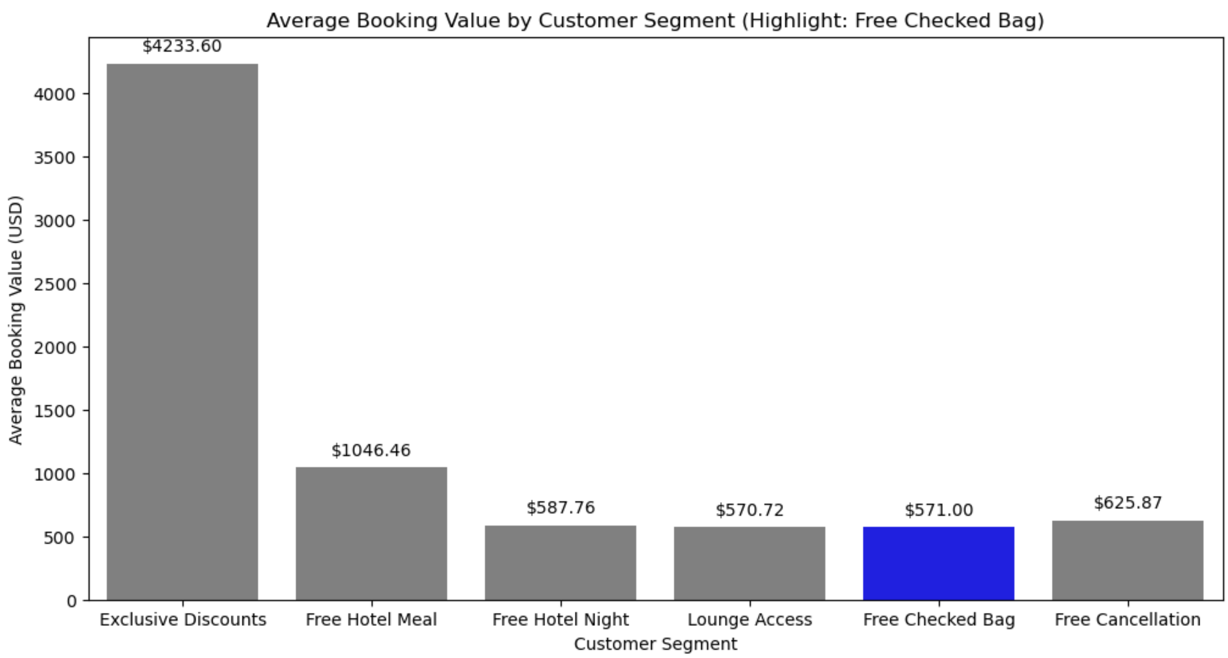

Average Booking Value for Lounge Access Segment
Frequency
0
2
4
6
8
10
12
14
200
400
600
800
1000
1200
Average Booking Value (USD)

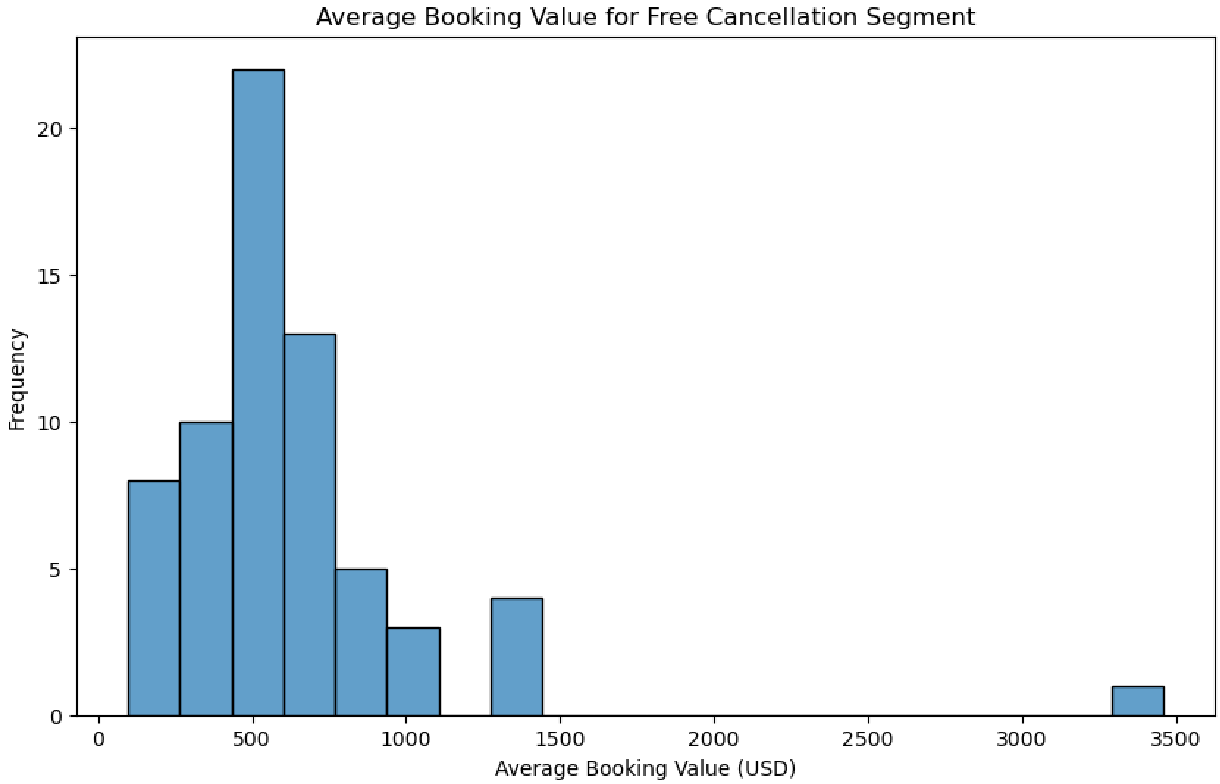

Box Plot of Average Booking Value for Lounge Access Segment
200
400
600
800
1000
1200
Average Booking Value (USD)

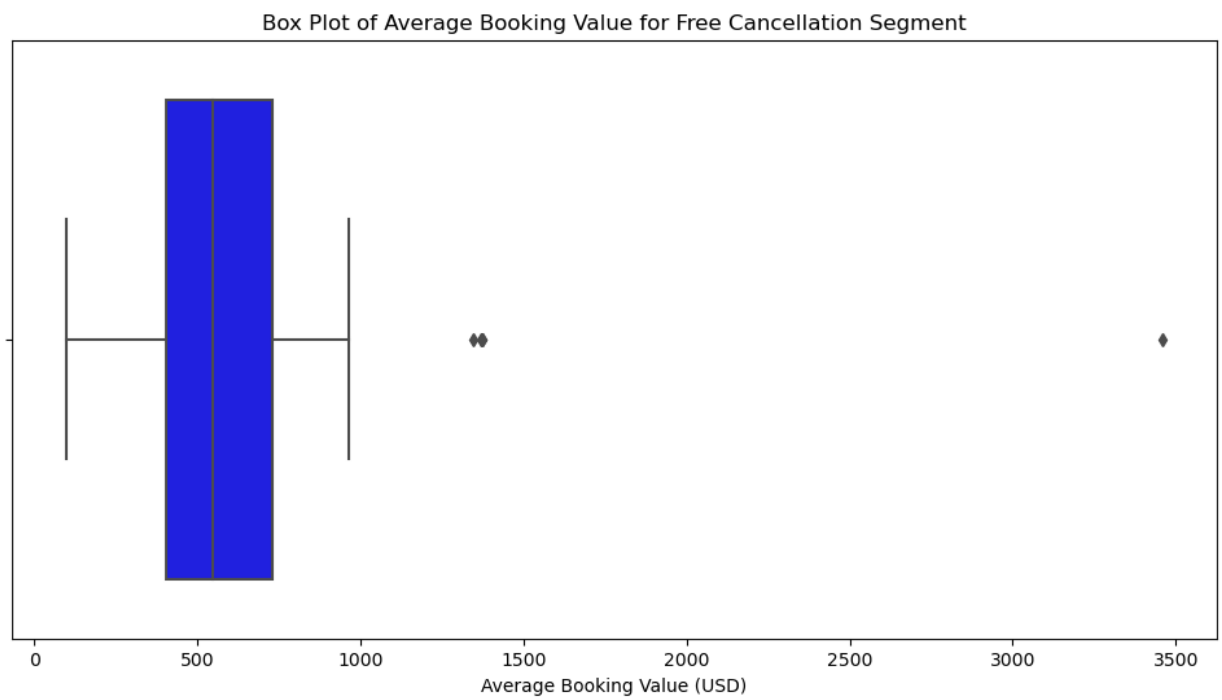

Average Booking Value by Customer Segment (Highlight: Lounge Access)
Average Booking Value (USD)
Customer Segment
Exclusive Discounts
Free Hotel Meal
Free Hotel Night
Lounge Access
Free Checked Bag
Free Cancellation
$4233.60
$1046.46
$587.76
$570.72
$571.00
$625.87

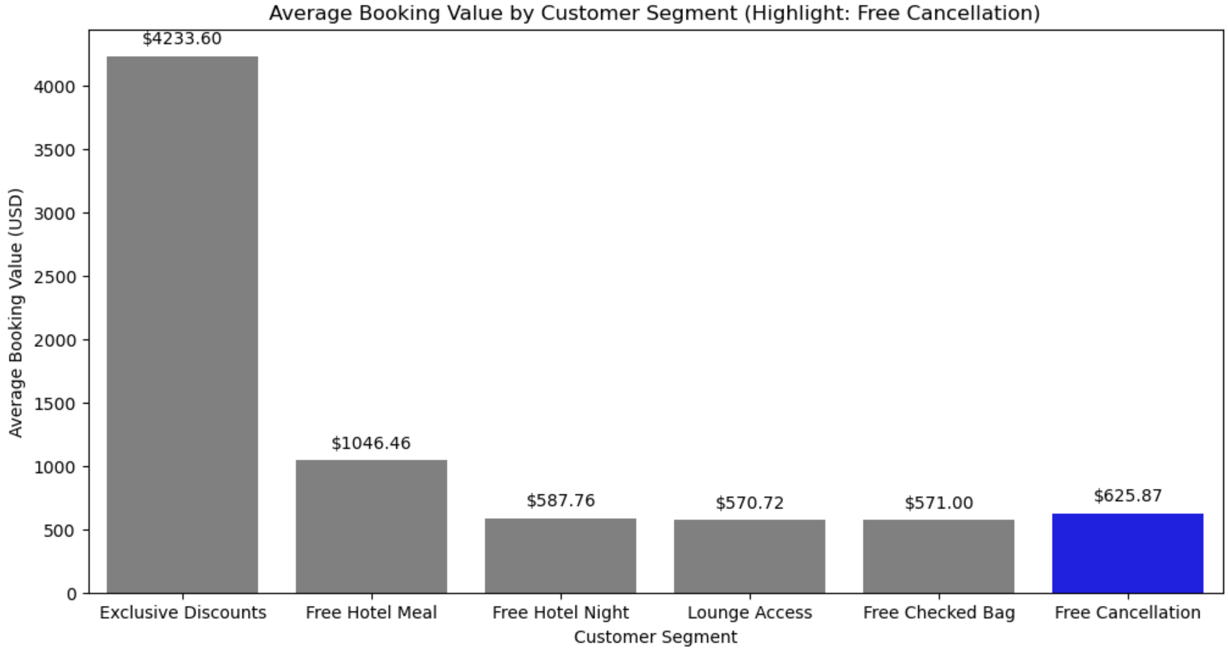

Average Booking Value for Free Checked Bag Segment
Frequency
Average Booking Value (USD)


Box Plot of Average Booking Value for Free Checked Bag Segment
0
500
1000
1500
2000
2500
3000
3500
4000
Average Booking Value (USD)


Average Booking Value by Customer Segment (Highlight: Free Checked Bag)
$4233.60
$1046.46
$587.76
$570.72
$571.00
$625.87
0
500
1000
1500
2000
2500
3000
3500
4000
Average Booking Value (USD)
Exclusive Discounts
Free Hotel Meal
Free Hotel Night
Lounge Access
Free Checked Bag
Free Cancellation
Customer Segment

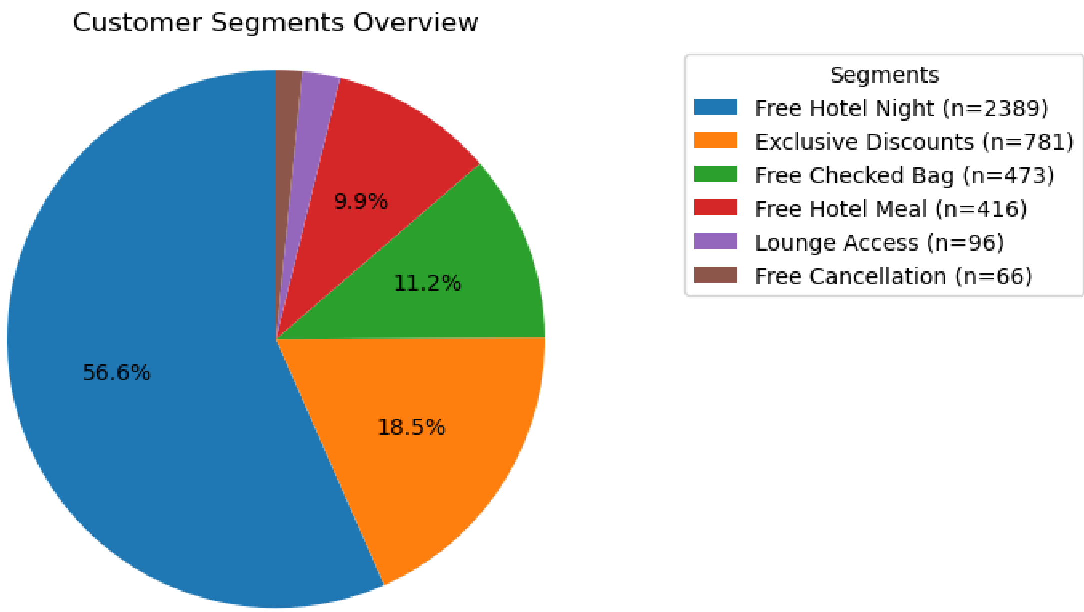

Average Booking Value for Free Cancellation Segment
Frequency
0
5
10
15
20
0
500
1000
1500
2000
2500
3000
3500
Average Booking Value (USD)


Box Plot of Average Booking Value for Free Cancellation Segment
0
500
1000
1500
2000
2500
3000
3500
Average Booking Value (USD)



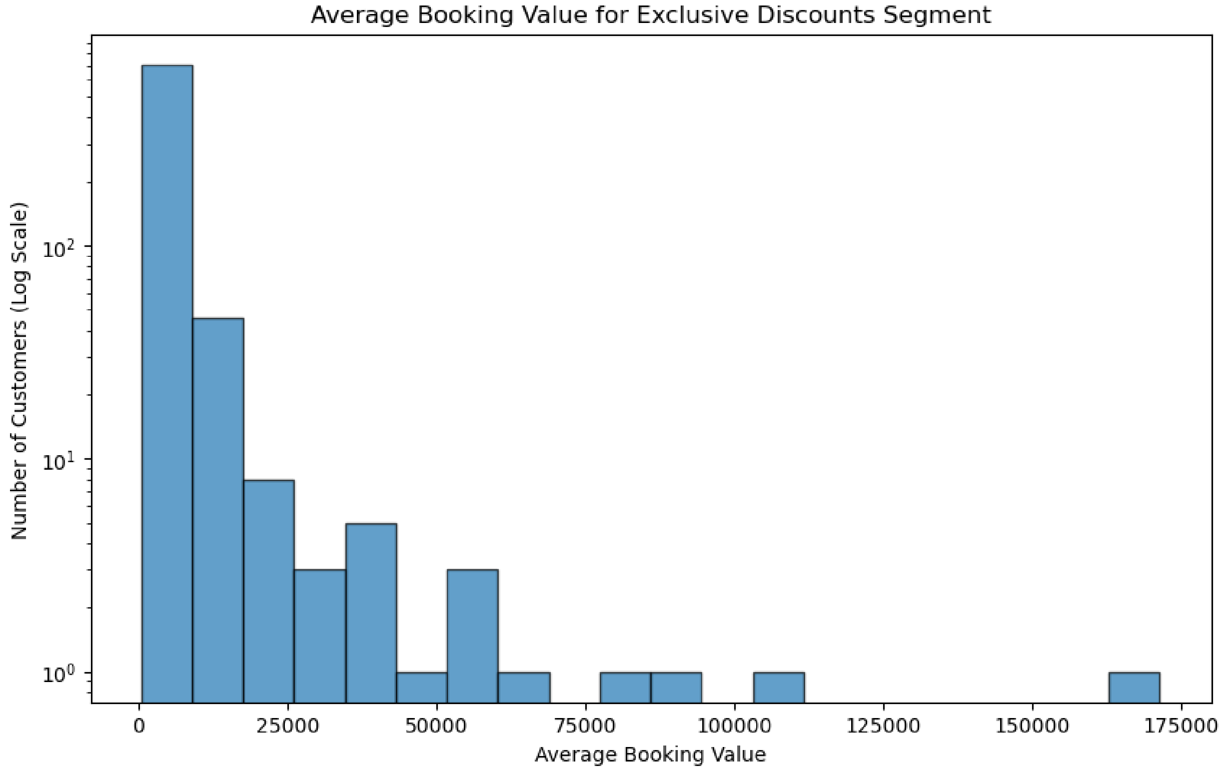
In [150…

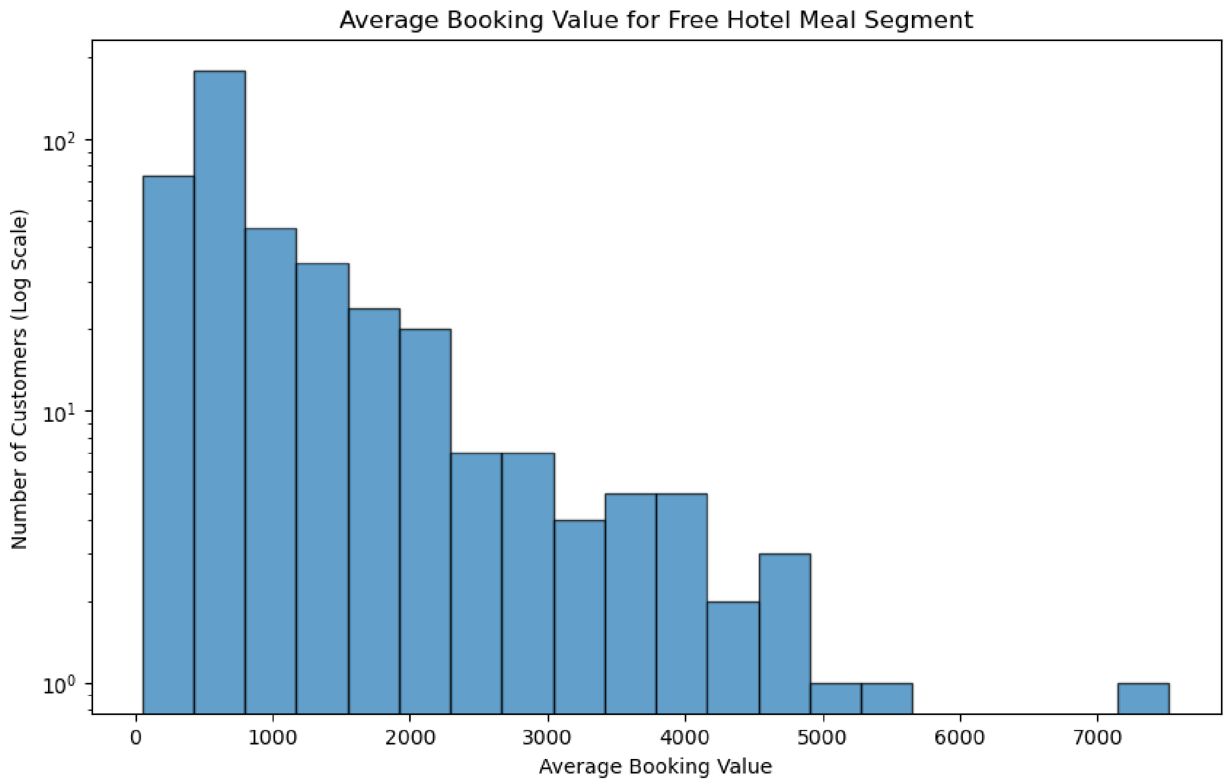
```
# Pie Chart for presentation with segment sizes

def func(pct, allvals):
    absolute = int(round(pct/100.*sum(allvals)))
    if pct < 5:  # threshold percentage to display
        return ""
    else:
        return "{:.1f}%".format(pct)

# Get the size of each segment
segment_sizes = df_marketing_segments['segment'].value_counts()

# Create labels for the legend including the counts
legend_labels = [f"{segment} (n={count})" for segment, count in zip(segment_sizes.inde

# Create the pie chart
fig1, ax1 = plt.subplots()
wedges, texts, autotexts = ax1.pie(segment_sizes, autopct=lambda pct: func(pct, segmen

# Equal aspect ratio ensures that pie is drawn as a circle.
ax1.axis('equal')

# Add title
plt.title('Customer Segments Overview')

# Add legend
ax1.legend(wedges, legend_labels, title="Segments", loc="upper left", bbox_to_anchor=(

# Show the pie chart
plt.savefig('segments_pie_chart.png', transparent=True, bbox_inches='tight')
plt.show()
```



In [166…

```
# logarithmic scale histogram

def plot_segment_histogram(df, segment_name):
    # Filter the DataFrame to only include rows from the specified segment
    segment_df = df[df['segment'] == segment_name]

    # Create the histogram
    plt.figure(figsize=(10, 6))
    plt.hist(segment_df['average_booking_value'], bins=20, edgecolor='black', alpha=0.

    # Set the y-axis to a logarithmic scale
    plt.yscale('log')

    # Add labels and title
    plt.xlabel('Average Booking Value')
    plt.ylabel('Number of Customers (Log Scale)')
    plt.title(f'Average Booking Value for {segment_name} Segment')

    # Show the plot
    plt.savefig(f'Log_Histogram_{segment_name.replace(" ", "_")}.png', transparent=Tru
    plt.show()

# Example usage
plot_segment_histogram(df_marketing_segments, 'Exclusive Discounts')
plot_segment_histogram(df_marketing_segments, 'Free Hotel Meal')
```





In [155…

```
# Quick check on the high value customers - how many trips has our highest value custo

# Filter the DataFrame to only include rows where the segment matches the segment_name
high_value_df = df_marketing_segments[df_marketing_segments['segment'] == 'Exclusive D

# Check if the DataFrame is empty
if high_value_df.empty:
    print("No data found for the segment 'Exclusive Discounts'.")
else:
    # Sort the DataFrame by 'total_value' in descending order
```

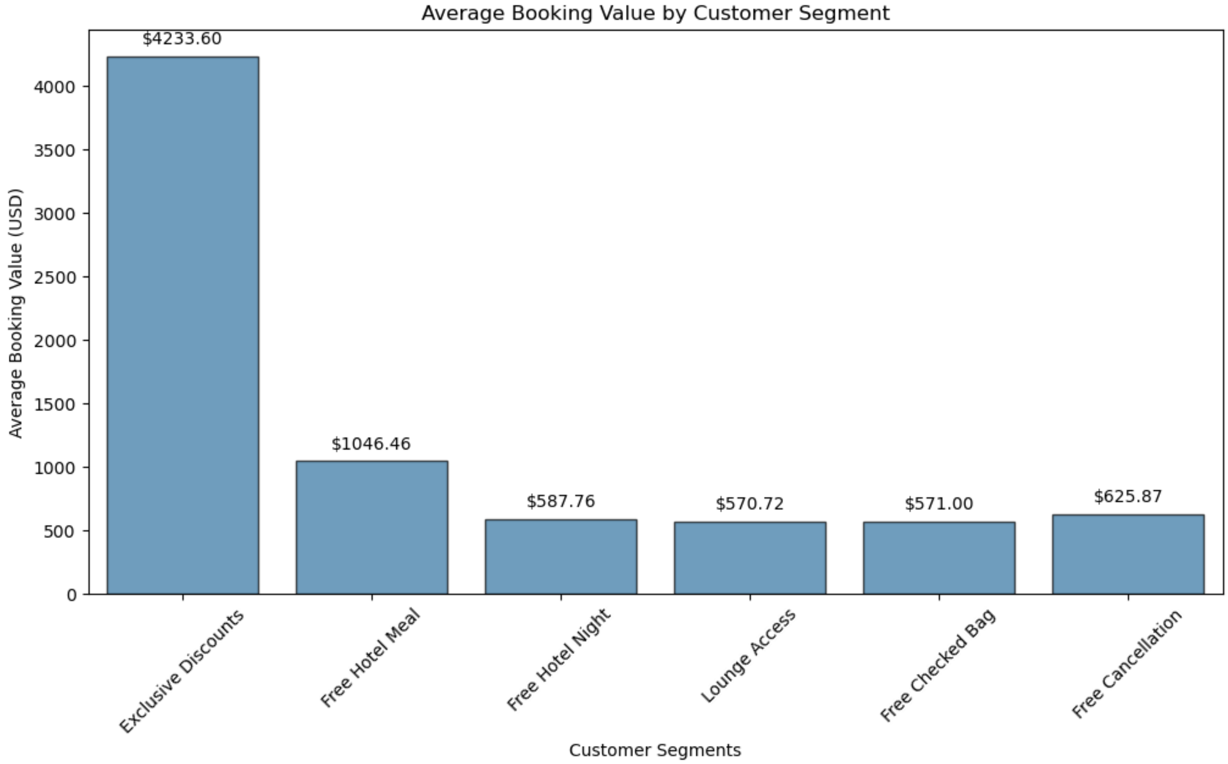
```
    high_value_df_sorted = high_value_df.sort_values(by='total_value', ascending=False

    # Get the number of trips booked by the top-spending customer
    top_customer_trips = high_value_df_sorted.iloc[0]['number_of_bookings']
    print(f"The top-spending customer in the 'Exclusive Discounts' segment has booked
```

```
The top-spending customer in the 'Exclusive Discounts' segment has booked 2 trips and
has spent 208658.11
```

In [167…

```
# Summary bar chart

# Create the bar chart
plt.figure(figsize=(12, 6))

# Use Seaborn's barplot function for the bar chart

# Create the bar plot
plt.figure(figsize=(12, 6))
ax = sns.barplot(x='segment', y='average_booking_value', data=df_marketing_segments, c


plt.xlabel('Customer Segments')
plt.ylabel('Average Booking Value (USD)')
plt.title('Average Booking Value by Customer Segment')
# Annotate each bar with the corresponding value
for p in ax.patches:
    ax.annotate(f"${p.get_height():.2f}",
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center',
                xytext=(0, 10),
                textcoords='offset points')

# Rotate x-axis labels for better visibility
plt.xticks(rotation=45)

# Save the plot
plt.savefig(f'summary bar.png', transparent=True, bbox_inches='tight')  # Save the plo
# Show the plot
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```


Average Booking Value by Customer Segment
$4233.60
$1046.46
$587.76
$570.72
$571.00
$625.87
Average Booking Value (USD)
0
500
1000
1500
2000
2500
3000
3500
4000
Exclusive Discounts
Free Hotel Meal
Free Hotel Night
Lounge Access
Free Checked Bag
Free Cancellation
Customer Segments

In [ ]: