Executive Summary

TravelTide

Intro & Background

TravelTide is a **rising star** in the online travel space. Since launching in April 2021, just after the peak of the COVID-19 pandemic, it's seen steady growth thanks to its **top-tier data aggregation and search technology**. Customers and industry analysts alike have praised the platform for offering one of the largest travel inventories in the e-booking world — a major draw for anyone looking to book flights or hotels with ease.

Objectives

Some parts of the TravelTide customer experience still need work, which has led to lower-than-hoped-for customer retention. **CEO Kevin Talanick** is keen to change that — he's focused on adding value for existing customers through a thoughtful marketing strategy. To help lead the charge, he's brought on **Elena Tarrant as Head of Marketing.** Her mission: design and launch a personalized rewards program that keeps customers coming back to TravelTide.
Our role is to explore whether the data backs up Elena's idea — that there are **specific customer groups** who'd be especially drawn to the **perks** she has in mind. From there, we'll aim to match each customer with the perk they're most likely to love.

Methodology

- Used **SQL and Python** for data exploration and cleaning
- Filtered the data to include:
  - Customers with **more than 7 sessions**
  - Data from **January 4th, 2023 onwards**
- **Removed cancelled bookings** and their original trips to avoid skewing results
- **Aggregated the data** at the user level using SQL
- Added two new columns:
  - **Segment** (based on behavior and demographics)
  - **Perk** (likely most appealing reward)
- Applied a simple **Decision Tree method** to assist with segmentation
- Used the final **CSV** to create visualizations in **Tableau**

Executive Summary

TravelTide

Key Findings

Segment Name	User Count	Key Traits	Proposed Perk
Family Travelers	1,432	Largest group, high bookings, moderate revenue; mix of solo & group travel	Kids stay free on next flight + hotel combo
Youth Travelers	991	Under 35, solo/duo travel, highest avg. revenue	Free checked bag + lounge access for solo trips
Young Solo Travelers	991	No kids, under 35, low bookings/revenue, mid baggage use	Free checked bag on next flight
Mid-Life Savers (No Discount)	1,459	Ages 35–55, no kids, high bookings, short trips, likely business travel	Free meal with next hotel booking
Mid-Life Savers (With Discount)	931	Ages 35–55, no kids, deal seekers, highest avg. discount rate	15% off next flight + hotel combo
Golden Explorers	285	55+, no kids, group travel possible, plan far ahead	10% off when booking 30+ days in advance
Last Minute Leavers	100	No completed trips, all cancellations, high engagement	Free cancellation on next booking
Wanderlust Watchers	456	No bookings or cancellations, high browsing time and clicks	Free drink on next flight

Recommendations

- Due to the limited timeframe of the data, behavioral patterns across segments were somewhat inconsistent. Rewards were assigned to each group using a combination of observed trends and strategic assumptions, with the aim of motivating future bookings.
- Expanding the dataset to include a full year and focusing on users with more than five sessions would likely reveal more consistent patterns and enhance the quality of the segmentation.

Detailed report

TravelTide

Sql Code to create a session level table

Only from 04th Jan and users with more then 7 sessions

```
WITH user_level AS (
  SELECT
    user_id,
    COUNT(session_id) AS total_sessions
  FROM sessions
  WHERE session_start >= '2023-01-04'
  GROUP BY user_id
  HAVING COUNT(session_id) > 7
),
session_full_level_table AS (
  SELECT
  s.session_id, s.user_id, s.trip_id, s.session_start, s.session_end, s.flight_discount, s.hotel_discount, s.flight_discount_amount, s.hotel_discount_amount,
  s.flight_booked, s.hotel_booked, s.page_clicks, s.cancellation, u.birthdate, u.gender, u.married, u.has_children, u.home_country, u.home_city, u.home_airport,
  u.home_airport_lat, u.home_airport_lon, u.sign_up_date,  h.check_in_time, h.check_out_time, h.hotel_per_room_usd,
f.origin_airport, f.destination, f.destination_airport, f.seats,
  f.return_flight_booked, f.departure_time, f.return_time, f.checked_bags, f.trip_airline, f.destination_airport_lat, f.destination_airport_lon, f.base_fare_usd,
  (haversine_distance(
        u.home_airport_lat,
        u.home_airport_lon,
        f.destination_airport_lat,
        f.destination_airport_lon
    )) AS distance_km,
  CASE
    WHEN check_out_time::date - check_in_time::date <1 THEN 1
    ELSE check_out_time::date - check_in_time::date
  END as nights_new,
  CASE
    WHEN rooms = 0 THEN 1
    ELSE rooms
  END as rooms_new,
  split_part(hotel_name,' - ', 1) AS hotel_name,
  split_part(hotel_name,' - ', 2) AS hotel_city,
  EXTRACT(YEAR FROM AGE('2023-07-31', u.sign_up_date)) * 12 + EXTRACT(MONTH FROM AGE('2023-07-31', u.sign_up_date)) AS mths_as_customer,
  EXTRACT(YEAR FROM AGE('2023-07-31', u.birthdate)) AS customer_age,
  (base_fare_usd / seats) AS cost_per_seat,
  EXTRACT(EPOCH FROM (session_end - session_start)) AS session_duration_seconds,
  (departure_time::DATE - session_end::DATE) AS days_to_flight_after_booking,
  (flight_discount_amount*base_fare_usd) AS dollars_saved_flights,
  (hotel_discount_amount*hotel_per_room_usd) AS dollars_saved_hotels,
  CASE
    WHEN return_time IS NOT NULL THEN 1
    ELSE 0
  END AS return_flight_bool,
  CASE
    WHEN departure_time IS NOT NULL THEN 1
    ELSE 0
  END AS departure_flight_bool
  FROM sessions s
  LEFT JOIN users u ON s.user_id = u.user_id
  LEFT JOIN hotels h ON s.trip_id = h.trip_id
  LEFT JOIN flights f ON s.trip_id = f.trip_id
  JOIN user_level ul ON s.user_id = ul.user_id
  WHERE session_start >= '2023-01-04'
)
SELECT *
FROM session_full_level_table
```

Joining and cleaning Sessions Table

- Joining all tables using a LEFT JOIN (except on user_level join)
- Selecting only the rows from 2023-01-04 onwards
- Selecting only the users from 2023-01-04 with more the 7 sessions
- Using haversine to calculate the distance from the home airport to destination airport
- Creating a new column for the correct nights (some negative or 0 when should be more), if it is 0 or less it is replaced with a 1 as per Katerina in class
- Create new columns for correct rooms, if it is 0 then replaced with a 1 as per Katerina in class
- Split the hotel name into 2 columns so it shows hotel and city seperately
- Created a column showing months user has been a customer (sign up date to end date of 2023-07-31)
- Created a new column showing customers age in years (calculated by using end date of 2023-07-31)
- Hotel Rate is per room per night
- Added the session_duration oin seconds (session end time less session start time)
- Added days_to_flight_after_booking
- Added dollars_saved_flights column
- Added in a column to create a boolean of 1 or 0 for both departing and returning flights

Detailed report

TravelTide

Working on the session level table

Python Analysis

- Ran checks for any inconstencies or errors in the final sessions table
- Fixed the columns that should be DATETIME
- Added a check in and departure period so that it is easy to see if user has prefered time of day to travel (Morning, Afternoon, Evening or nigh)
- Downloaded the updated file for record purposes
- Added a column for the full hotel cost (rate * rooms * nights)

```
# Correct date_time columns
sessions_level_df['session_start'] = pd.to_datetime(sessions_level_df['session_start'])
sessions_level_df['departure_time'] = pd.to_datetime(sessions_level_df['departure_time'])
sessions_level_df['return_time'] = pd.to_datetime(sessions_level_df['return_time'])
sessions_level_df['check_out_time'] = pd.to_datetime(sessions_level_df['check_out_time'])
```

Added a column to show the time of day for check_in and departure time, did this incase in the future I want to see prefered check in or departure time for user

```
# Adjust sessions level table so that check in time is in datetime format and no longer an object

sessions_level_df['check_in_time'] = pd.to_datetime(sessions_level_df['check_in_time'])

# Added a column called check_in_period which shows the period of the day the user checked in, incase want to use info later

def time_of_day(dt):
    hour = dt.hour
    if 0 <= hour <= 4:
        return 'morning'
    elif 5 <= hour <= 11:
        return 'morning'
    elif 12 <= hour <= 16:
        return 'afternoon'
    elif 17 <= hour <= 22:
        return 'evening'
    elif 23 <= hour <= 24:
        return 'evening'
    else:
        return 'No Check_in'

sessions_level_df['check_in_period'] = sessions_level_df['check_in_time'].apply(time_of_day)
```

```
# Added a column called departure_period which shows the period of the day the users flight departed

def time_of_day(dt):
    hour = dt.hour
    if 0 <= hour <= 4:
        return 'morning'
    elif 5 <= hour <= 11:
        return 'morning'
    elif 12 <= hour <= 16:
        return 'afternoon'
    elif 17 <= hour <= 22:
        return 'evening'
    elif 23 <= hour <= 24:
        return 'evening'
    else:
        return 'No Flight'

sessions_level_df['departure_period'] = sessions_level_df['departure_time'].apply(time_of_day)
```

Detailed report

TravelTide

Working on the session level table

```
# Add in the full hotel cost - cant do in orginal SQL as the columns dont exist yet (the rate we have is per room per night so multiplied by both to get full cost)

sessions_level_df['hotel_total_cost'] = sessions_level_df['hotel_per_room_usd']*sessions_level_df['nights_new']*sessions_level_df['rooms_new']
sessions_level_df
```

Created a chart to show the number of hotel trips per month 2023

```
import matplotlib.pyplot as plt
import seaborn as sns

# Create a year-month column and make it string
sessions_level_df['year_month'] = sessions_level_df['session_start'].dt.to_period('M').astype(str)

# Group by the months and show the total trips per month
monthly_counts = sessions_level_df.groupby('year_month')['trip_id'].nunique().reset_index()

# Rename columns for clarity
monthly_counts.columns = ['year_month', 'Distinct Trips']

# Convert year_month to datetime for filtering and sorting
monthly_counts['year_month'] = pd.to_datetime(monthly_counts['year_month'])

# Filter to include only up to July 2023
cutoff_date = pd.to_datetime('2023-07')
monthly_counts = monthly_counts[monthly_counts['year_month'] <= cutoff_date]

# Plots the chart with Seaborn and Matplotlib to see bookings per month and year over all the years
plt.figure(figsize=(10, 6))
sns.lineplot(data=monthly_counts, x='year_month', y='Distinct Trips', marker='o')
plt.xticks(rotation=45)
plt.title('Number of Hotel Trips per Month')
plt.xlabel('Year-Month')
plt.ylabel('Number of Hotel Trips')
plt.tight_layout()
plt.show()
```

Final query to create a users table with aggregations

- Does not include the cancelled trips or original trips linked to them - added a cancellations column to see which customers do have cancellations
- Used a subquery for total_cancellations to bring just the totals back for each customer from the cancelled_trips CTE
- Calculated the ADS (Avg dollar saved) per km travelled to get less biased result ((using flight_disount_amount*base_fare_uds)/distance_km)
- Used COALESCE on columns that have NULLS to make it look neater
- Used Round on columns with many decimal places, to make it look neater and easier to read
- Included both departure and return flights as a column each plus a total flights column with the 2 added together
- some users had hotel/flight discount marked as TRUE but no actual trip id (so no trip made) so have excluded these from the total_flights/hotels_with_discount, avg_flight/hotel_discount_amount, sum_flight/hotel_discount_amount and discount_flight_proportion

Detailed report

TravelTide

Sql Code to create a user level aggregated table

```
WITH cancelled_trips AS (
    SELECT trip_id AS cancelled_trip_id
    FROM sessions_level_df
    WHERE cancellation = True
)
SELECT

-- Customer Demographics

    user_id,
    birthdate,
    customer_age AS age_end_july_2023,
    gender,
    married,
    has_children,
    home_country,
    home_city,
    home_airport,
    home_airport_lat,
    home_airport_lon,
    sign_up_date,
    mths_as_customer,

-- General calculations on sessions, trips, page-clicks and cancellations

    AVG(session_duration_seconds) AS avg_session_duration_seconds,
    SUM(session_duration_seconds) AS sum_session_duration_seconds,
    COUNT(DISTINCT trip_id) AS trips_booked,
    COUNT(DISTINCT session_id) AS total_sessions,
    COALESCE(SUM(page_clicks), 0) AS total_page_clicks,
    ROUND(COALESCE(AVG(days_to_flight_after_booking),0),2) AS avg_days_booking_to_flight,
    (
        SELECT COUNT(DISTINCT trip_id)
        FROM sessions_level_df slu
        LEFT JOIN cancelled_trips cts ON slu.trip_id = cts.cancelled_trip_id
        WHERE slu.user_id = sl.user_id
            AND cts.cancelled_trip_id IS NOT NULL
    ) AS total_cancellations,
```

Detailed report

TravelTide

Sql Code to create a user level aggregated table

```
--Flight calculations

 SUM(CASE WHEN trip_id IS NOT NULL THEN flight_discount ELSE 0 END) AS total_flights_with_discount,
 ROUND(COALESCE(AVG(CASE WHEN trip_id IS NOT NULL THEN flight_discount_amount END),0),2) AS avg_flight_discount_amount,
 SUM(CASE WHEN trip_id IS NOT NULL THEN flight_discount_amount ELSE 0 END) AS sum_flight_discount_amount,
 ROUND(
 COALESCE(1.0 * SUM(CASE WHEN trip_id IS NOT NULL THEN flight_discount ELSE 0 END) /
 SUM(departure_flight_bool + return_flight_bool),0),3
 ) AS discount_flight_proportion,
 COALESCE(SUM(base_fare_usd), 0) AS sum_flights_cost,
 ROUND(COALESCE(AVG(base_fare_usd), 0),2) AS avg_flights_cost,
 COALESCE(SUM(departure_flight_bool),0) AS sum_departure_flights,
 COALESCE(SUM(return_flight_bool),0) AS sum_return_flights,
 COALESCE(SUM(departure_flight_bool),0) + COALESCE(SUM(return_flight_bool),0) AS total_flights_transactions,
 ROUND(COALESCE(AVG(dollars_saved_flights),0),2) AS ADS_flights,
 ROUND(COALESCE(SUM(flight_discount_amount*base_fare_usd)/SUM(distance_km),0),4) AS ADS_per_km,
 ROUND(COALESCE(SUM(distance_km), 0),2) AS sum_distance_km,
 ROUND(COALESCE(AVG(distance_km), 0),2) AS avg_distance_km,
 COALESCE(AVG(seats), 0) AS avg_seats_booked,
 COALESCE(SUM(seats), 0) AS sum_seats_booked,
 ROUND(COALESCE(AVG(checked_bags), 0), 2) AS avg_checked_bags,
 COALESCE(SUM(checked_bags), 0) AS sum_checked_bags,

-- Hotel calculations

 SUM(CASE WHEN trip_id IS NOT NULL THEN hotel_discount ELSE 0 END) AS total_hotels_with_discount,
 ROUND(COALESCE(AVG(CASE WHEN trip_id IS NOT NULL THEN hotel_discount_amount END),0),2) AS avg_hotel_discount_amount,
 SUM(CASE WHEN trip_id IS NOT NULL THEN hotel_discount_amount ELSE 0 END) AS sum_hotel_discount_amount,
 ROUND(
 COALESCE(1.0 * SUM(CASE WHEN trip_id IS NOT NULL THEN hotel_discount ELSE 0 END) /
 SUM(hotel_booked), 0),2
 ) AS discount_hotel_proportion,
 COALESCE(SUM(hotel_per_room_usd), 0) AS sum_hotel_cost_per_room_per_night,
 ROUND(COALESCE(AVG(hotel_per_room_usd), 0), 2) AS avg_hotel_cost_per_room_per_night,
 COALESCE(SUM(hotel_total_cost) , 0) AS sum_hotel_cost_total,
 ROUND(COALESCE(AVG(hotel_total_cost) , 0), 2) AS avg_hotel_cost_total,
 COALESCE(SUM(hotel_booked),0) AS total_hotel_transactions,
 COALESCE(AVG(hotel_discount_amount*hotel_per_room_usd), 0) AS ADS_hotels,
 COALESCE(AVG(nights_new), 0) AS avg_nights_booked,
 COALESCE(SUM(nights_new), 0) AS sum_nights_booked,
 COALESCE(AVG(rooms_new), 0) AS avg_rooms_booked,
 COALESCE(SUM(rooms_new), 0) AS sum_rooms_booked
FROM sessions_level_df sl
LEFT JOIN cancelled_trips ct ON sl.trip_id = ct.cancelled_trip_id
WHERE ct.cancelled_trip_id IS NULL
GROUP BY user_id, birthdate, gender, married, has_children, home_country, home_city, home_airport, home_airport_lat, home_airport_lon,
sign_up_date, mths_as_customer, customer_age
```

Detailed report

TravelTide

SQL and Python to create Final User table including Segmentation and Perks columns

```python
# Added the scaled ADS using python as when I tried to add the column using SQL I kept getting errors, probably wont use in segmentation but thought good to have incase

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
user_level_df['scaled_ADS'] = scaler.fit_transform(user_level_df[['ADS_flights']])
```

```
# Final query to create a table including the perks per customer

query = """

WITH segmentation_split AS (
   SELECT *,
      CASE
         WHEN trips_booked = 0 AND total_cancellations  = 0 THEN 'Wanderlust Watcher'
         WHEN trips_booked = 0 AND total_cancellations  > 0 THEN 'Last-Minute Leaver'
         WHEN trips_booked > 0 AND has_children = TRUE THEN 'Family Explorer'
         WHEN trips_booked > 0 AND has_children = FALSE AND age_end_july_2023 <= 35 THEN 'Youthful Adventurer'
         WHEN trips_booked > 0 AND has_children = FALSE AND age_end_july_2023 > 55 THEN 'Golden Explorer'
         WHEN trips_booked > 0 AND has_children = FALSE AND age_end_july_2023 BETWEEN 36 AND 55 AND avg_flight_discount_amount = 0 THEN 'Mid-Life
Spender'
         WHEN trips_booked > 0 AND has_children = FALSE AND age_end_july_2023 BETWEEN 36 AND 55 AND avg_flight_discount_amount > 0 THEN 'Mid-Life
Saver'
         ELSE 'No Perk'
      END AS segmentation
   FROM user_level_df
)
SELECT *,
      CASE
      WHEN segmentation = 'Last-Minute Leaver' THEN 'Free Cancellation'
      WHEN segmentation = 'Family Explorer' THEN 'Kids stay free on next booking flight & hotel combo'
      WHEN segmentation = 'Youthful Adventurer' THEN 'Free bag for all passengers on next flight booking'
      WHEN segmentation = 'Golden Explorer'  THEN '10% discount on next booking made 30 days in advance'
      WHEN segmentation = 'Mid-Life Spender' THEN 'Free meal with next hotel booking'
      WHEN segmentation = 'Mid-Life Saver' THEN '15% Discount with next booking flight & hotel combo'
      WHEN segmentation = 'Wanderlust Watcher' THEN 'Free drink on next flight'
      ELSE 'No Perk'
   END AS perk
FROM segmentation_split

"""
perk_result = pysqldf(query)
perk_result
perk_result.to_csv('user_level_df_incl_perk_segments.csv', index=False)
```