DA202 Customer Segmentation: Mastery Project (TravelTide) by Masterschool

Calculate Metrics and Segment Customers

Updated on October 3rd, 2023

Technical report

- This document will walk audience through ipynb for the project, which includes data cohorting, data cleaning (removing outliers), behavioral metrics, data scaling and K-Means analysis.
- Other documents for reference: executive summary, presentation and Tableau dashboard.

1. **Metadata**
   - Data is provided by Masterschool and details are available in the Python notebook.

2. **Project description**
   - Assist Elena Tarrant, Head of Marketing at TravelTide to put a personalized rewards program in motion. Elena hypothesized five perks: free hotel meal, 1 night free hotel with flight (aka free hotel night), free checked bag, no cancellation fees (aka free to cancel) and exclusive discounts.
   - Project introduction can be found here.
   - **Task overview:** calculate metrics and segment customers of TravelTide, a fictitious travel e-commerce platform into one of the five hypothesized perks, based on customers' behavioral metrics.

3. Advanced Tasks

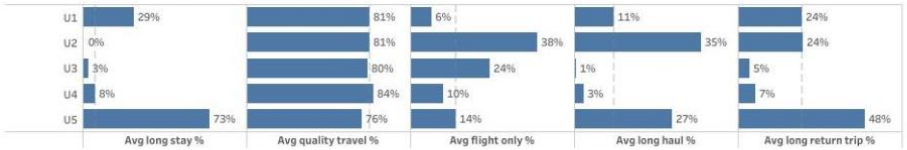
- Write a Better Haversine Distance Function – see **Cohort data (Section D) – part 4**.
- Make a Tableau Dashboard – see Tableau dashboard.
- Segment Customers with K Means – see Tableau dashboard and presentation. Extract from presentation:

Allocated perk vs Kmeans cluster - user data

	Free hotel meal	Free hotel night	Free checked bag	Free to cancel	Special offer travel	Grand Total
U5	78	12	14	14	11	129
U2	4	12	5			21
U3	32	110	292	224	715	1,373
U4	424	595	760	770	842	3,391
U1	411	147	261	172	93	1,084
Grand ..	949	876	1,332	1,180	1,661	5,998

General metrics and sub-metrics for VIP

Sub-metrics for exquisite perk metrics



Sub-metrics for standard perk metrics

K-Means

Approach and findings

- K-Means algo run with user behavioral sub-metrics
- Weak if any overlap of K-Means clusters U1-U5 with allocated perks
- U1/U4 users seem average on most themes. U1 above average for long stay. U4 sub-average for spend $/user
- U2 captures few users, thus insignificant
- U3 is high in cancel rate, relative discount and % of sessions failing to book (slippage)
- U5 outperforms strongly for extended round-trips and long hotel stays (7+ days), good for free hotel meal

Perks validated by K-Means? Unclear

- U3 exhibits mixed features for free to cancel and special offer travel, may mean those two should not be distinct perks behaviorally
- Difficult to pin-point where U1/U2/U4 should land

4. Steps to run ipynb

- **Daily ipynb initialization (Section A)**
  - Import relevant Python libraries in part 1.
  - Preset paths for importing and exporting various dfs in part 2.
  - Run ready-made functions in part 3. These functions will calculate distances (Haversine, Charles Karney's method), remove outliers, scale data, run K-Means, compute scaled behavioral metrics and more.
- **One-off data import (Section B)**
  - To import raw data from the postgres link provided by Masterschool for the first time, run steps in in parts 1 & 2 in section B. Data loading of the ipynb.
  - This will generate full df, and full.csv.
  - In future skip parts 1 & 2 in section B. Go directly to import full df from full.csv in the Express lane part of section B.

- **Exogenous info (Section C)**
  - Run part 1 to compile externally sourced destination country info (related the destination column in full df) into destinations df.
  - Run part 2 to compile externally sourced budget airline info (related the trip_airline column in full df) into airlines df.
  - Run part 3 to compile externally sourced hotel brand, hotel city, hotel country, hotel star rating and luxury indicator info (related the hotel_name_column in full df) into hotels df.

- **Cohort data (Section D) – parts 1-3**
  - Run part 1 to cohort data from full df by filtering for session_start >= 2023-01-04 and users with more than 7 sessions.

```
data = data[data['session_start'] >= pd.Timestamp('2023-01-04 00:00:00')]
```

```
data = data[data.groupby('user_id')['session_id'].transform('nunique') > 7]
```

- Run part 2 to merge previously mentioned destinations, airlines and hotels dfs.

```
data = pd.merge(data, destinations, on=['destination'], how='left')
data = pd.merge(data, airlines, on=['trip_airline'], how='left')
data = pd.merge(data, hotels, on=['hotel_name'], how='left')
```

- Run part 3 to generate indicator column use_trip_id (value==True means that the row info for the trip_id is not obsolete.

- **Cohort data (Section D) – part 4 – Advanced Task – Improved Haversine Function**
  - Flight distances in km are calculated in part 4.
  - Improved Haversine function used is found in part 3 in Section A, see below.
  - Improvement made into the original Haversine by being able to adjust for Earth's oblateness by using weighted average of equatorial and polar radii based on latitudes of two points
  - Additionally, a more accurate Charles Karney's method is included.

```
def distances(df, lat1_col='home_airport_lat', lon1_col='home_airport_lon',
lat2_col='destination_airport_lat', lon2_col='destination_airport_lon',
          flight_booked_col='flight_booked', return_flight_col='return_flight_booked', method='karney',
          short_haul_max=1287, medium_haul_max=4023, long_haul_max=10460):
    """
    * Haversine: Great-circle distance between two points on surface of a perfect sphere; (oblate==True) to
adjust for Earth's oblateness (use weighted average of equatorial and polar radii based on latitudes of two
points)
    * Charles Karney's Method: Use geographiclib library. Geodesic.WGS84.Inverse method from
geographiclib calculates distance between two points based on WGS-84 ellipsoidal model of the Earth.
Recent and more accurate method for geodetic computations than adjusted haversine. Accurate results for
nearly antipodal points. superior to previously commonly used Vincenty's method that might not converge
    """
    valid_methods = ['haversine_basic', 'haversine_oblate', 'karney']
    if method not in valid_methods and method is not None:
        print(f"Invalid method chosen: {method}. Choose from {', '.join(valid_methods)}.")
        return

    def is_true(val):                                                    # check if a value is true-like
        return val in [True, 'True', 'true', 1]
```

```
  def is_false_or_nan(val):                                                    # check if a value is false or NaN
    return val in [False, 'False', 'false', 0, 'NaN', 'nan', np.nan]

  for col in [lat1_col, lon1_col, lat2_col, lon2_col]:                         # ensure columns are of float type
    df[col] = df[col].astype(float)

  def haversine(lat1, lon1, lat2, lon2, oblate=False):                         # haversine estimate distances
    lat1, lon1, lat2, lon2 = map(np.radians, map(float, [lat1, lon1, lat2, lon2]))
    R = 6371 if not oblate else (1 - 1/298.257223563) * 6378.137 / np.sqrt(1 - (1/298.257223563)**2 *
np.sin((lat1 + lat2) / 2)**2)
    dLat, dLon = lat2 - lat1, lon2 - lon1
    a = np.sin(dLat / 2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dLon / 2)**2
    return R * 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))

  def categorize(distance):
    if math.isnan(distance) or distance == 0:
      return None, None
    flight_category = ('short haul' if distance <= short_haul_max else
              'medium haul' if distance <= medium_haul_max else
              'long haul' if distance <= long_haul_max else 'ultra long haul')
    flight_long_haul = distance > medium_haul_max
    return flight_category, flight_long_haul

  def calculate_all(row):                                  # main function
    if is_false_or_nan(row[flight_booked_col]):    # not accounting for cancellation, such that theoretical
flight distances still calculated
```

```
        return [np.nan]*8
    lat1, lon1, lat2, lon2 = row[lat1_col], row[lon1_col], row[lat2_col], row[lon2_col]
    h_basic, h_oblate = haversine(lat1, lon1, lat2, lon2), haversine(lat1, lon1, lat2, lon2, oblate=True)       # call the function for haversine both without and with oblate adjustment
    dist_karney = Geodesic.WGS84.Inverse(lat1, lon1, lat2, lon2)['s12'] / 1000                                  # charles karney's method, conver to km
    multiplier = 2 if (row[return_flight_col]==True) else 1                                                     # multiplier for return flight

    distance_km = {'haversine_basic': h_basic, 'haversine_oblate': h_oblate, 'karney': dist_karney}.get(method, 0)  # method is drawn from the keys of dict
    flown_km = distance_km * multiplier

    flight_category, flight_long_haul = categorize(distance_km)
    return [h_basic, h_oblate, dist_karney, method, flown_km, distance_km, flight_category, flight_long_haul]


  new_cols = pd.DataFrame(df.apply(calculate_all, axis=1).tolist(),                                             # apply main function
                          columns=['flown_km_haversine_basic', 'flown_km_haversine_oblate', 'flown_km_karney', 'distance_method', 'flown_km', 'distance_km', 'flight_category', 'flight_long_haul'])

  for col in new_cols.columns:
    df[col] = new_cols[col]

  return df
```

- For the purpose of cohort data, Karney's method is used by calling the function this way.

```
distances(data, lat1_col='home_airport_lat', lon1_col='home_airport_lon',
lat2_col='destination_airport_lat', lon2_col='destination_airport_lon',
        flight_booked_col='flight_booked', return_flight_col='return_flight_booked', method='karney',
        short_haul_max=1287, medium_haul_max=4023, long_haul_max=10460)
```

- Function also calculates flight_long_haul indicator column if the flight is long haul (distance over 4,023 km)

- **Cohort data (Section D) – cont'd**
  - Run parts 5 & 6 to generate time-based metrics and trip indicator columns, critical for behavioral metrics in subsequent sections.
  - Run part 7 to generate various transaction level economics including value_per_seat_per_km column that would be used to remove outliers.
  - Run part 8 to finalize the cohort data as data df, which will also be saved as data.csv, which can be imported in the Express lane part of this section.

- **Clean and remove outliers (Section E)**
    - Remove transactions with negative nights.

```
precut = data.loc[~(data['nights'] < 0)].copy()
```

    - Remove outliers (outside 2 standard deviation) for columns hotel_per_room_usd and 'value_per_seat_per_km.

```
economic_intensivity = ['hotel_per_room_usd', 'value_per_seat_per_km']

data_2_stdev, data_ex_2_stdev = remove_outliers(precut, col_list=economic_intensivity,
col_filter='use_trip_id', reference_col='session_id',
            methodology='stdev', threshold=2, whisker_size=0.5)
```

    - Generate clean version of data df as clean df

```
clean = data_2_stdev.copy() # assign to the choice of data sets above
```

- **User behavioral metrics (Section F)**
    - Run part 1 to (starting with clean df groupby user_id) calculate various user level metrics including grouped by related perks such as max_bags_per_seat for free checked bag, etc. This will complete user level metrics df called user df.
    - Run part 2 to investigate for each perk-based behavioral metric, the correlations between various sub-metrics (user level). Results of the correlation analysis saved here.
    - To calculate each perk-based behavioral metric (or index), the sub-metrics will be MinMax scales and then sum-weighted. The weights used will sum up to 1. More will be discussed subsequently. As a rule, if two sub-metrics are correlated at 0.70 or higher, their respective weights will be half of the weights of sub-metrics that don't correlate as strongly.

- **MinMax scale sub-metrics, calculate perk-based behavioral metric and run K-Means (Section G)**
    - Run part 1 to set up perk dictionary and generate perk df with details of the sub-metrics for each perk and their weights. The csv generated is here.
    - Run part 2 to calculate the PPC/VIP index which to generate VIP metric mentioned in presentation and bonus point for eligible VIP customers (those scoring 80-percentile or better also known as VIP customers).
    - The compute_index_scores function (found in part 3 in Section A) is called to MinMax scale the sub-metrics (with capping at 2 std dev) of PPC/VIP index and then sun-weighted per the weights in perk df to generate the a score which is valued between 0-1.
    - The compute_index_scores function is also called in part 3 to calculate the index/behavioral metric scores for exquisite perks: free hotel meal and free hotel night (previously 1 free night free hotel with flight).
    - Repeat for standard perks in part 4.
    - The MinMax scaled sub-metrics and overall perk-based behavioral metric / index scores are backed up here.
    - Run part 5 to collate overall perk-based behavioral / index scores and call standalone minmax function to MinMax scale the scores (without capping) for the final time.
    - Run part 6 to run K-Means algo on user level metrics in user df (can ignore K-Means on transaction data as this won't be presented in the Tableau dashboard).
    - Run part 7 to complete selection df that contains the details of use df, concatenated with the perk-based metric scores and final MinMax scaled index scores. The hypothetical original scores for exquisite perks also included. The csv version of selection df is here.
    - Part 8 is optional as the required visualization to show overlap between K-Means cluster and perks can be done in Tableau.