Technical_Document_final

November 1, 2022

Prepared for MSBA 6411

Lakshit Bajaj, Amlendu Kumawat, Shubham Midha, Nikita Saini and Kexin Yang

0.1 Background

The purpose of this document is to serve as a technical reference for a data-driven customer insights project done for Sun Country airlines. Using this document in conjunction with the Executive summary and PowerPoint slides, we can help them answer questions including but not limited to: - Understand the customers better - Provide customised support to the customers

Goals of this notebook:

- Provide step-wise exploration and cleaning of data
- Obtaining a customer level master data with their salient features
- Applying clustering techniques to understand features of customers better

CONTENTS :

- Exploring the data
- Tidying the data
- Creating New Features that have the potential to be used in Clustering
- Aggregating Data
- Addressing Compute Issues (Sub-sampling)
- Model Building
- Model Interpretation

0.2 Exploring the data

0.2.1 Importing libraries

[1]:
```
import pandas as pd
import os
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import seaborn as sns
from sklearn.model_selection import KFold
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

```python
import datetime

pd.options.display.max_rows = 1000
pd.options.display.max_columns = 200

from tqdm import tqdm
tqdm.pandas()

import warnings
warnings.filterwarnings('ignore')
```

0.2.2 Loading Data

[2]:
```python
os.chdir(r"D:\Lakshit\MSBA\sem2\MSBA 6411 Exploratory Data Analytics and␣
↪Visualization\Team project\01-Input")
```

[153]:
```python
suncountry_data=pd.read_csv('SunCountry.csv',low_memory=False)
suncountry_data.head(4)
```

[153]:
```
  PNRLocatorID      TicketNum  CouponSeqNbr ServiceStartCity ServiceEndCity  \
0       AAABJK  3377365159634             2              JFK            MSP
1       AAABJK  3377365159634             1              MSP            JFK
2       AAABMK  3372107381942             2              MSP            SFO
3       AAABMK  3372107381942             1              SFO            MSP

  PNRCreateDate ServiceStartDate PaxName  \
0    2013-11-23       2013-12-13  BRUMSA
1    2013-11-23       2013-12-08  BRUMSA
2    2014-02-04       2014-02-23  EILDRY
3    2014-02-04       2014-02-20  EILDRY

                                     EncryptedName GenderCode  birthdateid  \
0  4252554D4241434B44696420493F7C2067657420746869…          F      35331.0
1  4252554D4241434B44696420493F7C2067657420746869…          F      35331.0
2  45494C4445525344696420493F7C20676574207468697…          M      46161.0
3  45494C4445525344696420493F7C20676574207468697…          M      46161.0

    Age PostalCode BkdClassOfService   TrvldClassOfService  \
0  66.0        NaN             Coach                 Coach
1  66.0        NaN             Coach           First Class
2  37.0        NaN             Coach  Discount First Class
3  37.0        NaN             Coach  Discount First Class

        BookingChannel  BaseFareAmt  TotalDocAmt  UFlyRewardsNumber  \
0      Outside Booking       234.20          0.0                NaN
1      Outside Booking       234.20          0.0                NaN
2  SCA Website Booking       293.96        338.0                NaN
```

```
3  SCA Website Booking       293.96       338.0                NaN

  UflyMemberStatus CardHolder BookedProduct EnrollDate MarketingFlightNbr  \
0              NaN        NaN        CHEOPQ        NaN                244
1              NaN        NaN        CHEOPQ        NaN                243
2              NaN        NaN           NaN        NaN                397
3              NaN        NaN           NaN        NaN                392

  MarketingAirlineCode StopoverCode
0                   SY            O
1                   SY          NaN
2                   SY            O
3                   SY          NaN
```

[3]:
```
suncountry_data.describe()
```

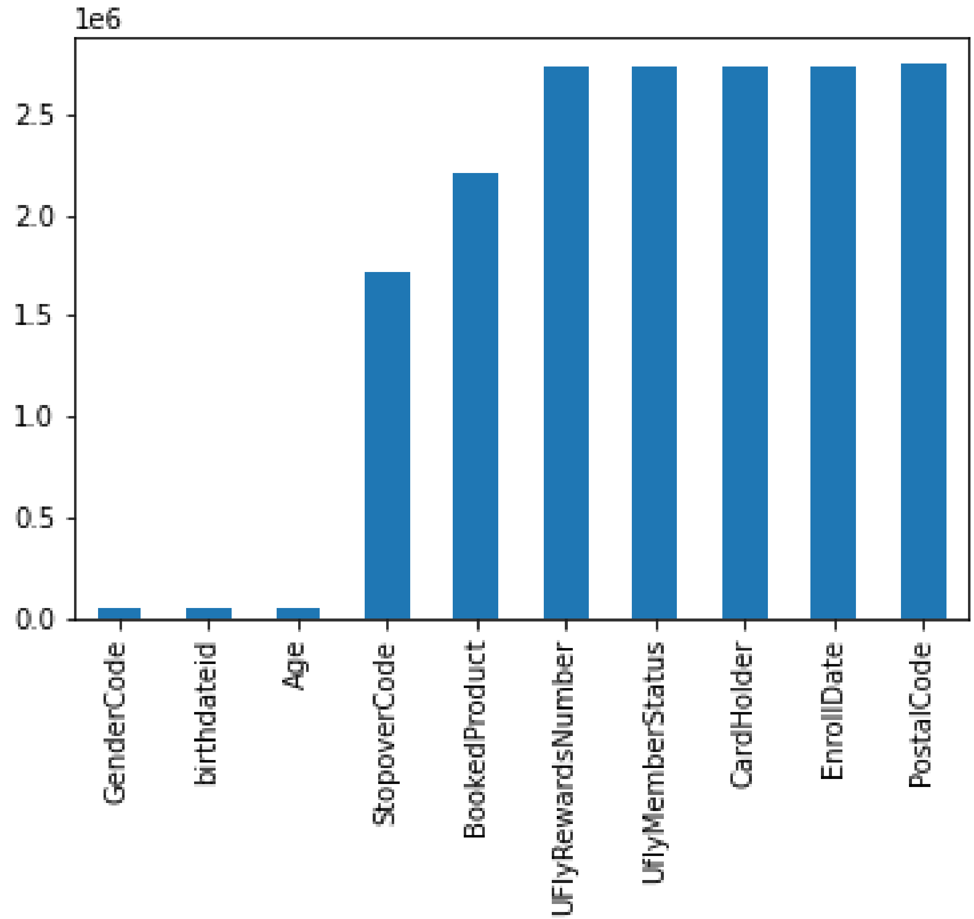
[3]:
```
          TicketNum  CouponSeqNbr   birthdateid           Age   BaseFareAmt  \
count  3.435388e+06  3.435388e+06  3.391389e+06  3.391389e+06  3.435388e+06
mean   3.374303e+12  1.463528e+00  4.498228e+04  4.004857e+01  2.877286e+02
std    2.575632e+09  5.752532e-01  7.072362e+03  1.935122e+01  1.824781e+02
min    3.372052e+12  1.000000e+00 -6.752900e+05 -2.883000e+03  0.000000e+00
25%    3.372107e+12  1.000000e+00  3.962000e+04  2.600000e+01  1.748800e+02
50%    3.372108e+12  1.000000e+00  4.508800e+04  4.000000e+01  2.732200e+02
75%    3.377293e+12  2.000000e+00  5.025100e+04  5.500000e+01  3.702300e+02
max    3.379578e+12  8.000000e+00  1.112840e+06  2.012000e+03  4.342000e+03

        TotalDocAmt  UFlyRewardsNumber
count  3.435388e+06       6.944800e+05
mean   3.148554e+02       2.042421e+08
std    2.121591e+02       1.482674e+07
min    0.000000e+00       1.000002e+08
25%    1.898000e+02       2.008601e+08
50%    3.022000e+02       2.029681e+08
75%    4.146000e+02       2.103819e+08
max    1.757200e+04       2.410863e+08
```

0.2.3 Missing Values Check

- Missing values are very high for StopOverCode, BookedProduct, UFlyMemeberStatus, CardHolder, EnrollDate, PostalCode.
- Age, GenderCode and birthdateids missing is what concerns us , we deal with these missing values later.

[4]:
```
missing = suncountry_data.isnull().sum()
missing = missing[missing > 0]
missing.sort_values(inplace=True)
missing.plot.bar()
```

[4]: <AxesSubplot:>



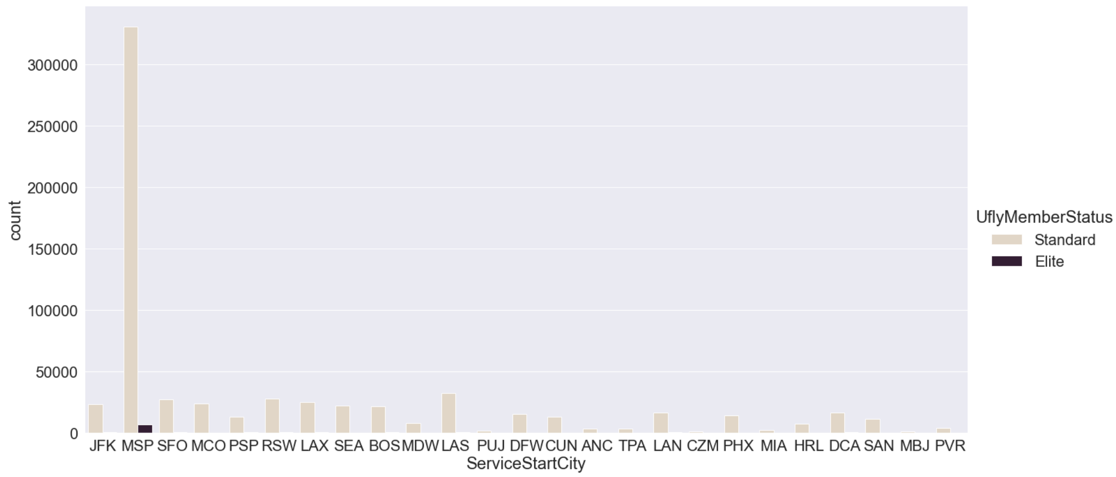
0.2.4 Distribution of Categorical Variables

[21]:
```
#Filtering for most frequent start service cities.
freq_start_service_city= suncountry_data.ServiceStartCity.
 ↪value_counts()[suncountry_data.
                                                                    ␣
 ↪ServiceStartCity.value_counts()>10000].index.to_list()

suncountry_cities= suncountry_data[suncountry_data.ServiceStartCity.
 ↪isin(freq_start_service_city)]
```

[30]:
```
plt.figure(figsize=(20,10))
sns.set(font_scale=2)
sns.catplot(data=suncountry_cities, x="ServiceStartCity", kind="count",␣
 ↪palette="ch:.25",height=10,hue='UflyMemberStatus', aspect=2)
```

[30]: <seaborn.axisgrid.FacetGrid at 0x7fbaaa88e280>

<Figure size 1440x720 with 0 Axes>

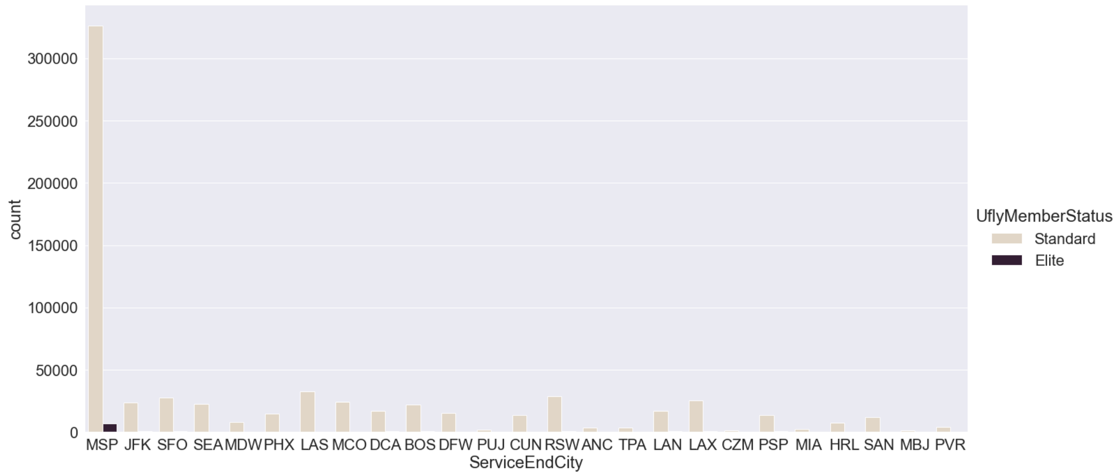


As expected, we have highest count for flights with ServiceStartCity MSP, also MSP has most concentration of our Elite members too.

[34]:
```
#Filtering for most frequent end service cities.
freq_start_service_city= suncountry_data.ServiceEndCity.
 ↪value_counts()[suncountry_data.
                                                                              ␣
 ↪ServiceEndCity.value_counts()>10000].index.to_list()

suncountry_cities= suncountry_data[suncountry_data.ServiceEndCity.
 ↪isin(freq_start_service_city)]
```

[35]:
```
sns.catplot(data=suncountry_cities, x="ServiceEndCity", kind="count",␣
 ↪palette="ch:.25",height=10,hue='UflyMemberStatus', aspect=2)
```

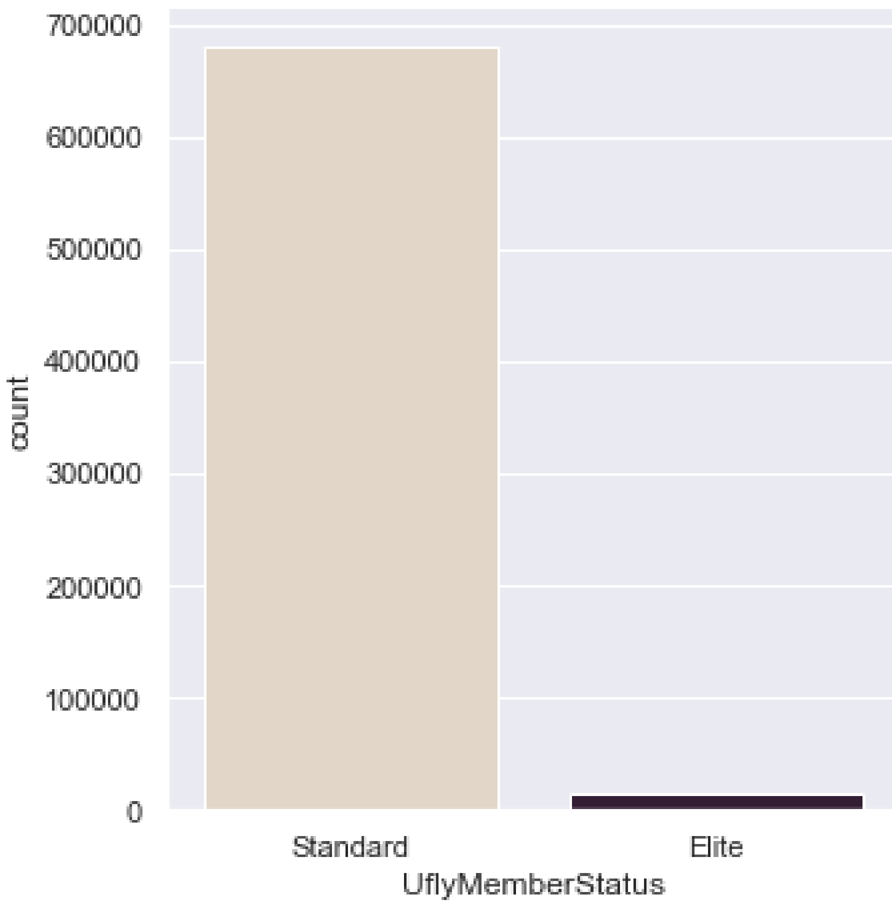
[35]: <seaborn.axisgrid.FacetGrid at 0x7fb99a75ea30>



As expected, we have highest count forServiceEndCity MSP, also MSP has most concentration of our Elite members too.

[36]:
```
sns.set(font_scale=1)
sns.catplot(data=suncountry_data, x="UflyMemberStatus", kind="count",
    palette="ch:.25")
```

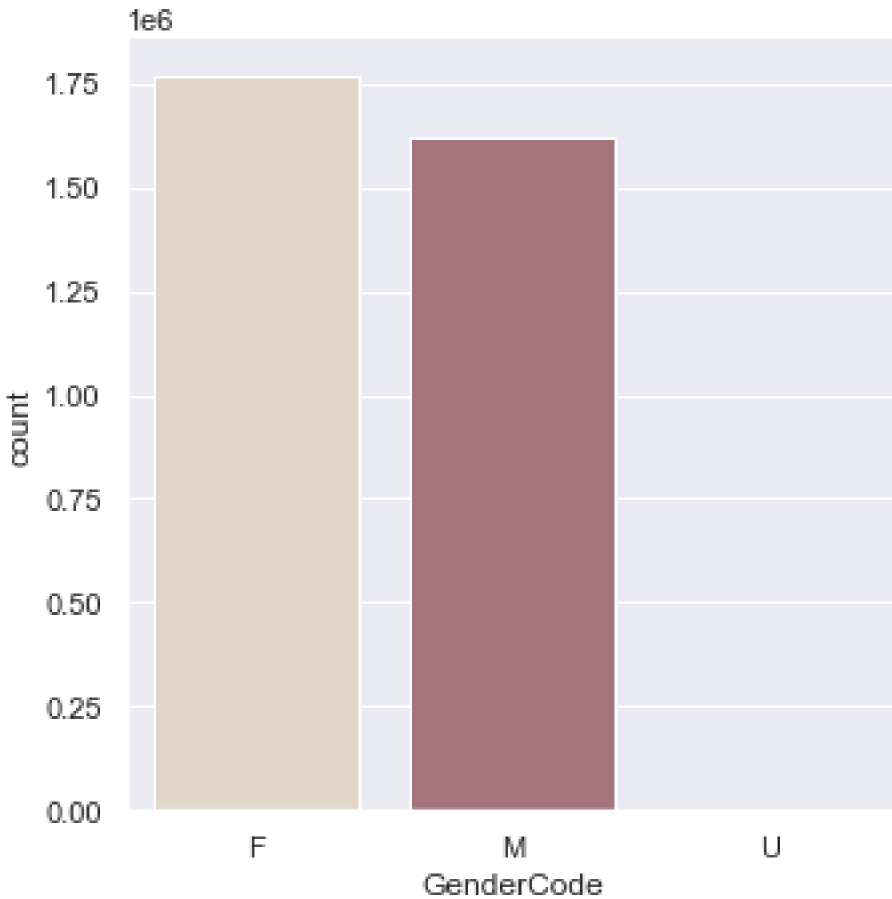
[36]: <seaborn.axisgrid.FacetGrid at 0x7fb9c813f970>



Data has extremely skewed distribution for UflyMemberStatus, Elite members data is very less compared to Standard class

[38]:
```
sns.set(font_scale=1)
sns.catplot(data=suncountry_data, x="GenderCode", kind="count", palette="ch:.
↪25")
```

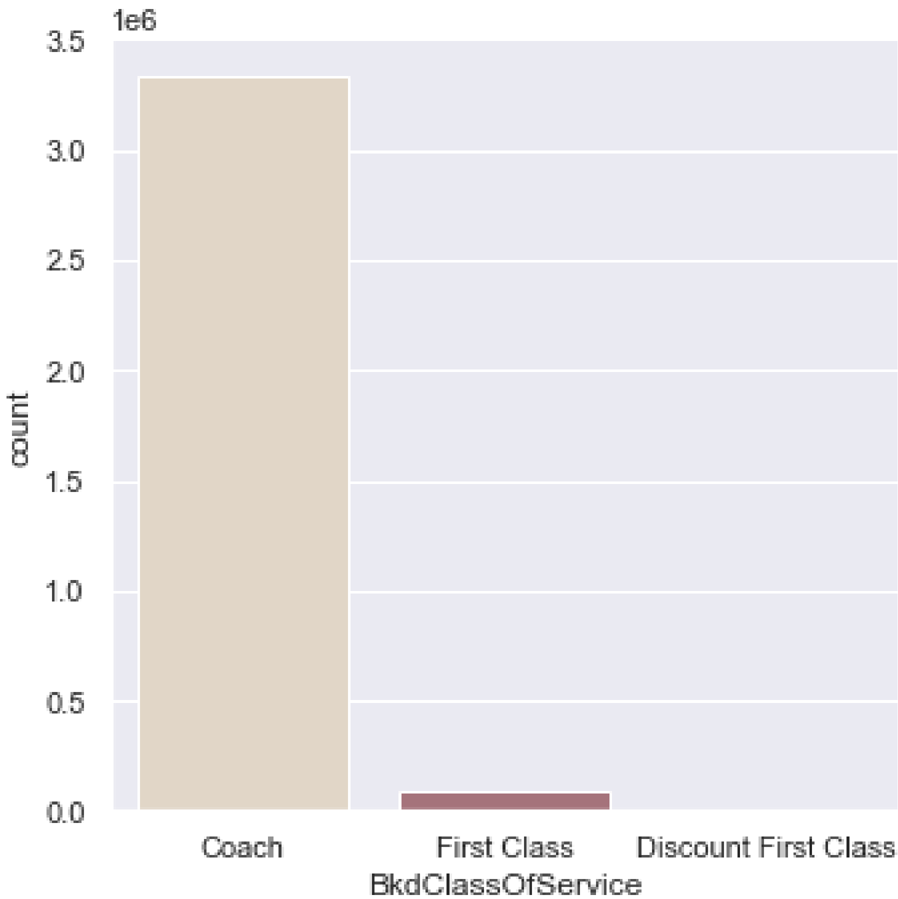
[38]: <seaborn.axisgrid.FacetGrid at 0x7fba9c3f76d0>



We have fairly equal occurrences for 'F' and 'M' in the data with some 'U' unidentified Gender values.

[42]:
```
sns.set(font_scale=1)
sns.catplot(data=suncountry_data, x="BkdClassOfService", kind="count",␣
↪palette="ch:.25")
```

[42]: <seaborn.axisgrid.FacetGrid at 0x7fb994cd2490>

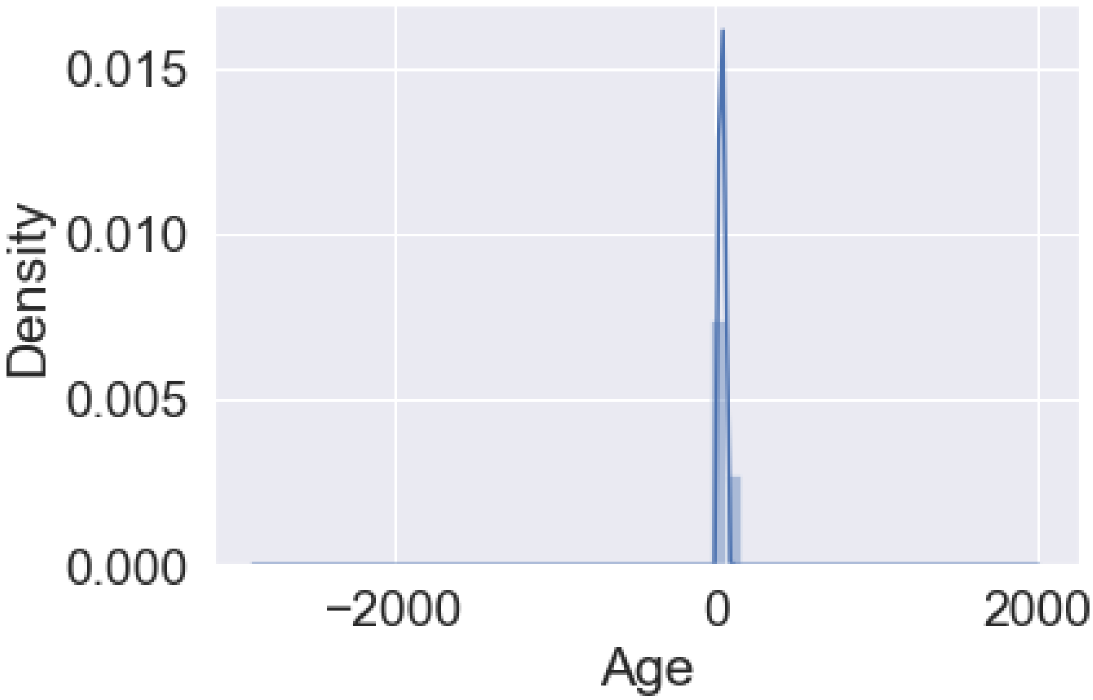


We have fairly skewed dataset in terms of category representation of BkdClassOfService .

0.2.5 Distribution of Numerical Variables

[158]:
```
sns.distplot(suncountry_data['Age'])
```

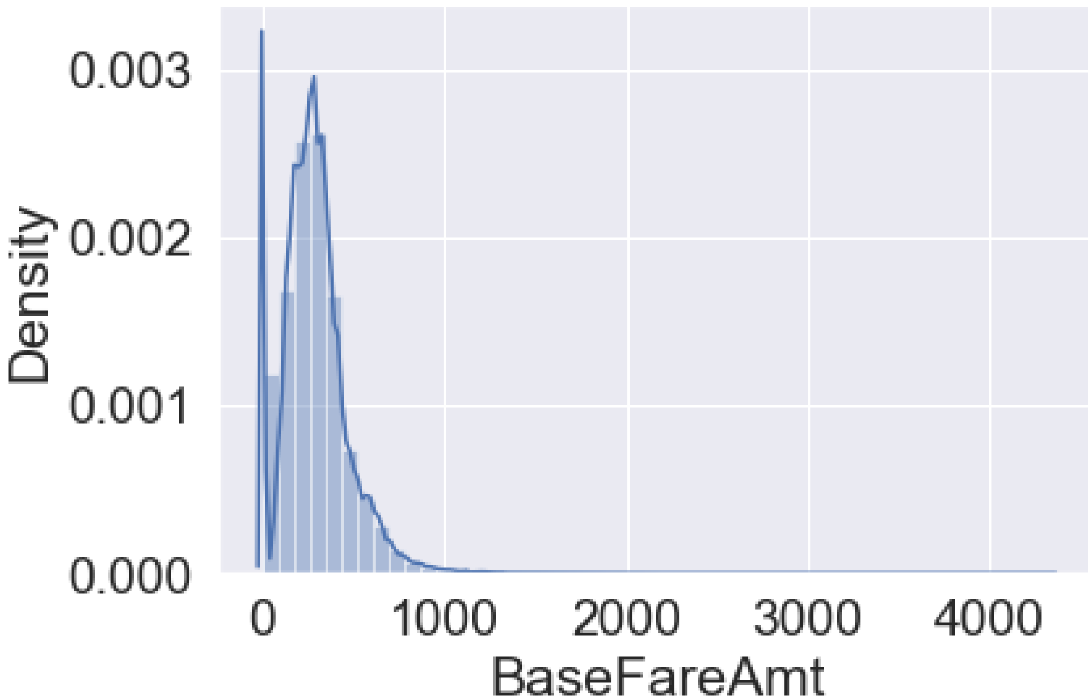
[158]: <AxesSubplot:xlabel='Age', ylabel='Density'>



Raw distribution of Age suggests some outlier ages, which we need to deal with

[160]:
```
sns.distplot(suncountry_data['BaseFareAmt'])
```

[160]: <AxesSubplot:xlabel='BaseFareAmt', ylabel='Density'>



We observe a bi-modal distribution of BaseFareAmtand also some anomalously high ticket fares for a single ticket.

0.3 Tidying the data

We tidied the data for using further: This included removing duplicates, handling missing values and removing some other unneccesary rows

Removing duplicated rows and data where airline is not Sun Country Airlines

[45]:
```
n_row_raw = len(suncountry_data)
print("Number of rows in raw data is: ",n_row_raw)
suncountry_data.drop_duplicates(inplace=True)

n_row_dedupe = len(suncountry_data)
print("Number of rows in after removing duplicates: ",n_row_dedupe)

suncountry_data=suncountry_data[suncountry_data.MarketingAirlineCode=='SY'].
 ↪reset_index(drop=True)
print("Number of rows after keeping only Sun Country Airlines:␣
 ↪",len(suncountry_data))
```

```
Number of rows in raw data is:  3435388
Number of rows in after removing duplicates:  3292499
```

```
Number of rows after keeping only Sun Country Airlines:  3287521
```

Removing rows where we have more than one encrypted name of a single ticket num This was done as we cannot have more than one passenger on one Ticket number; more than one passenger can have the same PNR number but each Ticket Number should have just one passenger

[46]:
```
df_group_tickets= suncountry_data.groupby('TicketNum')

xd=df_group_tickets.agg({'EncryptedName':'nunique','TotalDocAmt':'sum'})
ticket_num_to_rmv= xd[xd['EncryptedName']>1].index.to_list()

suncountry_data = suncountry_data[~suncountry_data.TicketNum.
 ↪isin(ticket_num_to_rmv)]

n_row_ticket = len(suncountry_data)

print("Number of rows after removing anomolous rows: ",n_row_ticket)

n_removed_ticket = n_row_dedupe - n_row_ticket
print("Number of rows removed with multiple passengers on Single Ticket Num: ",␣
 ↪n_removed_ticket)
print("Number of TicketNums removed: ", len(ticket_num_to_rmv))

print("Total rows removed so far: ",n_row_raw - n_row_ticket)
```

```
Number of rows after removing anomolous rows:  3279819
Number of rows removed with multiple passengers on Single Ticket Num:  12680
Number of TicketNums removed:  2024
Total rows removed so far:  155569
```

Some other observations

- A PNRLocatorID can have multiple TicketNum, multiple start and end city combinations, multiple encrypted name Eg: AAAZLM, LHIYVH
- A TicketNum can have multiple PNRLocatorID, multiple start and end city combinations, multiple encrypted name. Eg : 3372107529080
- A person making booking 4 round trips can have CouponSeqNbr ranging from 1-8, on same TicketNum and PNRLocatorID
- Of total 1,874,486 unique ticket numbers, we only have 2,024 tickets having multiple passengers travelling on them

We also observed some cases where the itinary was repeated for a person. These were another set of duplicates that we could eliminate

[47]:
```
repeated_bookings = suncountry_data.
 ↪groupby(['PNRLocatorID','TicketNum','CouponSeqNbr','ServiceStartCity'
```

```
↪,'ServiceEndCity','PNRCreateDate','ServiceStartDate','EncryptedName']).
↪count().reset_index()
dup_list = repeated_bookings[repeated_bookings.TrvldClassOfService>1].TicketNum.
↪to_list()

dup_data =suncountry_data[suncountry_data.TicketNum.isin(dup_list)].
↪sort_values(by=['TicketNum','CouponSeqNbr'])
#Just to check the results of duplicates
dup_data.head(4)
```

[47]:

```
        PNRLocatorID      TicketNum  CouponSeqNbr ServiceStartCity  \
1555618       PMTBTM  3372106145162             1              MSP
1555690       PMTBTM  3372106145162             1              MSP
1555617       PMTBTM  3372106145162             2              MCO
1555689       PMTBTM  3372106145162             2              MCO

        ServiceEndCity PNRCreateDate ServiceStartDate PaxName  \
1555618            MCO    2012-04-26       2013-02-16  JANKLA
1555690            MCO    2012-04-26       2013-02-16  JANKLA
1555617            MSP    2012-04-26       2013-02-23  JANKLA
1555689            MSP    2012-04-26       2013-02-23  JANKLA

                                             EncryptedName GenderCode  \
1555618  4A414E4B4F57534B4944696420493F7C20676574207468…          M
1555690  4A414E4B4F57534B4944696420493F7C20676574207468…          M
1555617  4A414E4B4F57534B4944696420493F7C20676574207468…          M
1555689  4A414E4B4F57534B4944696420493F7C20676574207468…          M

         birthdateid   Age PostalCode BkdClassOfService TrvldClassOfService  \
1555618      47244.0  33.0        NaN             Coach               Coach
1555690      37923.0  58.0        NaN             Coach               Coach
1555617      47244.0  33.0        NaN             Coach               Coach
1555689      37923.0  58.0        NaN             Coach               Coach

               BookingChannel  BaseFareAmt  TotalDocAmt  UFlyRewardsNumber  \
1555618  Reservations Booking       437.94       492.38                NaN
1555690  Reservations Booking       437.94       492.38                NaN
1555617  Reservations Booking       437.94       492.38                NaN
1555689  Reservations Booking       437.94       492.38                NaN

        UflyMemberStatus CardHolder BookedProduct EnrollDate  \
1555618              NaN        NaN           NaN        NaN
1555690              NaN        NaN           NaN        NaN
1555617              NaN        NaN           NaN        NaN
1555689              NaN        NaN           NaN        NaN
```

```
        MarketingFlightNbr MarketingAirlineCode StopoverCode
1555618                341                   SY          NaN
1555690                341                   SY          NaN
1555617                342                   SY            O
1555689                342                   SY            O
```

[48]:
```python
print("Number of rows before removing cases with itinary repeated: "
 ↪,n_row_ticket)

suncountry_data.
 ↪drop_duplicates(subset=['PNRLocatorID','TicketNum','CouponSeqNbr','ServiceStartCity'
 ↪,'ServiceEndCity','PNRCreateDate','ServiceStartDate','EncryptedName'],inplace=True)
n_row_itinary = len(suncountry_data)

print("Number of rows after removing rows with duplicated itinary: "
 ↪,n_row_itinary)
```

```
Number of rows before removing cases with itinary repeated:  3279819
Number of rows after removing rows with duplicated itinary:  3258837
```

Removing records where either Age and Gender are Null

[49]:
```python
print("Number of rows before removing Nulls in Age and GenderCode: "
 ↪,n_row_itinary)
suncountry_data=suncountry_data.dropna(subset=['GenderCode', 'Age']).
 ↪reset_index(drop=True)
n_row_age_gender = len(suncountry_data)
print("Number of rows after removing Nulls in Age and GenderCode: "
 ↪,n_row_age_gender)
```

```
Number of rows before removing Nulls in Age and GenderCode:  3258837
Number of rows after removing Nulls in Age and GenderCode:  3235186
```

Based on our exploration, a combination of EncryptedName and birthdateid could be used as an identifier for a passenger. We are trying to eliminate the few records with missing values

0.4 Creating new columns

We created some new columns to be used as features, while also simultaneously checking if we could eliminate certain rows as a part of tidying the data further

Column 1: Booking Channel This channel showed How the passenger booked the flight. If this is showing a 3 letter code, it's most likely booked at that airport. We wanted to see a pattern of flight booking by the passengers using this column

[61]:
```
suncountry_data['BookingChannel'] = suncountry_data['BookingChannel'].str.
 ↪strip()
suncountry_data['BookingChannel_len'] = suncountry_data['BookingChannel'].
 ↪apply(len)
suncountry_data['BookingChannel_2'] = np.
 ↪where(suncountry_data['BookingChannel']=='UFO', 'Cancun',
                                          np.
 ↪where(suncountry_data['BookingChannel_len']==3,
                                                 ␣
 ↪'Airport',suncountry_data['BookingChannel']))

#suncountry_data[['BookingChannel','BookingChannel_2']].drop_duplicates().
 ↪sort_values(by=['BookingChannel_2','BookingChannel'])
```

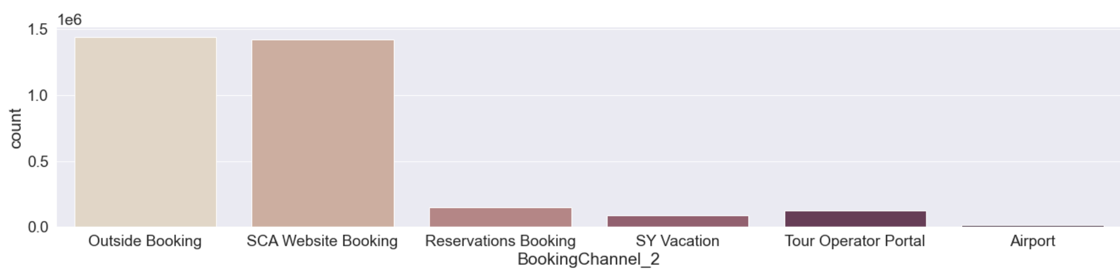
Filtering out rows for "Cancun" in booking channel We did not see any significant driop in the number of rows on dropping "Cancun" as booking channel.

[62]:
```
#Cleaning the records containing cancun

print("Number of rows before removing Cancun as a booking Channel: ",␣
 ↪n_row_age_gender)

suncountry_data = suncountry_data[~suncountry_data.BookingChannel_2.
 ↪isin(['Cancun'])]

n_row_cancun = len(suncountry_data)

print("Number of rows after removing Cancun as a booking Channel: ",␣
 ↪n_row_cancun)
```

```
Number of rows before removing Cancun as a booking Channel:  3235186
Number of rows after removing Cancun as a booking Channel:  3235039
```

[63]:
```
sns.set(font_scale=1.75)
sns.catplot(data=suncountry_data, x="BookingChannel_2", kind="count",␣
 ↪palette="ch:.25",aspect=4)
```

[63]:
```
&lt;seaborn.axisgrid.FacetGrid at 0x7fb7397bbee0&gt;
```



Booking channel distribution now looks interpretable and it seems majority of bookings were made either outside or using website

Column 2 :Encoding Booked product The "BookedProduct" column contained information about the discount code used, we used it to see how many customers booked a flight with a discount code

[64]:
```
suncountry_data['BookedProduct_flag']=np.where(suncountry_data['BookedProduct'].
↪isna(),0,1)
suncountry_data['BookedProduct_flag'].unique()
```

[64]:
```
array([1, 0])
```

0.5 Creating New Features that have the potential to be used in Clustering

For modeling customer behaviour, we created the following columns; deriving from either the columns we just created or the original columns.

[65]:
```
#suncountry_data = suncountry_data.
↪drop(columns=['BookingChannel_len','PaxName','MarketingFlightNbr','MarketingAirlineCode','B
```

Creating a city_dict to map the city strings to numbers and making paired destinations This feaure was not used eventually, but we explored it to check paired destinations, due to increased complexity we dropped using it.

[67]:
```
c1=list(suncountry_data.ServiceStartCity.unique())
c2=list(suncountry_data.ServiceEndCity.unique())
new_c = list(set(c1 + c2))
```

[68]:
```
city_dict = dict(zip(new_c, range(1,len(new_c)+1)))
suncountry_data['start_map']=suncountry_data.ServiceStartCity.map(city_dict)
suncountry_data['end_map']=suncountry_data.ServiceEndCity.map(city_dict)
```

Making a column for paired destinations!! We tried to find unique pairs of origin-destination pairs , this was not very useful in creating clusters, but useful to interpret the clusters later.

[71]:
```
def pair_destinations(c1,c2):
    lst = sorted([c1,c2])
    qw=""
    for person in lst:
        qw+=str(person)
    return qw
```

```
suncountry_data['paired_destinations']= suncountry_data.apply(lambda row:␣
↪pair_destinations(row['ServiceStartCity'],row['ServiceEndCity']) , axis=1)
```

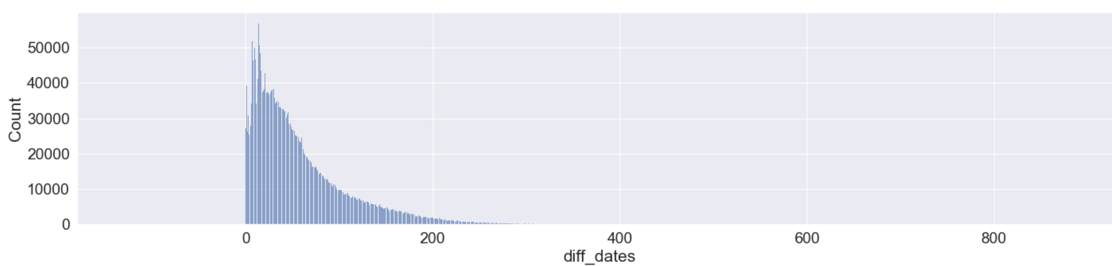
[74]:
```
pair_dest=list(suncountry_data.paired_destinations.unique())
pair_dest_dict = dict(zip(pair_dest, range(1,len(pair_dest)+1)))
suncountry_data['paired_destinations']=suncountry_data['paired_destinations'].
↪map(pair_dest_dict)
```

FEATURE 1: Difference between service date and PNR date The motivation behind creating this variable was to see if there was a meaningful pattern in the customers' flight date and booking date; we wanted to ascertain what kind of passengers were booking the flight ahead in time and how many were booked close to their flight date

[58]:
```
# Creating columns for features
suncountry_data['pnr_dates'] = pd.to_datetime(suncountry_data['PNRCreateDate'],␣
↪format='%Y-%m-%d')
suncountry_data['service_dates'] = pd.
↪to_datetime(suncountry_data['ServiceStartDate'], format='%Y-%m-%d')
suncountry_data['diff_dates']=(suncountry_data['service_dates'] -␣
↪suncountry_data['pnr_dates']).dt.days
```

[60]:
```
sns.set(font_scale=1.75)
sns.displot(data=suncountry_data, x="diff_dates", palette="ch:.25",aspect=4)
```

[60]:
```
<seaborn.axisgrid.FacetGrid at 0x7fb7fec5e610>
```



The distribution of diff_dates looks to be exponentially decreasing, with most data concetrated around the just in time bookings.

FEATURE 2: Making a new column getting the actual flight counts for each passenger

[73]:
```
flight_count_data= pd.DataFrame({'flight_count': suncountry_data.
↪groupby(['EncryptedName','birthdateid']).size()}).reset_index()
flight_count_data.sort_values(by=␣
↪['EncryptedName','birthdateid','flight_count']).head(2)
```

[73]:
```
                                       EncryptedName  birthdateid  \
0  4120414C52484D414E44696420493F7C20676574207468…      47687.0
1  414142454C44696420493F7C2067657420746869732072…      50666.0

   flight_count
0             1
1             1
```

FEATURE 3: Making a new column for a group travel versus solo based on a PNR LocatorID - ordinal variable.

- This feature was created to check for an individual customer - of all the trips taken how many trips were taken solo versus in Groups.
- We anticipated that if any kind of group has majority of solo members or group members then we can tweak our recommendations for the clusters accordingly.
- We used a custom mapping to make three categories for this column. 1- Solo Traveller 2- Travelling in Small Groups 3- Large Groups . Category 2 can indicate towards families - couples/ families traveling together.

[77]:
```
suncountry_data['group_travel']=suncountry_data.groupby(['PNRLocatorID'],
                                                                         ␣
 ↪sort=False)['EncryptedName'].transform(lambda x: x.nunique())
```

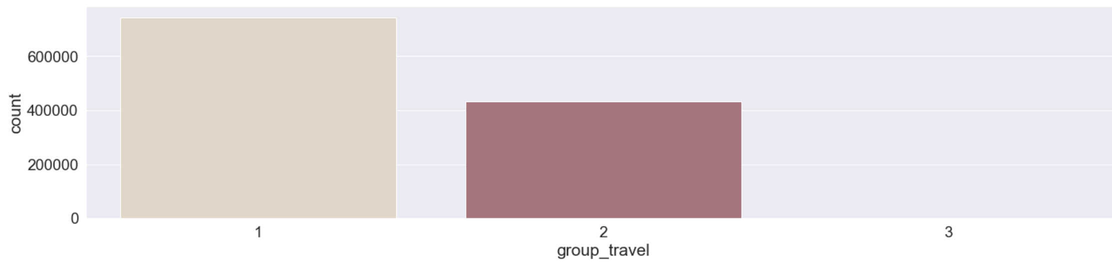
[78]:
```
gt_df=pd.DataFrame(suncountry_data.
 ↪groupby(['PNRLocatorID'],sort=False)['EncryptedName'].nunique().
 ↪reset_index())
```

[79]:
```
gt_df.columns=['PNRLocatorID', 'group_travel']
def map_group_size(x):
    if x==1 :
        return 1
    elif x<= 8 : # small groups
        return 2
    else :
        return 3 #large groups
gt_df['group_travel'] = gt_df['group_travel'].apply(map_group_size)
gt_df['group_travel'] .value_counts()
```

[79]:
```
1    744164
2    434074
3       992
Name: group_travel, dtype: int64
```

[81]:
```
sns.set(font_scale=1.75)
sns.catplot(data=gt_df, x="group_travel",kind='count', palette="ch:.
 ↪25",aspect=4)
```

[81]: <seaborn.axisgrid.FacetGrid at 0x7fb692fe3ee0>



We see people travelling in bigger groups is very low, and we have highest numbers of customers traveling solo.

FEATURE 4: Trip Duration

[80]:
```
td_df=pd.DataFrame(suncountry_data.groupby(['EncryptedName',
                                            'birthdateid','TicketNum'],sort=False)
                             ['diff_dates'].apply(lambda x: x.max() -x.min()).
 ↪reset_index())

td_df.columns=['EncryptedName', 'birthdateid', 'TicketNum', 'trip_duration']
```

FEATURE 5: Whether a passenger had a Round Trip or not

- We created this column to check whether the passengers had any round trips or not, we wanted to use them to see of all the trips that a passenger makes how many does s/he book for a round trip versus one-way.
- We currently categorize them into 4 categories, One way trips, One way Trips with Stops, Round Trip, Round Trip with Stops. We later club them into RT and OW, to find of all the trips a customer makes how many of them are RTs.
- Assumption - We assume that a person books round trip at once and does not one-way first and a return later in time. If a person booked a return at a later time that was not taken care by the following code.

[83]:
```
def check_round_trip(grouped_df):
    start_city=grouped_df.loc[grouped_df.CouponSeqNbr.
 ↪idxmin(),'ServiceStartCity']
    end_city=grouped_df.loc[grouped_df.CouponSeqNbr.idxmax(),'ServiceEndCity']
    max_coupon = grouped_df.CouponSeqNbr.max()

    if max_coupon ==1 :
        return 'OW'
    elif (start_city==end_city) :
        if max_coupon == 2 :
```

```
            return 'RT'
        else :
            return 'RT_with_stops'
    else :
        return 'OW_with_stops'
```

[84]:
```
suncountry_subset=suncountry_data[['EncryptedName',
                                   ␣
 ↪'birthdateid','TicketNum','ServiceStartCity','ServiceEndCity','CouponSeqNbr']]
data_RT= pd.DataFrame(suncountry_subset.
 ↪groupby(['EncryptedName','birthdateid','TicketNum']).
 ↪progress_apply(check_round_trip).reset_index())
```

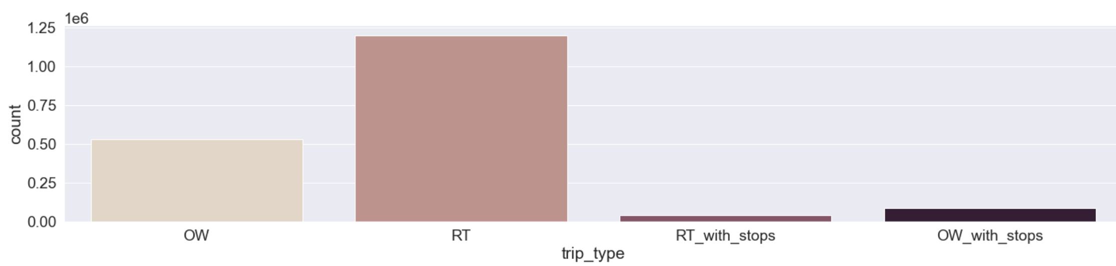
```
100%|                 | 1859548/1859548 [11:17<00:00, 2743.67it/s]
```

[85]:
```
data_RT.columns = ['EncryptedName', 'birthdateid', 'TicketNum', 'trip_type']
data_RT.trip_type.value_counts()
```

[85]:
```
RT                 1201120
OW                  529777
OW_with_stops        89054
RT_with_stops        39597
Name: trip_type, dtype: int64
```

[86]:
```
sns.set(font_scale=1.75)
sns.catplot(data=data_RT, x="trip_type",kind='count', palette="ch:.25",aspect=4)
```

[86]:
```
<seaborn.axisgrid.FacetGrid at 0x7fb73b077610>
```



We observe most of the tickets booked were round trips, and trips with stops were minimal.

Time of the day booking While trying to ascertain the day of booking to ascertain the day of booking and see if there is a pattern, this was considered in hope that we might have some differentiation in customer segements who tend to book on maybe weekends or specifically wednesdays as the flights are cheaper around that.

[87]:
```
suncountry_data['day_of_week']=suncountry_data.pnr_dates.dt.day_of_week
suncountry_data['day_of_week'].value_counts()
```

[87]:
```
1    634389
2    552510
0    516132
3    492623
4    415366
6    337174
5    286845
Name: day_of_week, dtype: int64
```

FEATURE 6: Frequent travel season Q1,Q2,Q3,Q4

- We wanted to analyze the customers in terms of in which season did they take maximum flights, was it in holiday season or non-holiday one. So first we build Q1,Q2,Q3,Q4 as 4 categories to get the season of travel and later categorize them as holiday versus non-holiday travel.

[88]:
```
def travel_time(x):
    if x in [11,12,1] :
        return 'Q4'
    elif x in [8,9,10] :
        return 'Q3'
    elif x in [5,6,7] :
        return 'Q2'
    elif x in [2,3,4] :
        return 'Q1'
```

[91]:
```
suncountry_data['season']=suncountry_data.service_dates.dt.month
suncountry_data['season']=suncountry_data['season'].apply(travel_time)
```

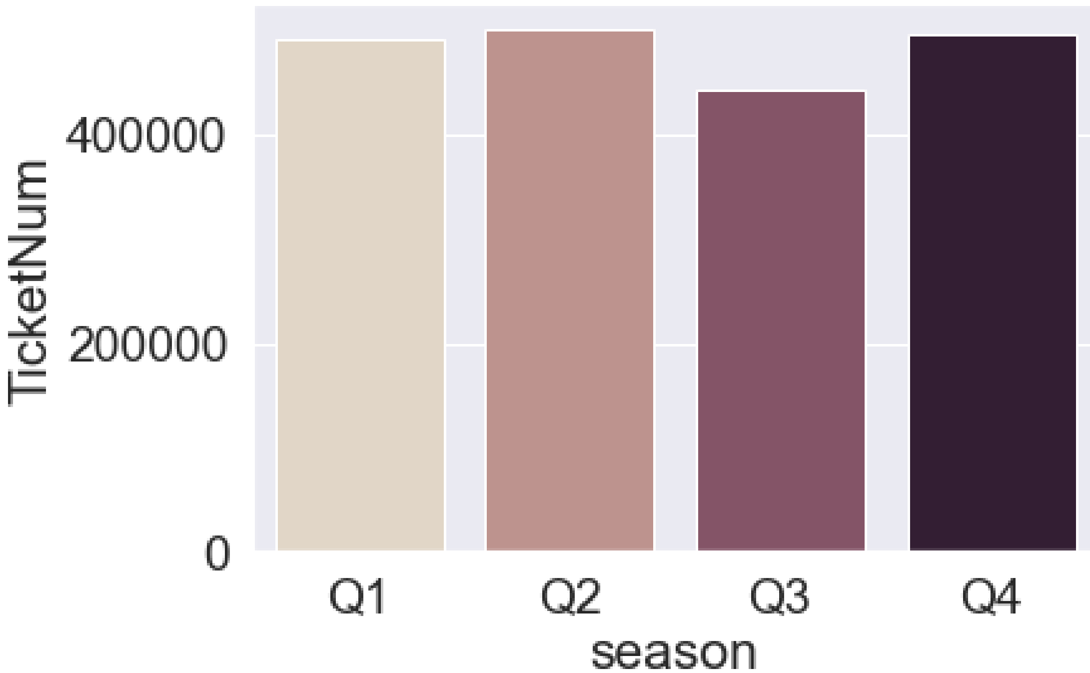
[103]:
```
data_season = pd.DataFrame(suncountry_data.groupby(['season']).agg({'TicketNum':
 ↪'nunique'})).reset_index()
```

[106]:
```
data_season.TicketNum
```

[106]:
```
0    490860
1    500547
2    443073
3    496712
Name: TicketNum, dtype: int64
```

[113]:
```
sns.set(font_scale=1.75)
sns.barplot(data=data_season, x="season",y='TicketNum', palette="ch:.25")
```

[113]: `<AxesSubplot:xlabel='season', ylabel='TicketNum'>`



Interestingly we observed almost equal amount of tickets booked across all quarters

[116]:
```python
loaded_sc = suncountry_data.copy()
```

Aggregating Data at Ticket-encrypted-birthdate level We aggregated the data at TicketNum-EncryptedName-birthdateid level to get their first, mean or min features so that this dataframe can aid us later to get the data at passenger level.

[120]:
```python
aggregated_ticket_data = pd.DataFrame(loaded_sc.groupby(['TicketNum',␣
↪'EncryptedName','birthdateid']).agg({
    'PNRLocatorID':'first','GenderCode':'first','Age':
↪'first','BkdClassOfService':'first','TrvldClassOfService':'first',
    'BaseFareAmt':'mean','TotalDocAmt':'mean', 'UflyMemberStatus':
↪'first','BookingChannel_2':'first',
    'diff_dates':'min','BookedProduct_flag':'first',
    'season':'first','day_of_week':'first'
}).reset_index())
```

We had earlier created the DataFrames that had information about round trips, trip duration and whether the passenger took trips individually or solo. We will now merge these DataFrames with the aggregated ticket data

[121]:
```python
aggregated_ticket_data=aggregated_ticket_data.merge(td_df,on=['EncryptedName',␣
↪'birthdateid', 'TicketNum'], how='left')
```

```python
aggregated_ticket_data=aggregated_ticket_data.merge(gt_df,on='PNRLocatorID',␣
 ↪how='left')
aggregated_ticket_data=aggregated_ticket_data.
 ↪merge(data_RT,on=['EncryptedName', 'birthdateid', 'TicketNum'], how='left')

aggregated_ticket_data.head(5)
```

[121]:

```
       TicketNum                                      EncryptedName  \
0  3372052115142  4D454E44455344696420493F7C20676574207468697320…
1  3372052793801  4E4F474144696420493F7C206765742074686973207269…
2  3372052793802  4E4F474144696420493F7C206765742074686973207269…
3  3372053842388  544F4D4348554B44696420493F7C206765742074686973…
4  3372053842389  544F4D4348554B44696420493F7C206765742074686973…

   birthdateid PNRLocatorID GenderCode   Age BkdClassOfService  \
0      42116.0       JTFZOT          F  47.0             Coach
1      37870.0       BZGKWB          M  59.0             Coach
2      38963.0       BZGKWB          F  56.0             Coach
3      45763.0       GNVPVL          F  37.0             Coach
4      45419.0       GNVPVL          M  38.0             Coach

  TrvldClassOfService  BaseFareAmt  TotalDocAmt UflyMemberStatus  \
0               Coach        258.6       308.18             None
1               Coach        468.0       618.22         Standard
2               Coach        468.0       618.22         Standard
3               Coach        240.0       288.96             None
4               Coach        240.0       288.96             None

  BookingChannel_2  diff_dates  BookedProduct_flag season  day_of_week  \
0  Outside Booking         376                   0     Q2            5
1  Outside Booking         255                   0     Q1            4
2  Outside Booking         255                   0     Q1            4
3  Outside Booking         208                   0     Q1            6
4  Outside Booking         208                   0     Q1            6

   trip_duration  group_travel trip_type
0              4             1        RT
1             28             2        RT
2             28             2        RT
3             13             2        RT
4             13             2        RT
```

Create dummies for categorical columns, because we would now look at them in terms of of all the trips that a customer took how many were RT, how many were in group etc.

[123]:
```
aggregated_ticket_data=pd.get_dummies(aggregated_ticket_data, columns␣
 ↪=['BkdClassOfService',
                                                                    'BookingChannel_2',
                                                                    'season', 'trip_type']
                                     ,drop_first=False)
```

0.5.1 Creating Passenger level data

After cleaning the raw data, we created a master dataset at a passenger level such that it had the basic features we would need for behavioral segmentation, e.g. for the BaseFareAmt, we kept the mean and sum, summed up different trips made on BkdClassOfService etc. at the EncryptedName-birthdateid level

[124]:
```
passenger_level_ns= aggregated_ticket_data.
 ↪groupby(['EncryptedName','birthdateid']).agg({
                                                    'TicketNum':'nunique',
                                                    'GenderCode':'first',
                                                    'Age':'mean',
                                                    'BaseFareAmt':['mean','sum'],␣
 ↪'TotalDocAmt':['sum','mean'],
                                                    'BkdClassOfService_Coach':'sum',
                                                    'BkdClassOfService_Discount First␣
 ↪Class':'sum',
                                                    'BkdClassOfService_First Class':
 ↪'sum',
                                                    'BookingChannel_2_Airport':'sum',
                                                    'BookingChannel_2_Outside Booking':
 ↪'sum',
                                                    'BookingChannel_2_Reservations␣
 ↪Booking':'sum',
                                                    'BookingChannel_2_SCA Website␣
 ↪Booking':'sum',
                                                    'BookingChannel_2_SY Vacation':
 ↪'sum',
                                                    'BookingChannel_2_Tour Operator␣
 ↪Portal':'sum',
                                                    'BookedProduct_flag':'sum',
                                                    'diff_dates':'mean',
                                                    'season_Q1': 'sum',
                                                    'season_Q2': 'sum',
                                                    'season_Q3': 'sum',
                                                    'season_Q4': 'sum',
                                                    'day_of_week': lambda x :x.
 ↪mode()[0],
                                                    'trip_duration':'mean',
                                                    'group_travel': lambda x :x.
 ↪mode()[0],
```

```
                                                   'trip_type_OW': 'sum',
                                                   'trip_type_OW_with_stops': 'sum',
                                                   'trip_type_RT': 'sum',
                                                   'trip_type_RT_with_stops': 'sum',
                                                   'UflyMemberStatus':'first'
                                                   })
```

[125]:
```
passenger_level_ns=passenger_level_ns.reset_index()
passenger_level_ns.columns = ["_".join(col_name).rstrip('_') for col_name in␣
 ↪passenger_level_ns.columns]
```

[126]:
```
cust_data_raw = passenger_level_ns.copy()
```

Outlier Removal Now that we have the customer data, we will do some more "housekeeping" steps, e.g. outlier removal and creating some other columns

[128]:
```
cust_data_1 = cust_data_raw[cust_data_raw['GenderCode_first'].isin(['M', 'F'])]
```

[129]:
```
#Age: keep [0-100]
cust_data_2 =␣
 ↪cust_data_1[(cust_data_1['Age_mean']>=0)&(cust_data_1['Age_mean']<=100)]
```

[130]:
```
cust_data_2['BkdClassOfService_firstclass'] =␣
 ↪cust_data_2['BkdClassOfService_Discount First Class_sum'] +␣
 ↪cust_data_2['BkdClassOfService_First Class_sum']
```

0.6 Enhancing the Customer Level Features

Creating columns for number of trips in the holiday vs non-holiday months

[132]:
```
cust_data_2['non_holiday_sum'] = cust_data_2['season_Q1_sum'] +␣
 ↪cust_data_2['season_Q3_sum']
cust_data_2['holiday_sum'] = cust_data_2['season_Q2_sum'] +␣
 ↪cust_data_2['season_Q4_sum']
```

Columns for the number of bookings done on website vs others

[133]:
```
cust_data_2['website_booking']=cust_data_2['BookingChannel_2_SCA Website␣
 ↪Booking_sum']
cust_data_2['outside_booking']=cust_data_2['BookingChannel_2_Airport_sum']+cust_data_2['Bookin
 ↪Booking_sum']+cust_data_2['BookingChannel_2_Reservations␣
 ↪Booking_sum']+cust_data_2['BookingChannel_2_SY␣
 ↪Vacation_sum']+cust_data_2['BookingChannel_2_Tour Operator Portal_sum']
```

Assigning "Non-member" status to customers with blank UflyMemberstatus

[134]:
```
cust_data_2.UflyMemberStatus_first = cust_data_2.UflyMemberStatus_first.
 ↪fillna("Non-member")
```

Creating columns for Return and one-way flights

[135]:
```
cust_data_2['RT']=␣
↪cust_data_2['trip_type_RT_sum']+cust_data_2['trip_type_RT_with_stops_sum']
cust_data_2['OT']=␣
↪cust_data_2['trip_type_OW_sum']+cust_data_2['trip_type_OW_with_stops_sum']
```

Creating proportions of Flights using different metrics, such as Return vs One way, Website booking vs outside booking It is obvious that for complimentary columns, the sum of proportions should be 1

[136]:
```
cust_data_2['prop_BkdClassOfService_Coach_sum']=cust_data_2['BkdClassOfService_Coach_sum']/
↪cust_data_2['TicketNum_nunique']
cust_data_2['prop_BkdClassOfService_firstclass']=cust_data_2['BkdClassOfService_firstclass']/
↪cust_data_2['TicketNum_nunique']

cust_data_2['prop_website_booking']=cust_data_2['website_booking']/
↪cust_data_2['TicketNum_nunique']
cust_data_2['prop_outside_booking']=cust_data_2['outside_booking']/
↪cust_data_2['TicketNum_nunique']

cust_data_2['prop_holiday_sum'] = cust_data_2['holiday_sum']/
↪cust_data_2['TicketNum_nunique']
cust_data_2['prop_non_holiday_sum'] = cust_data_2['non_holiday_sum']/
↪cust_data_2['TicketNum_nunique']

cust_data_2['prop_RT']=cust_data_2['RT']/cust_data_2['TicketNum_nunique']
cust_data_2['prop_OT']=cust_data_2['OT']/cust_data_2['TicketNum_nunique']
```

[137]:
```
#renaming the columns appropriately
simplified_columns_dict = { 'TicketNum_nunique':'num_flights',
                           'GenderCode_first':'gender',
                           'Age_mean':'age',
                           'BaseFareAmt_mean':'basefare_mean',
                           'BaseFareAmt_sum':'basefare_sum',
                           'TotalDocAmt_sum':'totalamt_sum',
                           'TotalDocAmt_mean':'totalamt_mean',
                           'diff_dates_mean':'date_diff',
                            'BookedProduct_flag_sum':'bookedproduct'}

cust_data_2 = cust_data_2.rename(columns = simplified_columns_dict)
```

[138]:
```
cust_data_2['booked_product_perc']= cust_data_2['bookedproduct']/
↪cust_data_2['num_flights']
cust_data_2['ow_with_stops_perc']= cust_data_2['trip_type_OW_with_stops_sum']/
↪(cust_data_2['trip_type_OW_with_stops_sum']+cust_data_2['trip_type_OW_sum'])
```

```
cust_data_2['passenger_type'] = np.
 ↪where(cust_data_2['group_travel_<lambda>']==1, "Individual","Group")
```

[139]:
```
load_passenger = cust_data_2.copy()
print("Total number of unique customers is:", len(load_passenger))
```

```
Total number of unique customers is: 1515510
```

0.7 Addressing Compute Issues

0.7.1 Sub-Sampling

Now that we have the data at customer level with all the features made, we will sub-sample the data because clustering on ~1.5M unique users is computaionally very expensive. Ideally, the sub-sample should be representative of the original data; i.e. the proportion of rows falling into a particular category in the original data should be the same as that in the new, smaller subset of data.

There were several factors that we wanted to stratify sample on but we found that using the below options gave us reasonable distribution that mimiced the actual data given.

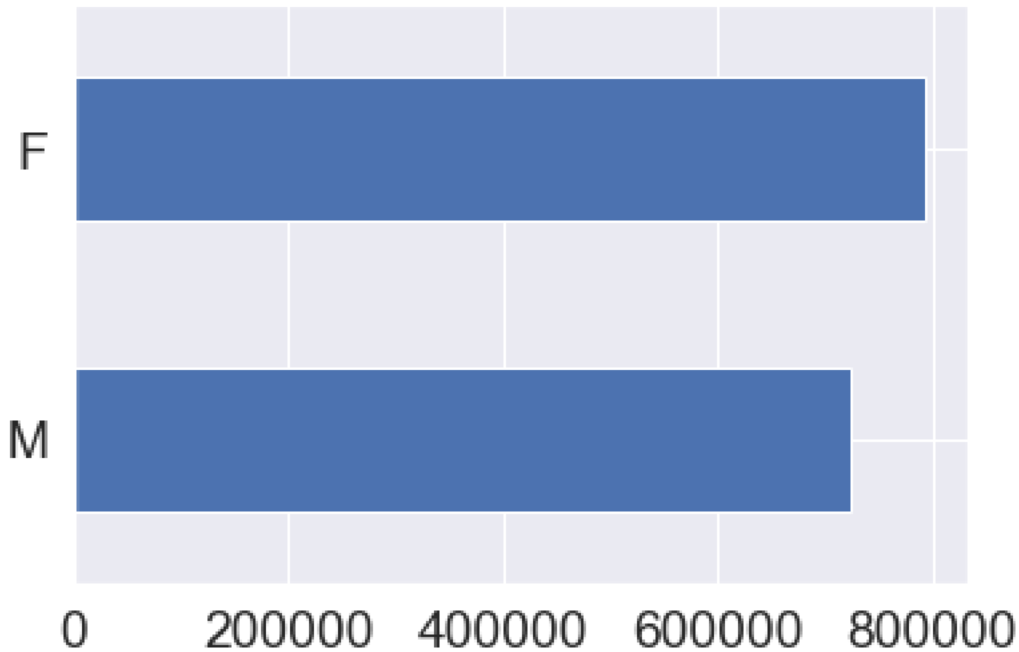
[243]:
```
#Creating a sub-sample of 10% of the data

load_passenger_sampled_final = load_passenger.
 ↪groupby(['age','num_flights','gender',
                                                'group_travel_<lambda>'],
                                                group_keys=False).progress_apply(lambda␣
 ↪x: x.sample(frac=0.1,random_state=1))
```

[206]:
```
load_passenger_sampled_copy=load_passenger_sampled_final.copy()
```

To check for distribution of the columns before and after subsampling

[207]:
```
load_passenger.gender.value_counts().sort_values().plot(kind = 'barh')
```

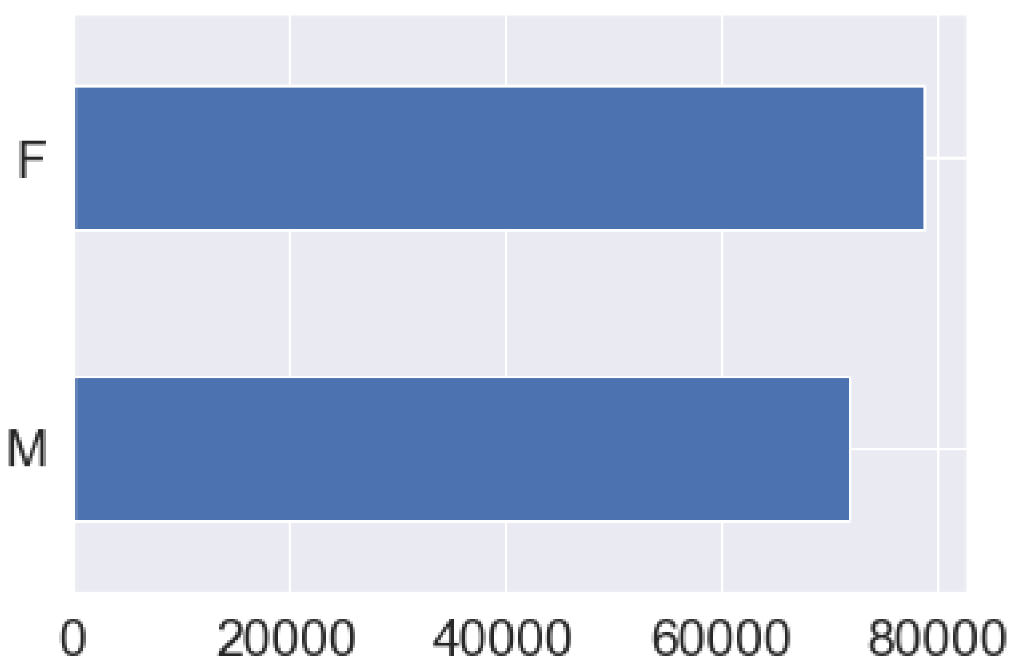
[207]: <AxesSubplot:>



[208]:

```
load_passenger_sampled_final.gender.value_counts().sort_values().plot(kind =␣
↪'barh')
```

[208]: <AxesSubplot:>



After Sub-Sampling the distirbution for gender looks to be similar.

Bi-modal distribution of Age remains intact

[212]:
```
load_passenger[['num_flights']].describe()
```

[212]:

	num_flights
count	1.515510e+06
mean	1.226412e+00
std	8.503878e-01
min	1.000000e+00
25%	1.000000e+00
50%	1.000000e+00
75%	1.000000e+00
max	1.050000e+02

[213]:
```
load_passenger_sampled_final[['num_flights']].describe()
```

[213]:

	num_flights
count	150564.000000
mean	1.184858
std	0.536939
min	1.000000
25%	1.000000
50%	1.000000
75%	1.000000
max	7.000000

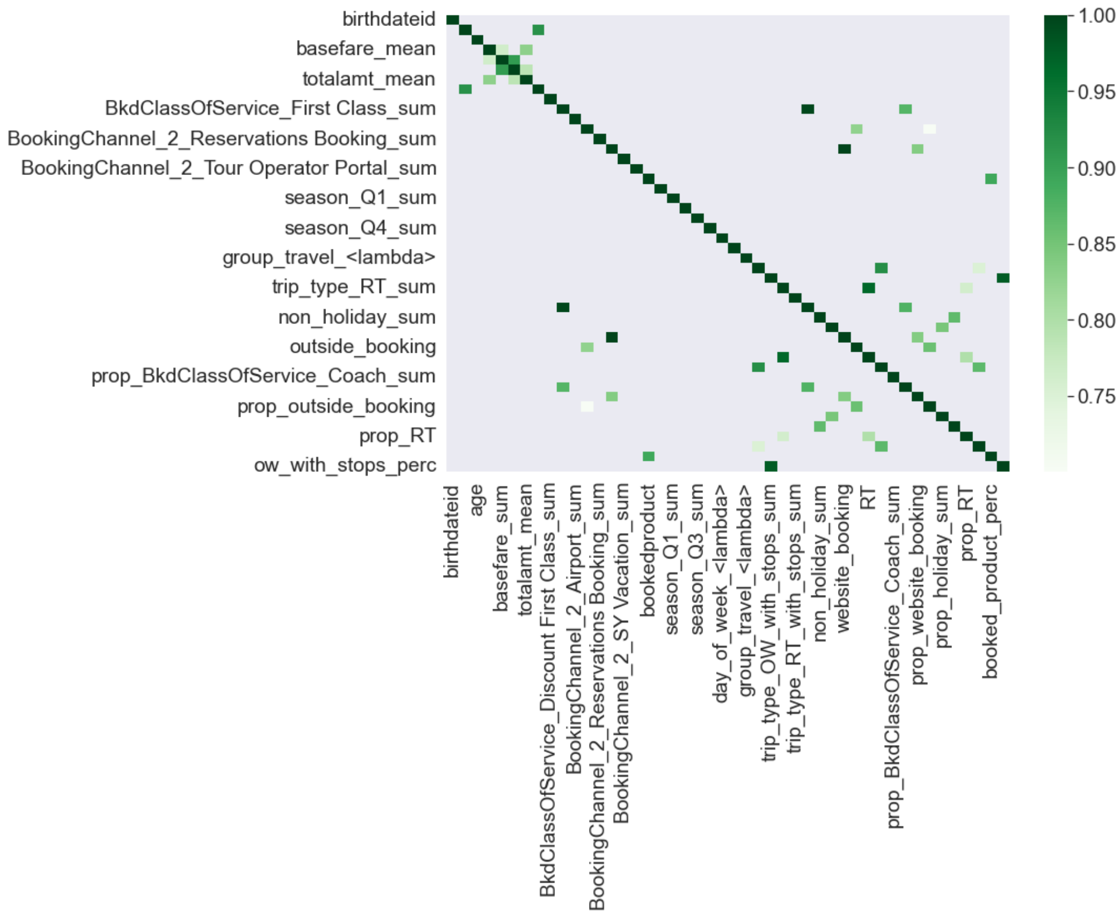
Mean number of flights stay Intact

0.8 Modeling the Sampled Data

[244]:
```
#load_passenger_sampled_final.corr()[load_passenger_sampled_final.corr()>0.8]
```

[246]:
```
corr = load_passenger_sampled_copy.corr()

kot = corr[corr>=.7]
plt.figure(figsize=(12,8))
sns.heatmap(kot, cmap="Greens")
```

[246]: <AxesSubplot:>



We decided to analyze the correlations among the features and moved forward with the below ones. 'UflyMemberStatus_first','age','gender','flight_count','basefare_mean','booked_product_perc', 'date_diff','prop_website_booking','prop_holiday_sum','prop_RT','passenger_type','ow_with_stops_perc', 'prop_BkdClassOfService_firstclass'

[215]:
```
flight_count_data=pd.read_csv('flight_count_data.csv')
load_passenger_sampled_final=pd.
↪merge(load_passenger_sampled_final,flight_count_data,on=['EncryptedName',␣
↪'birthdateid'])
```

[216]:
```
load_passenger_sampled_final=load_passenger_sampled_final[['UflyMemberStatus_first','age','ger
                                                                        ␣
↪'flight_count','basefare_mean','booked_product_perc',
                                                                         'date_diff',
                                                                        ␣
↪'prop_website_booking','prop_holiday_sum',
```

```
↪'prop_RT','passenger_type','ow_with_stops_perc',
↪'prop_BkdClassOfService_firstclass']]
```

[217]:
```
load_passenger_sampled_final['ow_with_stops_perc'] = np.
↪where(load_passenger_sampled_final.ow_with_stops_perc.isna(),
                                                            0.0,
                                              load_passenger_sampled_final.
↪ow_with_stops_perc)
```

[218]:
```
numeric_cols= ['flight_count','basefare_mean','date_diff','age']
```

[219]:
```
len(load_passenger_sampled_final)
```

[219]:
```
150564
```

Scaling the Data

[220]:
```
from sklearn.preprocessing import StandardScaler
load_passenger_sampled_final[numeric_cols] = StandardScaler().
↪fit_transform(load_passenger_sampled_final[numeric_cols])
```

[221]:
```
load_passenger_sampled_final.head()
```

[221]:
```
  UflyMemberStatus_first       age gender  flight_count  basefare_mean  \
0             Non-member -2.071869      F     -0.072047      -0.189513
1             Non-member -2.071869      F     -1.075630      -0.249122
2             Non-member -2.071869      F     -0.072047      -0.303370
3             Non-member -2.071869      F     -0.072047       0.244122
4             Non-member -2.071869      F     -0.072047      -0.525607

   booked_product_perc  date_diff  prop_website_booking  prop_holiday_sum  \
0                  0.0   0.507360                   0.0               0.0
1                  0.0  -0.667978                   0.0               0.0
2                  0.0  -1.040168                   0.0               1.0
3                  0.0  -0.805101                   0.0               0.0
4                  0.0  -0.687567                   0.0               0.0

   prop_RT passenger_type  ow_with_stops_perc  \
0      1.0     Individual                 0.0
1      0.0     Individual                 0.0
2      0.0     Individual                 1.0
3      1.0     Individual                 0.0
4      1.0     Individual                 0.0

   prop_BkdClassOfService_firstclass
```

```
0                                 0.0
1                                 0.0
2                                 0.0
3                                 0.0
4                                 0.0
```

[222]:
```
load_passenger_sampled_final.columns[[0,2,10]]
```

[222]:
```
Index(['UflyMemberStatus_first', 'gender', 'passenger_type'], dtype='object')
```

GMM does not handle categorical data well, K-mediods is very good when it comes to clustering customers as the representative point of the cluster is an actual point. But because of the computational constraints we could not do the calculations optimally even on sampled data so we moved forward with choosing k-prototypes as it gave us the best results in terms of computational efficiency and cluster cohesiveness.

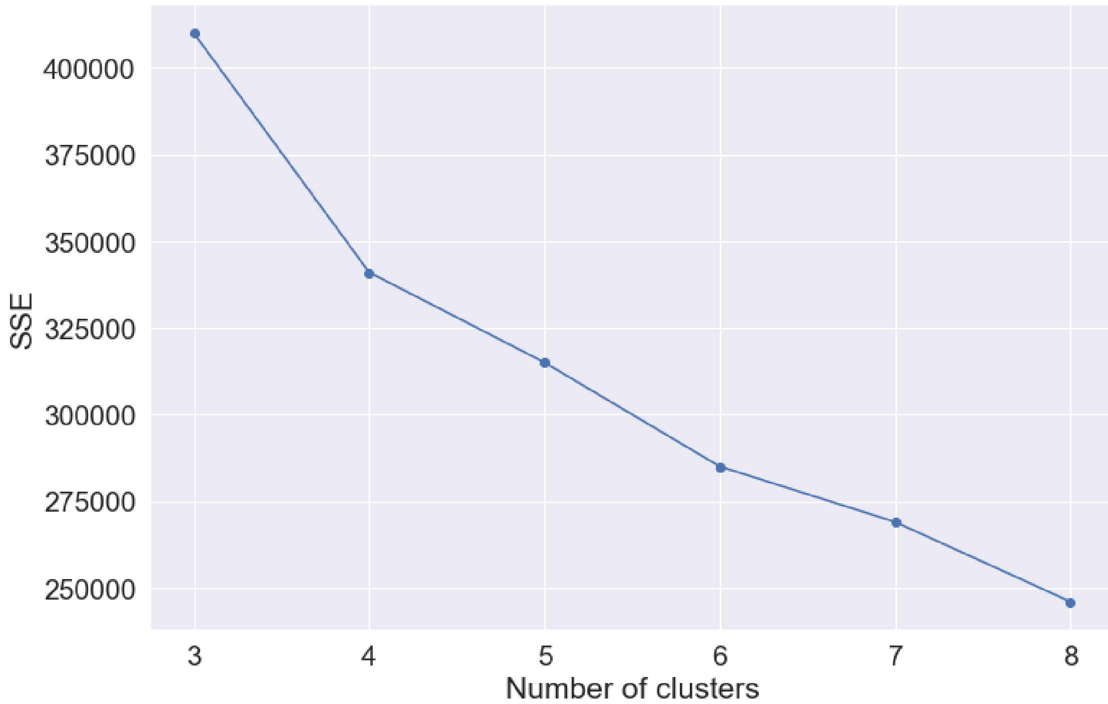
Elbow Curve

[183]:
```
categorical_index=[0,2,10]
```

[182]:
```
# Function for plotting elbow curve
def plot_elbow_curve(start, end, data):
    no_of_clusters = list(range(start, end+1))
    cost_values = []
    cost_values_dict = {}

    for k in no_of_clusters:
        kp_model = KPrototypes(n_clusters=k,random_state=10)
        kp_model.fit_predict(data.values, categorical=categorical_index)
        cost_values.append(kp_model.cost_)
        cost_values_dict[k] = kp_model.cost_

    sns.set_theme(style="whitegrid", palette="bright", font_scale=1.2)
    plt.figure(figsize=(12, 8))
    ax = sns.lineplot(x=no_of_clusters, y=cost_values, marker="o", dashes=False)
    ax.set_title('Elbow Plot', fontsize=18)
    ax.set_xlabel('Number of clusters', fontsize=15)
    ax.set_ylabel('SSE', fontsize=15)
    plt.plot();
```

Looking at above curve we decided to go forward with k=5 and ran the k-prototypes with categorical columns specified.

[252]:
```
plot_elbow_curve(start=3,end=8,data = load_passenger_sampled_final )
```



[178]:
```
import time
start_time = time.time()
```

[179]:
```
from kmodes.kprototypes import KPrototypes
kp_model = KPrototypes(n_clusters=5,random_state=10)
cls_assignment = kp_model.fit_predict(load_passenger_sampled_final.values,␣
↪categorical=[0,2,10])
```

[200]:
```
load_passenger_sampled_final['results']=cls_assignment
load_passenger_sampled_final.to_csv('results_k5.csv')
```

[224]:
```
load_passenger_sampled_final=pd.read_csv('results_k5.csv')
```

[225]:
```
final_result_k5=pd.
↪merge(load_passenger_sampled_copy,load_passenger_sampled_final,left_index=True,right_index='
```

0.9 Cluster Interpretations

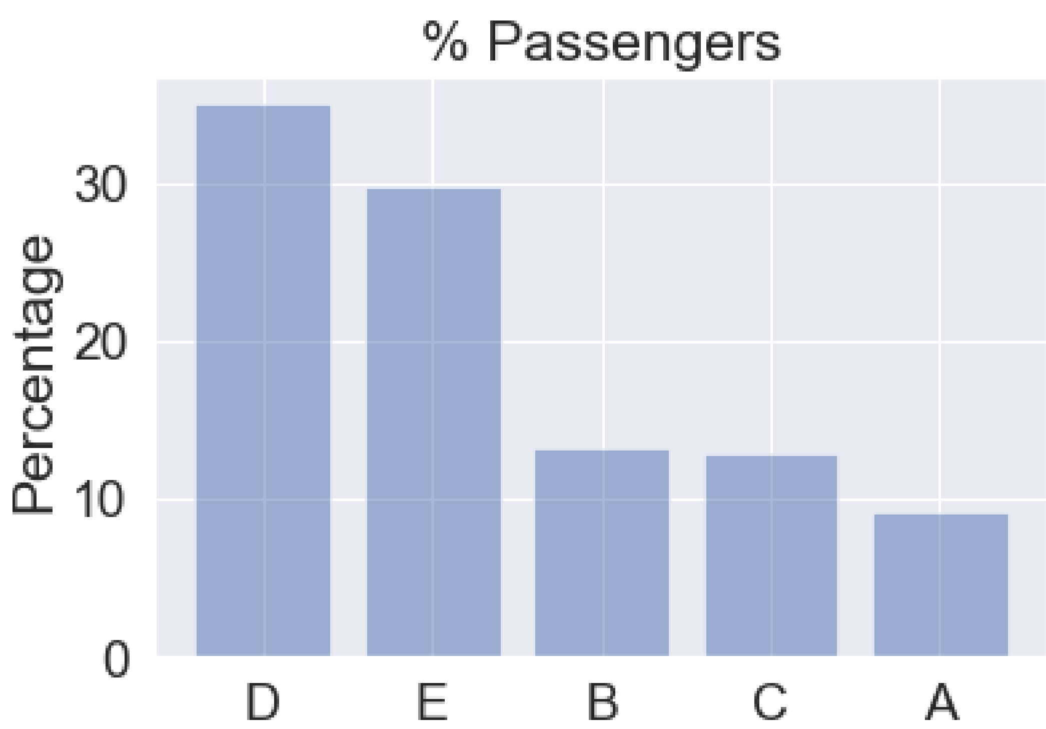
[229]:
```
cluster_data = pd.read_csv("clustering_results_mapped_LK.csv")
```

[230]:
```
cluster_data['cluster'] = np.where(cluster_data['results']==0,'A',
                                   np.where(cluster_data['results']==2,'B',
                                            np.
↪where(cluster_data['results']==3,'C',
```

```
                                                        np.
↪where(cluster_data['results']==1,'D','E'))))
```

Customer Count

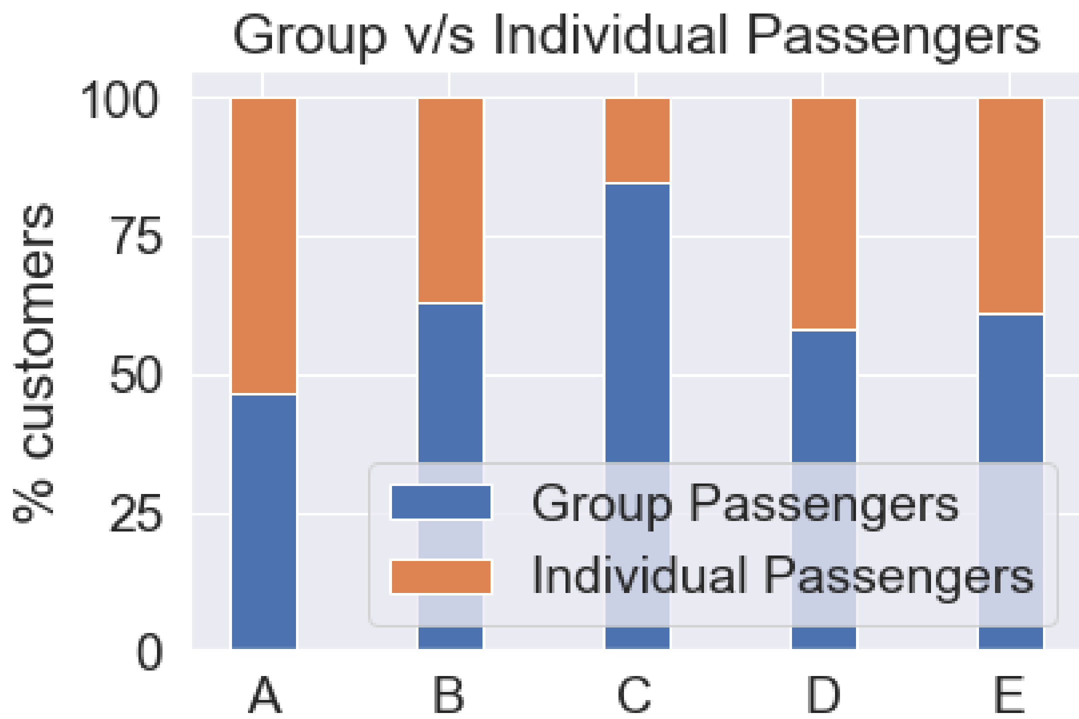
```
[231]: cust_count = pd.DataFrame(cluster_data.cluster.value_counts()).reset_index()
cust_count.columns = ['cluster','count_pass']
cust_count['perc'] = round(cust_count['count_pass']*100/
 ↪cust_count['count_pass'].sum(),2)
cust_count

y_pos = np.arange(len(cust_count))
performance = list(cust_count['perc'])

plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, list(cust_count['cluster']))
plt.ylabel('Percentage')
plt.title('% Passengers')


plt.show()
```



Group v/s Individual Passengers

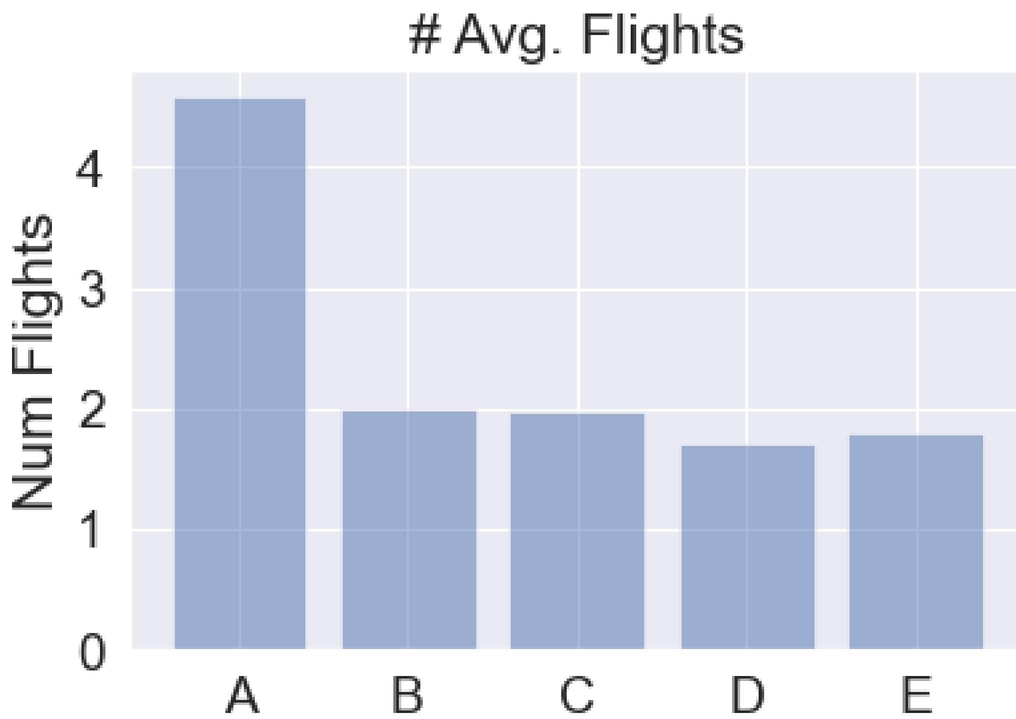
[240]:

```python
cluster_data.passenger_type_x.unique()
cluster_data['pass'] = 1
pass_type = cluster_data.groupby(['cluster','passenger_type_x']).agg({'pass':'sum'}).reset_index()
pass_type2 = pass_type.merge(cust_count[['cluster','count_pass']], how = "left", on = ['cluster'])
pass_type2['perc_pass'] = round(pass_type2['pass']*100/pass_type2['count_pass'],2)

labels = list(pass_type2['cluster'].unique())
group_pass = list(pass_type2[pass_type2['passenger_type_x']=='Group']['perc_pass'].unique())
individual_pass = list(pass_type2[pass_type2['passenger_type_x']=='Individual']['perc_pass'].unique())

width = 0.35       # the width of the bars: can also be len(x) sequence

fig, ax = plt.subplots()

ax.bar(labels, group_pass, width, label='Group Passengers')
ax.bar(labels, individual_pass, width,  bottom=group_pass,
       label='Individual Passengers')

ax.set_ylabel('% customers')
ax.set_title('Group v/s Individual Passengers')
ax.legend(loc='lower right')

plt.show()
```



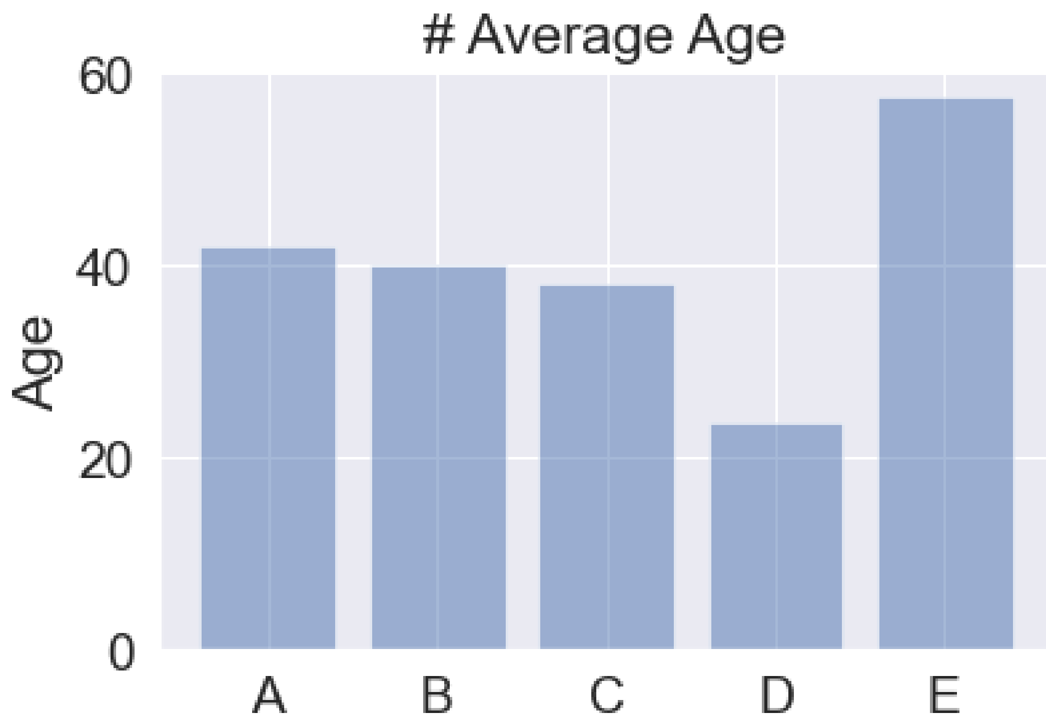
Average Number of Flyers

[236]:
```
avg_flights = cluster_data.groupby(['cluster']).agg({'flight_count_x':'mean'}).
    ↪reset_index()
avg_flights['flight_count_x'] = round(avg_flights['flight_count_x'],2)
y_pos = np.arange(len(avg_flights))
performance = list(avg_flights['flight_count_x'])

plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, list(avg_flights['cluster']))
plt.ylabel('Num Flights')
plt.title('# Avg. Flights')

plt.show()
```



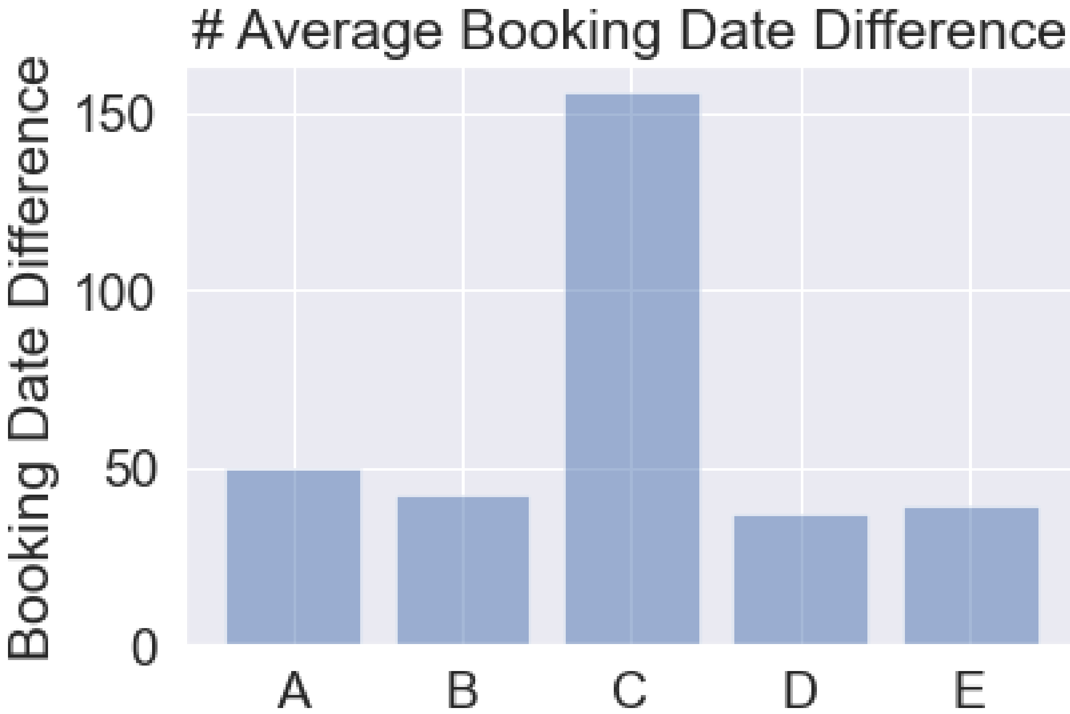
Average Age (Years)

[241]:
```
avge_age = cluster_data.groupby(['cluster']).agg({'age_x':'mean'}).reset_index()
avge_age['age_x'] = round(avge_age['age_x'],2)

y_pos = np.arange(len(avge_age))
performance = list(avge_age['age_x'])

plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, list(avge_age['cluster']))
plt.ylabel('Age')
plt.title('# Average Age')

plt.show()
```



Average Booking Date Difference

[242]:
```
avg_date_diff = cluster_data.groupby(['cluster']).agg({'date_diff_x':'mean'}).
    ↪reset_index()
avg_date_diff['date_diff_x'] = round(avg_date_diff['date_diff_x'],2)

y_pos = np.arange(len(avg_date_diff))
performance = list(avg_date_diff['date_diff_x'])

plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, list(avg_date_diff['cluster']))
plt.ylabel('Booking Date Difference')
plt.title('# Average Booking Date Difference')

plt.show()
```



Cluster 0 :

Fly 2.5 times others

Mostly Ufly Members (40%)

Mostly book return tickets (86%)

Cluster 1 & 4 :

Low Value Customers (1&4)

Young (D: 30%) and Old (E: 35%)

Book last-minute - Others book 2.2 times earlier

Book low-price – Others pay 1.9 times Base Fare than these

Cluster 2 :

Pay 2.1 times Base Fare than others

Book 6.6 times First Class than others

Mostly book return tickets (90%)

Cluster 3:

Book 3.7 times earlier than others

Mostly Group Travelers (85%)

Mostly book return tickets (86%)

```
[ ]:
```