Week 7 Assignment

DSC 630

Bellevue University

Yograj Karki

DSC630, Assignment 7.2: Create Optimal Hotel Recommendations

Online travel agencies are scrambling to meet the artificial intelligence driven personalization standard set by companies like Amazon and Netflix. In addition, the world of online travel has become a highly competitive space where brands try to capture our attention (and wallet) with recommending, comparing, matching, and sharing. For this assignment, we would like to create the optimal hotel recommendations for Expedia's users that are searching for a hotel to book. For this assignment, you need to predict which "hotel cluster" the user is likely to book, given his (or her) search details.

The data set can be found at Kaggle: Expedia Hotel Recommendations. To get started, I would suggest exploring the file train.csv, which contains the logs of user behavior. There is another file named destinations.csv, which contains information related to hotel reviews made by users. There is a lot of data here, and making an accurate prediction is rather difficult, e.g., simply running a standard prediction algorithm will probably yield below 10% accuracy. Stary by doing some exploratory analysis of this data to help understand how to make a prediction on the hotel cluster the user is likely to select. Then, split train.csv into a training and test set (feel free to select a smaller random subset of train.csv). Then, build at least two prediction models from the training set, and report the accuracies on the test set. As I mentioned, this is a difficult problem, so be creative with your solutions. You might want to try building your own predictor rather than a standard predictor model, e.g., a random forest. The purpose of this project is not necessarily to get great results but to understand the nuances and challenges of such problems.

Exploratory Data Analysis

In [1]:
```
# Loading Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# read CSV file
# filename = 'expedia-hotel-recommendations/train.csv'
# df = pd.read_csv(file)

# count no. of lines
# print("Number of lines in the file: ", len(df))
```

Sampling the rows of the csv file since it's very large (taking 6.7 GB+ memory)

In [3]:
```
import random
random.seed(7)
import random

n = 37670293 #number of records in file
s = 100000 # arbitrary sample size
filename = "expedia-hotel-recommendations/train.csv"
skip = sorted(random.sample(range(1, n+1), n-s))
df = pd.read_csv(filename, skiprows=skip)
```

In [4]:
```
df.shape
```

Out[4]:
```
(100000, 24)
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 24 columns):
 #   Column                     Non-Null Count   Dtype  
---  ------                     --------------   -----  
 0   date_time                  100000 non-null  object 
 1   site_name                  100000 non-null  int64  
 2   posa_continent             100000 non-null  int64  
 3   user_location_country      100000 non-null  int64  
 4   user_location_region       100000 non-null  int64  
 5   user_location_city         100000 non-null  int64  
 6   orig_destination_distance  64092 non-null   float64
 7   user_id                    100000 non-null  int64  
 8   is_mobile                  100000 non-null  int64  
 9   is_package                 100000 non-null  int64  
 10  channel                    100000 non-null  int64  
 11  srch_ci                    99899 non-null   object 
 12  srch_co                    99899 non-null   object 
 13  srch_adults_cnt            100000 non-null  int64  
 14  srch_children_cnt          100000 non-null  int64  
 15  srch_rm_cnt                100000 non-null  int64  
 16  srch_destination_id        100000 non-null  int64  
 17  srch_destination_type_id   100000 non-null  int64  
 18  is_booking                 100000 non-null  int64  
 19  cnt                        100000 non-null  int64  
 20  hotel_continent            100000 non-null  int64  
 21  hotel_country              100000 non-null  int64  
 22  hotel_market               100000 non-null  int64  
 23  hotel_cluster              100000 non-null  int64  
dtypes: float64(1), int64(20), object(3)
memory usage: 18.3+ MB
```

In [6]:
```
df.head()
```

Out[6]:

	date_time	site_name	posa_continent	user_location_country	user_location_region	user_location_city	orig_destination_distance	user_id	is_
0	2014-11-23 17:17:42	30	4	195	991	47725	NaN	1048	
1	2014-05-26 12:22:03	2	3	66	174	21356	2103.8393	7523	
2	2013-04-02 11:09:50	24	2	3	50	5703	NaN	9616	
3	2014-09-12 12:14:54	2	3	66	348	48862	3864.2730	10106	
4	2014-08-26 07:53:13	2	3	66	322	48585	224.9042	14864	

5 rows × 24 columns

In [7]:
```
# Drop any rows containing null values
df.dropna(axis=0, inplace=True)
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 64037 entries, 1 to 99999
Data columns (total 24 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   date_time                  64037 non-null  object 
 1   site_name                  64037 non-null  int64  
 2   posa_continent             64037 non-null  int64  
 3   user_location_country      64037 non-null  int64  
 4   user_location_region       64037 non-null  int64  
 5   user_location_city         64037 non-null  int64  
 6   orig_destination_distance  64037 non-null  float64
 7   user_id                    64037 non-null  int64  
 8   is_mobile                  64037 non-null  int64  
 9   is_package                 64037 non-null  int64  
 10  channel                    64037 non-null  int64  
 11  srch_ci                    64037 non-null  object 
 12  srch_co                    64037 non-null  object 
 13  srch_adults_cnt            64037 non-null  int64  
 14  srch_children_cnt          64037 non-null  int64  
 15  srch_rm_cnt                64037 non-null  int64  
 16  srch_destination_id        64037 non-null  int64  
 17  srch_destination_type_id   64037 non-null  int64  
 18  is_booking                 64037 non-null  int64  
 19  cnt                        64037 non-null  int64  
 20  hotel_continent            64037 non-null  int64  
 21  hotel_country              64037 non-null  int64  
 22  hotel_market               64037 non-null  int64  
 23  hotel_cluster              64037 non-null  int64  
dtypes: float64(1), int64(20), object(3)
memory usage: 12.2+ MB
```

Graph Analysis

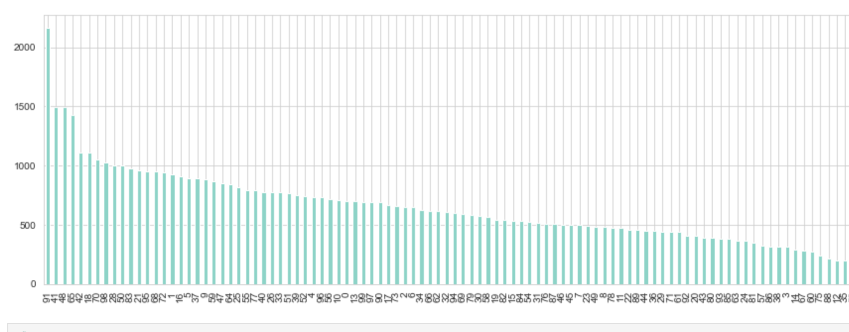
In [9]:
```
# Load Libraries
import matplotlib.pyplot as plt
import seaborn as sns
```

In [10]:
```
# Set up chart style
sns.set_style("whitegrid")

# Set up the figure size
%matplotlib inline
plt.rcParams['figure.figsize'] = (10, 5)
```

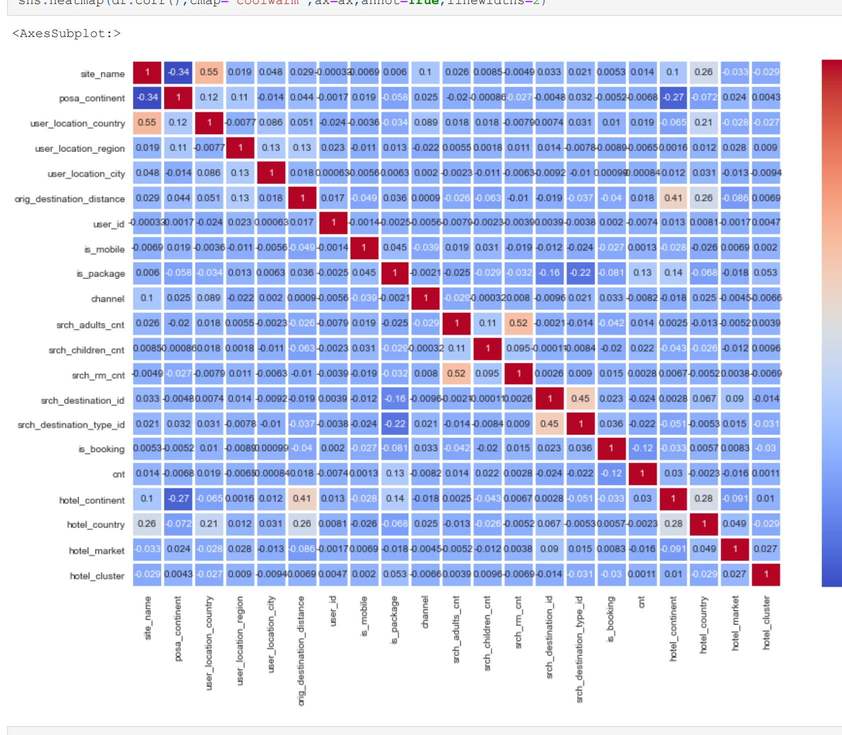
In [11]:
```
# Plot frequency for each hotel_clusters
df["hotel_cluster"].value_counts().plot(kind='bar',colormap="Set3",figsize=(15,5))
```

Out[11]:
```
<AxesSubplot:>
```



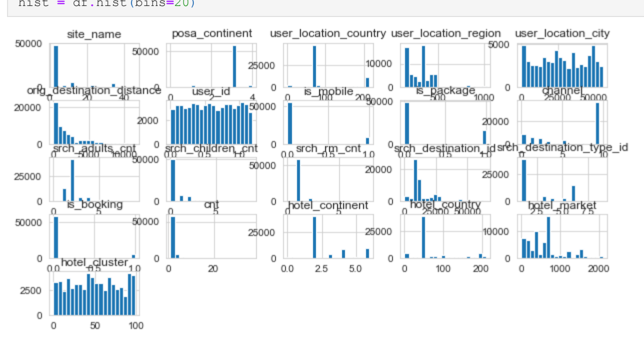
In [12]:
```
# heatmap
fig, ax = plt.subplots()
fig.set_size_inches(15, 10)
sns.heatmap(df.corr(),cmap='coolwarm',ax=ax,annot=True,linewidths=2)
```

Out[12]:
```
<AxesSubplot:>
```



In [ ]:

In [13]:
```
# plot histograms
hist = df.hist(bins=20)
```



Feature Engineering

In [14]:
```
# Dropping the user_id column as it has no significance for the model
df = df.drop(['user_id'], axis=1)
```

In [15]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 64037 entries, 1 to 99999
Data columns (total 23 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   date_time                  64037 non-null  object 
 1   site_name                  64037 non-null  int64  
 2   posa_continent             64037 non-null  int64  
 3   user_location_country      64037 non-null  int64  
 4   user_location_region       64037 non-null  int64  
 5   user_location_city         64037 non-null  int64  
 6   orig_destination_distance  64037 non-null  float64
 7   is_mobile                  64037 non-null  int64  
 8   is_package                 64037 non-null  int64  
 9   channel                    64037 non-null  int64  
 10  srch_ci                    64037 non-null  object 
 11  srch_co                    64037 non-null  object 
 12  srch_adults_cnt            64037 non-null  int64  
 13  srch_children_cnt          64037 non-null  int64  
 14  srch_rm_cnt                64037 non-null  int64  
 15  srch_destination_id        64037 non-null  int64  
 16  srch_destination_type_id   64037 non-null  int64  
 17  is_booking                 64037 non-null  int64  
 18  cnt                        64037 non-null  int64  
 19  hotel_continent            64037 non-null  int64  
 20  hotel_country              64037 non-null  int64  
 21  hotel_market               64037 non-null  int64  
 22  hotel_cluster              64037 non-null  int64  
dtypes: float64(1), int64(19), object(3)
memory usage: 11.7+ MB
```

I'm going to create some additional features from date columns

- stay_dur: number of duration of stay
- Cin_day: Check-in day
- Cin_month: Check-in month
- Cin_year: Check-out year

In [16]:
```
def convert_date_into_days(df):
    df['srch_ci'] = pd.to_datetime(df['srch_ci'])
    df['srch_co'] = pd.to_datetime(df['srch_co'])
    df['date_time'] = pd.to_datetime(df['date_time'])

    ## New columns
    df['stay_dur'] = (df['srch_co'] - df['srch_ci']).astype('timedelta64[D]')
    # For hotel check-in
    # Month, Year, Day
    df['Cin_day'] = df["srch_ci"].apply(lambda x: x.day)
    df['Cin_month'] = df["srch_ci"].apply(lambda x: x.month)
    df['Cin_year'] = df["srch_ci"].apply(lambda x: x.year)
```

In [17]:
```
# Applying the function
convert_date_into_days(df)
```

In [18]:
```
df.head(1)
```

Out[18]:

	date_time	site_name	posa_continent	user_location_country	user_location_region	user_location_city	orig_destination_distance	is_mobile
1	2014-05-26 12:22:03	2	3	66	174	21356	2103.8393	1

1 rows × 27 columns

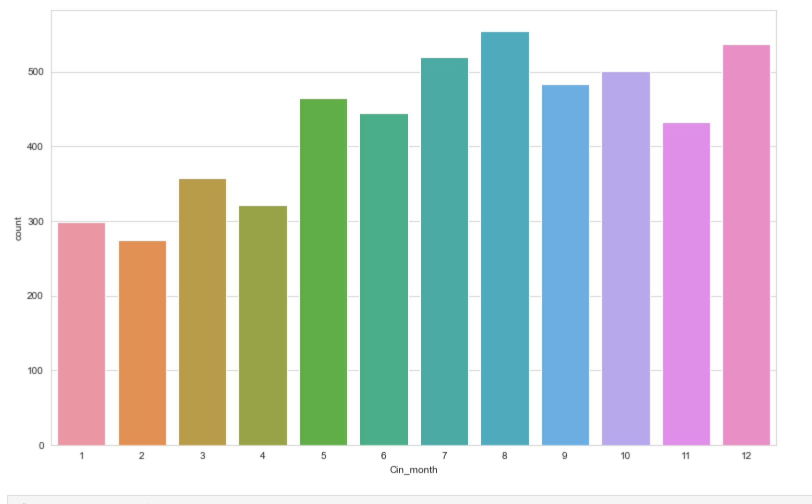
In [19]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 64037 entries, 1 to 99999
Data columns (total 27 columns):
 #   Column                     Non-Null Count  Dtype         
---  ------                     --------------  -----         
 0   date_time                  64037 non-null  datetime64[ns]
 1   site_name                  64037 non-null  int64         
 2   posa_continent             64037 non-null  int64         
 3   user_location_country      64037 non-null  int64         
 4   user_location_region       64037 non-null  int64         
 5   user_location_city         64037 non-null  int64         
 6   orig_destination_distance  64037 non-null  float64       
 7   is_mobile                  64037 non-null  int64         
 8   is_package                 64037 non-null  int64         
 9   channel                    64037 non-null  int64         
 10  srch_ci                    64037 non-null  datetime64[ns]
 11  srch_co                    64037 non-null  datetime64[ns]
 12  srch_adults_cnt            64037 non-null  int64         
 13  srch_children_cnt          64037 non-null  int64         
 14  srch_rm_cnt                64037 non-null  int64         
 15  srch_destination_id        64037 non-null  int64         
 16  srch_destination_type_id   64037 non-null  int64         
 17  is_booking                 64037 non-null  int64         
 18  cnt                        64037 non-null  int64         
 19  hotel_continent            64037 non-null  int64         
 20  hotel_country              64037 non-null  int64         
 21  hotel_market               64037 non-null  int64         
 22  hotel_cluster              64037 non-null  int64         
 23  stay_dur                   64037 non-null  float64       
 24  Cin_day                    64037 non-null  int64         
 25  Cin_month                  64037 non-null  int64         
 26  Cin_year                   64037 non-null  int64         
dtypes: datetime64[ns](3), float64(2), int64(22)
memory usage: 13.7 MB
```

In [20]:
```
# Count the bookings in each month
fig, ax = plt.subplots()
fig.set_size_inches(13, 8)
sns.countplot('Cin_month',data=df[df["is_booking"] == 1],order=list(range(1,13)),ax=ax)
```

```
/Users/ykarki1/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the fol
lowing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and
passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

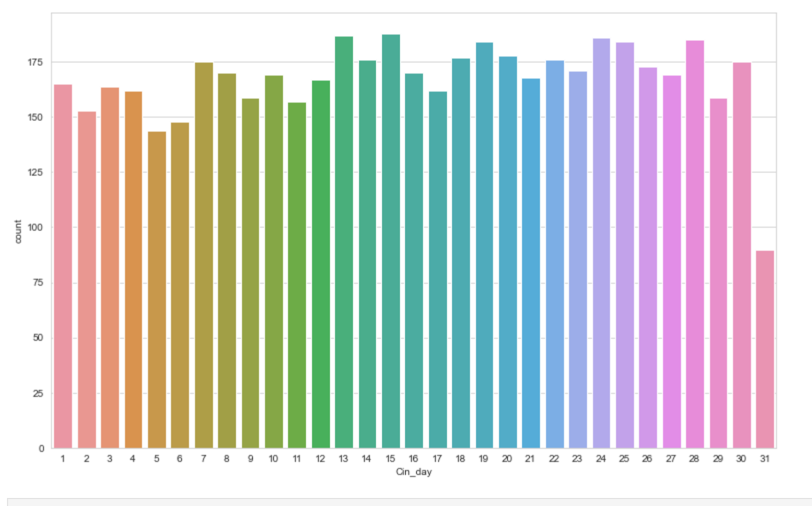
Out[20]:
```
<AxesSubplot:xlabel='Cin_month', ylabel='count'>
```



In [21]:
```
# Count the bookings as per the day
fig, ax = plt.subplots()
fig.set_size_inches(13, 8)
sns.countplot('Cin_day',data=df[df["is_booking"] == 1],order=list(range(1,32)),ax=ax)
```

```
/Users/ykarki1/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the fol
lowing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and
passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

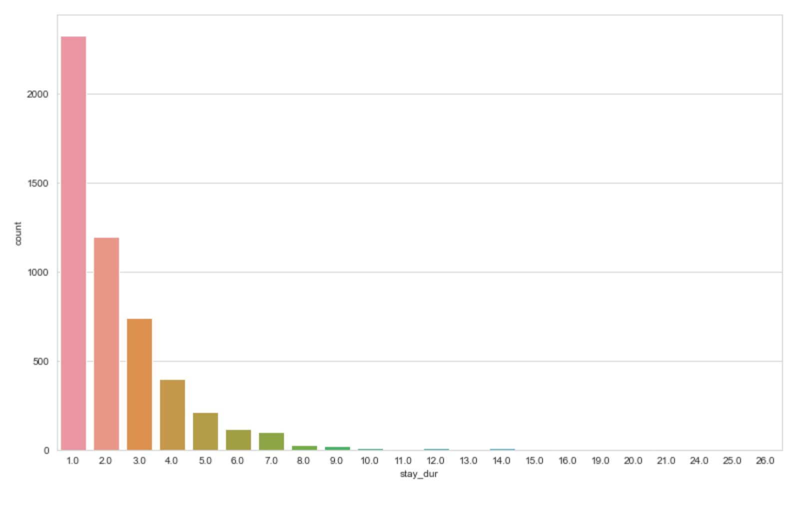
Out[21]:
```
<AxesSubplot:xlabel='Cin_day', ylabel='count'>
```



In [22]:
```
# Count the bookings as per the stay_duration
fig, ax = plt.subplots()
fig.set_size_inches(13, 8)
sns.countplot('stay_dur',data=df[df["is_booking"] == 1],ax=ax)
```

```
/Users/ykarki1/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the fol
lowing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and
passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[22]:
```
<AxesSubplot:xlabel='stay_dur', ylabel='count'>
```



Missing values

In [23]:
```
# Check the percentage of NaN in dataset
#total = df.isnull().sum().sort_values(ascending=False)
#percent = (df.isnull().sum()/df['hotel_cluster'].count()).sort_values(ascending=False)*100
#missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
#missing_data
```

In [24]:
```
# Fill nan with the day which has max occurence
#df['Cin_day'] = df['Cin_day'].fillna(26.0)
#df['Cin_month'] = df['Cin_month'].fillna(8.0)
#df['Cin_year'] = df['Cin_year'].fillna(2014.0)
#df['stay_dur'] = df['stay_dur'].fillna(1.0)

# Fill average values in place for nan, fill with mean
#df['orig_destination_distance'].fillna(df['orig_destination_distance'].mean(), inplace=True)
```

In [25]:
```
# Get list of categorical variable columns
catCols = ['site_name', 'posa_continent', 'user_location_country',
           'user_location_region', 'user_location_city',
           'channel', 'srch_destination_id', 'srch_destination_type_id',
           'hotel_continent', 'hotel_country', 'hotel_market']
```

In [26]:
```
# Function to convert to categorical
def to_category(col, df=df):
    df[col] = df[col].astype('category')

for col in catCols:
    to_category(col)
```

In [27]:
```
# Converts a column to binary based on matching / not matching value
def cat_to_binary(row, col, val):
    if row[col] == val:
        return 1
    return 0


# Converts anything over a particular value to a certain value
def bin_vals(row, col, val):
    if row[col] > val:
        return val
    return row[col]
```

In [28]:
```
# Data Transformations and new feature creation
df['site_name_2'] = df.apply(lambda row: cat_to_binary(row, 'site_name', 2), axis=1)
df['posa_continent_3'] = df.apply(lambda row: cat_to_binary(row, 'posa_continent', 3), axis=1)
df['user_location_country_66'] = df.apply(lambda row: cat_to_binary(row, 'user_location_country', 66), axis=1)
df['user_location_region'] = df.apply(lambda row: bin_vals(row, 'user_location_region', 500), axis=1)
df['hotel_country'] = df.apply(lambda row: cat_to_binary(row, 'hotel_country', 50), axis=1)
```

In [29]:
```
# Look at variables
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 64037 entries, 1 to 99999
Data columns (total 30 columns):
 #   Column                     Non-Null Count  Dtype         
---  ------                     --------------  -----         
 0   date_time                  64037 non-null  datetime64[ns]
 1   site_name                  64037 non-null  category      
 2   posa_continent             64037 non-null  category      
 3   user_location_country      64037 non-null  category      
 4   user_location_region       64037 non-null  int64         
 5   user_location_city         64037 non-null  category      
 6   orig_destination_distance  64037 non-null  float64       
 7   is_mobile                  64037 non-null  int64         
 8   is_package                 64037 non-null  int64         
 9   channel                    64037 non-null  category      
 10  srch_ci                    64037 non-null  datetime64[ns]
 11  srch_co                    64037 non-null  datetime64[ns]
 12  srch_adults_cnt            64037 non-null  int64         
 13  srch_children_cnt          64037 non-null  int64         
 14  srch_rm_cnt                64037 non-null  int64         
 15  srch_destination_id        64037 non-null  category      
 16  srch_destination_type_id   64037 non-null  category      
 17  is_booking                 64037 non-null  int64         
 18  cnt                        64037 non-null  int64         
 19  hotel_continent            64037 non-null  category      
 20  hotel_country              64037 non-null  int64         
 21  hotel_market               64037 non-null  category      
 22  hotel_cluster              64037 non-null  int64         
 23  stay_dur                   64037 non-null  float64       
 24  Cin_day                    64037 non-null  int64         
 25  Cin_month                  64037 non-null  int64         
 26  Cin_year                   64037 non-null  int64         
 27  site_name_2                64037 non-null  int64         
 28  posa_continent_3           64037 non-null  int64         
 29  user_location_country_66   64037 non-null  int64         
dtypes: category(9), datetime64[ns](3), float64(2), int64(16)
memory usage: 12.1 MB
```

In [30]:
```
df.columns
```

Out[30]:
```
Index(['date_time', 'site_name', 'posa_continent', 'user_location_country',
       'user_location_region', 'user_location_city',
       'orig_destination_distance', 'is_mobile', 'is_package', 'channel',
       'srch_ci', 'srch_co', 'srch_adults_cnt', 'srch_children_cnt',
       'srch_rm_cnt', 'srch_destination_id', 'srch_destination_type_id',
       'is_booking', 'cnt', 'hotel_continent', 'hotel_country', 'hotel_market',
       'hotel_cluster', 'stay_dur', 'Cin_day', 'Cin_month', 'Cin_year',
       'site_name_2', 'posa_continent_3', 'user_location_country_66'],
      dtype='object')
```

In [31]:
```
# Load libraries
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [32]:
```
final_df = df[['user_location_region', 'user_location_city',
        'orig_destination_distance', 'is_mobile', 'is_package', 'channel',
        'srch_adults_cnt', 'srch_children_cnt',
        'srch_rm_cnt', 'srch_destination_id', 'srch_destination_type_id',
        'is_booking', 'cnt', 'hotel_continent', 'hotel_country', 'hotel_market',
        'hotel_cluster', 'stay_dur', 'Cin_day', 'Cin_month', 'Cin_year',
        'site_name_2', 'posa_continent_3', 'user_location_country_66']].copy()
```

Split data for feature selection:

In [33]:
```
# Set up features target sets
X = final_df.drop(['hotel_cluster'], axis=1)
y = final_df['hotel_cluster']

# Encode the target variables
le = LabelEncoder()
y = le.fit_transform(y)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =0.25, random_state=7)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test  = scaler.transform(X_test)
```

Feature Selection

In [34]:
```
# Feature selection using F-statistic
from sklearn.feature_selection import f_classif
f = f_classif(X_train, y_train)[0]
```

In [35]:
```
# Feature selection using mutual information
from sklearn.feature_selection import mutual_info_classif
mi = mutual_info_classif(X_train, y_train)
```

In [36]:
```
# Feature selection using logistic regression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(max_iter=500).fit(X_train, y_train)
```

In [37]:
```
# Feature selection using light gbm
from lightgbm import LGBMClassifier
lgbm = LGBMClassifier(
    objective = 'multiclass',
    metric = 'multi_logloss',
    importance_type = 'gain'
).fit(X_train, y_train)
```

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
<ipython-input-37-73703950729a> in <module>
      1 # Feature selection using light gbm
----> 2 from lightgbm import LGBMClassifier
      3 lgbm = LGBMClassifier(
      4     objective = 'multiclass',
      5     metric = 'multi_logloss',

~/opt/anaconda3/lib/python3.8/site-packages/lightgbm/__init__.py in <module>
      6 from pathlib import Path
      7 
----> 8 from .basic import Booster, Dataset, Sequence, register_logger
      9 from .callback import early_stopping, log_evaluation, print_evaluation, record_evaluation, reset_parame
ter
     10 from .engine import CVBooster, cv, train

~/opt/anaconda3/lib/python3.8/site-packages/lightgbm/basic.py in <module>
    108 
    109 
--> 110 _LIB = _load_lib()
    111 
    112 

~/opt/anaconda3/lib/python3.8/site-packages/lightgbm/basic.py in _load_lib()
     99     if len(lib_path) == 0:
    100         return None
--> 101     lib = ctypes.cdll.LoadLibrary(lib_path[0])
    102     lib.LGBM_GetLastError.restype = ctypes.c_char_p
    103     callback = ctypes.CFUNCTYPE(None, ctypes.c_char_p)

~/opt/anaconda3/lib/python3.8/ctypes/__init__.py in LoadLibrary(self, name)
    457 
    458     def LoadLibrary(self, name):
--> 459         return self._dlltype(name)
    460 
    461 cdll = LibraryLoader(CDLL)

~/opt/anaconda3/lib/python3.8/ctypes/__init__.py in __init__(self, name, mode, handle, use_errno, use_last_erro
r, winmode)
    379 
    380         if handle is None:
--> 381             self._handle = _dlopen(self._name, mode)
    382         else:
    383             self._handle = handle

OSError: dlopen(/Users/ykarki1/opt/anaconda3/lib/python3.8/site-packages/lightgbm/lib_lightgbm.so, 6): Library
 not loaded: /usr/local/opt/libomp/lib/libomp.dylib
  Referenced from: /Users/ykarki1/opt/anaconda3/lib/python3.8/site-packages/lightgbm/lib_lightgbm.so
  Reason: image not found
```

Store Results

In [ ]:
```
# Create DF to store feature ranking info
ranking = pd.DataFrame(index = range(X_train.shape[1]))

# Store ranking info for each feature from each method
ranking['feat'] = X.columns
ranking['f'] = pd.Series(f, index = ranking.index).fillna(0).rank(ascending = False)
ranking['mi'] = pd.Series(mi, index = ranking.index).fillna(0).rank(ascending = False)
ranking['logreg'] = pd.Series(np.abs(logreg.coef_).mean(axis = 0), index = ranking.index).rank(ascending = Fals
ranking['lightgbm'] = pd.Series(lgbm.feature_importances_, index = ranking.index).rank(ascending = False)
# ranking['mrmr'] = pd.Series(list(range(1, len(mrmr) + 1)) + [X_train.shape[1]] * (X_train.shape[1] - len(mrmr

ranking = ranking.replace(to_replace = ranking.max(), value = X_train.shape[1])
ranking.to_csv('ranking.csv', index = False)
```

Compare

In [ ]:
```
# Sum the rankings across methods
ranking['total'] = ranking.sum(axis=1)
# Sort by sum to get an overall idea of the most/least useful features
ranking.sort_values('total')
```

In [ ]:
```
# Get the highest ranked Features
hiRank = ranking[ranking['total'] <= 110].sort_values('total')
```

In [ ]:
```
# Get a list of the highest ranked features
hiRankList = hiRank['feat'].tolist()
# Fix that weird 53 that popped up in there
hiRankList = np.array(hiRankList)
hiRankList = np.where(hiRankList == '53', 'user_location_region', hiRankList)
# Convert array back to list
hiRankList = hiRankList.tolist()
```

In [ ]:
```
# Combine top 15 from correlation and features into one list of columns
combined = list(set(hiCorrList[:15]).union(set(hiRankList[:15])))
combined
```

Run modeling using dataframes with high feature rank, high correlation, and a mix of both:

In [ ]:
```
# Create a dataframe for use in modeling
# modelDF = ranDF[combined]
modelDF = ranDF[hiRankList]
# modelDF = ranDF[hiCorrList]
```

In [ ]:
```
# Save data
modelDF.to_csv('modelDF100k.csv', index=False)
```

In [ ]:
```
# load saved data
modelDF = pd.read_csv('modelDF100k.csv')
ranDF = pd.read_csv('ranDF.csv')
```

Split train / test data

In [ ]:
```
# Set up features target sets
X = modelDF # check back for hicorr?
y = ranDF.hotel_cluster

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =0.3,
                                                    random_state=42)

# Standardize Features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test  = scaler.transform(X_test)
```

In [ ]:

Build Baseline Prediction Models

In [ ]:
```
# Set up dictionary for model results
models_dict = {}
```

Logistic Regression

In [ ]:
```
# Load libraries
from sklearn.linear_model import LogisticRegression

# Create classification model
logistic = LogisticRegression(random_state=42,
                              n_jobs=-1)
```

In [ ]:
```
# Fit model
models_dict['Logistic'] = logistic.fit(X_train, y_train)
```

Random Forest

In [ ]:
```
# Load libraries
from sklearn.ensemble import RandomForestClassifier

# Create classification model
rfclassifier = RandomForestClassifier(random_state=42,
                                      n_jobs=-1)
```

In [ ]:
```
# Fit model
models_dict['RandomForest'] = rfclassifier.fit(X_train, y_train)
```

Decision Tree

In [ ]:
```
# Load libraries
from sklearn.tree import DecisionTreeClassifier

# Create classification model
decisiontree = DecisionTreeClassifier(random_state=42,
                                      class_weight='balanced')
```

In [ ]:
```
# Fit model
models_dict['DecisionTree'] = decisiontree.fit(X_train, y_train)
```

AdaBoost

In [ ]:
```
# Load libraries
from sklearn.ensemble import AdaBoostClassifier

# Create classification model
adaboost = AdaBoostClassifier(random_state=42)
```

In [ ]:
```
# Fit model
models_dict['AdaBoost'] = adaboost.fit(X_train, y_train)
```

Support Vector Classifier (SVC)

In [ ]:
```
# Load libraries
from sklearn.svm import SVC

# Create classification model
svc = SVC(random_state=42)
```

In [ ]:
```
# Fit model
models_dict['SVC'] = svc.fit(X_train, y_train)
```

Gaussian Naive Bayes Classifier

In [ ]:
```
# Load libraries
from sklearn.naive_bayes import GaussianNB

# Create Gaussian naive Bayes object
nBayes = GaussianNB()
```

In [ ]:
```
# Fit model
models_dict['GaussianNB'] = nBayes.fit(X_train, y_train)
```

MLPClassifier

In [ ]:
```
# Load libraries
from sklearn.neural_network import MLPClassifier

# Create classification model
mlp = MLPClassifier()
```

In [ ]:
```
# Fit model
models_dict['MLP'] = mlp.fit(X_train, y_train)
```

In [ ]:

Compare Accuracies

In [ ]:
```python
# Create lists of models and accuracy scores
modelName = []
score = []
for key in models_dict:
    modelName += [key]
    model = models_dict[key]
    value = model.score(X_test, y_test)*100
    score += [round(value, 2)]

# Create DataFrame of results
d = {'Model': modelName, 'Accuracy': score}
results = pd.DataFrame(d).sort_values(by=['Accuracy'], ascending=False)
results
```

The Decision Tree model performed best with highly correlated variables, all others performed as good or better with highly ranked features, with the mix landing in-between.

In [ ]:

In [ ]:
```python
# Set up dictionary for model results
models_dict = {}
```

The following were tuned using initial random sample of 100,000 records:

Decision Tree

In [ ]:
```python
# Load libraries
from sklearn.tree import DecisionTreeClassifier

# Create decision tree regressor object
decisiontree = DecisionTreeClassifier(random_state=42)

# Get a baseline model
baseline = decisiontree.fit(X_train, y_train)

# Create range of candidate penalty hyperparameter values
parameter_space = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [None, 5, 8, 10],
    'min_samples_split': [2, 4, 8],
    'min_samples_leaf': [1, 2, 4],
    'max_leaf_nodes': [30, 100, 300, 1000, None],
    'class_weight': ['balanced', None],
    'max_features': ['sqrt', 'log2', None],
}
grid = GridSearchCV(decisiontree, parameter_space, verbose=2, n_jobs=-1, cv=5)
```

In [ ]:
```python
# Fit models
grid_result = grid.fit(X_train, y_train)
```

In [ ]:
```python
# Show best parameters
print('Best parameters found:\n', grid_result.best_params_, '\n')
# Get accuracy scores
baseScore = round(baseline.score(X_test, y_test)*100, 2)
score = round(grid_result.score(X_test, y_test)*100, 2)
print(f"Baseline Accuracy:\t{baseScore}")
print(f"Tuned Accuracy:\t\t{score}")
```

AdaBoost

In [ ]:
```python
# Load libraries
from sklearn.ensemble import AdaBoostClassifier

# Create classifier
adaboost = AdaBoostClassifier(random_state=42)

# Get a baseline model
baseline = adaboost.fit(X_train, y_train)

# Create range of candidate penalty hyperparameter values
parameter_space = {
    'n_estimators': [100, 300, 1000, 3000],
    'learning_rate': [0.1, 1, 10],
    'algorithm': ['SAMME', 'SAMME.R'],
}
grid = GridSearchCV(adaboost, parameter_space, n_jobs=-1, cv=5)
```

In [ ]:
```python
# Fit models
grid_result = grid.fit(X_train, y_train)
```

In [ ]:
```python
# Show best parameters
print('Best parameters found:\n', grid_result.best_params_, '\n')
# Get accuracy scores
baseScore = round(baseline.score(X_test, y_test)*100, 2)
score = round(grid_result.score(X_test, y_test)*100, 2)
print(f"Baseline Accuracy:\t{baseScore}")
print(f"Tuned Accuracy:\t\t{score}")
```

Random Forest

In [ ]:
```python
# Load libraries
from sklearn.ensemble import RandomForestClassifier

# Create classifier
rfc = RandomForestClassifier(random_state=42,
                             n_jobs=-1)

# Get a baseline model
baseline = rfc.fit(X_train, y_train)

# Create range of candidate penalty hyperparameter values
parameter_space = {
    'n_estimators': [30, 100, 300, 1000],
    'criterion': ['gini', 'entropy'],
    'max_features': ['sqrt', 'log2', None],
}
grid = GridSearchCV(rfc, parameter_space, n_jobs=-1, cv=5)
```

In [ ]:
```python
# Fit models
grid_result = grid.fit(X_train, y_train)
```

In [ ]:
```python
# Show best parameters
print('Best parameters found:\n', grid_result.best_params_, '\n')
# Get accuracy scores
baseScore = round(baseline.score(X_test, y_test)*100, 2)
score = round(grid_result.score(X_test, y_test)*100, 2)
print(f"Baseline Accuracy:\t{baseScore}")
print(f"Tuned Accuracy:\t\t{score}")
```

Regression

In [ ]:
```python
# Load libraries
from sklearn.linear_model import LogisticRegression

# Create logistic regression
logistic = LogisticRegression(random_state=42,
                              max_iter=300,
                              multi_class='auto',
                              n_jobs=-1)

# Get a baseline model
baseline = logistic.fit(X_train, y_train)

# Create range of candidate penalty hyperparameter values
parameter_space = {
    'penalty': ['l1', 'l2', 'elasticnet', 'none'],
    'tol': [1e-2, 1e-3, 1e-4],
    'C': np.logspace(0, 2, 3),
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
}
grid = GridSearchCV(logistic, parameter_space, n_jobs=-1, cv=5)
```

In [ ]:
```python
# Fit models
grid_result = grid.fit(X_train, y_train)
```

In [ ]:
```python
# Show best parameters
print('Best parameters found:\n', grid_result.best_params_, '\n')
# Get accuracy scores
baseScore = round(baseline.score(X_test, y_test)*100, 2)
score = round(grid_result.score(X_test, y_test)*100, 2)
print(f"Baseline Accuracy:\t{baseScore}")
print(f"Tuned Accuracy:\t\t{score}")
```

Support Vector Classifier (SVC)

In [ ]:
```python
# Load libraries
from sklearn.svm import SVC

# Create support vector classifier
svc = SVC(random_state=42,
          max_iter=10000)

# Get a baseline model
baseline = svc.fit(X_train, y_train)

# Create range of candidate penalty hyperparameter values
parameter_space = {
    'C': [1, 3, 10, 30, 100],
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'gamma': ['scale', 'auto'],
    'shrinking': [True, False],
    'tol': [1e-3, 1e-4, 1e-5],
    'decision_function_shape': ['ovo', 'ovr'],
    'break_ties': [True],
}
grid = GridSearchCV(svc, parameter_space, n_jobs=-1, cv=5)
```

In [ ]:
```python
# Fit models
grid_result = grid.fit(X_train, y_train)
```

In [ ]:
```python
# Show best parameters
print('Best parameters found:\n', grid_result.best_params_, '\n')
# Get accuracy scores
baseScore = round(baseline.score(X_test, y_test)*100, 2)
score = round(grid_result.score(X_test, y_test)*100, 2)
print(f"Baseline Accuracy:\t{baseScore}")
print(f"Tuned Accuracy:\t\t{score}")
```

Modeling with Tuned Parameters

In [ ]:
```python
# Set up dictionary for model results
models_dict = {}
```

Logistic Regression

In [ ]:
```python
# Load libraries
from sklearn.linear_model import LogisticRegression

# Create classification model
logistic = LogisticRegression(C=1.0,
                              penalty='l1',
                              solver='liblinear',
                              tol=0.001,
                              random_state=42,
                              n_jobs=-1)
```

In [ ]:
```python
# Fit model
models_dict['Logistic'] = logistic.fit(X_train, y_train)
```

Random Forest

In [ ]:
```python
# Load libraries
from sklearn.ensemble import RandomForestClassifier

# Create classification model
rfclassifier = RandomForestClassifier(criterion='entropy',
                                      max_features=None,
                                      n_estimators=1000,
                                      random_state=42,
                                      n_jobs=-1)
```

In [ ]:
```python
# Fit model
models_dict['RandomForest'] = rfclassifier.fit(X_train, y_train)
```

Decision Tree

In [ ]:
```python
# Load libraries
from sklearn.tree import DecisionTreeClassifier

# Create classification model
decisiontree = DecisionTreeClassifier(class_weight=None,
                                      criterion='entropy',
                                      max_depth=None,
                                      max_features=None,
                                      max_leaf_nodes=300,
                                      min_samples_leaf=1,
                                      min_samples_split=2,
                                      splitter='best',
                                      random_state=42,)
```

In [ ]:
```python
# Fit model
models_dict['DecisionTree'] = decisiontree.fit(X_train, y_train)
```

AdaBoost

In [ ]:
```python
# Load libraries
from sklearn.ensemble import AdaBoostClassifier

# Create classification model
adaboost = AdaBoostClassifier(algorithm='SAMME',
                              learning_rate=1,
                              n_estimators=1000,
                              random_state=42)
```

In [ ]:
```python
# Fit model
models_dict['AdaBoost'] = adaboost.fit(X_train, y_train)
```

Support Vector Classifier (SVC)

In [ ]:
```python
# Load libraries
from sklearn.svm import SVC

# Create classification model
svc = SVC(C=1,
          break_ties=True,
          decision_function_shape='ovr',
          gamma='scale',
          kernel='rbf',
          shrinking=True,
          tol=0.001,
          random_state=42)
```

In [ ]:
```python
# Fit model
models_dict['SVC'] = svc.fit(X_train, y_train)
```

Gaussian Naive Bayes Classifier

In [ ]:
```python
# Load libraries
from sklearn.naive_bayes import GaussianNB

# Create Gaussian naive Bayes object
nBayes = GaussianNB(var_smoothing=0.1)
```

In [ ]:
```python
# Fit model
models_dict['GaussianNB'] = nBayes.fit(X_train, y_train)
```

MLPClassifier

In [ ]:
```python
# Load libraries
from sklearn.neural_network import MLPClassifier

# Create classification model
mlp = MLPClassifier(tol=1e-5,
                    solver='adam',
                    learning_rate_init=0.01,
                    hidden_layer_sizes=(32, 68, 24),
                    epsilon=1e-9,
                    early_stopping=True,
                    alpha=1e-5,
                    activation='logistic')
```

In [ ]:
```python
# Fit model
models_dict['MLP'] = mlp.fit(X_train, y_train)
```

In [ ]:

Compare Accuracies

In [ ]:
```python
# Create lists of models and accuracy scores
modelName = []
score = []
for key in models_dict:
    modelName += [key]
    model = models_dict[key]
    value = model.score(X_test, y_test)*100
    score += [round(value, 2)]

# Create DataFrame of results
d = {'Model': modelName, 'Accuracy': score}
results = pd.DataFrame(d).sort_values(by=['Accuracy'], ascending=False)
results
```

In [ ]: