Contents

1. Loading data and package

```
import numpy as np
import pandas as pd
import datetime as dt
import matplotlib.pyplot as plt
import seaborn as sns
```

```
train=pd.read_csv('train.csv')
```

```
train.shape
```

```
(841115, 47)
```

```
X_test=pd.read_csv('test.csv')
```

```
X_test.shape
```

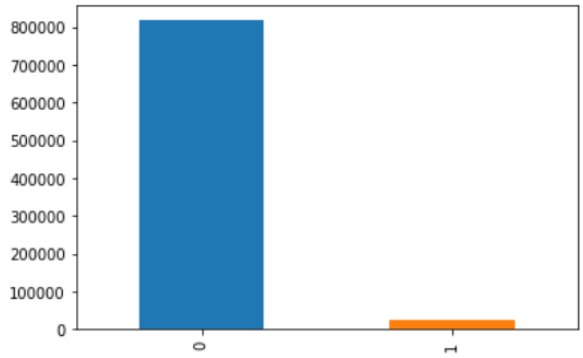
```
(351544, 47)
```

2. Exploration data analysis

I plot the number of records of both booking and non-booking in bar chart. Booking records occupy **2.85%.** Test set doesn't have booking label which is the result I need to predict from model. The class distribution is imbalanced, I decided to downsample the non-booking records in machine learning section to re-balance the class distribution.

```
train['prop_booking_bool'].value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x23897c32a90>
```



Let's get first intuition about which attribute correlated most with booking. I list 20 most correlated features. Apart from ID, it is reasonable that day_deal , review, date, price, brand and starratting are quite important for customer decision, so these features will be analysed in the EDA process.

```
correlation=train.corr()['prop_booking_bool'].abs().sort_values(ascending=False)
correlation[:20]
```

```
prop_booking_bool                   1.000000
prop_dotd_bool                      0.051169
prop_review_count                   0.049420
prop_room_capacity                  0.021394
srch_mobile_bool                    0.018481
prop_imp_drr                        0.015859
prop_brand_bool                     0.007988
srch_dest_longitude                 0.006189
prop_review_score                   0.005910
srch_hcom_destination_id            0.004373
srch_children_cnt                   0.004145
prop_key                            0.003361
srch_visitor_visit_nbr              0.003230
prop_market_id                      0.002608
prop_starrating                     0.002591
prop_price_without_discount_local   0.002273
srch_rm_cnt                         0.002247
srch_co_day                         0.002182
srch_los                            0.002135
srch_ci_day                         0.002060
Name: prop_booking_bool, dtype: float64
```

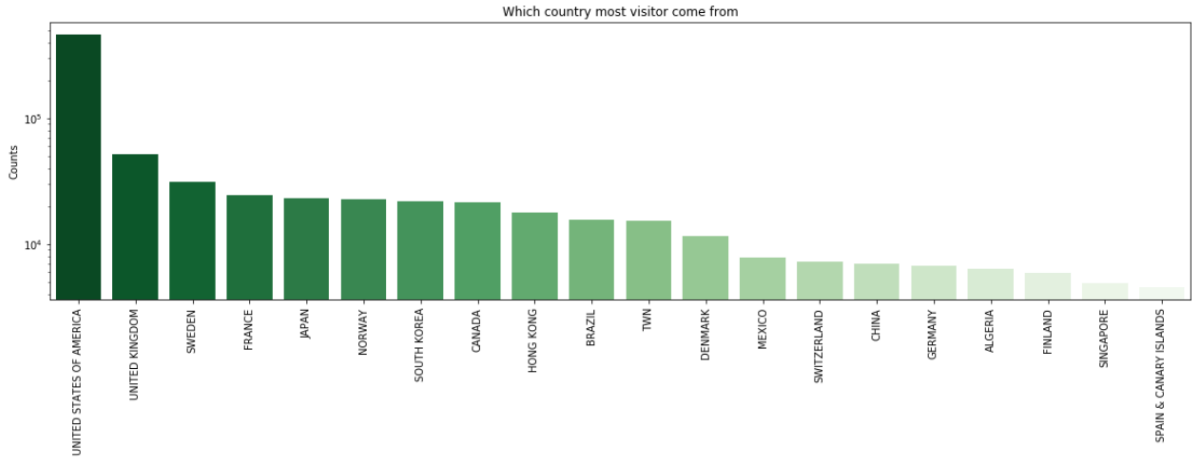
2.1 Country analysis

First, I analysed where customers are from, as customers from the same country are likely to have similar preferences. It looks like most of customers are from **USA** occupying **55.3%** of total population. There are total of **151** countries in the dataset.

```
train['srch_visitor_loc_country'].nunique()
```

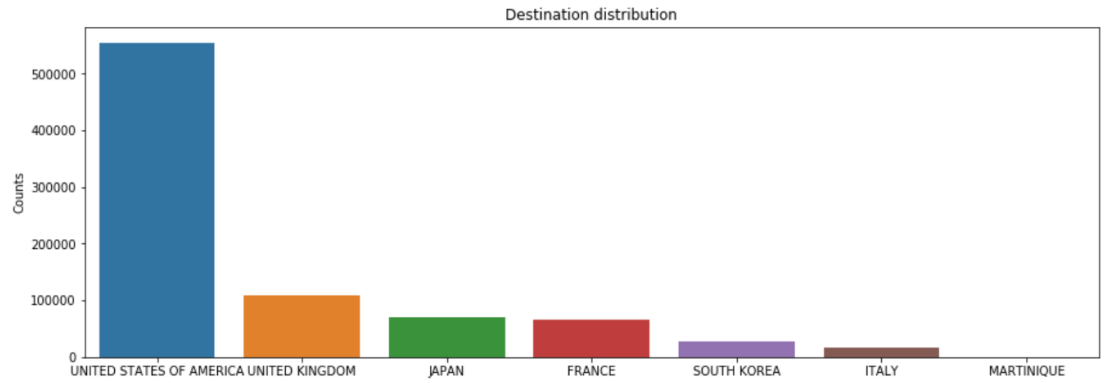
```
151
```

```
country_counts = train['srch_visitor_loc_country'].value_counts().sort_values(ascending=False).iloc[:20]
plt.figure(figsize=(20,5))
sns.barplot(country_counts.index, country_counts.values, palette="Greens_r")
plt.ylabel('Counts')
plt.title('Which country most visitor come from')
plt.xticks(rotation=90)
plt.yscale('log')
```




```
country_counts = train['prop_country'].value_counts().sort_values(ascending=False)
plt.figure(figsize=(15,5))
sns.barplot(country_counts.index, country_counts.values)
plt.ylabel('Counts')
plt.title('Destination distribution')
#plt.yscale('log')
```

```
Text(0.5, 1.0, 'Destination distribution')
```



There are **7** country destinations in the dataset. Most customers choose to travel to **USA** occupying **66%** of total population, combining figures of customers from which country, it is possible that quite a lot of USA customers travelling domestic.

2.2 Check in and out date analysis

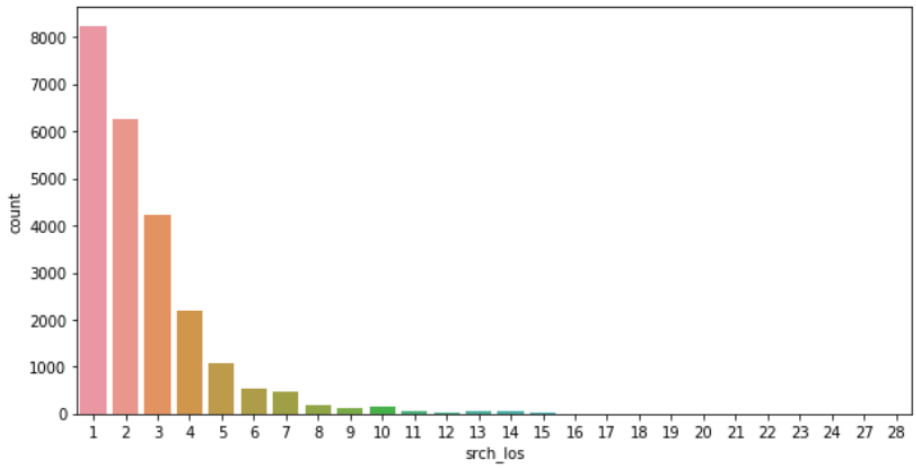
```
def convert_date_into_days(df):
    df['srch_ci'] = pd.to_datetime(df['srch_ci'])
    df['srch_co'] = pd.to_datetime(df['srch_co'])
    df['srch_local_date'] = pd.to_datetime(df['srch_local_date'])
    df['srch_date_time'] = pd.to_datetime(df['srch_date_time'])

    # For hotel check-in
    # Month, Year, Day
    df['Cin_month'] = df["srch_ci"].apply(lambda x: x.month)
    df['Cin_year'] = df["srch_ci"].apply(lambda x: x.year)
```

```
convert_date_into_days(train)
```

First, time information inferred from datetime could be useful in the prediction, so I convert time value to datetime object and build check-in month and year feature. Check-in day information already existed in dataset. Then I plot duration of stay distribution of customers who booked hotel and it turned out that most customers stayed for short period of time below a week. Its very strange customers check-in day concentrated at the first week of a month. It needs more information to interpret this phenomenon.

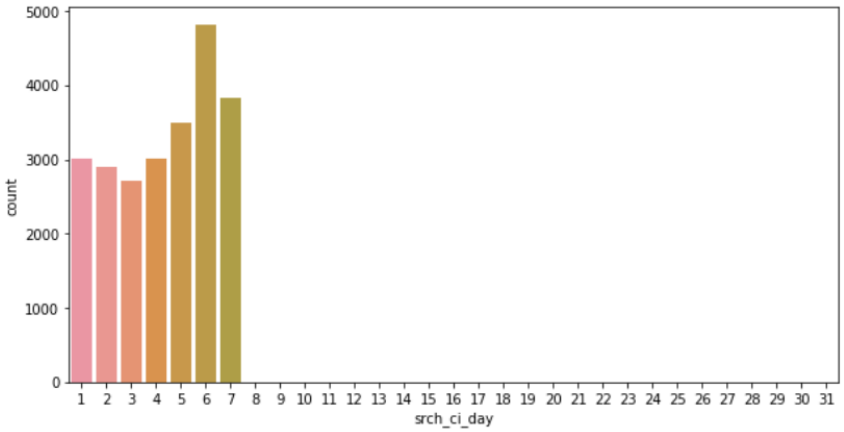
```
# Count the bookings as per the stay_duration
fig, ax = plt.subplots()
fig.set_size_inches(10, 5)
sns.countplot('srch_los',data=train[train['prop_booking_bool'] == 1],ax=ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x224835f0eb8>
```



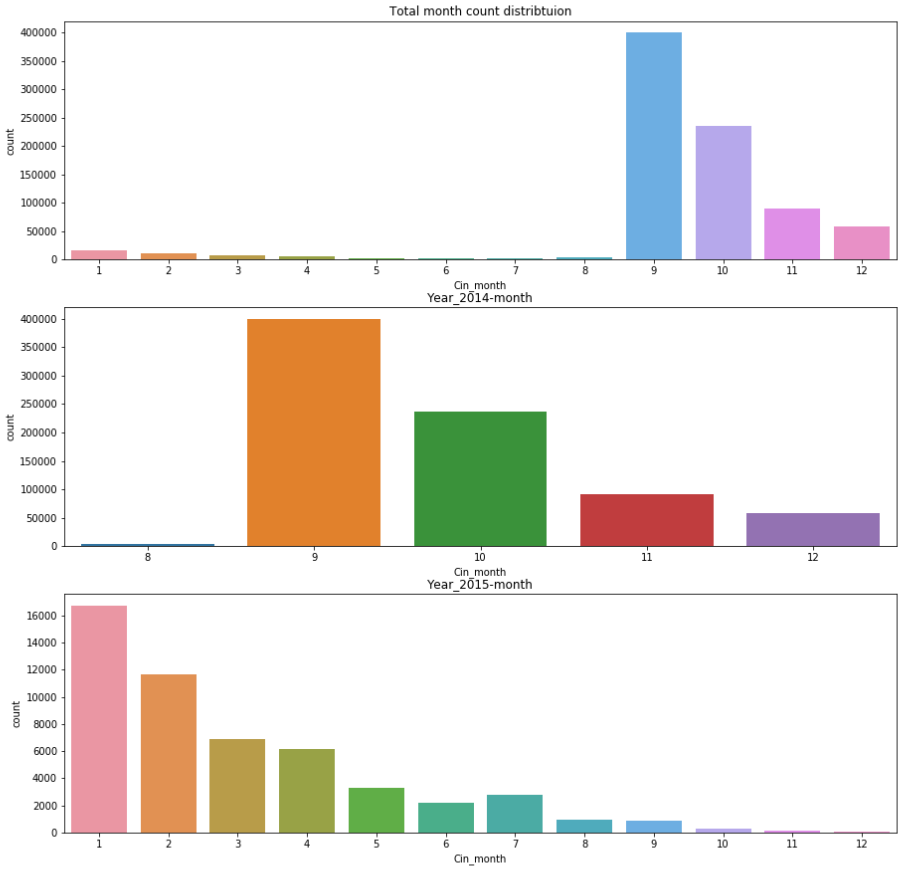

```
# Count the bookings as per the day
fig, ax=plt.subplots()
fig.set_size_inches(10,5)
sns.countplot('srch_ci_day', data=train[train['prop_booking_bool'] == 1], order=list(range(1,32)),ax=ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x22480c385c0>
```



Apart from check-in day, I also plot check-in month distribution in the year of 2014, 2015 and total respectively. The most busies month in 2014 is September and January in 2015. Check-in months distribute quite evenly in year 2015 but customers only checked in month 8,9,10,11 and 12 in year 2014. Consider that huge amount of data available should cover each check-in month situation, this is very strange. Its clear most of data comes from year 2014 (**93.8%** of total counts) and hence heavily skew the total month count distribution as well.

```
fig, (axis1, axis2,axis3)=plt.subplots(3,1,figsize=(15,15))

data_2014=train.loc[(train['Cin_year']==2014)]
data_2015=train.loc[(train['Cin_year']==2015)]

sns.countplot('Cin_month',data=train[train['prop_booking_bool'] == 1],ax=axis1).set(title='Total month count distribtuion')
sns.countplot('Cin_month',data=data_2014[data_2014['prop_booking_bool'] == 1],ax=axis2).set(title='Year_2014-month')
sns.countplot('Cin_month',data=data_2015[data_2015['prop_booking_bool'] == 1],ax=axis3).set(title='Year_2015-month')
fig.savefig('month_count_distribution')
```



2.3 hotel analysis

From my personal perspective, when I book hotel online, I am mostly interested in deal, starratting and review score. I think these three attributes are very important when it comes to selecting a hotel to stay. Therefore, its necessary to analyse them accordingly.

- Day deal analysis

I calculated booking rate in two scenarios, one is had day deal having **16.64%** booking rate compared to the other no day deal having only **2.8%** booking rate. This looks quite normal as people naturally pursue lower price deal.

```
deal=train.loc[train['prop_dotd_bool']==1]
percentage = deal.loc[deal['prop_booking_bool']==1].shape[0]/deal.shape[0]*100
print('percentage of people booking the property with daily deal is %.2f'%percentage)
```

```
percentage of people booking the property with daily deal is 16.64
```

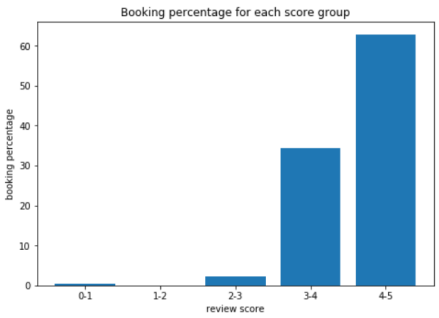
```
deal=train.loc[train['prop_dotd_bool']==0]
percentage = deal.loc[deal['prop_booking_bool']==1].shape[0]/deal.shape[0]*100
print('percentage of people booking the property without daily deal is %.2f'%percentage)
```

```
percentage of people booking the property without daily deal is 2.80
```

- Review analysis

Above 60% customers choose the highest rating hotel, and more than 90% customers choose hotel with rating greater than 3 (total rating is 5). Very few people choose low rating hotel which make sense as reviews from others are important information for decision.

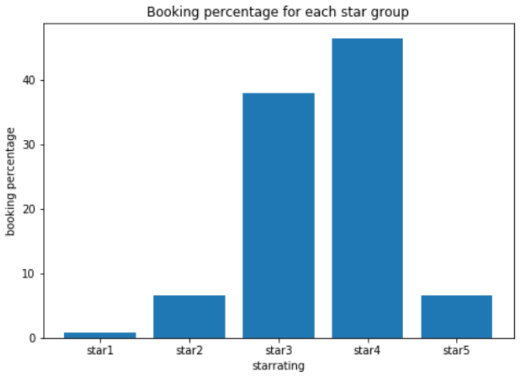
```
review_percent=[]
train_book=train.loc[train['prop_booking_bool']==1]

for i in range(5):
    score=train_book.loc[(train['prop_review_score']>=i)&(train['prop_review_score']<i+1)]
    percent = score.shape[0]/train_book.shape[0]*100
    review_percent.append(percent)

fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
langs = ['0-1', '1-2', '2-3', '3-4', '4-5']
ax.bar(langs,review_percent)
plt.xlabel('review score')
plt.ylabel('booking percentage')
plt.title('Booking percentage for each score group')
plt.show()
```



- Star ratting analysis

Most customers choose hotels of star 3 and 4 indicating most traveller like to live in a quite high-quality hotel for the trip.

```
star_percent=[]
train['prop_starrating']=train['prop_starrating'].astype(int)
train_book=train.loc[train['prop_booking_bool']==1]

for i in range(5):
    star=train_book.loc[train['prop_starrating']==i+1]
    percent = star.shape[0]/train_book.shape[0]*100
    star_percent.append(percent)

fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
langs = ['star1', 'star2', 'star3', 'star4', 'star5']
ax.bar(langs,star_percent)
plt.xlabel('starrating')
plt.ylabel('booking percentage')
plt.title('Booking percentage for each star group')
plt.show()
```



2.4 Price analysis

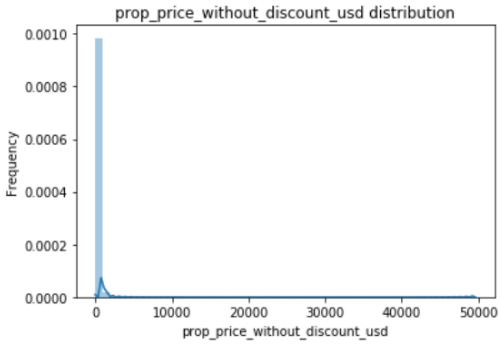
I decide to only include price in USD currency for two reasons:

1. Most customers **55.3%** are from US and **66%** travel to US
2. Currency column has **15.9%** missing values, drop currency attribute

I filled the missing value of price column based on group of hotel star ratting and then calculated mean price value in the group for filling the missing value.

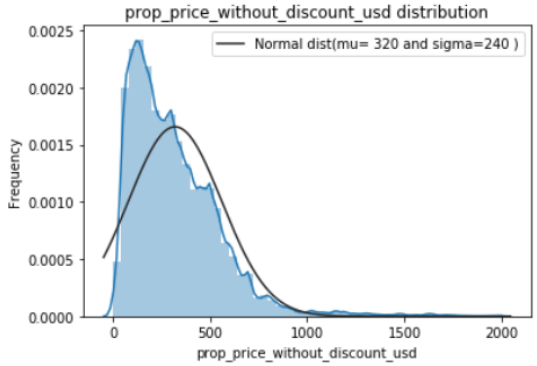
```
train['prop_price_without_discount_usd']=train.groupby('prop_starrating')['prop_price_without_discount_usd'].
transform(lambda x: x.fillna(x.median()))
train['prop_price_with_discount_usd']=train.groupby('prop_starrating')['prop_price_with_discount_usd'].
transform(lambda x: x.fillna(x.median()))
```

I find out the original price entry has some extreme outliers that affect price distribution (highly skewed). As I might want to create features later on that are based on historical prices, this is very disruptive. I decided to constrain the price limit to **2000USD** so to eliminate outliers. And resulting dataset is **99.4%** of original dataset, in other words, outliers is about **0.6%.**

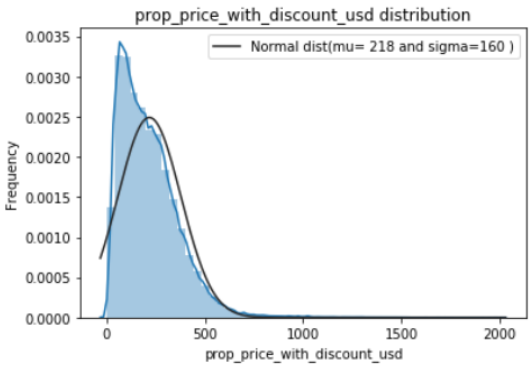



```
from scipy.stats import norm

train=train.loc[train['prop_price_without_discount_usd']<=2000]

sns.distplot(train['prop_price_without_discount_usd'], fit=norm)
(mu, sigma) = norm.fit(train['prop_price_without_discount_usd'])
print('mu = %d and sigma = %d' % (mu, sigma))
plt.legend(['Normal dist(mu= %d and sigma=%d )'% (mu, sigma)],
           loc='best')
plt.ylabel('Frequency')
plt.title('prop_price_without_discount_usd distribution')

print("Skewness: %f" % train['prop_price_without_discount_usd'].skew())
print("Kurtosis: %f" % train['prop_price_without_discount_usd'].kurt())
```

```
mu = 320 and sigma = 240
Skewness: 1.921149
Kurtosis: 6.604318
```



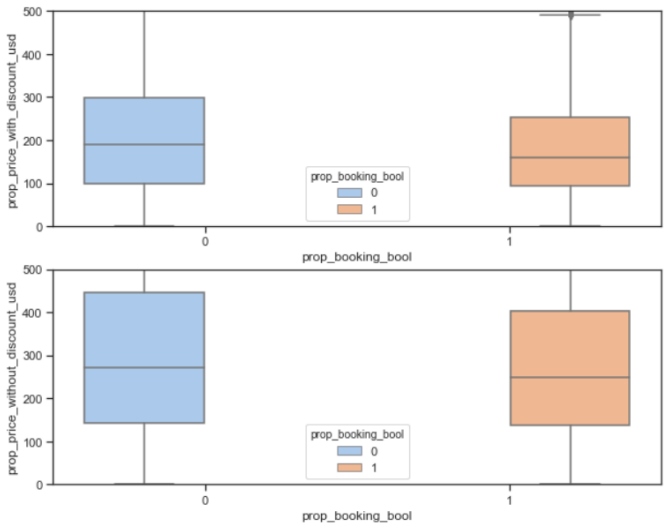

```
mu = 218 and sigma = 160
Skewness: 2.155320
Kurtosis: 10.859306
```



Looking at the kurtosis score, we can see that there is a very nice peak. However, looking at the skewness score, we can see that the sale prices deviate from the normal distribution. But since I am not predicting the price, I don't need to fix this to normal distribution. On average, the price no matter discounted or not received booking is always lower than those of did not book.

```
fig,(axis1, axis2)=plt.subplots(2,1, figsize=(10,8))
sns.set(style='ticks',palette='pastel')
axis1.set_ylim([0,500])
axis2.set_ylim([0,500])
sns.boxplot(x='prop_booking_bool', y='prop_price_with_discount_usd', hue='prop_booking_bool', ax=axis1, data=train)
sns.boxplot(x='prop_booking_bool', y='prop_price_without_discount_usd', hue='prop_booking_bool',ax=axis2, data=train)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2389ab0f9e8>
```



3. Clean data and feature engineering

Firstly, I concatenated the test dataset with training dataset to do feature pre-processing. The dataset was split into categorical and numerical dataset for cleaning and will be merged later.

```
X_test['prop_booking_bool']=-1
```

```
data = pd.concat([X_train, X_test]).reset_index(drop=True)
```

3.1 Categorical data

First, time information inferred from datetime could be useful in the prediction, so I convert it to datetime object and build check-in month and year feature. I think check-in feature and duration of stay is more relevant to prediction, I keep this two datetime feature for prediction. Then, I delete other datetime column and ID column of categorical dataset as it doesn't affect machine learning training.

```
cat_cols = [col for col in train.columns if train[col].dtype == 'O']
cat_cols
```

```
['srch_date_time',
 'srch_visitor_id',
 'srch_visitor_loc_country',
 'srch_visitor_loc_region',
 'srch_visitor_loc_city',
 'srch_visitor_wr_member',
 'srch_posa_continent',
 'srch_posa_country',
 'srch_ci',
 'srch_co',
 'srch_device',
 'srch_currency',
 'prop_super_region',
 'prop_continent',
 'prop_country',
 'srch_local_date']
```

```
cat_df = train[cat_cols]
convert_date_into_days(cat_df)
```

drop unnecessary columns, these columns won't be useful in analysis and prediction

```
visitor_id=cat_df['srch_visitor_id']

cat_df.drop(['srch_date_time','srch_visitor_id','srch_ci','srch_co','srch_local_date'] , axis = 1, inplace = True)
```

- Check missing value

srch_posa_continent and **srch_visitor_wr_member** have over **50%** missing value and they cannot be inferred from other features, so they will be deleted. The same as **srch_visitor_loc_region.** I decided to use USD as the price currency in the training procedure to avoid information duplication and most customers **55.3%** are from US and **66%** travel to US from EDA, so **srch_currency** is deleted. I also deleted useless feature like some 'region' information that are redundant since we have more granularity 'country' information.

```
total=cat_df.isnull().sum().sort_values(ascending=False)
percentage= (cat_df.isnull().sum()/cat_df.shape[0]).sort_values(ascending=False)
missing=pd.concat([total,percentage],axis=1, keys=['Total','Percent'])
missing.head()
```

	Total	Percent
srch_posa_continent	485248	0.576910
srch_visitor_wr_member	444878	0.528915
srch_currency	134104	0.159436
srch_visitor_loc_region	123	0.000146
Cin_year	0	0.000000

```
cat_df.drop(['srch_posa_continent','srch_visitor_wr_member','srch_currency','srch_visitor_loc_region',
             'srch_visitor_loc_city','prop_super_region','prop_continent','srch_visitor_loc_region'], axis = 1, inplace = True)
```

Now, the categorical data contains only 6 feature columns.

```
cat_df.head()
```

	srch_visitor_loc_country	srch_posa_country	srch_device	prop_country	Cin_month	Cin_year
0	TWN	TAIWAN, REPUBLIC OF CHINA	DESKTOP	JAPAN	2	2015
1	TWN	TAIWAN, REPUBLIC OF CHINA	DESKTOP	JAPAN	2	2015
2	TWN	TAIWAN, REPUBLIC OF CHINA	DESKTOP	JAPAN	2	2015
3	TWN	TAIWAN, REPUBLIC OF CHINA	DESKTOP	JAPAN	2	2015
4	TWN	TAIWAN, REPUBLIC OF CHINA	DESKTOP	JAPAN	2	2015

- Encoding for categorical data

Label encoding for categorical feature which has order, eg. Year and month

One-hot encoding for categorical features which are nominal.

```
cat_df['Cin_year'] = cat_df['Cin_year'].map({2015: 0, 2014: 1})
```

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

features = [x for x in cat_df.columns if x not in ['Cin_month','Cin_year']]
le = LabelEncoder()

for feat in features:
    value=list(cat_df[feat].unique())
    le.fit(value)
    cat_df[feat]=le.fit_transform(cat_df[feat])
```

```
cat_df.head()
```

	srch_visitor_loc_country	srch_posa_country	srch_device	prop_country	Cin_month	Cin_year
0	138	58	0	2	2	0
1	138	58	0	2	2	0
2	138	58	0	2	2	0
3	138	58	0	2	2	0
4	138	58	0	2	2	0

There are total of 6 column features in categorial dataset now.

3.2 Numerical data

```
num_df = train.drop(columns = cat_cols, axis = 1)
num_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 836230 entries, 0 to 841114
Data columns (total 31 columns):
srch_id                             836230 non-null int64
prop_key                            836230 non-null int64
srch_visitor_visit_nbr              836230 non-null int64
```

- Missing value

```
total=num_df.isnull().sum().sort_values(ascending=False)
percentage= (num_df.isnull().sum()/num_df.shape[0]).sort_values(ascending=False)
missing=pd.concat([total,percentage],axis=1, keys=['Total','Percent'])
missing.head()
```

	Total	Percent
prop_price_with_discount_local	18	0.000033
srch_adults_cnt	18	0.000033
prop_price_without_discount_local	18	0.000033
srch_children_cnt	18	0.000033
prop_hostel_bool	0	0.000000

From the EDA analysis, I know most customers **55.3%** are from US and **66%** travel to US. I dropped currency feature. Therefore, I deleted **prop_price_without_discount_local, prop_price_with_discount_local.** Fill **0** for the rest of missing value.

```
columns=['prop_price_without_discount_local','prop_price_with_discount_local']
num_df.drop(columns, axis=1, inplace=True)
```

Filling missing value

I already filled **prop_price_without/with_discount_USD** missing value based on group of hotel star rating in the price analysis section.

train['prop_price_without_discount_usd']=train.groupby('prop_starrating')['prop_price_without_discount_usd'].transform(lambda x: x.fillna(x.median()))

```
num_df.fillna(0, inplace=True)
```

```
num_df.isnull().sum()
```

```
srch_id                         0
prop_key                        0
srch_visitor_visit_nbr          0
```

- Drop useless features

```
columns=['srch_dest_longitude','srch_dest_latitude','srch_co_day','prop_imp_drr',
         'srch_mobile_bool', 'prop_travelad_bool','srch_mobile_app','prop_market_id']
num_df.drop(columns, axis=1, inplace=True)
```

```
num_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 836230 entries, 0 to 841114
Data columns (total 21 columns):
srch_id                         836230 non-null int64
prop_key                        836230 non-null int64
srch_visitor_visit_nbr          836230 non-null float64
srch_hcom_destination_id        836230 non-null int64
```

- Normalization

Since now we are analysing numerical data, I want to know the variation of each feature and then normalize features with large variation.

```
num_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 836230 entries, 0 to 841114
Data columns (total 21 columns):
srch_id                         836230 non-null int64
prop_key                        836230 non-null int64
srch_visitor_visit_nbr          836230 non-null int64
srch_hcom_destination_id        836230 non-null int64
srch_ci_day                     836230 non-null int64
srch_los                        836230 non-null int64
srch_bw                         836230 non-null int64
srch_adults_cnt                 836230 non-null float64
srch_children_cnt               836230 non-null float64
srch_rm_cnt                     836230 non-null int64
prop_dotd_bool                  836230 non-null int64
prop_price_without_discount_usd 836230 non-null float64
prop_price_with_discount_usd    836230 non-null float64
prop_booking_bool               836230 non-null int64
prop_brand_bool                 836230 non-null int64
prop_starrating                 836230 non-null float64
prop_market_id                  836230 non-null int64
prop_room_capacity              836230 non-null int64
prop_review_score               836230 non-null float64
prop_review_count               836230 non-null float64
prop_hostel_bool                836230 non-null int64
```

I defined a normalize function to normalize the numeric features which has high standard deviation, although that step was not important for the tree-based algorithms. I tried creating some features

with PCA hoping to reduce feature dimension but maintain the maximum feature variety. But I abandoned that after they failed to boost performance.

```
def normalize(col):
    max_value = col.max()
    min_value = col.min()

    col = (col - min_value) / (max_value - min_value)
    return col
```

```
num_df['prop_review_count'] = normalize(num_df['prop_review_count'])
num_df['prop_room_capacity'] = normalize(num_df['prop_room_capacity'])
num_df['prop_price_without_discount_usd'] = normalize(num_df['prop_price_without_discount_usd'])
num_df['prop_price_with_discount_usd'] = normalize(num_df['prop_price_with_discount_usd'])
num_df['srch_bw'] = normalize(num_df['srch_bw'])
num_df['srch_visitor_visit_nbr'] = normalize(num_df['srch_visitor_visit_nbr'])
num_df['srch_los'] = normalize(num_df['srch_los'])
num_df['srch_ci_day'] = normalize(num_df['srch_ci_day'])
```

- Group features for better understanding.

Customers who evaluate review_score in the same range tend to have similar behaviour.

```
col = 'prop_review_score'
conditions  = [ num_df[col] >= 4,
                (num_df[col] < 4) & (num_df[col]>=3),
                (num_df[col] < 3) & (num_df[col]>=2),
                (num_df[col] < 2) & (num_df[col]>=1),
                num_df[col] <=1]
choices     = [ 4.5, 3.5, 2.5, 1.5, 0.5]
num_df['prop_review_score'] = np.select(conditions, choices, 0)
```

```
col = 'prop_room_capacity'
conditions  = [ num_df[col] >= 4000,
                (num_df[col] < 4000) & (num_df[col]>=3000),
                (num_df[col] < 3000) & (num_df[col]>=1000),
                (num_df[col] < 1000) & (num_df[col]>=0),
                num_df[col] <0]
choices     = [4000, 3500, 2000, 500, 0]
num_df['prop_room_capacity'] = np.select(conditions, choices, 0)
```

```
num_df['srch_visitor_visit_nbr']=np.where((num_df['srch_visitor_visit_nbr']>100), 100, num_df['srch_visitor_visit_nbr'])
```

- Merge data and analyse correlation

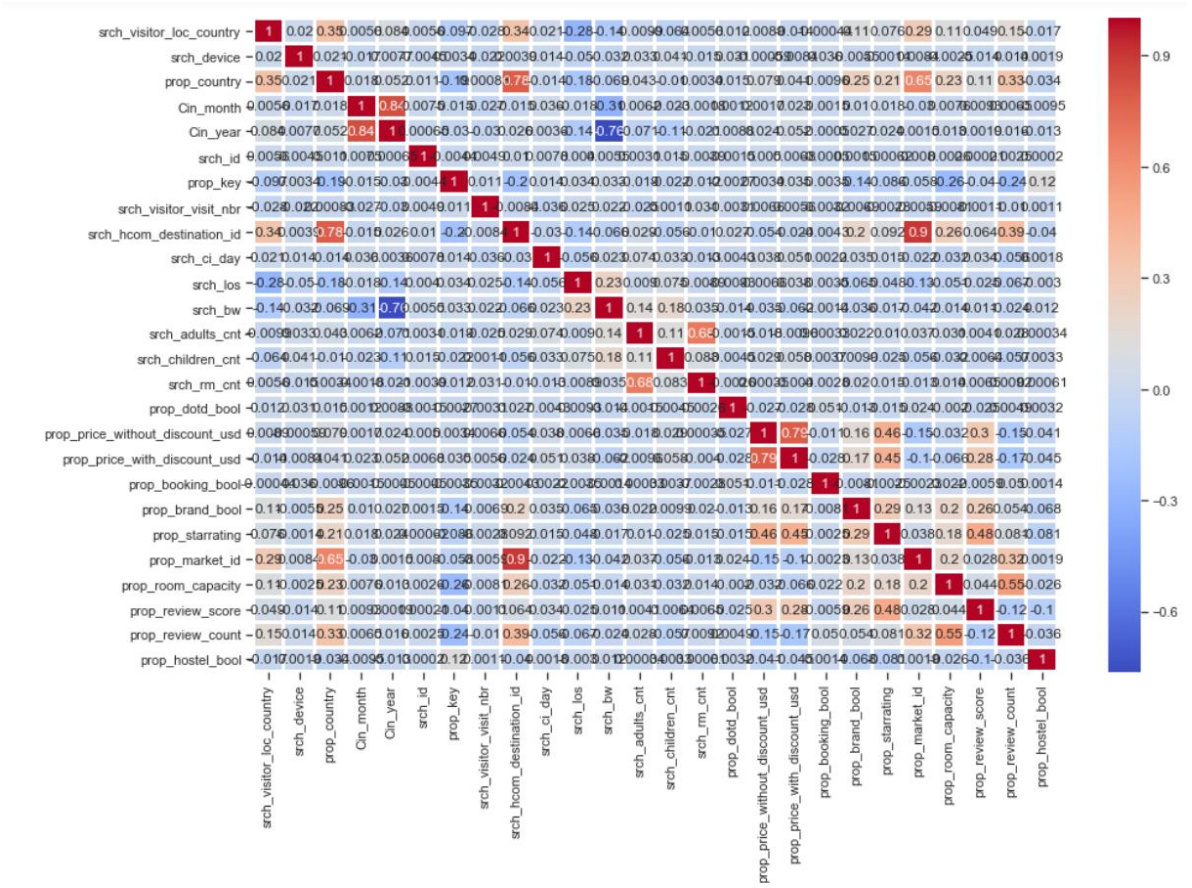
```
X_train = pd.concat([cat_df, num_df], axis = 1)
```

```
X_train=X_train.loc[X_train['srch_adults_cnt']>0]
```

```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 836212 entries, 0 to 841114
Data columns (total 27 columns):
srch_visitor_loc_country          836212 non-null int32
srch_posa_country                 836212 non-null int32
```

```
fig,ax=plt.subplots(figsize=(15,10))
sns.heatmap(X_train.corr(),cmap='coolwarm',ax=ax,annot=True, linewidths=2)
```



This tells us that no columns correlate linearly with booking rate. This makes sense, because there is no linear ordering to booking rate. For example, booking a specific hotel isn't tied to having a higher srch_hcom_destination_id. Unfortunately, this means that techniques like linear regression and logistic regression won't work well on our data, because they rely on linear correlations between predictors and targets.

4. Machine Learning

4.1 Balance dataset

Since the class distribution is extremely imbalanced with only **2.85%** booking=1 samples, I decided to down sample the non-booking records in machine learning section to re-balance the class distribution in the training process. I realized that this method reduced a lot of data and there are other more useful pre-processing techniques to deal with imbalanced dataset but due to the time limit, I selected the most simple one which is re-balancing.

```
book_indices = X_train[X_train['prop_booking_bool']==1].index

random_indices = np.random.choice(book_indices, len(X_train.loc[X_train['prop_booking_bool'] == 1]), replace=False)
book_sample = X_train.loc[random_indices]

not_book = X_train[X_train['prop_booking_bool'] == 0].index
random_indices = np.random.choice(not_book, sum(X_train['prop_booking_bool']), replace=False)
not_book_sample = X_train.loc[random_indices]

xtrain = pd.concat([not_book_sample, book_sample], axis=0)

print("Percentage of not  book: ", len(xtrain[xtrain['prop_booking_bool']==0])/len(xtrain))
print("Percentage of book: ", len(xtrain[xtrain['prop_booking_bool']==1])/len(xtrain))
print("Total number of records in resampled data: ", len(xtrain))
```

```
Percentage of not  book:  0.5
Percentage of book:  0.5
Total number of records in resampled data:  47550
```

4.2 XGBoost

The first machine learning algorithm I tried to use is XGBoost. XGBoost is short for Extreme Gradient Boosting and is an efficient implementation of the stochastic gradient boosting machine learning algorithm. There are 4 reasons I choose XGBoost as the first ML algorithm.

Reason1: we don't need to worry about feature scaling issue in tree algorithms. They are in general scale-invariant. Although I did normalization in the *num_df* to bring numerical features into the same scale. But there are also label encoded categorical features that span quite large scale. This characteristic also has advantage in dealing with outliers.

Reason2: XgBoost, don't assume any normal distribution and can work on raw data. It makes piecewise constant predictions, so are not particularly susceptible high leverage points. The dataset has a lot of features values distance far away from majority.

Reason3: XGBoost had advantage of handling sparsity. The fact that I filled 0 for some missing value or NaN value and feature engineering like one-hot encoding categorical features caused a lot of zero entries. XGBoost can deal with sparsity for our dataset.

Reason4 XGBoost can be used to select features. A benefit of using gradient boosting is that after the boosted tree are constructed, it is relatively straight forward to retrieve importance score for each attribute. Generally, importance provides a score that indicates how useful or valuable each feature was in the construction of the boosted trees within the model. XGBoost has built-in function *get_fscore()* to generate list of scores for each features. The more a feature is used to make key decisions, the higher its relative importance. I used feature importance score to select features for training neural network (NN) in the next section.

- Model selection

Tunning hyper-parameter is a critical step in machine learning model building process. Particularly, XGBoost has a lot of hyper-parameters to tune, default value is not suitable for every kind of dataset. I selected *RandomimzedSearchCV* to search over parameter settings by cross-validation. It has built-in method best_pramas_ to tell us the best parameters from the validation dataset. The task is a binary classification task, so I choose binary logistic function as objectives.

I set **0.8** and **0.85** for *'subsample'* and *'colsample_bytree'* to introduce randomness in the model training process so that model is less prone to over-fitting.

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from xgboost import XGBClassifier  # wrapper
import scipy.stats as st

params_sk = {
    'objective': 'binary:logistic',
    'subsample': 0.8,
    'colsample_bytree': 0.85,
    'seed': 42}

skrg = XGBClassifier(**params_sk)


params_grid = {"n_estimators": st.randint(50, 500),
               'max_depth': st.randint(6, 30)
               }
search_sk = RandomizedSearchCV(skrg, params_grid, cv=5, scoring='roc_auc', random_state=1, n_iter=10)
search_sk.fit(xtrain, Y)
```

- Build XGBoost model

After RandomizedSearchCV, I found out the best parameters to build the model as shown below:

```
# best parameters
print("best parameters:", search_sk.best_params_)
print("best score:", search_sk.best_score_)
```

```
best parameters: {'max_depth': 6, 'n_estimators': 194}
best score: 0.7018477113581255
```

```
import xgboost as xgb
import operator

dtrain = xgb.DMatrix(X_train, y_train)
dtest = xgb.DMatrix(X_test, y_test)

watchlist = [(dtrain, 'train'), (dtest, 'validate')]
early_stop = 20
```

```
params_new = {**params_sk, **search_sk.best_params_}

model_final = xgb.train(params_new, dtrain, evals=watchlist, early_stopping_rounds=early_stop, verbose_eval=True)
```

- Model evaluation

I split the 30% of training dataset into validation dataset to evaluate model performance.

```
def print_evaluation_metrics(trained_model,trained_model_name,X_test,y_test):
    predicted_values = trained_model.predict(X_test)
    print(metrics.classification_report(y_test,predicted_values))
    print("Accuracy Score : ",metrics.accuracy_score(y_test,predicted_values))
    print("---------------------------------------\n")
```

```
              precision    recall  f1-score   support

           0       0.66      0.65      0.66      4782
           1       0.65      0.66      0.66      4728

    micro avg       0.66      0.66      0.66      9510
    macro avg       0.66      0.66      0.66      9510
 weighted avg       0.66      0.66      0.66      9510

 Accuracy Score :  0.6588853838065194
```

Note that precision and recall are quite similar which means that the model is performing generally well on both classes. This could be due to the fact that I balanced the class distribution for the

training dataset. Then I ran the model on test dataset and produce the estimated probability of each sample belonging to which class (*y_pred*) using sigmoid function.

```
# prediction to testing data
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

X_test = xgb.DMatrix(X_test)
y_pred = model_final.predict(X_test)
```

```
y_pred
```

```
array([0.5042258 , 0.5856177 , 0.4715971 , ..., 0.59938085, 0.52030426,
       0.5696452 ], dtype=float32)
```

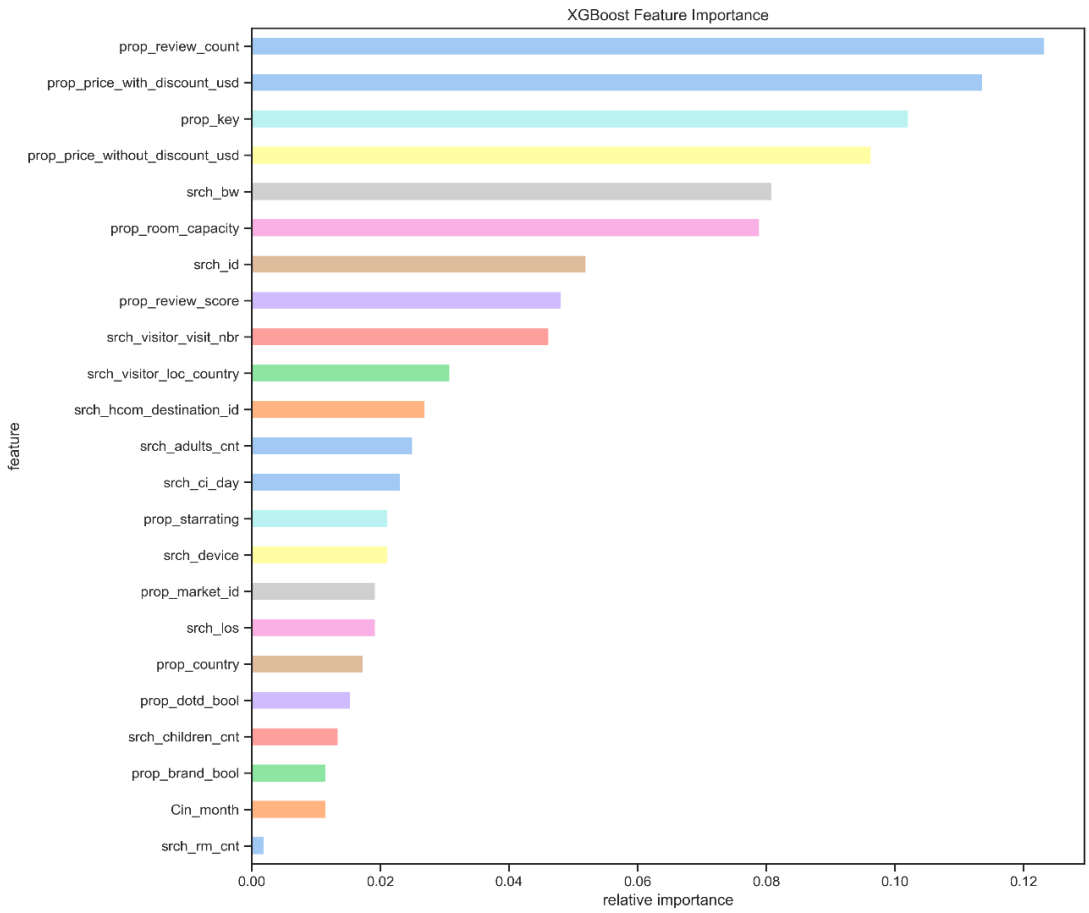
- Feature importance

As I mentioned before, XGBoost can automatically provide estimates of feature importance that help use interpret model and select features. So I defined a plot function to draw the relative importance of each feature.

```
def feature_importance_plot(importance_sorted, title):
    df = pd.DataFrame(importance_sorted, columns=['feature', 'fscore'])
    df['fscore'] = df['fscore'] / df['fscore'].sum()

    plt.figure()
    # df.plot()
    df.plot(kind='barh', x='feature', y='fscore',
            legend=False, figsize=(12, 10))
    plt.title('XGBoost Feature Importance')
    plt.xlabel('relative importance')
    plt.tight_layout()
    plt.savefig(title + '.png', dpi=300)
    plt.show()
```

```
def xgb_importance(X_train, X_test, y_train, y_test, xgb_params, ntree, early_stop, plot_title):

    dtrain = xgb.DMatrix(X_train, y_train)
    dtest = xgb.DMatrix(X_test, y_test)

    watchlist = [(dtrain, 'train'), (dtest, 'validate')]

    xgb_model = xgb.train(xgb_params, dtrain, ntree, evals=watchlist,
                          early_stopping_rounds=early_stop, verbose_eval=5)

    importance = xgb_model.get_fscore()
    importance_sorted = sorted(importance.items(), key=operator.itemgetter(1))
    feature_importance_plot(importance_sorted, plot_title)
```

```
print('-----Xgboost Using Datetime Features Only------',
      '\n---Grid Search model feature importance---')
importance = model_final.get_fscore()
importance_sorted = sorted(importance.items(), key=operator.itemgetter(1))
fig1 = feature_importance_plot(importance_sorted, 'feature importance')
plt.show()
```



From the importance figure, we can clearly see the importance of each feature apart from ID and Key value, such as price and review score, which makes sense. I choose the top-rated features and use entity embedding method in NN model to try to further improve the model score.

4.3 Entity embedding neural network

- Why deep neural network?

From my understanding, features are more valuable if they can interact with each other in the recommendation task. For example, customers who have viewed the hotel in the same destination and even the same hotel may have similar preferences. Firstly, deep neural network is used to capture high-order interaction between features and generate the model result through fully connected layers. Secondly, from universal approximation theorems we know neural networks can approximate almost any complex decision boundary and almost sure convergence as the number of samples goes to infinity. DNN is a feasible option in terms of these two points. But the challenge arises from label encoding and hot encoding.

- Why entity embedding?

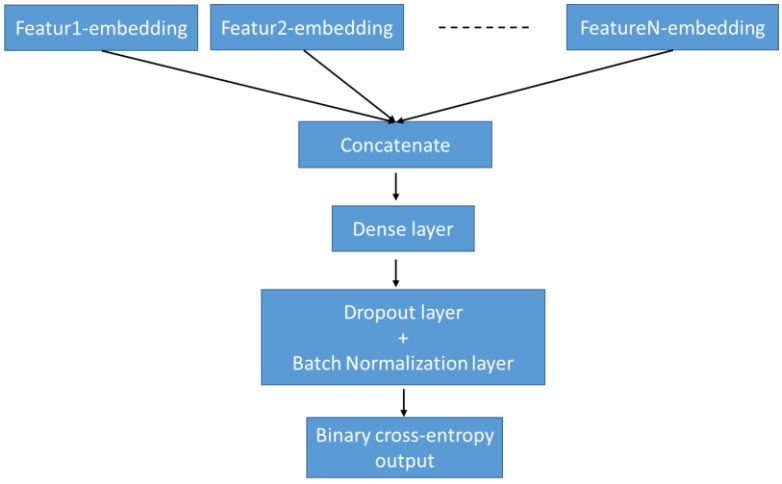
Often the label encoding integers are just used for convenience to label the different states and have no information in themselves. We consider input data with sparse and dense features. The inputs are mostly categorical features, e.g. "country=usa". Such features are encoded as one-hot vectors e.g. "[0,1,0]"; however, this leads to excessively high-dimensional feature spaces for large

vocabulary. This issue also applied to numerical feature such as month and year. In NN, each neuron received input x from previous layer and perform linear weight w calculation $(wx + b)$. Higher value of x have more contribution to the next layer. But month 2 is not necessarily have more valuable information than month 1.

I used encoder network to learn lower-dimensional embedding of input through mutual information in my PhD work. The task of embedding is to map high dimension data to a lower dimensional space where values with similar function output are close to each other. The similarity function can be cosine similarity function, Euclidean distances-based function and etc. The intuition behind predicting booking probability is to learn the similarity of user behaviour. Hence, I believed the ideal of embedding could also be useful for this task. Then I found the ideal of entity embedding that are very popular in recommendation system. The entity embedding can map N*K dimensional vector (hot-encoded vector) to 1*K dimension embedding vector at the same time captures some of the semantics of the input by placing semantically similar input close together in the embedding space through deep NN training.

- Model architecture

I first label encoded all selected column features based on XGBoost feature importance score to prepare for entity embedding. After using entity embeddings to represent all discrete categorical variables, all embedding layers were concatenated. The merged layer was treated like a normal input layer in neural networks and other layers can be built on top of it. In this way, the entity embedding layer learns about the intrinsic properties of each category, while the deeper layers form complex combinations of them. I used binary cross-entropy as loss function. The reason is that cross-entropy penalize loss heavily as it calculates negative log of the probability. Therefore, it increases losses exponentially if the predicted probability of the true class gets closer to zero. The model architecture is shown below:



Dropout layer is added to reduce over-fitting. Batch normalization is added to improve model stability.

- Model implementation

Firstly, I imported all necessary tensorflow and sklearn package.

```
import pandas as pd
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
from tensorflow.keras import layers
from tensorflow.keras.models import Model
from tensorflow.keras import callbacks
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold
import tensorflow as tf
from sklearn import metrics
```

Next, I imported cleaned train and test csv file and merged them to label encoded column features. After feature encoding, I split data file back to train and test dataset.

```
train=pd.read_csv('x_train.csv')
x_test = pd.read_csv('x_test.csv')

x_test['prop_booking_bool'] = -1
data = pd.concat([train, x_test]).reset_index(drop=True)

features = [x for x in train.columns if x not in ['srch_id','prop_key','prop_booking_bool',
                                                  'prop_price_without_discount_usd','prop_price_with_discount_usd']]

for feat in features:
    lbl_enc = LabelEncoder()
    train[feat] = lbl_enc.fit_transform(train[feat].fillna("-1").astype(str).values)
    train[feat] = lbl_enc.fit_transform(train[feat])
```

```
x_train = train[train.prop_booking_bool != -1].reset_index(drop=True)
test = train[train.prop_booking_bool == -1].reset_index(drop=True)
test_data = [test.loc[:, features].values[:, k] for k in range(test.loc[:, features].values.shape[1])]
```

I used build-in embedding method from tensorflow keras module to create embedding layers for each feature column. The embedding dimension was defined as the half of unique values but with limit of 50. Embedding layers were concatenated for next layer processing. The layers after concatenation layer were normal deep NN layers following the same procedure of deep NN model building process.

```
def create_model(data, catcols):
    inputs = []
    outputs = []
    for c in catcols:
        num_unique_values = int(data[c].nunique())
        embed_dim = int(min(np.ceil((num_unique_values) / 2), 50))
        inp = layers.Input(shape=(1,))
        out = layers.Embedding(num_unique_values + 1, embed_dim, name=c)(inp)
        out = layers.Reshape(target_shape=(embed_dim,))(out)
        inputs.append(inp)
        outputs.append(out)

    x = layers.Concatenate()(outputs)
    x = layers.BatchNormalization()(x)

    x = layers.Dense(500, activation="relu")(x)
    x = layers.Dropout(0.3)(x)
    x = layers.BatchNormalization()(x)

    y = layers.Dense(1, activation="sigmoid")(x)

    model = Model(inputs=inputs, outputs=y)
    return model
```

K-fold is a resampling technique without replacement, the advantage of this is that each sampling point will be used for training and validation once, which yields a lower-variance estimate of the model performance than holdout method. I used 5 fold to split data in train/test sets using *StratifieldKFold* from sklearn.model_selection module. Then the average of recorded scores is calculated as the performance metric for the model.

I choose ROC AUC metric to evaluate model performance. ROC are useful tools to select models for classification based on their performance with respect to FPR and TPR, which are computed by shifting the decision threshold of the classifier. Based on ROC curve, we can compute ROC Area under the Curve (ROC AUC) to characterize the performance of the model.

```
test_preds = np.zeros((len(test)))

skf = StratifiedKFold(n_splits=5, shuffle=True)
for train_index, test_index in skf.split(x_train, x_train.prop_booking_bool):
    X_train, X_test = train.iloc[train_index, :], train.iloc[test_index, :]
    X_train = X_train.reset_index(drop=True)
    X_test = X_test.reset_index(drop=True)
    y_train, y_test = X_train.prop_booking_bool.values, X_test.prop_booking_bool.values

    model = create_model(x_train, features)
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=metrics.roc_auc_score)

    X_train = [X_train.loc[:, features].values[:, k] for k in range(X_train.loc[:, features].values.shape[1])]
    X_test = [X_test.loc[:, features].values[:, k] for k in range(X_test.loc[:, features].values.shape[1])]

    es = callbacks.EarlyStopping(monitor='val_auc', min_delta=0.001, patience=5,
                                 verbose=1, mode='max', baseline=None, restore_best_weights=True)


    model.fit(X_train,
              y_train,
              validation_data=(X_test, y_test),
              verbose=2,
              batch_size=20,
              callbacks=es,
              epochs=10
              )

    test_fold_preds = model.predict(test_data)
    test_preds +=test_fold_preds.ravel()
```

The training loss is converged and auc score reached at 0.91 at epoch 9. However, validation loss is not converged and auc_score is only 0.62. One reason could be overfitting. After I tried L2-regualrizeation and different dropout rate, it did not improve much. The other reason could come from the dataset itself that need more feature processing, or further deep NN hyper-parameter fine-tunning. I also need to do more literature review to understand how embedding works in recommendation system.