Machine Learning Model for Airbnb Income Prediction

February 3, 2019

0.1 I. Introduction

Airbnb is a great platform that provides people online marketplace and service to arrange or offer lodging. As a travel enthusiast, Airbnb is always my first choice when I am planning a trip. Hosts need to provide details for their listed houses so that guests can use filters on the website to search for their preferred accomodations. For potential hosts, they must be very interested in how much they could earn from listing their houses on Airbnb. As far as I know, there is no such a model in public for predicting the yield of a new house on Airbnb. So, the object of this project is to apply machine learning models to help potential hosts gain some intuitions about the yield of their listed houses.

Fortunately, Inside Airbnb has already aggregated all the publicly available informations from Airbnb site for public discussion. So, the dataset obtained from this website directly should be a good starting point for my machine learning model. In particular, I will the dataset collected in Los Angeles city compiled on 06 December, 2018. When selecting features for machine learning model, besides the variables provided in the datasets, the featured photo on the listing's website and the description of listing can be crucial for attracting more guests. So, I will analyze featured photos and text mining on the descriptions and add these two new features to improve the machine learning model.

The project will be described as follows: 1. Exploratory data analysis and data preprocessing. 2. Feature engineering. 3. Machine learning model. 4. Model evaulation.

```
In [7]: # load the dataset
        import pandas as pd

        df = pd.read_csv('listings.csv')
        print ('There are {} rows and {} columns in the dataset'.format(*df.shape))
        df.head(3)
```

```
There are 43047 rows and 96 columns in the dataset
```

```
Out[7]:      id                         listing_url       scrape_id last_scraped  \
        0   109   https://www.airbnb.com/rooms/109  20181206172531   2018-12-07
        1   344   https://www.airbnb.com/rooms/344  20181206172531   2018-12-07
        2  2708  https://www.airbnb.com/rooms/2708  20181206172531   2018-12-06

                                                    name  \
        0  Amazing bright elegant condo park front *UPGRA...
```

```
1                  Family perfect;Pool;Near Studios!
2  Gold Memory Foam Bed & Breakfast in West Holly...

                                             summary  \
0  *** Unit upgraded with new bamboo flooring, br...
1  This home is perfect for families; aspiring ch...
2  Our best memory foam pillows you'll ever sleep...

                                               space  \
0  *** Unit upgraded with new bamboo flooring, br...
1  Cheerful & comfortable; near studios, amusemen...
2  Flickering fireplace display heater.  Decorate...

                                         description experiences_offered  \
0  *** Unit upgraded with new bamboo flooring, br...                none
1  This home is perfect for families; aspiring ch...                none
2  Our best memory foam pillows you'll ever sleep...                none

                                neighborhood_overview        ...          \
0                                                 NaN        ...
1  Quiet-yet-close to all the fun in LA! Hollywoo...        ...
2  We are minutes away from the Mentor Language I...        ...

  requires_license license            jurisdiction_names instant_bookable  \
0                f     NaN         {"Culver City"," CA"}                f
1                f     NaN                           NaN                t
2                f     NaN  {"City of Los Angeles"," CA"}                t

  is_business_travel_ready          cancellation_policy  \
0                        f  strict_14_with_grace_period
1                        f                     flexible
2                        f  strict_14_with_grace_period

   require_guest_profile_picture require_guest_phone_verification  \
0                              t                                f
1                              f                                f
2                              f                                f

   calculated_host_listings_count  reviews_per_month
0                               1               0.02
1                               1               0.13
2                               2               0.24

[3 rows x 96 columns]
```

In [8]:
```
df.columns
```

Out[8]:
```
Index(['id', 'listing_url', 'scrape_id', 'last_scraped', 'name', 'summary',
       'space', 'description', 'experiences_offered', 'neighborhood_overview',
       'notes', 'transit', 'access', 'interaction', 'house_rules',
       'thumbnail_url', 'medium_url', 'picture_url', 'xl_picture_url',
       'host_id', 'host_url', 'host_name', 'host_since', 'host_location',
       'host_about', 'host_response_time', 'host_response_rate',
       'host_acceptance_rate', 'host_is_superhost', 'host_thumbnail_url',
       'host_picture_url', 'host_neighbourhood', 'host_listings_count',
       'host_total_listings_count', 'host_verifications',
       'host_has_profile_pic', 'host_identity_verified', 'street',
       'neighbourhood', 'neighbourhood_cleansed',
       'neighbourhood_group_cleansed', 'city', 'state', 'zipcode', 'market',
       'smart_location', 'country_code', 'country', 'latitude', 'longitude',
       'is_location_exact', 'property_type', 'room_type', 'accommodates',
       'bathrooms', 'bedrooms', 'beds', 'bed_type', 'amenities', 'square_feet',
       'price', 'weekly_price', 'monthly_price', 'security_deposit',
       'cleaning_fee', 'guests_included', 'extra_people', 'minimum_nights',
       'maximum_nights', 'calendar_updated', 'has_availability',
       'availability_30', 'availability_60', 'availability_90',
       'availability_365', 'calendar_last_scraped', 'number_of_reviews',
       'first_review', 'last_review', 'review_scores_rating',
       'review_scores_accuracy', 'review_scores_cleanliness',
       'review_scores_checkin', 'review_scores_communication',
       'review_scores_location', 'review_scores_value', 'requires_license',
       'license', 'jurisdiction_names', 'instant_bookable',
       'is_business_travel_ready', 'cancellation_policy',
       'require_guest_profile_picture', 'require_guest_phone_verification',
       'calculated_host_listings_count', 'reviews_per_month'],
      dtype='object')
```

0.2 II. Exploratory data analysis and data preprocessing

0.2.1 Data cleaning

There are 49056 observations and 96 columns in the dataset. However, not all the columns are needed for machine learning model. Especially, for a new house, there won't be any information about reviews. So columns containing informations about reviews should be dropped. These features are "review_scores_rating", "review_scores_accuracy", "review_scores_cleanliness", "review_scores_checkin", "review_scores_communication", "review_scores_location", "review_scores_value", "reviews_per_month". After carefully considering each features, these features are kept for further data analysis: > - **listing_url:** from the url, photos of the houses can be scraped. Needless to say, a comfortable featured photo of the apartment can attract more viewers and improve the yield. - **description:** description with more details about the apartment can help tourists to make the decision. - **latitude, longitude:** these two columns provide the information about the location. There are some other columns such as "transit", "zipcode", "street" are actually closely related to the location. - **property_type, room_type, bathrooms, bedrooms, bed_type, square_feet, amenities:** these columns describe the properties of the house, such as how large is the aparment, how many bathrooms or bedrooms it has. - **guests_included, cleaning_fee, extra_people, minimum_nights, maximum_nights, availability_365, cancellation_policy:** these columns provide informations about the policy of booking a room. The house with more flexible policy may be more prefered for some tourists who are not so sure about their schedules. - **reviews_per_month:** this column is kept because it will be used later for calculating the yield. - **scrape_id:** this id is kept for later image scraping.

The data cleaning process will be performed as follows: 1. Drop all the unnecessary columns. 2. "cleaning_fee","extra_people","price" have the dollar sign before the number. Need to remove the "\$" and change the datetype from string to numerical values. 3. "property_type" has many categories, however, most of them only have few observations, so those categories can be combined into one category and name it "Other". 4. Handle missing values. First, columns including "bathrooms","bedrooms","cleaning_fee" and "reviews_per_month" have NULL values. They can be filled in with the median. There is also a column: "square_feet" whose majority of observations is missing, so this feature can be deleted. 5. Check the distribution of variables. The distribution of "available_365" shows that some houses are only available for a few days within a year. Rooms that only available for a short time are not considered in this project.

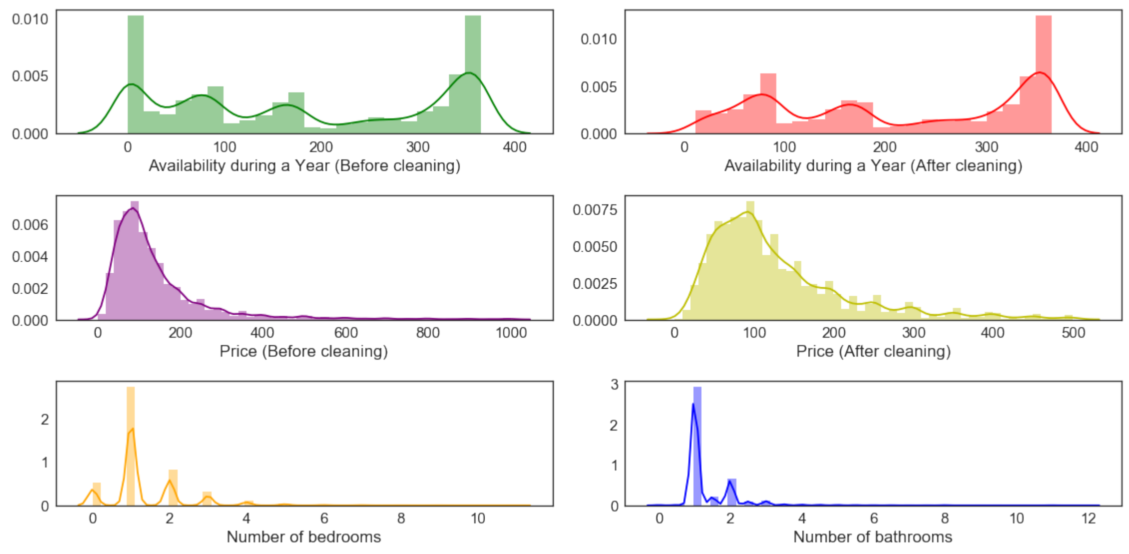
```
In [9]: import os
        import numpy as np
        import re
        import math
        import matplotlib as mpl
        import matplotlib.pyplot as plt
        import seaborn as sns

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

```
In [10]: # where to save figures and results
         ROOT_DIR = os.path.dirname(os.path.realpath('__file__'))
         Image_Path = os.path.join(ROOT_DIR,'Images')

         if not os.path.exists('Images'):
             os.makedirs('Images')
         Image_path = os.path.join('Images')

         def save_fig(fig_id, tight_layout=True):
             path = os.path.join(Image_path,fig_id + ".png")
             print("Saving figure", fig_id)
             if tight_layout:
                 plt.tight_layout()
             plt.savefig(path, format='png', dpi=300)
```

```
In [13]: # drop all the unnecessary columns
         feature_to_keep = ['listing_url','id','description','latitude','longitude','property_
                            'bedrooms','bed_type','price','square_feet','guests_included','clea
                            'maximum_nights','availability_365','cancellation_policy','reviews_
         new_df = df[feature_to_keep]

         # remove the dollar sign before "cleaning_fee", "extra_people", "price" and change th
         feature_to_remove_dollar = ['cleaning_fee','extra_people','price']
```

```
new_df[feature_to_remove_dollar] = new_df[feature_to_remove_dollar].replace('\$','',re
new_df[feature_to_remove_dollar] = new_df[feature_to_remove_dollar].apply(pd.to_numeri

# merge small catergories in property_type into one category "Other"
Other = ['Bed and breakfast','Resort','Boutique hotel','Guesthouse','Hostel','Hotel',
         'Tent','Cottage','Camper/RV','Cabin','Casa particular (Cuba)','Nature lodge'
         'Island','Earth house']
new_df['property_type'].loc[new_df['property_type'].isin(Other)] = "Other"

# drop the column "square_feet"
new_df = new_df.drop('square_feet', axis = 1)

# fill NaN with median value for 'bathrooms', 'bedrooms', 'cleaning_fee', 'price'
new_df['bathrooms'] = new_df['bathrooms'].fillna(new_df['bathrooms'].median())
new_df['bedrooms'] = new_df['bedrooms'].fillna(new_df['bedrooms'].median())
new_df['cleaning_fee'] = new_df['cleaning_fee'].fillna(new_df['cleaning_fee'].median()
new_df['price'] = new_df['price'].fillna(new_df['price'].median())

# there are 523 rows missing description, drop those rows
new_df = new_df.dropna()

# EDA of other variables and drop rows with availability_365 smaller than 10
%matplotlib inline
fig,axs = plt.subplots(ncols = 2, nrows = 3, figsize = (16,8))
plt.subplots_adjust(left=0, bottom=0, right=1, top=0.9,hspace=0.5,wspace=0.3)
sns.set(style = "white",font_scale=1.5)

sns.distplot(pd.Series(new_df['availability_365'],name = "Availability during a Year
sns.distplot(pd.Series(new_df['price'], name = "Price (Before cleaning)"), color = "pu

# remove houses that are only available for a short time and houses with high prices
new_df = new_df[new_df['availability_365']>10]
new_df = new_df[new_df['price']<500]

sns.distplot(pd.Series(new_df['availability_365'],name = "Availability during a Year
sns.distplot(pd.Series(new_df['price'],name = "Price (After cleaning)"),color = "y", a
sns.distplot(pd.Series(new_df['bathrooms'],name = "Number of bathrooms"),color = 'blue
sns.distplot(pd.Series(new_df['bedrooms'], name = "Number of bedrooms"), color = "oran

save_fig("Distribution_of_variables")

print ("Dataset has {} rows and {} columns.".format(*new_df.shape))
```

```
Saving figure Distribution_of_variables
Dataset has 27826 rows and 20 columns.
```



After cleaning up the data, the new dataset now has 27826 rows and 20 columns without any missing values.

0.2.2 Yield calculation

Inside Airbnb's "San Francisco Model" will be used for yield calculation. The caculation is as follows: > Yield = Average length of stay × Price × Number of reviews × 12 Months / Review_rate

Here is how the website explained the model: > Inside Airbnb's **"San Francisco Model"** uses as a modified methodology as follows: - A **review rate of 50%** is used to convert **reviews** to **estimated bookings.** - An **average length of stay** is configured for each city, and this, multiplied by the **estimated bookings** for each listings over a period gives the **occupancy rate** - Where statements have been made about the average length of stay of Airbnb guests for a city, this was used. - For example, Airbnb reported 5.5 nights as the average length of stay for guests using Airbnb in San Francisco. - Where no public statements were made about average stays, a value of **3 nights per booking** was used. - If a listing has a **higher minimum nights** value than the average length of stay, the minimum nights value was used instead. - The **occupancy rate** was **capped at 70%** - relatively high, but reasonable number for a highly occupied "hotel". - **Number of nights** booked or availble per year for the **high availability** and **frequently rented** metrics and filters were generally aligned with a city's short term rental laws designed to **protect residential housing.**

In our case, the **Average length of stay** will be 3 nights since there is no reported value. Also, if the minimum night is higher than 3 days, the average length of stay will be the value of minimum nights. 50% will be used as the review rate. The **Price** in the model should be the sum of 'price' and 'cleaning_fee' in the dataset.

```
In [14]: # calculate the Yield using San Francisco Model
         review_rate = 0.5
         new_df['average_length_of_stay'] = [3 if x < 3 else x for x in new_df['minimum_nights
         new_df['yield'] = new_df['average_length_of_stay']*(new_df['price']+new_df['cleaning_

         # reviews_per_month can be dropped now
```

```
new_df = new_df.drop('reviews_per_month',axis = 1)
new_df.head(3)

# save the current dataframe into a csv file
new_df.to_csv('cleaned_df.csv')
```

0.3 III. Feature engineering

0.3.1 Image analysis on featured photos

In most cases, hosts on Airbnb will upload some photos of their houses. These photos, especially the featured photo on the website, are extremely important to attract more viewers. An ideal photo should have desirable resolution and also be aesthetically attractive. Here I will use **NIMA: Neural Image Assessment** to score image quality. In NIMA, a deep convolutional neural network (CNN) is trained to predict whether the image will rated by a viewer as looking good (technically) and attractive (aesthetically).

To assess both resolution and perceptual quality, the model first initialize weights from object recognition networks, such as ImageNet, to understand general classification of objects. Then the perceptual quality assessment is achieved by fine-tuning on annotated data. This NIMA model gives a distribution of ratings for a given image on scale of 1 to 10 and also assign the probabilities. NIMA has been tested on Aesthetic Visual Analysis (AVA) datasets, and the rank given by NIMA matches closely the mean scores given by human raters.

Here, I will use the pre-trained the NIMA model Github to predict the image score for each featured photo on the website and this score will be incorporated as a new feature for machine learning model. The workflow will be as follows:

1. Use beautiful soup to scrape images from the url link of the listed houses.
2. Predict the image score use NIMA model.

```
In [15]: import requests
         from bs4 import BeautifulSoup
         import urllib.request
         import scipy.misc

In [18]: listings = new_df['listing_url']

In [34]: new_df = new_df.reset_index()

In [39]: # extract the url for the feature photo from 'listings_url'
         listings = new_df['listing_url']
         image_link = {}
         for i in range(len(listings)):
             file_url = new_df['listing_url'][i]
             page = requests.get(file_url)
             soup = BeautifulSoup(page.text,"html.parser")
             img_tags = soup.find_all('img')
             img_urls = [img['src'] for img in img_tags]
             for url in img_urls:
                 if not url.startswith("https://a0.muscache.com/im/pictures/"):
                     continue
                 image_link[file_url] = url
                 break
```

```
In [69]: # add this featured photo url to the dataframe
         new_df['image_link'] = new_df['listing_url'].map(image_link)
         new_df.to_csv('cleaned_df_with_image_link.csv')

         #some listings are no longer availble, so their image_link is missing.
         new_df = new_df.dropna()

         # set up the path for the photos output
         ROOT_DIR = os.path.dirname(os.path.realpath('__file__'))
         Photo_Path = os.path.join(ROOT_DIR,'Photos')

         if not os.path.exists('Photos'):
             os.makedirs('Photos')
         Photo_path = os.path.join('Photos')

         # scraping images from the link
         df_image = new_df[['id','image_link']].reset_index()

         new_df = new_df.drop(['level_0'],axis = 1)
         new_df = new_df.reset_index()

         for i in range(len(df_image)):
            # link = df_image['image_link'][i]
             url_link = new_df['listing_url'][i]
            # print (url_link)
             link = image_link[url_link]
             photo_id = df_image['id'][i]
             image_name = os.path.join(Photo_Path,str(photo_id)+str('.jpg'))

             if not os.path.isfile(image_name):
                 f = open(image_name,'wb')
                 f.write(requests.get(link).content)
                 f.close()
```

```
In [73]: # take random samples and check if their NIMA scores make sense
         sample = df_image['image_link'][65]
         photo_id = df_image['id'][65]
         image_name = os.path.join(Photo_path, str(photo_id)+str('.jpg'))
         img = scipy.misc.imread(image_name)
         plt.imshow(img)
```

```
Out[73]: <matplotlib.image.AxesImage at 0x152bf94e0>
```

```
In [78]: # use NIMA model to score images
         import tensorflow as tf
         from keras.models import Model
         from keras.layers import Dense, Dropout
         from keras.applications.mobilenet import MobileNet
         from keras.applications.mobilenet import preprocess_input
         from keras.preprocessing.image import load_img, img_to_array
         from utils import mean_score, std_score

         NIMA_dic = {}
         image_name = os.path.join(Photo_path, str(photo_id)+str('.jpg'))

         with tf.device('/CPU:0'):
             base_model = MobileNet((None, None, 3), alpha=1, include_top=False, pooling='avg'
             x = Dropout(0.75)(base_model.output)
             x = Dense(10, activation='softmax')(x)

             model = Model(base_model.input, x)
             model.load_weights('weights/mobilenet_weights.h5')

             for i in range(len(df_image)):
                 try:
                     photo_id = df_image['id'][i]
                     image_name = os.path.join(Photo_path, str(photo_id)+str('.jpg'))
```

```
                img = load_img(image_name)
                x = img_to_array(img)
                x = np.expand_dims(x, axis=0)
                x = preprocess_input(x)

                scores = model.predict(x, batch_size=1, verbose=0)[0]

                mean = mean_score(scores)
                std = std_score(scores)
                NIMA_dic[photo_id] = mean
                #print("NIMA Score : %0.3f +- (%0.3f)" % (mean, std))
            except:
                pass
```

```
In [88]: # add NIMA_score to new_df
         new_df['NIMA_score'] = new_df['id'].map(NIMA_dic)

         # fill Null values with median
         new_df['NIMA_score'] = new_df['NIMA_score'].fillna(new_df['NIMA_score'].median())

         # save file into a csv
         new_df.to_csv('new_df_withNIMA.csv')
```

0.3.2 Sentiment analysis on 'description'

Description of the houses also has a great impact on guest's decision. An appropriate description can not only provide viewers with more details of the room but also leave them good impressions of the living environment using phrases such as "comfortable", "lovely bedroom", "bright and sunny room". So this part will focus on extraccting useful features from description. **Nature language processing (NLP)** and **topic modeling** will be carried out to analyze the text in 'description'.

Topic model is a widely used text-mining tools to discover the abstract "topics" hidden in a collection of documents. Here, **Latent Dirichlet Allocation (LDA)** will be used to discover topics in each description. In LDA model, a generative Bayesian inference model is used to assign each document with a probability distribution over topics, where topics are probability over words.

Before topic modeling, the number of corpus in each description needs to be reduced. Non-english words, stop words and non-alphanumeric strings will be removed. The remaining corpus will also be lemmatised so that only important and meaningful words will be kept later sentiment analysis. The corpus then needs to be converted into a **Document-term-matrix**, where each row corresponding to the documents and column corresponding to the terms.

The pipeline of topic modeling on text of description will be as follows: 1. Tokenize words, remove non-english words, stop words and non-alphanumeric strings, convert all letters to lower case, and lemmatize words. 2. Convert the remaining corpus into Document Term Matrix. 3. Apply LDA model to model topics. 4. Use pyLDAvis.gensim to visualize topics. 5. Assign each observation with the topics with highest probability.

```
In [89]: import nltk
         from nltk import word_tokenize
```

```
        from nltk.corpus import stopwords
        from nltk.stem import WordNetLemmatizer
        from nltk.tokenize import RegexpTokenizer

        from gensim.corpora.dictionary import Dictionary
        from gensim.models.tfidfmodel import TfidfModel
        from gensim.models.ldamodel import LdaModel

        import json
        import pyLDAvis.gensim
        pyLDAvis.enable_notebook()

In [90]: def preprocess_text(corpus):
             processed_corpus = []
             english_words = set(nltk.corpus.words.words())
             english_stopwords = set(stopwords.words('english'))
             wordnet_lemmatizer = WordNetLemmatizer()
             tokenizer = RegexpTokenizer(r'[A-Za-z|!]+')
             for row in corpus:
                 sentences = []
                 word_tokens = tokenizer.tokenize(row)
                 word_tokens_lower = [t.lower() for t in word_tokens]
                 word_tokens_lower_english = [t for t in word_tokens_lower if t in english_wor
                 word_tokens_no_stops = [t for t in word_tokens_lower_english if not t in engli
                 word_tokens_no_stops_lemmatized = [wordnet_lemmatizer.lemmatize(t) for t in wo
                 for word in word_tokens_no_stops_lemmatized:
                     if len(word) >= 2:
                         sentences.append(word)
                 processed_corpus.append(sentences)
             return processed_corpus

         def pipline(processed_corpus):
             dictionary = Dictionary(processed_corpus)
             doc_term_matrix = [dictionary.doc2bow(listing) for listing in processed_corpus]
             return dictionary, doc_term_matrix

         def lda_topic_model(doc_term_matrix,dictionary,num_topics = 3, passes = 2):
             LDA = LdaModel
             ldamodel = LDA(doc_term_matrix,num_topics = num_topics, id2word = dictionary, pas
             return ldamodel

         def topic_feature(ldamodel,doc_term_matrix,df,new_col,num_topics):
             docTopicProbMat = ldamodel[doc_term_matrix]
             docTopicProbDf = pd.DataFrame(index = df.index, columns = range(0,num_topics))
             for i,doc in enumerate(docTopicProbMat):
                 for topic in doc:
                     docTopicProbDf.iloc[i,topic[0]] = topic[1]
             docTopicProbDf = docTopicProbDf.fillna(0)
             docTopicProbDf[new_col] = docTopicProbDf.idxmax(axis=1)
             df_topics = docTopicProbDf[new_col]
             df_new = pd.concat([df,df_topics],axis = 1)
             return df_new
```

```
In [113]: corpus_description = new_df['description'].astype(str)

          # use nlp package to process the text in description
          processed_corpus_description = preprocess_text(corpus_description)

          # generate the doc_term_matrix for lda model
          dictionary_description, doc_term_matrix_description = pipline(processed_corpus_descr

          # lda model for topic modeling
          ldamodel_description = lda_topic_model(doc_term_matrix_description,dictionary_descrip

          # add the topic feature to the dataframe
          final_df = topic_feature(ldamodel_description,doc_term_matrix_description,new_df,new_

          # visualization of the lda model and save it as html page
          p_description = pyLDAvis.gensim.prepare(ldamodel_description, doc_term_matrix_descrip
          pyLDAvis.save_html(p_description,'lda_description.html')

          # save the file
          final_df.to_csv('final_df_with_topicmodel.csv')
```

```
In [95]: pyLDAvis.display(p_description)
```

```
Out[95]: <IPython.core.display.HTML object>
```

pyLDAvis package is a great package to visualize the LDA model. The area of the circles means the prevalence of each topic. Here I chose the cluster the corpus into three topics. The red bar represents the estimated term frequency within selected topic and the blue bar represents the overall term frequency. In topic 1, the prevalent term is about layout of the room, for example, there are words "kitchen", "bathroom", "bedroom". Topic 2 is about the living environment because it has words "new","private","space","large". Topic 3 is correlated with location or transit with words "subway", "walk","away". There are some overlaps among these three topics, which can be improved to better serve the machine learninng model. At this moment, I will go ahead with the current model.

0.4 IV. Machine learning model

In this part, **random forest algorithm** will be applied to the clean dataset and predict the yield. Random forest model can capture the nonlinear relationships in the dataset and it is a complex model to provide high accuracy.

To measure the accuracy of the model, MSE (mean squared error) is used as evaluation metrics. The target for prediction is "yield". "price" and "reviews_per_month", "average_length_of_stay" have strong correlation with "yield" because they are used for yield calculation. Catergorical features also need to be converted to numerical features so that they can be fed into machine learning algorithms. To split the whole dataset into a training set and a testing set, the dataset will be randomly shuffled first and 25% will be used as the splitting ratio.

Once the model have been applied and trained, they will be compared based on MSE value. The smaller MSE, the better accuracy. The best algorithm will be chosen and the model will be further fine-tuned using GridSearchCV function in scikit-learn. The to-do list in this part is:

1. Clean-up the dataset: separate the "yield" from the dataset and save it as the target, drop "price", "average_length_of_stay" and "reviews_per_month", convert catergorical variables into numerical features. Other columns including "level_0", "id", "listing_url", "description","image_link" can be dropped as well since they are not needed any more.
2. In order to evaluate how the two new features captured from photo and description of the listings can improve the performance of the model, both models with and without these two new features will be built.
3. Randomly shuffle the dataset to remove inherent order and split the dataset into a training set and a test set using 75:25 ratio.
4. Use linear regression, decision tree, and random forest separately to train the model and calculate the MSE value.
5. Select the model with lowest MSE value for further refinement.

```
In [131]: final_df =  pd.read_csv('final_df_with_topicmodel.csv')
```

```
In [98]: # features to keep
         cols_to_keep = ['latitude','longitude','accommodates','bathrooms','bedrooms','guests_i
                         'maximum_nights','property_type','room_type','bed_type','cancellation_
         model_df = final_df[cols_to_keep]

         # convert strings to dummies
         categorical_feats = ['property_type','room_type','bed_type','cancellation_policy']
         model_df = pd.get_dummies(model_df,columns = categorical_feats,drop_first = False)

         # separate the target variable "yield" from the dataset
         target = model_df['yield']
         X_df = model_df.drop(['yield'],axis = 1)
```

0.4.1 Random forest

```
In [136]: # split the training set and testing set
          from sklearn.model_selection import train_test_split
          from sklearn.metrics import mean_squared_error,r2_score
          seed = 42
          X_train,X_test,y_train,y_test = train_test_split(X_df,target,random_state=seed)
```

```
In [138]: from sklearn.ensemble import RandomForestRegressor

          for K in [1,3,5,7,10,15,20]:
              rf_reg = RandomForestRegressor(random_state = seed, bootstrap = True, criterion =
                                             max_features = 'auto', min_samples_split = 5, n_es
              y_rf_pred = rf_reg.predict(X_test)
```

```
            print ("Max_depth = " + str(K))
            print("Mean squared error: %.3f" %mean_squared_error(y_test,y_rf_pred))
            print("Variance score: %.3f" %r2_score(y_test,y_rf_pred))
```

```
Max_depth = 1
Mean squared error: 4235835851.867
Variance score: 0.060
Max_depth = 3
Mean squared error: 4219722807.763
Variance score: 0.063
Max_depth = 5
Mean squared error: 4230893974.845
Variance score: 0.061
Max_depth = 7
Mean squared error: 4242613796.671
Variance score: 0.058
Max_depth = 10
Mean squared error: 4229651455.101
Variance score: 0.061
Max_depth = 15
Mean squared error: 4216468094.997
Variance score: 0.064
Max_depth = 20
Mean squared error: 4213626070.418
Variance score: 0.065
```

0.4.2 Implementation: Incorporating the two engineered features

Two new features engineered from photos and descriptions of the houses will be incorporated into the model.

0.4.3 Random forest

```
In [132]: # drop unnecessary columns
          cols_to_keep =  ['latitude','longitude','accommodates','bathrooms','bedrooms','guests
                           'maximum_nights','property_type','room_type','bed_type','cancellatio
          refine_df = final_df[cols_to_keep]

          # convert strings to numerical features
          categorical_feats = ['property_type', 'room_type', 'bed_type', 'cancellation_policy'
          refine_df = pd.get_dummies(refine_df, columns = categorical_feats, drop_first = False

          # separate the target variable "yield" from the dataset
          target = refine_df['yield']
          refine_df = refine_df.drop(['yield'], axis = 1)

In [141]: # split the training set and testing set
          from sklearn.model_selection import train_test_split
          from sklearn.metrics import mean_squared_error,r2_score
          seed = 42
          X_train,X_test,y_train,y_test = train_test_split(refine_df,target,random_state=seed)

In [142]: from sklearn.ensemble import RandomForestRegressor

          for K in [1,3,5,7,10,15,20]:
              rf_reg = RandomForestRegressor(random_state = seed, bootstrap = True, criterion =
                                             max_features = 'auto', min_samples_split = 5, n_es
              y_rf_pred = rf_reg.predict(X_test)

              print ("Max_depth = " + str(K))
              print("Mean squared error: %.3f" %mean_squared_error(y_test,y_rf_pred))
              print("Variance score: %.3f" %r2_score(y_test,y_rf_pred))
```

```
Max_depth = 1
Mean squared error: 4235835851.867
Variance score: 0.060
Max_depth = 3
Mean squared error: 4226199045.345
Variance score: 0.062
Max_depth = 5
Mean squared error: 4241483004.705
Variance score: 0.058
Max_depth = 7
Mean squared error: 4238381313.703
Variance score: 0.059
Max_depth = 10
Mean squared error: 4214493407.211
Variance score: 0.064
Max_depth = 15
Mean squared error: 4192146357.324
Variance score: 0.069
Max_depth = 20
Mean squared error: 4184092191.506
Variance score: 0.071
```

By comparing the MSE score and the variance score, after adding the two new features: NIMA_score and description_topic, the performance of the random forest model got improved.

0.5 V. Fine tuning and model evaluation

As expected, random forest gave the lowest MSE and highest variance score. So this part will focusing on fine tuning the model and test how robust the model is. It will be structured into three parts:

1. Using GridSearchCV to fine tuning the model using random forest regressor.

2. Check the importance of each feature in the dataset, especially the two features from image analysis and text mining.
3. Test the robustness of the model by using a different seed.

```
In [145]: from sklearn.model_selection import GridSearchCV

          param_grid = {"n_estimators" :[150,175,200,225,250,300],
                        "criterion": ['mse'],
                        "max_features": ['auto'],
                        "max_depth": [5,7,9,11,15,20,25,30],
                        "min_samples_split":[4,6,8,10,12],
                        "bootstrap":[True]}

          rf_fine = RandomForestRegressor(random_state = seed)
          rf_cv = GridSearchCV(rf_fine,param_grid,cv=5).fit(X_train,y_train)
          y_rf_cv_pred = rf_cv.predict(X_test)
          print("Mean squared error: %.3f" % mean_squared_error(y_test, y_rf_cv_pred))
          print('Variance score: %.3f' % r2_score(y_test, y_rf_cv_pred))
          print("Best Parameters: {}".format(rf_cv.best_params_))
```

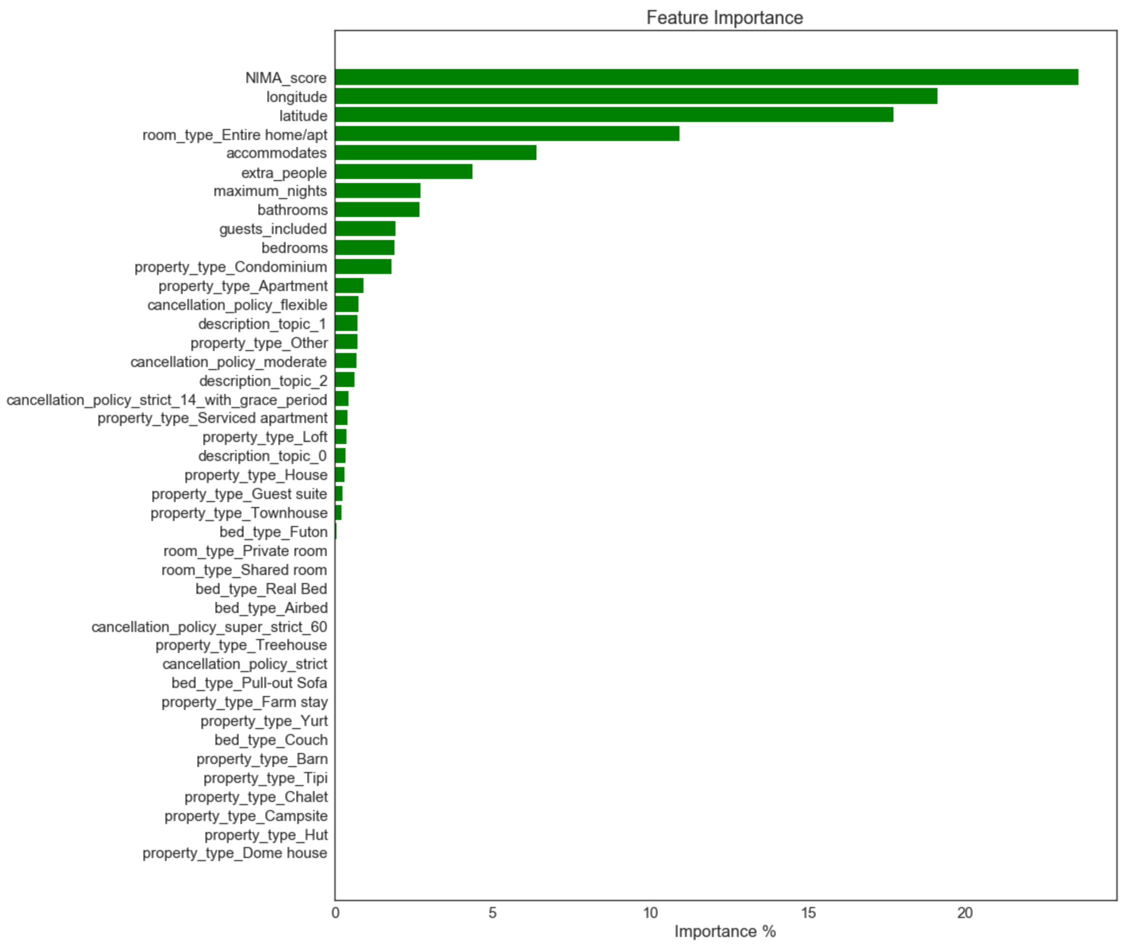
```
Mean squared error: 4177564595.927
Variance score: 0.073
Best Parameters: {'bootstrap': True, 'criterion': 'mse', 'max_depth': 25, 'max_features': 'auto
```

```
In [179]: rf_final = rf_cv.best_estimator_
          feature_import = rf_final.feature_importances_*100
          feature_import = pd.DataFrame(list(zip(feature_import,X_train.columns.values)))
          feature_import = feature_import.sort_values(by=0,axis=0,ascending=False)
          feature_import.columns = ['importance %','feature']
          print(feature_import[:20])
```

```
    importance %                          feature
8      23.591967                       NIMA_score
1      19.125749                        longitude
0      17.729351                         latitude
26     10.940065          room_type_Entire home/apt
2       6.383180                     accommodates
6       4.358200                     extra_people
7       2.714976                   maximum_nights
3       2.674377                        bathrooms
5       1.931795                  guests_included
4       1.883962                         bedrooms
13      1.800505          property_type_Condominium
9       0.924392            property_type_Apartment
34      0.753243        cancellation_policy_flexible
40      0.726294                description_topic_1
20      0.705806                property_type_Other
35      0.684130        cancellation_policy_moderate
```

```
41      0.615718                                 description_topic_2
37      0.441630  cancellation_policy_strict_14_with_grace_period
21      0.398022                    property_type_Serviced apartment
19      0.358925                                  property_type_Loft
```

```
In [184]: features = feature_import['feature']
          importances = feature_import['importance %']

          fig,ax = plt.subplots(figsize=(14,16))
          y_pos = np.arange(len(features))
          ax.barh(y_pos, importances, align='center',color='green')
          ax.set_yticks(y_pos)
          ax.set_yticklabels(features)
          ax.invert_yaxis()  # labels read top-to-bottom
          ax.set_xlabel('Importance %')
          ax.set_title('Feature Importance')

          plt.show()
```



From the feature importance rank, it's very nice to see that **NIMA score** has a importance of **23.59%** and it is the most important feature in this model. Location has a combined importance of 36.86% - 19.13% from longitude and 17.73% from latitude, which make sense to me. A convenient location can be very attractive for viewers. Other features such as "accommodates" also occupied 6.38% importance. The other feature **"description_topic"** also has combined **>1%** of importance (sum of "description_topic_1" and "description_topic_2"). This information shows that there are valuable information in the photo and description text.

To test the robustness of the model, random_state for shuffling dataset will be changed, the ratio of training set and test set will also be changed to 0.2.

```
In [189]: random_state = 65
          X_train,X_test,y_train,y_test = train_test_split(refine_df,target,test_size = 0.2,ra
```

```
In [190]: # Fit and make prediction
          rf_final.fit(X_train, y_train)
          rf_y_pred = rf_final.predict(X_test)

          # The mean squared error
          print("Mean squared error: %.2f" % mean_squared_error(y_test, rf_y_pred))
          # Explained variance score: 1 is perfect prediction
          print('Variance score: %.2f' % r2_score(y_test, rf_y_pred))
```

```
Mean squared error: 4046491479.02
Variance score: 0.06
```

There is no significant difference after adjusting the random state and proportions of training set and test set, which demonstrate that the final model is robust.

0.6 VI. Conclusion and reflection

The original goal of this project is to apply machine learning algorithms to give potential hosts some insights on how much they can earn from listing their beloved houses on Airbnb. The information from Inside Airbnb is definitely very helpful. Combined my own experience of browsing accommodations in Airbnb, I added two additional features: image score and topic modeling from web photos and descriptions. It turned out that these two features actually contain lots of valuable informations and play important roles when building machine learning models. Of couse, my solution is not perfect, here are two points I would like to spend more time on further improving my model.

1. There are some overlaps among the three topics, so potential improvement would be implement the topic modeling methods. It would be worthwhile comparing LDA with other algorithms, such as Non-negative matrix factorization.

2. Should I consider time effect? If a host gets very positive reviews from first few guests, it's possible that new viewers will also consider choosing their houses. How should I predict time series?

3. Try other machine learning models.

4. How to convert this to a real data product? A web application ,which allow potential hosts to upload their photos and other informations about their houses so that they can check the expected income given their input information, can be the next step.