Boston Airbnb Analysis

As one of the most popular online listing service for all-kinds of leases around the world. Set up an Airbnb business may be easy, however it is difficult to make it successful.

We are going to apply data analytics in order to explore if Airbnb is viable or not. Also, we will build a recommendation system based on customer's reviews towards the listings they have ever experienced.

The following questions will be answered:
✔️How much do Airbnb hosts make?
✔️What are the best types of property to rent?
✔️What is the best time to rent?
✔️Which are the best areas to rent?
✔️What should you write in a listing name to attract more attention?
✔️What features are important for increasing the revenue?

Data Loading & Cleaning

Dataset:

Listings(3585,95):contains all the properties from Boston Airbnb, including features from all kinds of text description, prices, locations, and even information of hosts.

The dataset has too many columns, some of which are useless for our analysis, so we decide to select out 14 columns as our target features for our analysis.

The format of all the data is string with special symbols such as %,$. That's why we are going to remove the symbol and transform string into float.
Eg:

```
# Convert string of pecentages to floats
listings.host_response_rate = listings.host_response_rate.str.replace('%','')
listings.host_response_rate = listings.host_response_rate.astype(float)/100
```

```
# Convert string of prices to floats
listings.price=listings.price.str.replace('$','')
listings.price=listings.price.str.replace(',','').astype(float)
```

Next, we need to fill the null value with reasonable ones.

```
listings.isnull().any()
```

```
id                          False
host_response_time           True
host_response_rate           True
host_is_superhost           False
host_neighbourhood           True
property_type                True
room_type                   False
accommodates                False
price                       False
security_deposit             True
cleaning_fee                 True
minimum_nights              False
review_scores_rating         True
cancellation_policy         False
dtype: bool
```

For numerical columns, we fill the null with the mean number

```
# fill the null numerical data with mean
listings['host_response_rate'] = listings['host_response_rate'].fillna(listings['host_response_rate'].mean())
listings['security_deposit'] = listings['security_deposit'].fillna(listings['security_deposit'].mean())
listings['cleaning_fee'] = listings['cleaning_fee'].fillna(listings['cleaning_fee'].mean())
listings['review_scores_rating'] = listings['review_scores_rating'].fillna(listings['review_scores_rating'].mean())
```

For categorical columns, we fill the null with mode of the column

```
# fill null categorical data with mode
listings['host_response_time'] = listings['host_response_time'].fillna(listings['host_response_time'][listings['host_re
listings['host_neighbourhood'] = listings['host_neighbourhood'].fillna(listings['host_neighbourhood'][listings['host_ne
listings['property_type'] = listings['property_type'].fillna(listings['property_type'][listings['property_type'].notnul
```

Finally, the dataset comes to:

	id	host_response_time	host_response_rate	host_is_superhost	host_neighbourhood	property_type	room_type	accommodates	price
0	12147973	within an hour	0.949891	f	Roslindale	House	Entire home/apt	4	250.0
1	3075044	within an hour	1.000000	f	Roslindale	Apartment	Private room	2	65.0
2	6976	within a few hours	1.000000	t	Roslindale	Apartment	Private room	2	65.0
3	1436513	within a few hours	1.000000	f	Allston-Brighton	House	Private room	4	75.0
4	7651065	within an hour	1.000000	t	Roslindale	House	Private room	2	79.0

security_deposit	cleaning_fee	minimum_nights	review_scores_rating	cancellation_policy
324.698212	35.000000	2	91.916667	moderate
95.000000	10.000000	2	94.000000	moderate
324.698212	68.380145	3	98.000000	moderate
100.000000	50.000000	1	100.000000	moderate
324.698212	15.000000	2	99.000000	flexible

Calendar(1308890,4):Calendar records properties' price per day from 2016-09-06 to 2017-09-05 and we also transform the data in price column from string into float. And similarly, we fill the null value.

```
# finding the matched price in table listing and fill the null value
def fill_price(calendar,listings):
    calendar['price'] = calendar['price'].fillna(0)

    price = calendar['price']
    listing_id = calendar['listing_id']
    num = len(price)

    for i in range(0,num):
        if price[i] == 0:
            list_id = listing_id[i]
            list_price = listings[listings['id'] == list_id].price
```

The calendar data comes to:

	listing_id	date	available	price
0	12147973	2017/9/5	f	250
1	12147973	2017/9/4	f	250
2	12147973	2017/9/3	f	250
3	12147973	2017/9/2	f	250
4	12147973	2017/9/1	f	250

Reviews(68275,6):The dataset of reviews collected the comments of user's towards properties from 2009-3-21 to 2016-9-6. The number of reviews for a property and the content are both important for our analysis, we are going to talk about them later.

	listing_id	id	date	reviewer_id	reviewer_name	comments
0	1178162	4724140	2013-05-21	4298113	Olivier	My stay at islam's place was really cool! Good...
1	1178162	4869189	2013-05-29	6452964	Charlotte	Great location for both airport and city - gre...
2	1178162	5003196	2013-06-06	6449554	Sebastian	We really enjoyed our stay at Islams house. Fr...
3	1178162	5150351	2013-06-15	2215611	Marine	The room was nice and clean and so were the co...
4	1178162	5171140	2013-06-16	6848427	Andrew	Great location. Just 5 mins walk from the Airp...

Data Statistictics:

As our final purpose is to increase revenue, we need to estimate the annual revenue for every property.

Before that ,we could have a glimpse at average price during the whole period, based on data from the table 'calendar' which has record the properties' price per day from 2016-09-06 to 2017-09-05.

And we decided to display the variation by average monthly price. So first, we transform the date format into '%Y-%m' (only year and month).

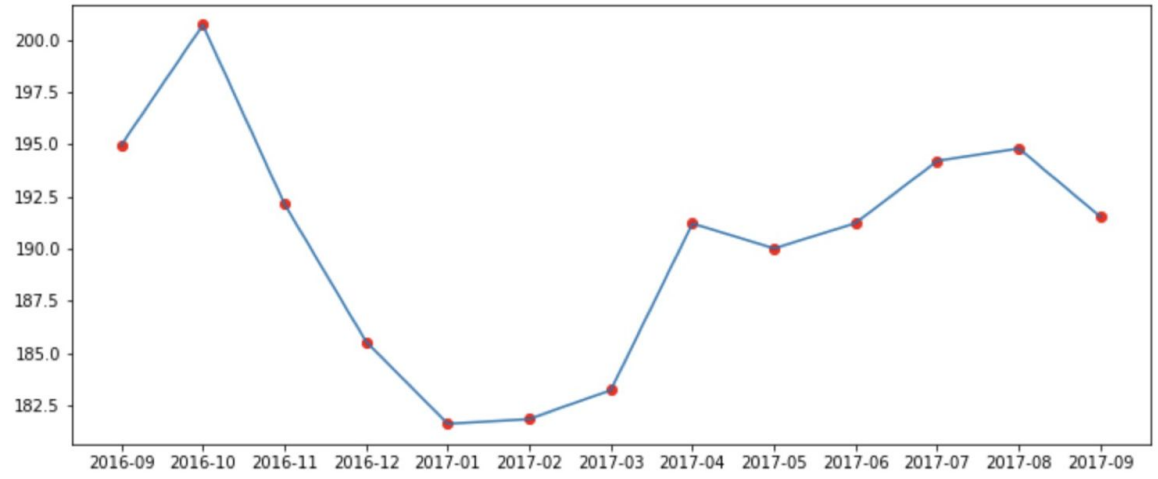
```
# transform date into only year with month
calendar['date'] = pd.to_datetime(calendar['date'],format= "%Y-%m-%d")
calendar['date'] = calendar['date'].apply(lambda x:x.strftime("%Y-%m"))
calendar['date'].head()
```

```
0    2017-09
1    2017-09
2    2017-09
3    2017-09
4    2017-09
```

Then use this column to group by and get the count of the properties, the sum of the price for each price. So for each month, the average monthly price = sum of the price / count of the properties.

```
# calculate the average price for month
calendar_month = calendar.groupby('date',as_index = False).agg({'listing_id': 'count', 'price': 'sum'})
calendar_month.rename(columns={'listing_id': 'amount', 'price': 'sum_month_price'},inplace=True)
calendar_month['average'] = calendar_month['sum_month_price']/calendar_month['amount']
```

We plot the data get the trend of the price:



We can see the price from October to January continuously decreased and reached the lowest position. It started to increase significantly at March to April and reached the highest price at July and August. This trend is quite reasonable, because Boston's weather starts to turn cold at October and be quite frozen around January. So few people will choose to come to Boston in winter for a trip. But after March, the weather starts to turn warm and number of visitors may meet a peek at July and August because of summer vacation.

Revenue:

Here we come to the revenue, we certainly should know the price and the order amount. We didn't have a detailed transaction records' table,but we can estimate the number of bookings for each property based on the number of reviews they received during the time period.

And we can see that some value in column 'comment' actually is null, so we can assume that this table not only collect the records who has comment but also collect the records without comment.

```
reviews.isnull().any()
```

```
listing_id       False
id               False
date             False
reviewer_id      False
reviewer_name    False
comments          True
dtype: bool
```

We pick 2015-09-07 to 2016-09-06 as the year-period revenue. The revenue calculation format for each property is:

$$Revenue = review\_num\_2015 \times listing\_price \times min\_night$$

'review_num_2015' is the total records' number of the property in table 'review', 'listing_price' and 'min_night' is the matched price and the minimum required stay nights of the property in table 'listing' which has record the detailed information of each property.

We use group by in table 'review' to get 'review_num_2015':

```
reviews_2015 = reviews[(reviews.date >= '2015-01-01') & (reviews.date < '2016-01-01')]

# calculate each property's review number in 2015
reviews_2015_agg = reviews_2015.groupby(['listing_id'],as_index=False).agg({'id':'count'})
reviews_2015_agg.rename(columns={"id": "number_of_reviews_2015"},inplace=True)
reviews_2015_agg.head()
```

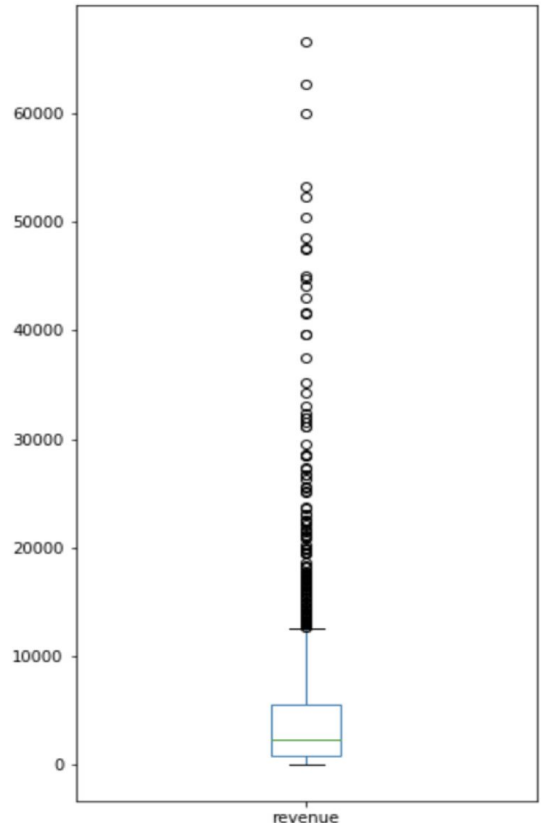
	listing_id	number_of_reviews_2015
0	3353	8
1	5506	3
2	6695	2
3	6976	7
4	8792	7

And then we merge the table above with table 'listing' and do the calculation:

```
# merge the review number into datatset listing
listing_agg = pd.merge(listings, reviews_2015_agg, how='inner', left_on='id',right_on='listing_id')
```

```
# calculate revenue
listing_agg['revenue'] = listing_agg['price']*listing_agg['minimum_nights']*listing_agg['number_of_reviews_2015']
```

We used the Box-plot to see the distribution of the revenue in 2015. (Because the value higher than 100000 (only 4 points) will cause the diagram to be too long and cause the box looks very small in the whole diagram, we just plot the diagram with the value lower than 100000, but it will not cause the changing of the distribution's shape):



```
listing_agg['revenue'].describe()
```

```
count      1571.000000
mean       5068.504774
std       10430.361009
min          32.000000
25%         880.000000
50%        2290.000000
75%        5625.000000
max      224224.000000
Name: revenue, dtype: float64
```

In 2015, the average estimated revenue for a Boston Airbnb host was 5068 dollars, but revenue disparity was prevalent. 50% of the hosts gained less than 2290 dollars in revenues, while the top Airbnb host in Boston received over 224,000 dollars of revenue in 2015! In fact, 4 of the highest hosts received at least 100,000 dollars in revenue during that year.

price	security_deposit	cleaning_fee	minimum_nights	review_scores_rating	cancellation_policy	number_of_reviews_2015	revenue
88.0	324.698212	68.380145	28	96.0	moderate	91	224224.0
88.0	324.698212	68.380145	28	96.0	moderate	65	160160.0
150.0	324.698212	70.000000	30	91.0	flexible	24	108000.0
695.0	600.000000	150.000000	3	96.0	strict	48	100080.0
60.0	324.698212	10.000000	30	90.0	strict	37	66600.0

From the table above, we can see that most of these top host of revenue has a high score rating, all of them are higher than 90. Also, their minimum nights are very high, except the forth one. But the forth one has a really high price. So we might think the revenue is quite relevant to the score rating, minimum night and price. For figure out which features are the most important for revenue, we will build a regressor tree to get feature importance.

Before stepping into the part of regression, we still focus on the visualization using Tableau to represent the secret from the countless number of data.

Data Visualization:

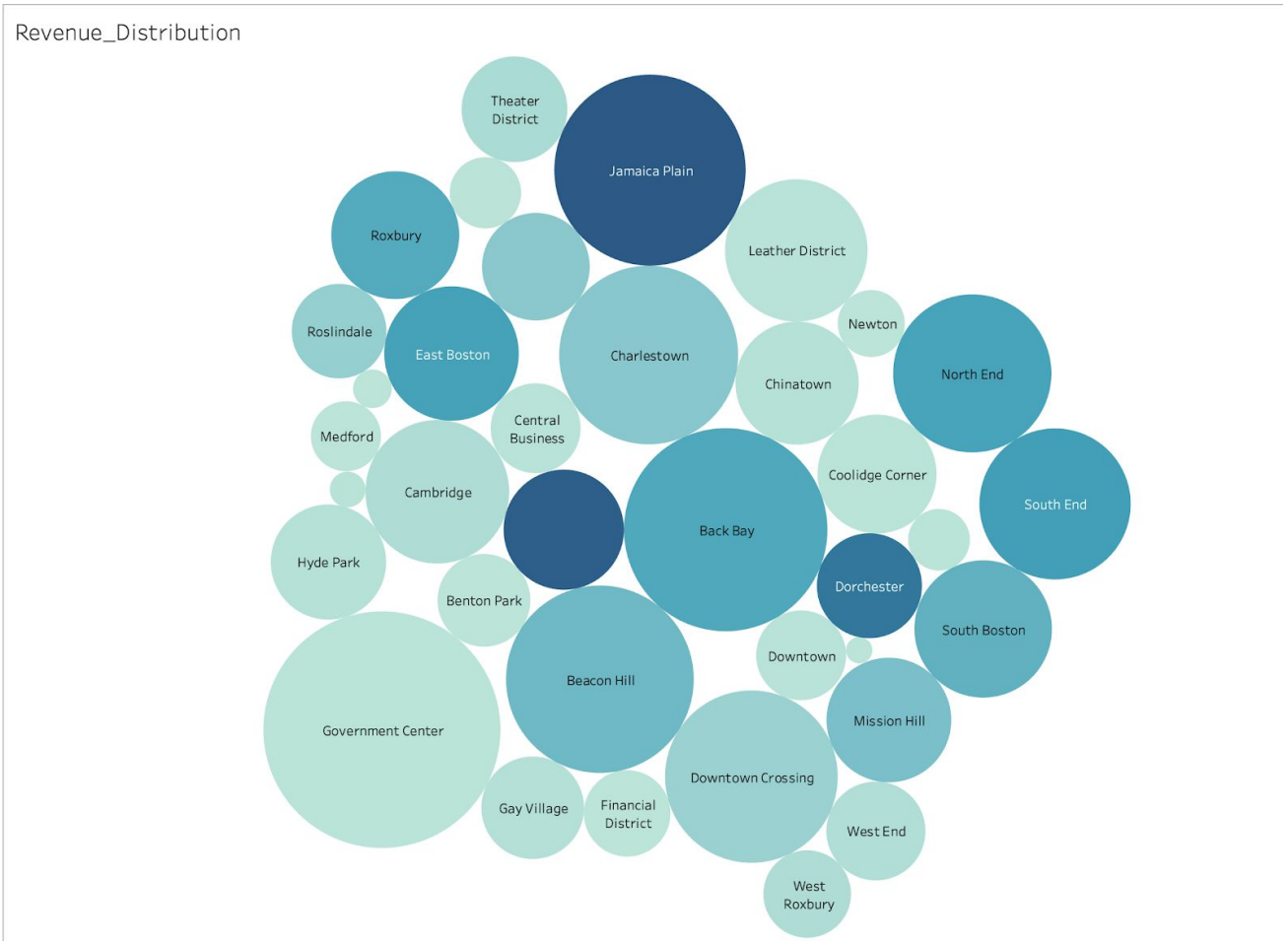
You should rent out an Entire home or apartment.

(Number:average revenue)

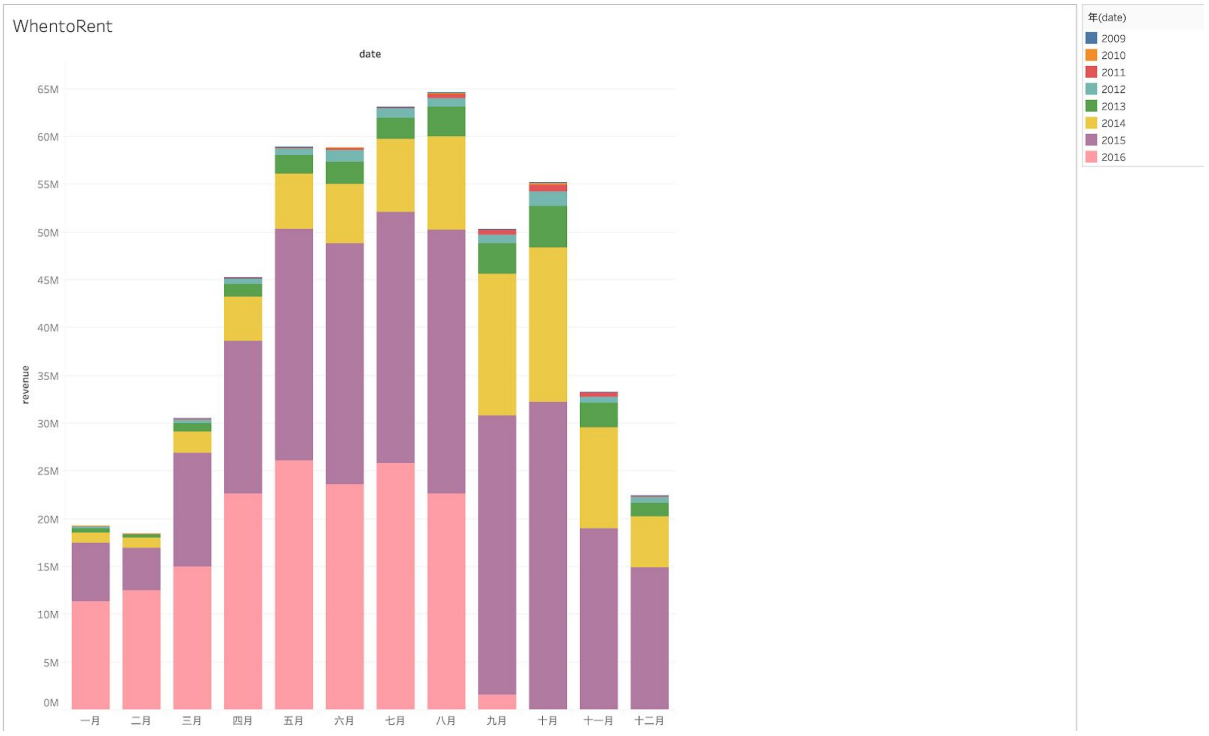
	room_type (listing_agg.csv)		
property_type..	Entire home/apt	Private room	Shared room
Apartment	10,945	4,649	2,787
Bed & Breakfast	12,723	5,043	
Boat	2,186	1,380	
Condominium	5,713		2,759
House	10,565	4,868	2,134
Loft	23,263	1,603	1,876
Other		2,309	2,500
Townhouse	3,906	3,054	
Villa		1,360	

Jamaica Plain is a good place for host to rent:

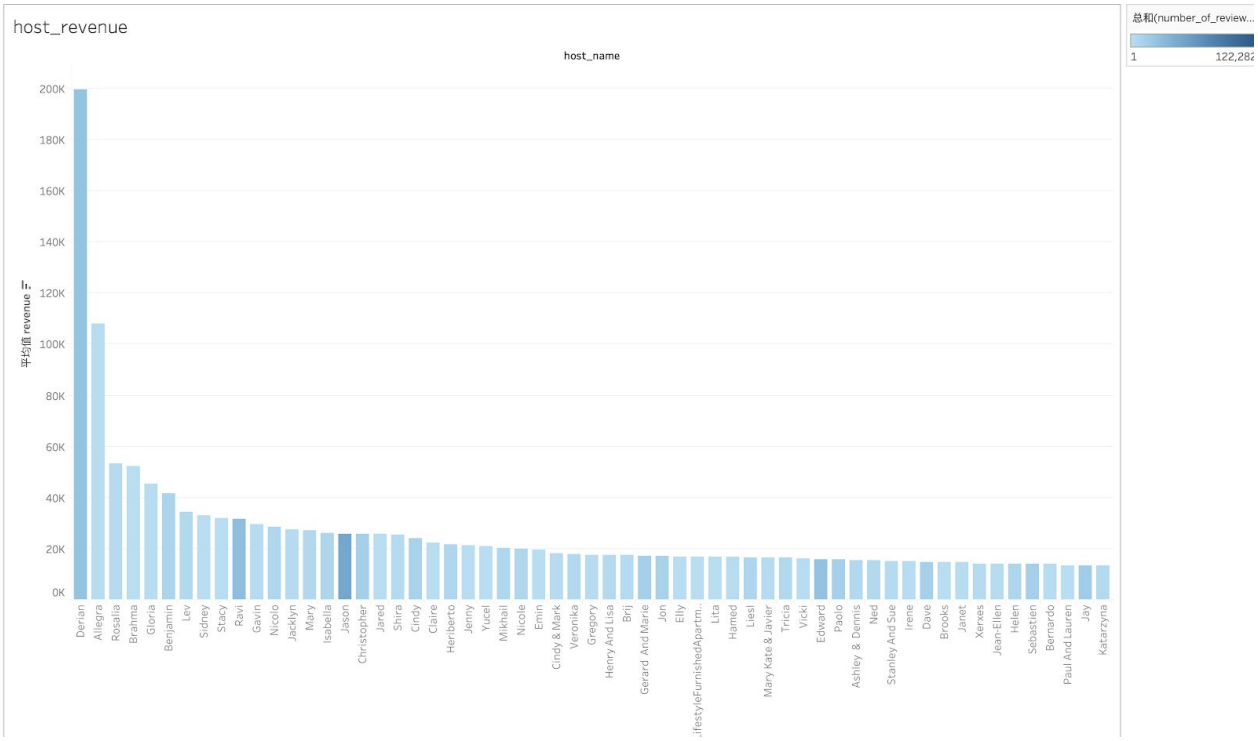
(Color depth:number of bookings; Size: average revenue)



May to October is the best time to rent:



Hosts are rich!

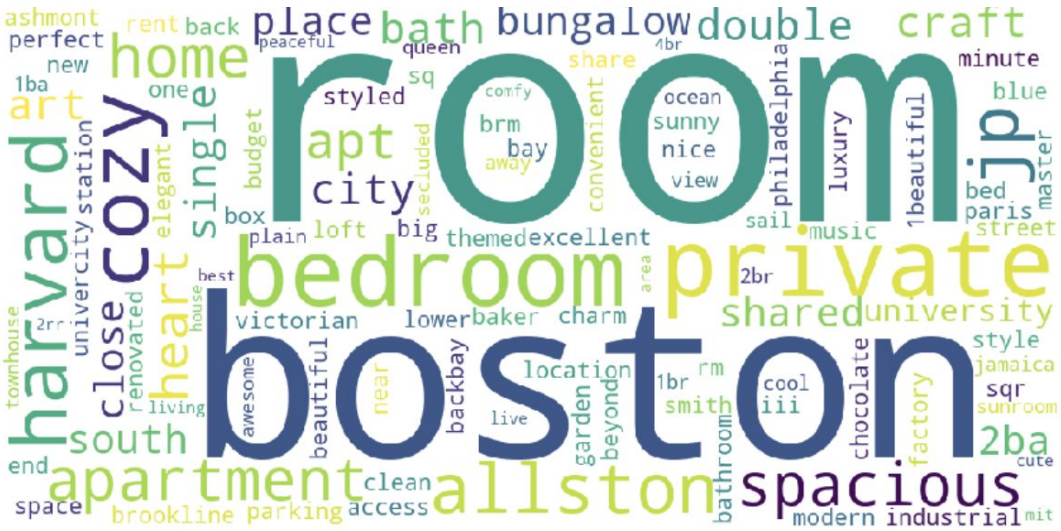


Mention popular locations in your listing name

When we look insight the name of listings, we are about to see that comparing with last50 property, the top50 mentions more about locations instead of just delivering the room itself.

Top50:

Last50:



Regression analysis:

For figuring out the importance of different feature to revenue, we build a random forest with regressor tree and the model to calculate the importance rank for us.

Data preprocessing：

One hot encoding

For regressor tree, we have to do one hot encoding to let the categorical data be numerical

```
# one hot encoding for categorical data
host_response_time = pd.get_dummies(df_regress['host_response_time'])
host_is_superhost = pd.get_dummies(df_regress['host_is_superhost'])
property_type = pd.get_dummies(df_regress['property_type'])
room_type = pd.get_dummies(df_regress['room_type'])
host_neighbourhood = pd.get_dummies(df_regress['host_neighbourhood'])
cancellation_policy = pd.get_dummies(df_regress['cancellation_policy'])
```

Normalization

Because when regressor tree calculating the importance of features, it will use MSE, and the scale of data will impact MSE, thus, also impact the final result, so we have to do the normalization.

```
#normalization for numerical data
df_regress = df_regress.apply(lambda x : (x-np.mean(x))/np.std(x))
```

Random Forest:

We split the dataset into 8:2 for training data and testing data, and build a random forest with 10 trees.

The final regression score is about 0.7 which is not too bad, though it's not the important point.

Result of feature importance with top 20:

```
 1) minimum_nights                 0.255209
 2) price                          0.167260
 3) moderate                       0.086956
 4) Jamaica Plain                  0.074361
 5) host_response_rate             0.054181
 6) review_scores_rating           0.041549
 7) is_spuer_host                  0.037677
 8) within an hour                 0.035759
 9) Back Bay                       0.028453
10) security_deposit               0.027343
11) cleaning_fee                   0.027095
12) Condominium                    0.026109
13) accommodates                   0.020666
14) Private room                   0.013947
15) East Boston                    0.013111
16) Beacon Hill                    0.009404
17) Entire home/apt                0.008762
18) North End                      0.008410
19) South End                      0.008389
20) strict                         0.006046
```

We can see the top 7 important features are minimum_nights, 'moderate'(cancellation policy), price, 'Jamaica Plain'(neighborhood), host_response_rate, review_scores_rating, is_spuer_host. It is really reasonable that when price, score rate, response rate and minimum nights are higher and when the host is a super host, the revenue can be higher. And the result also indicates that people will also pay attention to

the cancellation policy and prefer to moderate policy. 'Jamaica Plain' is one of the neighbourhood in boston and this result is a little bit surprised to us. We first thought Back bay will be the top in area but it is the eighth important feature, the second important area.

Sentiment Analysis:

Unsupervised learning:

The main idea behind unsupervised learning is that you don't give any previous assumptions and definitions to the model about the outcome of variables you feed into it — you simply insert the data (of course preprocessed before), and want the model to learn the structure of the data itself.

In this case, because every customer's review does not have the rating, that's why we are going to build an unsupervised learning model.

The main idea:

The main idea for unsupervised sentiment analysis is that negative and positive words usually are surrounded by similar words. Although there may exist exceptions like 'I love that because it is not boring at all', we focus on a large number of data, such exceptions will be discarded.
The perfect tool for such problem(of having words that are similar to their surrounding) is the one and only Word2Vec. We will be talking more details about that below.

Data Processing:

For the raw text in the comments, we need to clean them first including remove kind of meaningless symbols,numbers. Actually, it always serves as the first time for any natural language process, because unlike number, text is more complicated.

Word2Vec:

Word2Vec is a method to construct such an embedding which is obtained using Common Bag of Words(CBOW)
What are Word embeddings exactly? Loosely speaking, they are vector representations of a particular word
CBOW Model: This method takes the context of each word as the input and tries to predict the word corresponding to the context

```
8]: w2v_model = Word2Vec(min_count=3,
                         window=4,
                         size=300,
                         sample=1e-5,
                         alpha=0.03,
                         min_alpha=0.0007,
                         negative=20,
                         workers=multiprocessing.cpu_count()-1)
    start = time()

    w2v_model.build_vocab(sentences, progress_per=50000)

    print('Time to build vocab: {} mins'.format(round((time() - start) / 60, 2

    Time to build vocab: 0.27 mins
```

min_count = 3 ----remove most unusual words from training embeddings, like words ' ssssuuuuupppppeeeerrr' which actually stands for 'super', and doesn't need additional training.

window = 4 ----Word2Vec model will learn to predict given word from up 4 words to the left and up to 4 words to the right

size = 300 ---- size of hidden layer used to predict surroundings of embedded word, which also stands for dimensions of trained embeddings.

sample = 1e-5 ---- probability baseline for subsampling most frequent words from surrounding of embedded word.

negative= 20 ---- number of negative(ones that shouldn't have been predicted while modeling selected pair of words) words that will have their corresponding weight updated while training on specific training example, along with positive word.

K-means Clustering:

K-means clustering is a basic technique for data clustering, and it seemed most suitable for a given problem, as it takes as an input number of necessary clusters, and outputs coordinates of calculated clusters centroids (central points of discovered clusters)

In the given problem I used sklearn's implementation of K-means algorithm with 50 repeated starting points, to presumably prevent the algorithm from choosing wrong starting centroid coordinates, that would lead the algorithm to converge to not optimal clusters, and 1000 iterations of reassigning points to clusters.

```
model = KMeans(n_clusters=2, max_iter=1000, random_state=True, n_init=50).fit(X=word_vectors.vectors)
```

After running it on estimated word vectors, I got 2 centroid.Next, to check which cluster is relatively positive, and which negative, with use of gensim's most_similar method I checked what word vectors are most similar in terms of cosine similarity to coordinates of first cluster:

```
In [66]: word_vectors.similar_by_vector(model.cluster_centers_[0], topn=10, restric

Out[66]: [('serious_cleaning', 0.8859721422195435),
          ('touristy_spots', 0.8813823461532593),
          ('maintained_daily', 0.879956841468811),
          ('jazmyne', 0.8755927681922913),
          ('margarita', 0.875381350517273),
          ('square_footage', 0.8750718832015991),
          ('aiming', 0.8669908046722412),
          ('becouse', 0.8666425347328186),
          ('quick_stopover', 0.8605995178222656),
          ('go_freely', 0.8604874610900879)]
```

As we can see 10 closest words to cluster no. 0 in terms of cosine distance are the ones with positive sentiment.

Assigning cluster:

The next step is to assign each word sentiment score ---- negative or positive value(-1 or 1) based on the cluster to which they belong. To weigh this score I multiplied it by how close they were to their cluster(to weigh how potentially positive/negative they are). As the score that

K-means algorithm outputs is distance from both clusters, to properly weight them I multiplied them by the inverse of closeness score.

```
In [68]: words = pd.DataFrame(word_vectors.vocab.keys())
         words.columns = ['words']
         words['vectors'] = words.words.apply(lambda x: word_vectors.wv[f'{x}'])
         words['cluster'] = words.vectors.apply(lambda x: model.predict([np.array(x)]))
         words.cluster = words.cluster.apply(lambda x: x[0])

         /anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:3: DeprecationWarning: Call to deprecated `wv` (Attribut
         e will be removed in 4.0.0, use self instead).
           This is separate from the ipykernel package so we can avoid doing imports until

In [69]: words['cluster_value'] = [1 if i==0 else -1 for i in words.cluster]
         words['closeness_score'] = words.apply(lambda x: 1/(model.transform([x.vectors]).min()), axis=1)
         words['sentiment_coeff'] = words.closeness_score * words.cluster_value

In [70]: words[['words', 'sentiment_coeff']].to_csv('sentiment_dictionary.csv', index=False)
```

So far we are able to build sentiment_dictionary, in which every word has its weighted sentiment score. Words in the table below mostly end up in the correct cluster, though I must admit that many words didn't look so promising. Probably, the best option to correct it would be to normalize data properly or to create 3rd, neutral cluster for words that shouldn't have any sentiment at all assigned to them, but in order to not make this project too big, I didn't improve them, and it still worked pretty well, as you will see later.

Out[9]:

	words	sentiment_coeff
20670	serious_cleaning	1.597197
4002	touristy_spots	1.585309
16382	square_footage	1.578513
21835	maintained_daily	1.577958
24999	jazmyne	1.572394
14219	margarita	1.567519
27788	becouse	1.554773
22026	aiming	1.553439
27568	go_freely	1.551003
18224	quick_stopover	1.548995
21074	main_reasons	1.544407
7372	eased	1.540255
26611	music_box	1.539582

Then we will move to the part of prediction.

Tfidf weighting and sentiment prediction

In this step, we are going to calculate tfidf score of each word in each sentence with sklearn's TfidfVectorizer. This step is conducted to consider how unique every word is for every sentence, and increase positive/negative signal associated with words that are highly specific for given sentence in comparison to whole corpus.

```
tfidf = TfidfVectorizer(tokenizer=lambda y: y.split(), norm=None)
tfidf.fit(final_file.comments)
features = pd.Series(tfidf.get_feature_names())
transformed = tfidf.transform(final_file.comments)
```

Finally, all words in every sentence were on one hand replaced with their tfidf scores, and on the other with their corresponding weighted sentiment scores.

```
]: def create_tfidf_dictionary(x, transformed_file, features):
       vector_coo = transformed_file[x.name].tocoo()
       vector_coo.col = features.iloc[vector_coo.col].values
       dict_from_coo = dict(zip(vector_coo.col, vector_coo.data))
       return dict_from_coo

   def replace_tfidf_words(x, transformed_file, features):
       dictionary = create_tfidf_dictionary(x, transformed_file, features)
       return list(map(lambda y:dictionary[f'{y}'], x.comments.split()))
```

```
]: replaced_tfidf_scores = final_file.apply(lambda x: replace_tfidf_words(x, transformed, features), axis=1)
```

Gists above and below present functions for replacing words in sentences with their associated tfidf/sentiment scores, to obtain 2 vectors for each sentence

```
]: def replace_sentiment_words(word, sentiment_dict):
       try:
           out = sentiment_dict[word]
       except KeyError:
           out = 0
       return out
```

```
]: replaced_closeness_scores = final_file.comments.apply(lambda x: list(map(lambda y: replace_sentiment_words(y, sentiment
```

The dot product of such 2 sentence vectors indicated whether overall sentiment was positive or negative (if the dot product was positive, the sentiment was positive, and in opposite case negative).

```
In [14]: replaced_closeness_scores = final_file.comments.apply(lambda x: list(map(lambda y: replace_sentiment_words(y, sentiment
```

```
In [15]: replacement_df = pd.DataFrame(data=[final_file.listing_id,final_file.id,final_file.reviewer_id,final_file.comments,rep
         replacement_df.columns = ['listing_id','id','reviewer_id','comments','sentiment_coeff', 'tfidf_scores']
         replacement_df['sentiment_rate'] = replacement_df.apply(lambda x: np.array(x.loc['sentiment_coeff']) @ np.array(x.loc[
         replacement_df['prediction'] = (replacement_df.sentiment_rate>0).astype('int8')
```

```
In [16]: replacement_df.head()
```

Out[16]:

	listing_id	id	reviewer_id	comments	sentiment_coeff	tfidf_scores	sentiment_rate	prediction
0	1178162	4724140	4298113	my stay at islam s place was really cool ! goo...	[1.0340865918003272, 1.081973735252718, 1.0439...	[2.739815972569313, 1.9027477633752095, 2.3949...	221.908462	1
1	1178162	4869189	6452964	great location for both airport and city great...	[1.053914608905532, 1.046935095678005, 1.04800...	[3.663483429427597, 2.1612853767894076, 1.8429...	78.089494	1
2	1178162	5003196	6449554	we really_enjoyed our stay at_islams house fro...	[1.0380807097110565, 1.0655643933965595, 1.029...	[3.856259860314039, 4.848497120190661, 4.93764...	423.442734	1
3	1178162	5150351	2215611	the room was nice and clean and so were the co...	[1.045809937026322, 1.0431377329714395, 1.0693...	[3.69634184583942, 2.524888895273744, 1.361725...	142.581635	1
4	1178162	5171140	6848427	great location just 5_mins walk from the airpo...	[1.053914608905532, 1.046935095678005, 1.02714...	[1.8317417147137984, 2.1612853767894076, 6.154...	78.078184	1

User-based collaborative filter recommendation

We use sentiment rate got from sentiment analysis to do the recommendation.

	listing_id	reviewer_id	sentiment_rate
0	1178162	4298113	221.928332
1	1178162	6452964	78.096883
2	1178162	6449554	423.292134
3	1178162	2215611	142.634225
4	1178162	6848427	78.088891

The table above has 2827 unique properties and 63367 unique users, so there are too many users and properties in the table, and we just choose those users who has more than 5 records in the dataset, otherwise the data might be too sparse and too large.

We first used group by to get the record's number of each user then get all the data of the users who has more than 5 records.

```
# get these users's data in review_score table
users = a.index
df = pd.DataFrame()
for i in range (0,67):
    r = review_score[review_score['reviewer_id']== users[i]]
    df = pd.concat([df,r], ignore_index = True)
```

Now, it remained 278 properties and 67 users. Then we build a null table whose index is users' is and columns are properties' id and find out the matched sentiment rate for each pair of (user_id, listing_id) and fill the rate into the null table.

Result:

listing_id	5729845	4000384	6513924	4331214	3392423	3693850	31796	1544702	1391215	4090224	...	7181950
eviewer_id												
51538	58.385623	478.422061	538.146928	49.615487	225.045388	133.642223	0.000000	0.000000	0.000000	0.000000	...	0.0
114538	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	6.890452	44.184787	53.623242	213.883931	...	0.0
1089634	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0
1399007	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0
1695789	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0

rows × 278 columns

User-based collaborative filter is finding the similar users and recommend the item they like but you haven't tried before. So first, we have to calculate users' similarity. Cosine similarity measures the size of the difference between two individuals by using the cosine value of two vector angles in the vector space. When the cosine value is closer to 1, the two vectors are more similar.

$$\frac{x \bullet y}{\sqrt{x \bullet x}\sqrt{y \bullet y}}$$

We used sklearn's cosine_similarity to do this calculation and get the result:

reviewer_id	51538	114538	1089634	1399007	1695789	1812574	1911807	3230853	3751402	4491477	...	44445333
reviewer_id												
51538	1.0	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	...	0.0
114538	0.0	1.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.043669	0.0	...	0.0
1089634	0.0	0.0	1.000000	0.0	0.070807	0.0	0.0	0.0	0.000000	0.0	...	0.0
1399007	0.0	0.0	0.000000	1.0	0.000000	0.0	0.0	0.0	0.000000	0.0	...	0.0
1695789	0.0	0.0	0.070807	0.0	1.000000	0.0	0.0	0.0	0.000000	0.0	...	0.0

For example, the similarity of user 1089634 and user 1695789 is about 7%. It's not a big value but in true life, it's really hard to find two person who can be quite similar unless there only few items in the system and everyone doesn't have too much different choice. When the items are more in system, the average similarity should be lower.
Then we defined a function for finding the top 5 similar users

```
def find_top5_similar_users(user_id, user_similarity):
    similarity = user_similarity.loc[user_id]
    similarity = similarity.T.sort_values(ascending= False)
    top5 = similarity.iloc[1:6]
    return top5
```

and try user 51538 as example to get the recommended propertied for him.
First, we find the top 5 similar users for user 51538, and get the union of the properties that his similar users has a positive review rate.

```
# use user 51538 as example
# top 5 most similar users
similar_users = find_top5_similar_users(51538,user_similarity).index

#get the properties id that each similmar user has reivewed and also has positive rate
user_1 = review_score['listing_id'][(review_score['reviewer_id']== similar_users[0])&(review_score['sentiment_rate']>0)
user_2 = review_score['listing_id'][(review_score['reviewer_id']== similar_users[1])&(review_score['sentiment_rate']>0)
user_3 = review_score['listing_id'][(review_score['reviewer_id']== similar_users[2])&(review_score['sentiment_rate']>0)
user_4 = review_score['listing_id'][(review_score['reviewer_id']== similar_users[3])&(review_score['sentiment_rate']>0)
user_5 = review_score['listing_id'][(review_score['reviewer_id']== similar_users[4])&(review_score['sentiment_rate']>0)

# get a union of similar users' recommended properties
union = user_1 + user_2 + user_3 + user_4 + user_5
```

Then we removed the properties that user 51538 has reviewed from the union

```
#get the properties id that inputted user has reivewed
user = review_score['listing_id'][review_score['reviewer_id']== 51538].values

# get the difference set between the union and inputted user
# contains the properties that inputted user hasn't live and recommended by his similar users
rest = list(set(union).difference(user))
rest = pd.Series(rest)
```

At last, output all the rest properties and its recommended time

```
# recommended properties's id and recommended times
rest.value_counts().sort_values()
```

```
6765855     1
3703674     1
2378421     1
7988755     1
6347026     1
7823025     1
1067184     1
12309083    1
8193344     1
4530670     1
11397511    1
4583526     1
1391492     1
750438      1
8067585     1
4223387     1
7956634     1
2564544     1
dtype: int64
```