NYC_Final_Version

April 27, 2022

[41]:
```
!pip install rfpimp
```

```
Requirement already satisfied: rfpimp in c:\users\samsu\anaconda3\lib\site-
packages (1.3.7)
Requirement already satisfied: matplotlib in c:\users\samsu\anaconda3\lib\site-
packages (from rfpimp) (3.4.3)
Requirement already satisfied: scikit-learn in
c:\users\samsu\anaconda3\lib\site-packages (from rfpimp) (1.0.2)
Requirement already satisfied: numpy in c:\users\samsu\anaconda3\lib\site-
packages (from rfpimp) (1.20.3)
Requirement already satisfied: pandas in c:\users\samsu\anaconda3\lib\site-
packages (from rfpimp) (1.4.1)
Requirement already satisfied: python-dateutil>=2.7 in
c:\users\samsu\anaconda3\lib\site-packages (from matplotlib->rfpimp) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in
c:\users\samsu\anaconda3\lib\site-packages (from matplotlib->rfpimp) (8.4.0)
Requirement already satisfied: cycler>=0.10 in
c:\users\samsu\anaconda3\lib\site-packages (from matplotlib->rfpimp) (0.10.0)
Requirement already satisfied: pyparsing>=2.2.1 in
c:\users\samsu\anaconda3\lib\site-packages (from matplotlib->rfpimp) (3.0.4)
Requirement already satisfied: kiwisolver>=1.0.1 in
c:\users\samsu\anaconda3\lib\site-packages (from matplotlib->rfpimp) (1.3.1)
Requirement already satisfied: six in c:\users\samsu\anaconda3\lib\site-packages
(from cycler>=0.10->matplotlib->rfpimp) (1.16.0)
Requirement already satisfied: pytz>=2020.1 in
c:\users\samsu\anaconda3\lib\site-packages (from pandas->rfpimp) (2021.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\users\samsu\anaconda3\lib\site-packages (from scikit-learn->rfpimp) (2.2.0)
Requirement already satisfied: joblib>=0.11 in
c:\users\samsu\anaconda3\lib\site-packages (from scikit-learn->rfpimp) (1.1.0)
Requirement already satisfied: scipy>=1.1.0 in
c:\users\samsu\anaconda3\lib\site-packages (from scikit-learn->rfpimp) (1.7.1)
```

[1]:
```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier,KNeighborsRegressor
from sklearn.preprocessing import␣
↪LabelEncoder,OneHotEncoder,StandardScaler,MinMaxScaler
```

```
from sklearn.metrics import␣
↪mean_squared_error,accuracy_score,r2_score,mean_absolute_error
from datetime import datetime
from gensim.models import Word2Vec, word2vec
from math import sqrt
from sklearn.tree import DecisionTreeRegressor
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import RandomizedSearchCV,train_test_split
import pickle
from sklearn.ensemble import RandomForestRegressor
from folium import Choropleth, Circle, Marker
from folium.plugins import HeatMap, MarkerCluster
import math
from matplotlib_venn import venn2, venn2_circles
import lightgbm as lgb
from sklearn import metrics
import folium
from sklearn.model_selection import GroupKFold,StratifiedKFold
from sklearn.metrics import mean_absolute_error as mae
from nltk.corpus import stopwords
import nltk
import gensim.downloader
import re
from tqdm import tqdm
import joblib
from rfpimp import importances
```

[43]:
```
# nltk.download('punkt')
# nltk.download('stopwords')
# glove_vectors = gensim.downloader.load('glove-twitter-25')
# glove_vectors = gensim.load('glove-twitter-50')
```

1 LOADING DATA

[2]:
```
df =pd.read_csv("nycdata_final.csv")
vectors = joblib.load("glove25_vecs.pkl")
vec_text = joblib.load("glove25_vecs_text.pkl")
# tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
df.head()
```

[2]:
```
   Unnamed: 0    id                                      name  \
0           0  2539            Clean & quiet apt home by the park
1           1  2595                         Skylit Midtown Castle
2           2  3647           THE VILLAGE OF HARLEM…NEW YORK !
3           3  3831               Cozy Entire Floor of Brownstone
```

```
4          4  5022  Entire Apt: Spacious Studio/Loft by central park

   host_id    host_name neighbourhood_group neighbourhood  latitude  \
0     2787         John            Brooklyn    Kensington  40.64749
1     2845     Jennifer           Manhattan       Midtown  40.75362
2     4632    Elisabeth           Manhattan        Harlem  40.80902
3     4869  LisaRoxanne            Brooklyn  Clinton Hill  40.68514
4     7192        Laura           Manhattan   East Harlem  40.79851

   longitude        room_type  …  minimum_nights  number_of_reviews  \
0  -73.97237     Private room  …               1                  9
1  -73.98377  Entire home/apt  …               1                 45
2  -73.94190     Private room  …               3                  0
3  -73.95976  Entire home/apt  …               1                270
4  -73.94399  Entire home/apt  …              10                  9

   last_review reviews_per_month  calculated_host_listings_count  \
0   2018-10-19              0.21                               6
1   2019-05-21              0.38                               2
2          NaN               NaN                               1
3   2019-07-05              4.64                               1
4   2018-11-19              0.10                               1

   availability_365                geom  \
0               365  40.64749,-73.97237
1               355  40.75362,-73.98377
2               365   40.80902,-73.9419
3               194  40.68514,-73.95976
4                 0  40.79851,-73.94399

                                             address postcode  \
0  807, Friel Place, Windsor Terrace, Kings Count…  11218.0
1  Bryant Park, 6th Avenue, Theater District, Man…  10018.0
2  25, West 128th Street, East Harlem, Manhattan …  10027.0
3  188, Gates Avenue, Brooklyn, Kings County, New…  11238.0
4  1626, Park Avenue, East Harlem, Manhattan Comm…  10029.0

                                           add_text
0    807, Friel Place, Windsor Terrace, Kings County
1  Bryant Park, 6th Avenue, Theater District, Man…
2  25, West 128th Street, East Harlem, Manhattan …
3          188, Gates Avenue, Brooklyn, Kings County
4  1626, Park Avenue, East Harlem, Manhattan Comm…

[5 rows x 21 columns]
```

[4]:
```
df.isnull().sum()
```

[4]:
```
Unnamed: 0                            0
id                                    0
name                                 16
host_id                               0
host_name                            21
neighbourhood_group                   0
neighbourhood                         0
latitude                              0
longitude                             0
room_type                             0
price                                 0
minimum_nights                        0
number_of_reviews                     0
last_review                       10052
reviews_per_month                 10052
calculated_host_listings_count        0
availability_365                      0
geom                                  0
address                               0
postcode                             59
add_text                              0
dtype: int64
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 21 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Unnamed: 0                      48895 non-null  int64
 1   id                              48895 non-null  int64
 2   name                            48879 non-null  object
 3   host_id                         48895 non-null  int64
 4   host_name                       48874 non-null  object
 5   neighbourhood_group             48895 non-null  object
 6   neighbourhood                   48895 non-null  object
 7   latitude                        48895 non-null  float64
 8   longitude                       48895 non-null  float64
 9   room_type                       48895 non-null  object
 10  price                           48895 non-null  int64
 11  minimum_nights                  48895 non-null  int64
 12  number_of_reviews               48895 non-null  int64
 13  last_review                     38843 non-null  object
 14  reviews_per_month               38843 non-null  float64
 15  calculated_host_listings_count  48895 non-null  int64
 16  availability_365                48895 non-null  int64
```

```
 17   geom                            48895 non-null   object
 18   address                         48895 non-null   object
 19   postcode                        48836 non-null   float64
 20   add_text                        48895 non-null   object
dtypes: float64(4), int64(8), object(9)
memory usage: 7.8+ MB
```

[6]:
```
df.nunique()/df.shape[0]
```

[6]:
```
Unnamed: 0                        1.000000
id                                1.000000
name                              0.979753
host_id                           0.766070
host_name                         0.234216
neighbourhood_group               0.000102
neighbourhood                     0.004520
latitude                          0.389569
longitude                         0.301012
room_type                         0.000061
price                             0.013785
minimum_nights                    0.002229
number_of_reviews                 0.008058
last_review                       0.036077
reviews_per_month                 0.019164
calculated_host_listings_count    0.000961
availability_365                  0.007485
geom                              0.999509
address                           0.755681
postcode                          0.004847
add_text                          0.754065
dtype: float64
```

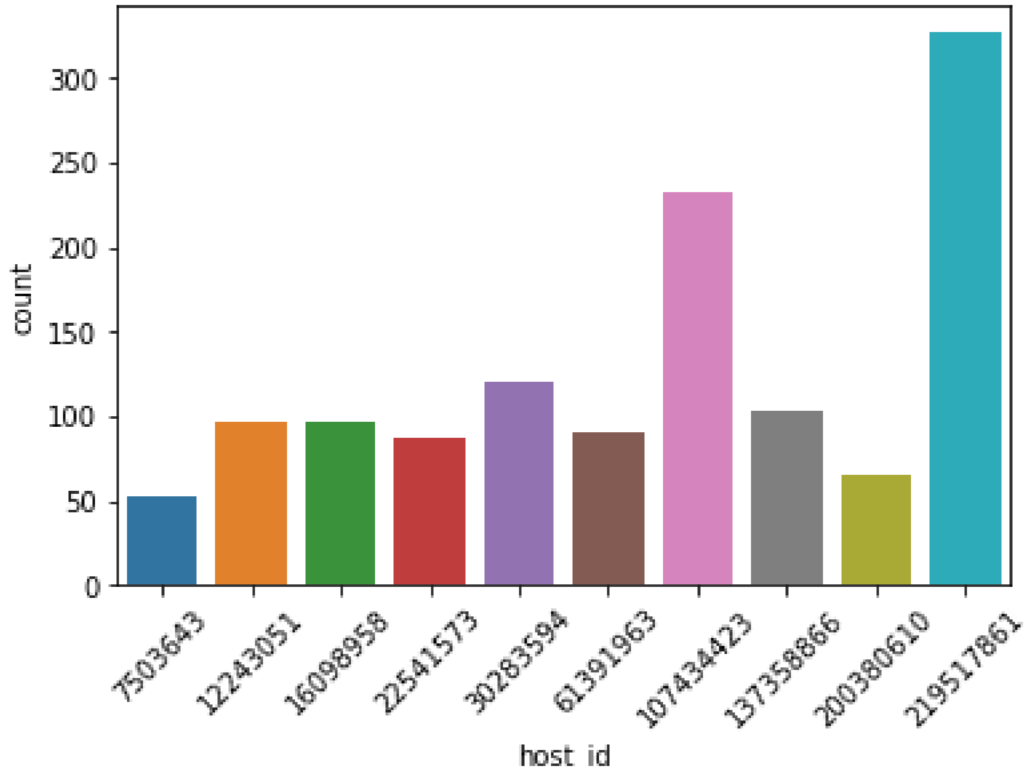
1.0.1 Data preprocessing

[7]:
```
df_host = pd.DataFrame(df['host_id'].value_counts().head(10))
df_host.reset_index(inplace = True)
df_host.rename(columns = {'index': 'host_id', 'host_id': 'count'}, inplace =␣
 ↪True)
```

[8]:
```
#Don't need this there are more than 37k unique hosts, i.e 76% unique hosts

sns.barplot(data = df_host, x = 'host_id', y = 'count')
plt.xticks(rotation = 45)
```

[8]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 [Text(0, 0, '7503643'),
  Text(1, 0, '12243051'),
```

```
 Text(2, 0, '16098958'),
 Text(3, 0, '22541573'),
 Text(4, 0, '30283594'),
 Text(5, 0, '61391963'),
 Text(6, 0, '107434423'),
 Text(7, 0, '137358866'),
 Text(8, 0, '200380610'),
 Text(9, 0, '219517861')])
```



[9]:
```python
df[df['host_id']==2787]
```

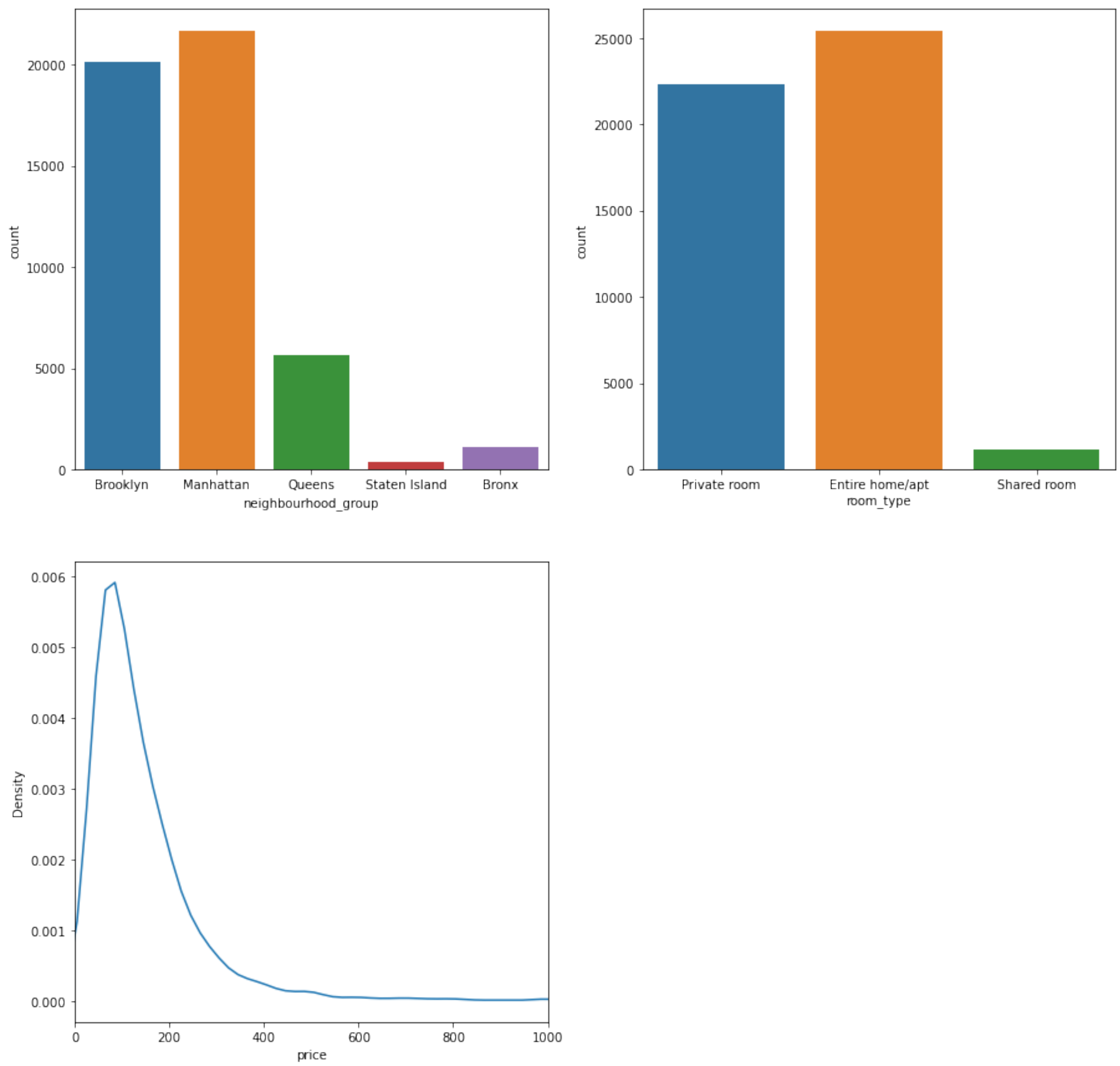
[9]:
```
       Unnamed: 0        id  \
0               0      2539
10372       10372   7937553
13583       13583  10160215
13688       13688  10267242
13963       13963  10593675
21556       21556  17263207

                                                   name  host_id host_name  \
0                      Clean & quiet apt home by the park     2787      John
10372  Riomaggiore Room. Queen Bedroom in Bklyn Townh…     2787      John
```

```
13583                              Torre del Lago Room.     2787      John
13688  Cinque Terre Room.  Clean and Quiet Queen Bedroom     2787      John
13963  La Spezia room.  Clean, quiet and comfortable bed     2787      John
21556   Brooklyn home.  Comfort and clean. Liguria room.     2787      John

      neighbourhood_group neighbourhood  latitude  longitude     room_type  \
0                Brooklyn    Kensington  40.64749  -73.97237  Private room
10372            Brooklyn   Bensonhurst  40.60951  -73.97622  Private room
13583            Brooklyn     Gravesend  40.60755  -73.97410  Private room
13688            Brooklyn     Gravesend  40.60810  -73.97541  Private room
13963            Brooklyn   Bensonhurst  40.60951  -73.97642   Shared room
21556            Brooklyn   Bensonhurst  40.60877  -73.97382  Private room

       …  minimum_nights  number_of_reviews  last_review reviews_per_month  \
0      …               1                  9   2018-10-19              0.21
10372  …               1                 21   2018-10-27              0.50
13583  …               1                 17   2019-06-26              0.40
13688  …               1                 24   2019-05-11              0.64
13963  …               1                 15   2018-09-29              0.43
21556  …               1                 19   2019-06-08              0.70

       calculated_host_listings_count  availability_365                geom  \
0                                   6               365  40.64749,-73.97237
10372                               6               153  40.60951,-73.97622
13583                               6               174   40.60755,-73.9741
13688                               6               180   40.6081,-73.97541
13963                               6               180  40.60951,-73.97642
21556                               6               360  40.60877,-73.97382

                                                address postcode  \
0      807, Friel Place, Windsor Terrace, Kings Count…  11218.0
10372  1552, West 2nd Street, Brooklyn, Kings County,…  11204.0
13583  1642, Dahill Road, Brooklyn, Kings County, New…  11223.0
13688  1613, West 2nd Street, Brooklyn, Kings County,…  11223.0
13963  1550, West 2nd Street, Brooklyn, Kings County,…  11204.0
21556  372, Avenue P, Brooklyn, Kings County, New Yor…  11223.0

                                             add_text
0      807, Friel Place, Windsor Terrace, Kings County
10372    1552, West 2nd Street, Brooklyn, Kings County
13583        1642, Dahill Road, Brooklyn, Kings County
13688    1613, West 2nd Street, Brooklyn, Kings County
13963    1550, West 2nd Street, Brooklyn, Kings County
21556            372, Avenue P, Brooklyn, Kings County

[6 rows x 21 columns]
```

How many are data points in each location and room types?

[10]:
```
plt.figure(figsize = (15, 15))
plt.subplot(221)
sns.countplot(data = df, x = 'neighbourhood_group')
plt.subplot(222)
sns.countplot(data = df, x = 'room_type')
plt.subplot(223)
df['price'].plot.density()
plt.xlabel('price')
plt.xlim(0,1000)
```

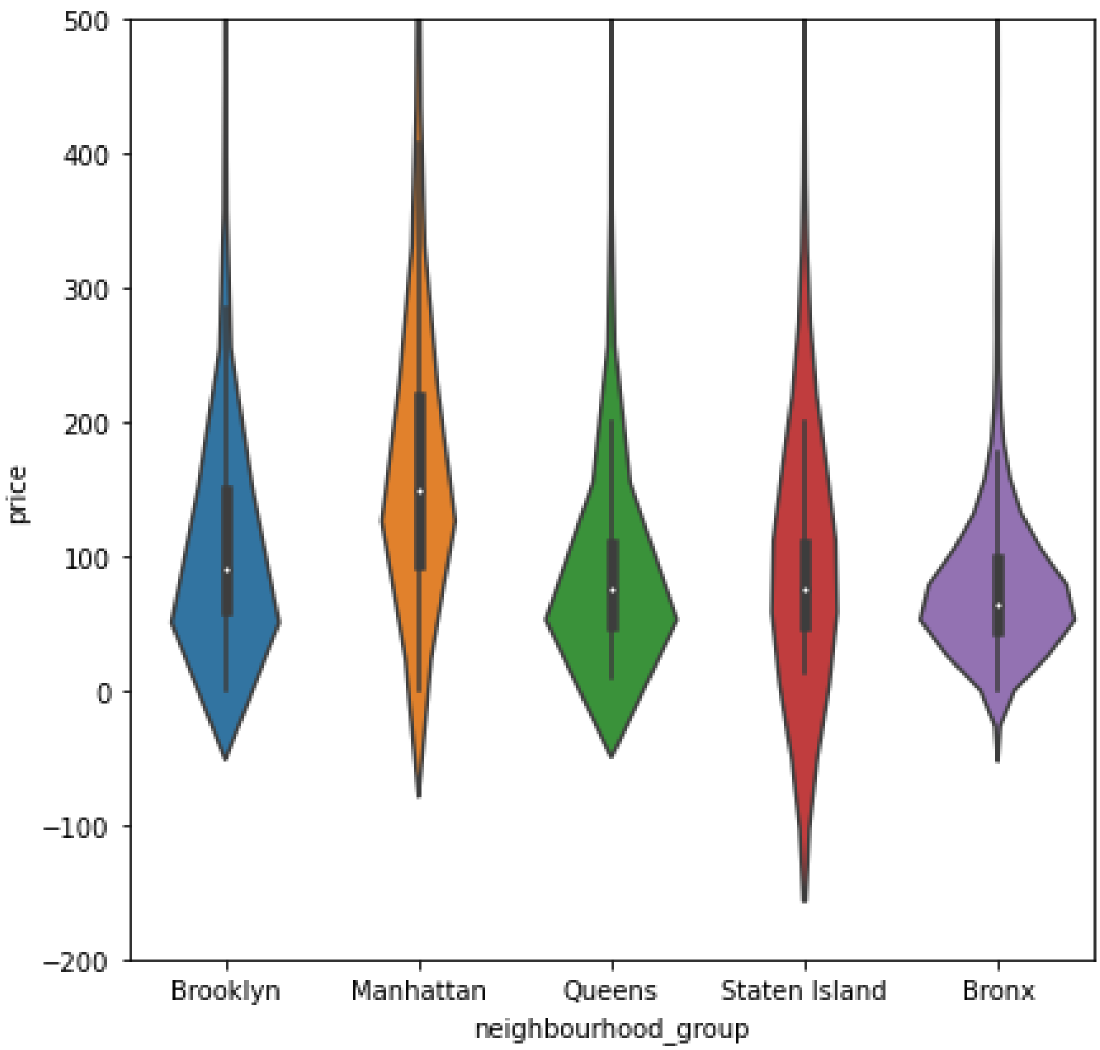
[10]: (0.0, 1000.0)



Neighbourhoods group and price

[11]:
```
plt.figure(figsize = (15, 15))
plt.subplot(221)
sns.violinplot(data=df, x='neighbourhood_group',y='price')
plt.ylim(-200,500)
```

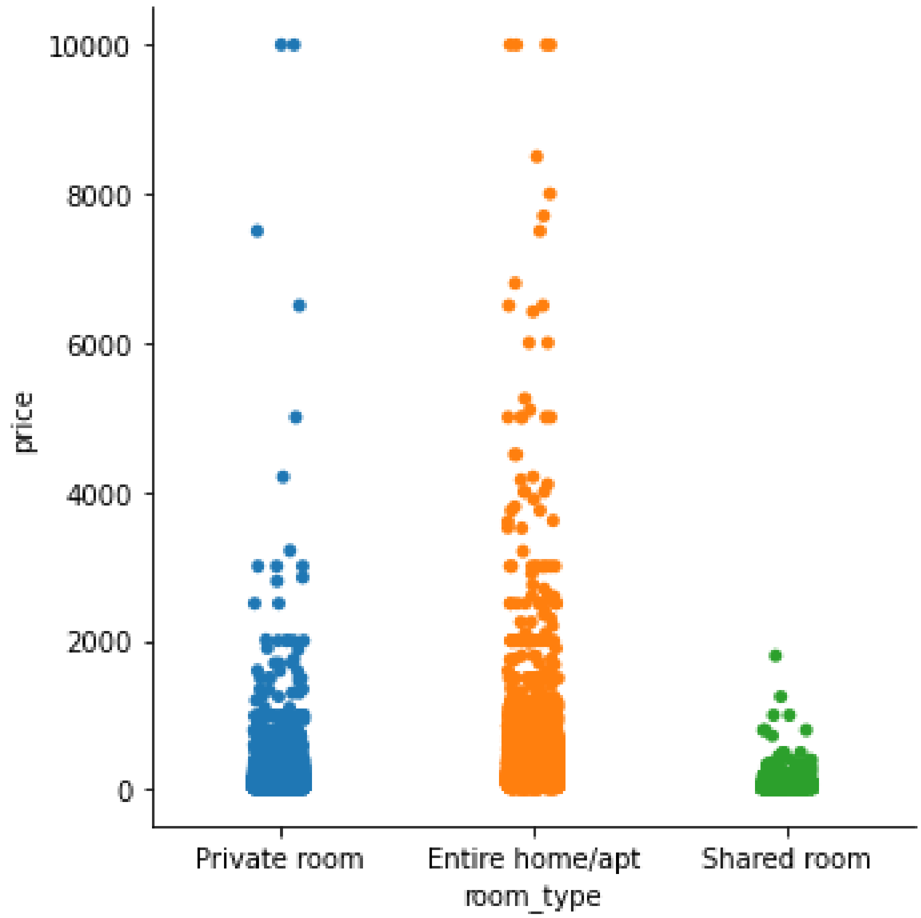
[11]: (-200.0, 500.0)



Room type and price

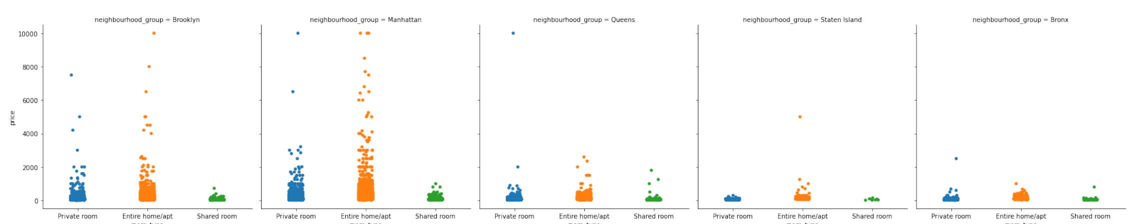
[12]:
```
sns.catplot(data = df, x = 'room_type', y = 'price')
```

[12]: <seaborn.axisgrid.FacetGrid at 0x21930a797f0>



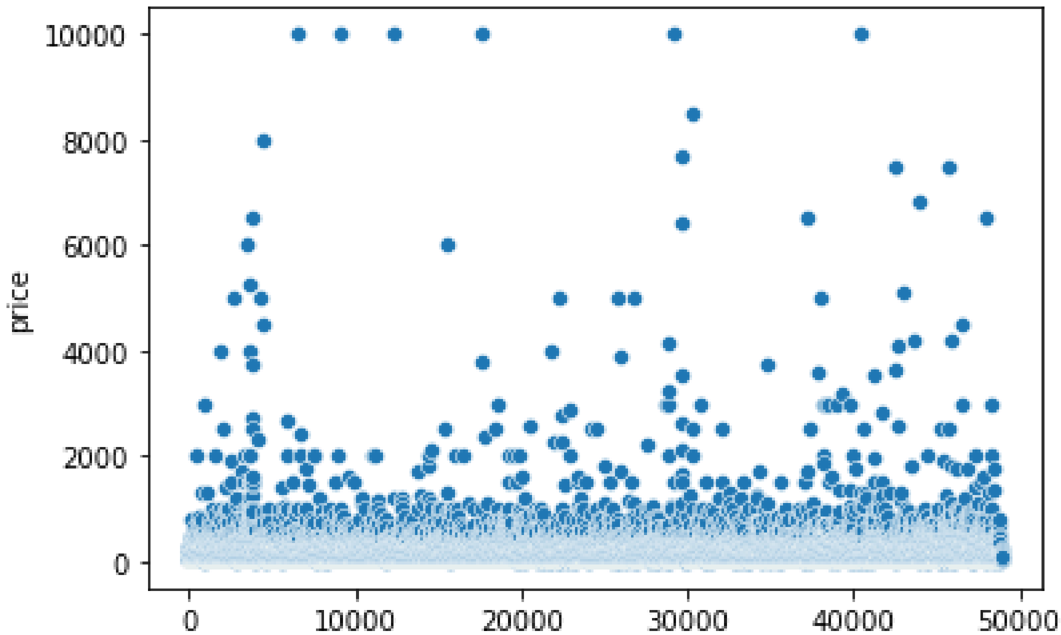

```
[13]: #Cat plot
      sns.catplot(x="room_type", y="price", col="neighbourhood_group", data=df)
```

```
[13]: <seaborn.axisgrid.FacetGrid at 0x2193bbb3fd0>
```



```
[14]: sns.scatterplot(y='price',x = df.index,data=df)
```

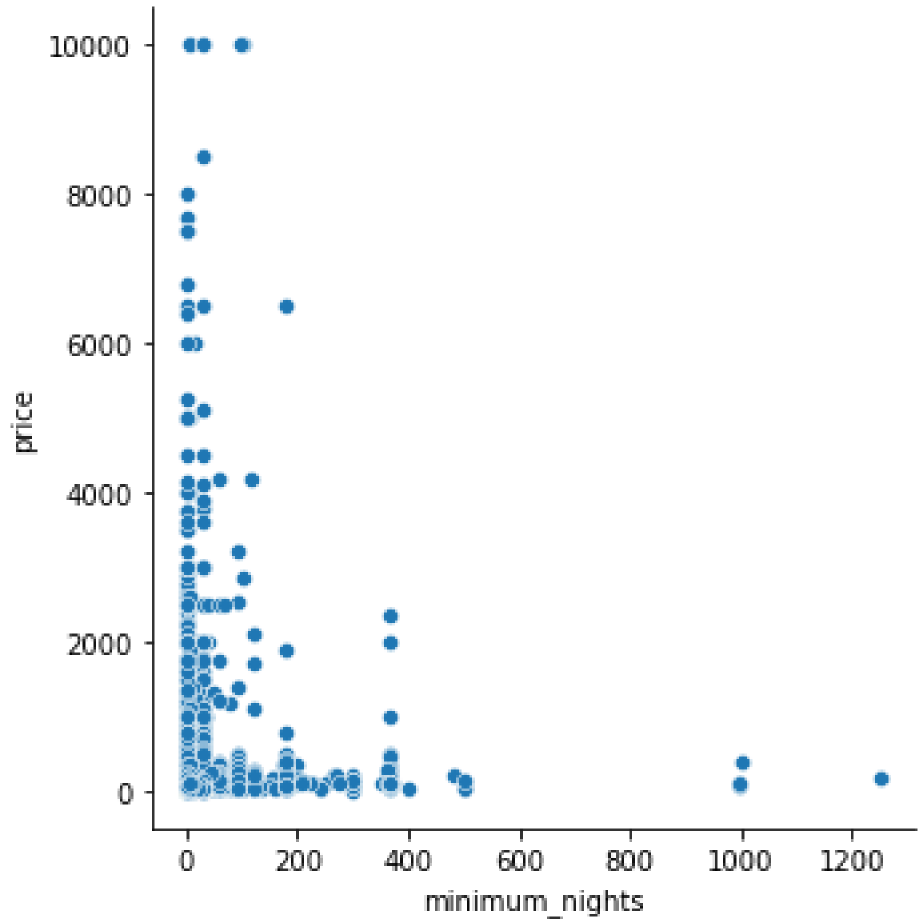
```
[14]: <AxesSubplot:ylabel='price'>
```



Minimum nights and price

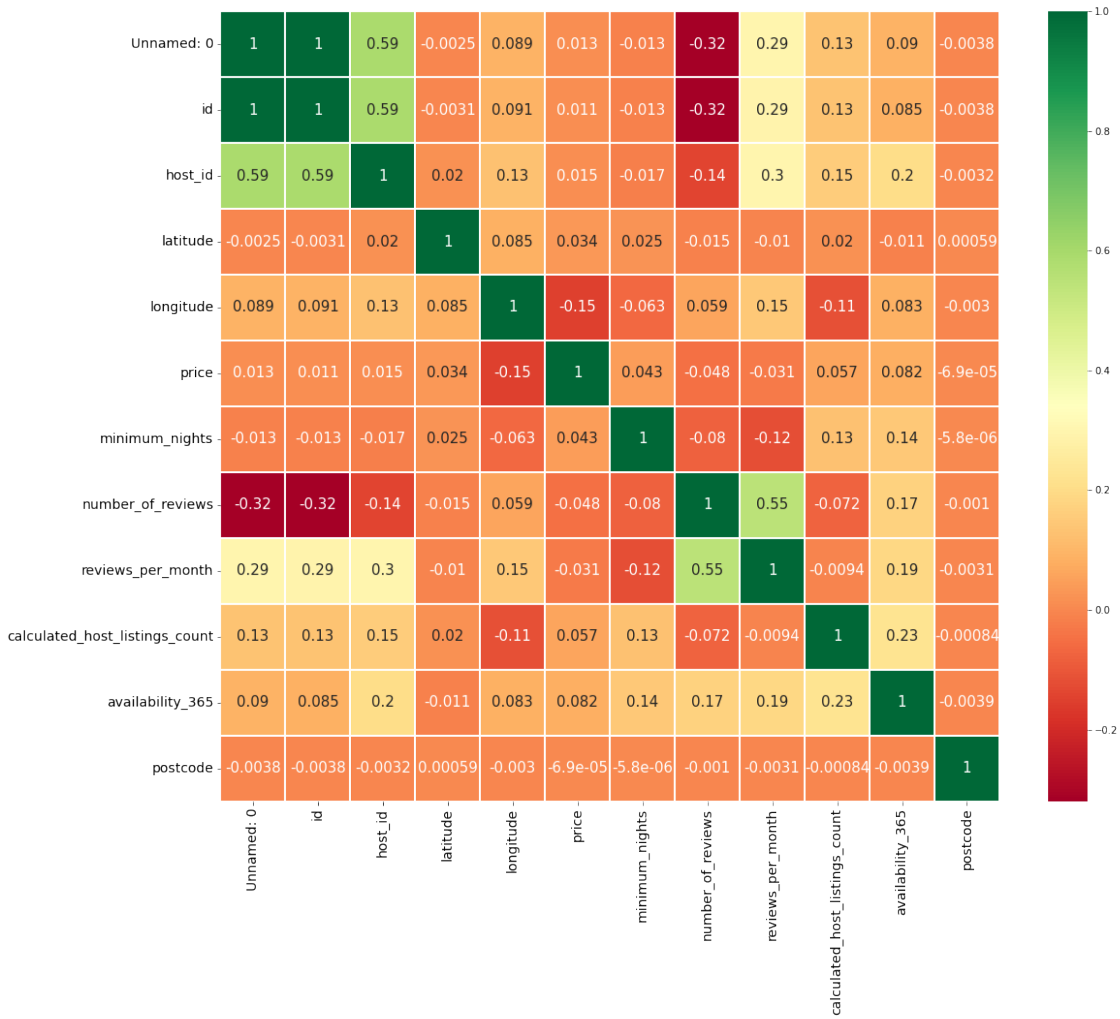
[15]:
```
sns.relplot(data=df,x='minimum_nights',y='price')
```

[15]: <seaborn.axisgrid.FacetGrid at 0x2193b3292b0>



[ ]:

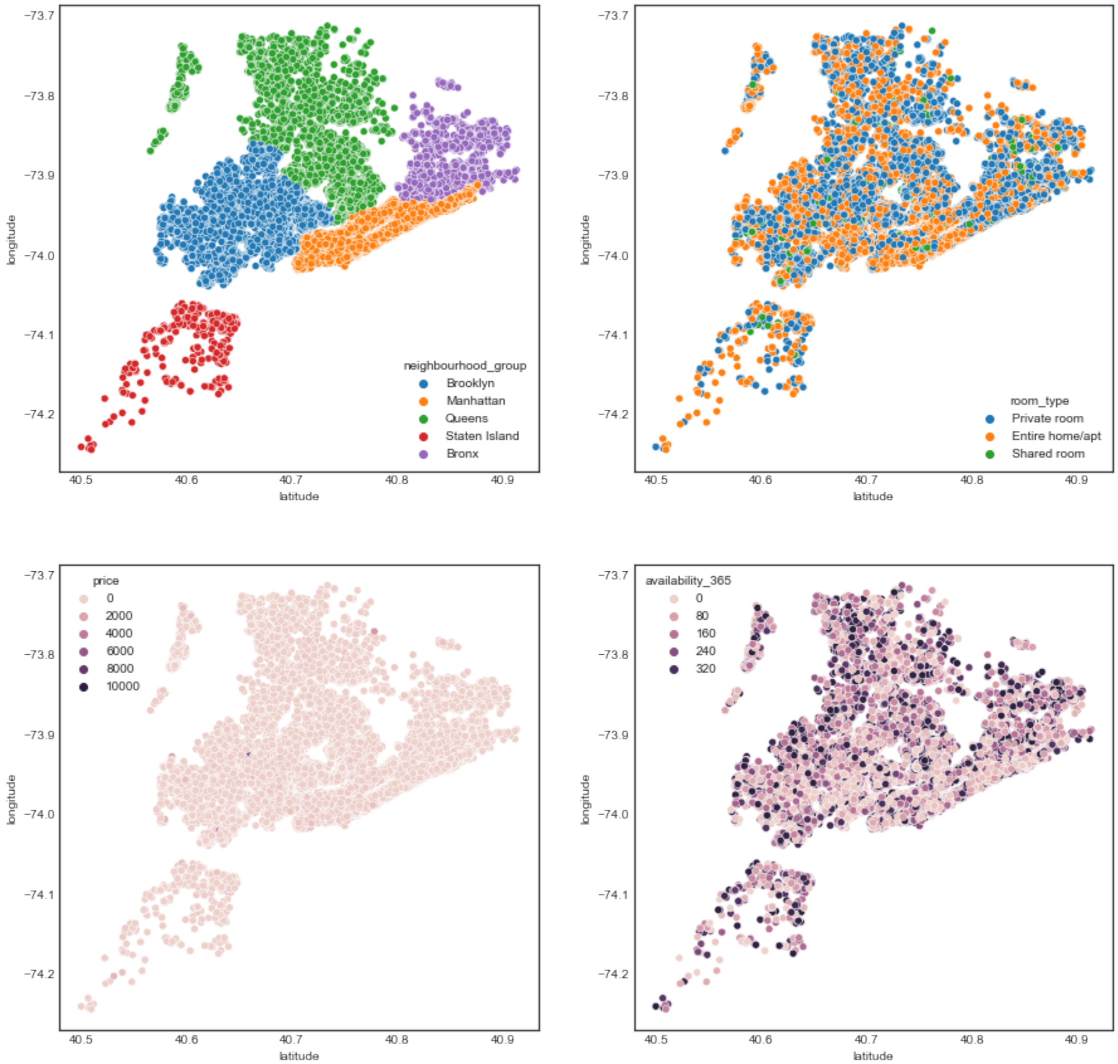
[16]:
```
corr = df.corr()
fig=sns.heatmap(corr,annot=True,cmap='RdYlGn',linewidths=0.2,annot_kws={'size':
 ↪15})
fig=plt.gcf()
fig.set_size_inches(18,15)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```



Area

[17]:
```
plt.figure(figsize = (15, 15))
plt.style.use('seaborn-white')
plt.subplot(221)
sns.scatterplot(x="latitude", y="longitude",hue="neighbourhood_group", data=df)
plt.subplot(222)
sns.scatterplot(x="latitude", y="longitude",hue="room_type", data=df)
plt.subplot(223)
sns.scatterplot(x="latitude", y="longitude",hue="price", data=df)
plt.subplot(224)
sns.scatterplot(x="latitude", y="longitude",hue="availability_365", data=df)
```

[17]: `<AxesSubplot:xlabel='latitude', ylabel='longitude'>`



Heat Maps

[18]:
```python
m_2 = folium.Map(location=[40.77,-73.99], tiles='cartodbpositron',␣
 ↪zoom_start=12)

# Adding a heatmap to the base map
HeatMap(data=df[['latitude', 'longitude']], radius=10).add_to(m_2)

# Displaying the map
m_2
```

[18]: <folium.folium.Map at 0x2193c4bac70>

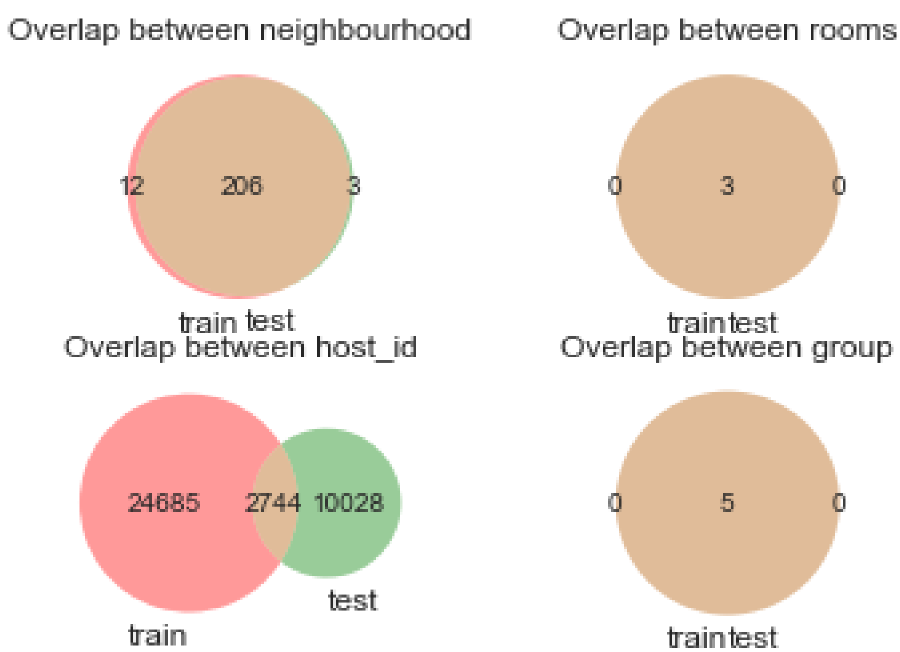
Bubble Map

[19]:
```
#Reference
#https://www.kaggle.com/alexisbcook/interactive-maps#Choropleth-maps
# Create a map
m_1 = folium.Map(location=[40.77,-73.99], tiles='cartodbpositron',␣
↪zoom_start=13)
m_1

# Add points to the map
mc = MarkerCluster()
for idx, row in df.iterrows():
    if not math.isnan(row['latitude']) and not math.isnan(row['longitude']):
        mc.add_child(Marker([row['latitude'], row['longitude']]))
m_1.add_child(mc)

# Display the map
m_1
```

[19]:
```
<folium.folium.Map at 0x2193f1d1130>
```

[20]:
```
## Some Test code,## So might have to split data by host_id , or maybe remove␣
↪that column completely

X_train,X_test,Y_train,Y_test= train_test_split(df,df['price'],test_size=0.
↪3,random_state=22)
fig, axs = plt.subplots(2, 2)
axs[0,0].set_title(f'Overlap between neighbourhood')
venn2([set(X_train['neighbourhood'].values), set(X_test['neighbourhood'].
↪values)],
      set_labels = ('train','test'),ax=axs[0,0])
axs[1,1].set_title(f'Overlap between group')
venn2([set(X_train['neighbourhood_group'].values),␣
↪set(X_test['neighbourhood_group'].values)],
      set_labels = ('train','test'),ax=axs[1,1])
axs[0,1].set_title(f'Overlap between rooms')
venn2([set(X_train['room_type'].values), set(X_test['room_type'].values)],
      set_labels = ('train','test'),ax=axs[0,1])
axs[1,0].set_title(f'Overlap between host_id')
venn2([set(X_train['host_id'].values), set(X_test['host_id'].values)],
      set_labels = ('train','test'),ax=axs[1,0])
```

[20]:
```
<matplotlib_venn._common.VennDiagram at 0x2195158efd0>
```



1.1 Word2Vec

[3]:
```
vectors
```

[3]:
```
array([[-1.13241   ,  0.13470681,  0.4152584 , …,  0.24678198,
        -0.149896  ,  0.07637601],
       [-1.5611467 , -0.05994999,  0.388486  , …,  0.20808266,
        -0.8337167 ,  0.55918   ],
       [-0.45280498,  0.09483501,  0.2230975 , …, -0.14181925,
        -1.0148975 ,  0.60518503],
       …,
       [-0.83779997, -0.11825749,  0.53628993, …, -0.33549747,
        -0.6932125 ,  0.0667375 ],
       [-0.9408907 ,  0.17652127,  0.49140212, …,  0.20132427,
         0.00465299,  0.078404  ],
       [-0.73716   , -0.15977   ,  0.32877797, …,  0.34656742,
         0.052904  ,  0.23506801]], dtype=float32)
```

[46]:
```
df = df[~df['name'].isnull()]
df.reset_index(inplace=True,drop=True)
df.head()
```

[46]:
```
   Unnamed: 0    id                                              name  \
0           0  2539                Clean & quiet apt home by the park
1           1  2595                             Skylit Midtown Castle
```

```
2           2  3647               THE VILLAGE OF HARLEM….NEW YORK !
3           3  3831                   Cozy Entire Floor of Brownstone
4           4  5022  Entire Apt: Spacious Studio/Loft by central park

   host_id    host_name neighbourhood_group neighbourhood  latitude  \
0     2787         John            Brooklyn    Kensington  40.64749
1     2845     Jennifer           Manhattan       Midtown  40.75362
2     4632    Elisabeth           Manhattan        Harlem  40.80902
3     4869  LisaRoxanne            Brooklyn  Clinton Hill  40.68514
4     7192        Laura           Manhattan   East Harlem  40.79851

   longitude        room_type  …  minimum_nights  number_of_reviews  \
0  -73.97237     Private room  …               1                  9
1  -73.98377  Entire home/apt  …               1                 45
2  -73.94190     Private room  …               3                  0
3  -73.95976  Entire home/apt  …               1                270
4  -73.94399  Entire home/apt  …              10                  9

   last_review reviews_per_month  calculated_host_listings_count  \
0   2018-10-19              0.21                               6
1   2019-05-21              0.38                               2
2          NaN               NaN                               1
3   2019-07-05              4.64                               1
4   2018-11-19              0.10                               1

   availability_365                geom  \
0               365  40.64749,-73.97237
1               355  40.75362,-73.98377
2               365   40.80902,-73.9419
3               194  40.68514,-73.95976
4                 0  40.79851,-73.94399

                                             address postcode  \
0  807, Friel Place, Windsor Terrace, Kings Count…  11218.0
1  Bryant Park, 6th Avenue, Theater District, Man…  10018.0
2  25, West 128th Street, East Harlem, Manhattan …  10027.0
3  188, Gates Avenue, Brooklyn, Kings County, New…  11238.0
4  1626, Park Avenue, East Harlem, Manhattan Comm…  10029.0

                                          add_text
0    807, Friel Place, Windsor Terrace, Kings County
1  Bryant Park, 6th Avenue, Theater District, Man…
2  25, West 128th Street, East Harlem, Manhattan …
3          188, Gates Avenue, Brooklyn, Kings County
4  1626, Park Avenue, East Harlem, Manhattan Comm…

[5 rows x 21 columns]
```

2 HELPER FUNCTIONS TO CREATE WORD EMBEDDINGS USING WORD2VEC

[47]:
```
def review_to_wordlist(review, remove_stopwords=False):
    """
    Convert a review to a list of words. Removal of stop words is optional.
    """
    # remove non-letters
    review_text = re.sub("[^a-zA-Z]"," ", review)

    # convert to lower case and split at whitespace
    words = review_text.lower().split()

    # remove stop words (false by default)
    if remove_stopwords:
        stops = set(stopwords.words("english"))
        words = [w for w in words if not w in stops]

    return words

def review_to_sentences(review, tokenizer, remove_stopwords=False):
    """
    Split review into list of sentences where each sentence is a list of words.
    Removal of stop words is optional.
    """
    # use the NLTK tokenizer to split the paragraph into sentences
    raw_sentences = tokenizer.tokenize(review.strip())

    # each sentence is furthermore split into words
    sentences = []
    for raw_sentence in raw_sentences:
        # If a sentence is empty, skip it
        if len(raw_sentence) > 0:
            sentences.append(review_to_wordlist(raw_sentence, remove_stopwords))

    return sentences
```

[48]:
```
def make_feature_vec(words, model, num_features):
    """
    Average the word vectors for a set of words
    """
    feature_vec = np.zeros((num_features,),dtype="float32")  # pre-initialize␣
↪(for speed)
    nwords = 0.
    index2word_set = set(model.index2word)  # words known to the model

    for word in words:
```

```
        if word in index2word_set:
            nwords = nwords + 1.
            feature_vec = np.add(feature_vec,model[word])

    feature_vec = np.divide(feature_vec, nwords)
    return feature_vec

def get_avg_feature_vecs(reviews, model, num_features):
    """
    Calculate average feature vectors for all reviews
    """
    counter = 0
    review_feature_vecs = np.zeros((len(reviews),num_features),␣
↪dtype='float32')  # pre-initialize (for speed)

    for review in tqdm(reviews):
        review_feature_vecs[counter] = make_feature_vec(review, model,␣
↪num_features)
        counter = counter + 1
    return review_feature_vecs
```

3 CREATING WORD2VEC EMBEDDINGS FOR FIELD "name" and "text"

[49]:

```
# train_sentences = []  # Initialize an empty list of sentences
# for review in df['name']:
#     train_sentences += review_to_sentences(review, tokenizer)

# joblib.dump(trainDataVecs, 'glove25_vecs.pkl')

# clean_train_reviews = []
# for review in tqdm(df_rmsle['add_text']):
#     clean_train_reviews.append(review_to_wordlist(review,␣
↪remove_stopwords=True))
# trainDataVecs = get_avg_feature_vecs(clean_train_reviews, glove_vectors,␣
↪num_features=25)
# joblib.dump(trainDataVecs, 'glove25_vecs_text.pkl')
```

Joining the dataframe

[50]:

```
df_host = pd.DataFrame(df['host_id'].value_counts().head(10))
df_host.reset_index(inplace = True)
df_host.rename(columns = {'index': 'host_id', 'host_id': 'count'}, inplace =␣
↪True)
df_host
```

```
[50]:     host_id  count
0  219517861    327
1  107434423    232
2   30283594    121
3  137358866    103
4   16098958     96
5   12243051     96
6   61391963     91
7   22541573     87
8  200380610     65
9    1475015     52
```

```
[51]: df_vecs = pd.DataFrame(vectors,columns = ['name_'+str(x) for x in␣
      ↪range(len(vectors[0]))])
      df_vecs
      df=pd.concat([df,df_vecs],axis=1)
      df
```

```
[51]:        Unnamed: 0        id  \
0                 0      2539
1                 1      2595
2                 2      3647
3                 3      3831
4                 4      5022
…               …       …
48874         48890  36484665
48875         48891  36485057
48876         48892  36485431
48877         48893  36485609
48878         48894  36487245

                                                   name   host_id  \
0                    Clean & quiet apt home by the park      2787
1                                 Skylit Midtown Castle      2845
2                   THE VILLAGE OF HARLEM…NEW YORK !      4632
3                       Cozy Entire Floor of Brownstone      4869
4      Entire Apt: Spacious Studio/Loft by central park      7192
…                                                   …       …
48874    Charming one bedroom - newly renovated rowhouse   8232441
48875      Affordable room in Bushwick/East Williamsburg   6570630
48876            Sunny Studio at Historical Neighborhood  23492952
48877               43rd St. Time Square-cozy single bed  30985759
48878  Trendy duplex in the very heart of Hell's Kitchen  68119814

          host_name neighbourhood_group       neighbourhood  latitude  \
0              John            Brooklyn          Kensington  40.64749
1          Jennifer           Manhattan             Midtown  40.75362
```

```
2         Elisabeth          Manhattan             Harlem  40.80902
3       LisaRoxanne           Brooklyn       Clinton Hill  40.68514
4             Laura          Manhattan        East Harlem  40.79851
…                 …                  …                  …         …
48874       Sabrina           Brooklyn  Bedford-Stuyvesant  40.67853
48875       Marisol           Brooklyn           Bushwick  40.70184
48876  Ilgar & Aysel         Manhattan             Harlem  40.81475
48877           Taz          Manhattan     Hell's Kitchen  40.75751
48878    Christophe          Manhattan     Hell's Kitchen  40.76404

       longitude        room_type  …   name_15   name_16   name_17  \
0      -73.97237     Private room  … -0.119630  0.358446 -0.261629
1      -73.98377  Entire home/apt  … -0.136938  0.252237  0.334670
2      -73.94190     Private room  … -0.116102  0.386901 -0.008850
3      -73.95976  Entire home/apt  …  0.165937  0.763173  0.220880
4      -73.94399  Entire home/apt  … -0.111114  0.420411  0.258372
…            …                …  …       …         …         …
48874  -73.94995     Private room  …  0.028466  0.922270  0.329282
48875  -73.93317     Private room  … -0.595992  0.515644 -0.038676
48876  -73.94867  Entire home/apt  …  0.002907  0.330335  0.248530
48877  -73.99112      Shared room  …  0.061727  0.279649 -0.251020
48878  -73.98933     Private room  …  0.198152  0.444624 -0.093494

        name_18   name_19   name_20   name_21   name_22   name_23   name_24
0     -0.280356 -0.491068  0.027326 -0.231274  0.246782 -0.149896  0.076376
1      0.581537  0.492677 -0.439593 -0.600883  0.208083 -0.833717  0.559180
2     -0.343070  0.420657 -0.462853 -0.727195 -0.141819 -1.014897  0.605185
3     -0.013177  0.223837 -0.803038 -0.059888  0.324075 -0.223083 -0.033134
4     -0.163367  0.125644 -0.549020 -0.607433  0.057689 -0.474711  0.152281
…           …         …         …         …         …         …         …
48874  0.208982 -0.254954 -0.295970 -0.460188 -0.461461 -0.378484 -0.501034
48875  0.313254  0.261840 -0.098324 -0.984138 -0.044194 -0.992851  0.026878
48876 -0.246872  0.447348 -0.435058 -0.518501 -0.335497 -0.693213  0.066738
48877 -0.583804 -0.011813 -0.748277 -0.191887  0.201324  0.004653  0.078404
48878  0.122938  0.370372 -0.341424 -0.067676  0.346567  0.052904  0.235068

[48879 rows x 46 columns]
```

Detecting outliers Outliers are unusual values in your dataset, and they can distort statistical analyses and violate their assumptions. Unfortunately, all analysts will confront outliers and be forced to make decisions about what to do with them. Given the problems they can cause, you might think that it's best to remove them from your data. But, that's not always the case. Removing outliers is legitimate only for specific reasons.Outliers can be very informative about the subject-area and data collection process. It's essential to understand how outliers occur and whether they might happen again as a normal part of the process or study area. Unfortunately, resisting the temptation to remove outliers inappropriately can be difficult. Outliers increase the variability in your data, which decreases statistical power. Consequently, excluding outliers can cause your results to become statistically significant. In our case, let's first visualize our data and decide on what to do with the outliers https://www.kaggle.com/benroshan/belong-anywhere-ny-airbnb-price-prediction

[52]:
```
Q1 = df['number_of_reviews'].quantile(0.25)
Q3 = df['number_of_reviews'].quantile(0.75)
IQR = Q3 - Q1    #IQR is interquartile range.

filter = (df['number_of_reviews'] >= Q1 - 1.5 * IQR) & (df['number_of_reviews']␣
↪<= Q3 + 1.5 *IQR)
df=df.loc[filter]


Q1 = df['reviews_per_month'].quantile(0.25)
Q3 = df['reviews_per_month'].quantile(0.75)
IQR = Q3 - Q1    #IQR is interquartile range.

filter = (df['reviews_per_month'] >= Q1 - 1.5 * IQR) & (df['reviews_per_month']␣
↪<= Q3 + 1.5 *IQR)
df_new=df.loc[filter]
df_new=df_new.reset_index(drop=True)
```

[53]:
```
df_rmsle = df_new.copy()
```

[54]:
```
df_rmsle.reset_index(inplace=True,drop=True)
```

[55]:
```
df_rmsle.shape
```

[55]:
```
(30127, 46)
```

Some feature preprocessing and encoding

[56]:
```
df_rmsle=pd.get_dummies(df_rmsle,columns=['neighbourhood_group','room_type'])
df_rmsle.drop(['id', 'name', 'host_name','last_review','neighbourhood'], axis =␣
↪1,inplace=True)

df_rmsle['reviews_per_month'] = df_rmsle['reviews_per_month'].
↪fillna(df_rmsle['reviews_per_month'].mean())
```

Joining the word vectors for the field "text"

[57]:
```
df_vecs = pd.DataFrame(vec_text,columns = ['text_'+ str(x) for x in␣
↪range(len(vec_text[0]))])
df_vecs.shape
df_rmsle=pd.concat([df_rmsle,df_vecs],axis=1)
```

[58]:
```
df_rmsle.columns
```

[58]: Index(['Unnamed: 0', 'host_id', 'latitude', 'longitude', 'price',
       'minimum_nights', 'number_of_reviews', 'reviews_per_month',
       'calculated_host_listings_count', 'availability_365', 'geom', 'address',
       'postcode', 'add_text', 'name_0', 'name_1', 'name_2', 'name_3',
       'name_4', 'name_5', 'name_6', 'name_7', 'name_8', 'name_9', 'name_10',
       'name_11', 'name_12', 'name_13', 'name_14', 'name_15', 'name_16',
       'name_17', 'name_18', 'name_19', 'name_20', 'name_21', 'name_22',
       'name_23', 'name_24', 'neighbourhood_group_Bronx',
       'neighbourhood_group_Brooklyn', 'neighbourhood_group_Manhattan',
       'neighbourhood_group_Queens', 'neighbourhood_group_Staten Island',
       'room_type_Entire home/apt', 'room_type_Private room',
       'room_type_Shared room', 'text_0', 'text_1', 'text_2', 'text_3',
       'text_4', 'text_5', 'text_6', 'text_7', 'text_8', 'text_9', 'text_10',
       'text_11', 'text_12', 'text_13', 'text_14', 'text_15', 'text_16',
       'text_17', 'text_18', 'text_19', 'text_20', 'text_21', 'text_22',
       'text_23', 'text_24'],
      dtype='object')

[59]:
```
df_rmsle.drop(['Unnamed: 0','geom', 'address','add_text'],axis=1,inplace=True)
```

[60]:
```
df_rmsle.dropna(inplace=True)
df_rmsle.reset_index(inplace=True,drop=True)
y_temp = df_rmsle['price']
df_rmsle.drop('price',inplace=True,axis=1)
```

[61]:
```
df_rmsle.shape
```

[61]: (29852, 67)

Testing with Groupkfold and rmse we can see the outliers affect the score by a lot.

[62]:
```
splits=5
gkf = GroupKFold(n_splits = splits)
groups = df_rmsle['host_id']
for fold_ind, (trn_idx, val_idx) in enumerate(gkf.split(df_rmsle, y_temp,␣
 ↪groups)):
    X_train,y_train = df_rmsle.iloc[trn_idx],y_temp.iloc[trn_idx]
    X_valid,y_valid = df_rmsle.iloc[val_idx],y_temp.iloc[val_idx]
    break
```

[63]:
```
splits=5
gkf = GroupKFold(n_splits = splits)
groups = X_train['host_id']
# test_preds = np.zeros((len(df_test), 1))
holdout_preds = np.zeros((len(X_valid), 1))
final_preds = []
```

```python
for fold_ind, (trn_idx, val_idx) in enumerate(gkf.split(X_train, y_train,␣
↪groups)):
    print("Fold {}".format(fold_ind))
    X_trn,y_trn = X_train.drop(['host_id','postcode'],axis=1).
↪iloc[trn_idx],y_train.iloc[trn_idx]
    X_val,y_val = X_train.drop(['host_id','postcode'],axis=1).
↪iloc[val_idx],y_train.iloc[val_idx]
    clf = lgb.LGBMRegressor(n_estimators=200,max_depth=5,random_state=22)
    clf.fit(X_trn, y_trn)
    final_preds.append(sqrt(mean_squared_error(y_pred=clf.
↪predict(X_val),y_true=y_val)))
    holdout_preds += clf.predict(X_valid.drop(['host_id','postcode'],axis=1)).
↪reshape(-1,1)
print("The CV RMSE :", sum(final_preds)/splits)
print("The test set score:",sqrt(mean_squared_error(y_pred=holdout_preds/
↪5,y_true=y_valid)))
```

```
Fold 0
Fold 1
Fold 2
Fold 3
Fold 4
The CV RMSE : 205.2388568894738
The test set score: 173.62379477694043
```

Groupkfold and MAE

[64]:
```python
splits=5
gkf = GroupKFold(n_splits = splits)
groups = X_train['host_id']
# test_preds = np.zeros((len(df_test), 1))
holdout_preds = np.zeros((len(X_valid), 1))
final_preds = []
for fold_ind, (trn_idx, val_idx) in enumerate(gkf.split(X_train, y_train,␣
↪groups)):
    print("Fold {}".format(fold_ind))
    X_trn,y_trn = X_train.drop(['host_id','postcode'],axis=1).
↪iloc[trn_idx],y_train.iloc[trn_idx]
    X_val,y_val = X_train.drop(['host_id','postcode'],axis=1).
↪iloc[val_idx],y_train.iloc[val_idx]
    clf = lgb.LGBMRegressor(n_estimators=100,max_depth=10,random_state=22)
    clf.fit(X_trn, y_trn)
    final_preds.append(mean_absolute_error(y_pred=clf.
↪predict(X_val),y_true=y_val))
    holdout_preds += clf.predict(X_valid.drop(['host_id','postcode'],axis=1)).
↪reshape(-1,1)
print("Groupkfold CV MAE with Lightgbm:",sum(final_preds)/splits)
```

```
print("The Test set score:",mean_absolute_error(y_pred=holdout_preds/
↪5,y_true=y_valid))
```

```
Fold 0
Fold 1
Fold 2
Fold 3
Fold 4
Groupkfold CV MAE with Lightgbm: 63.598494934812095
The Test set score: 61.95993887457325
```

4 THE FEATURE IMPORTANCE

[65]:
```
imp = importances(clf,X_valid.drop(["host_id",'postcode'],axis=1),y_valid)
```

[66]:
```
imp
```

[66]:
```
                                Importance
Feature
room_type_Entire home/apt         0.181969
longitude                         0.034861
minimum_nights                    0.030815
latitude                          0.030236
text_2                            0.017459
…                                    …
name_22                          -0.004410
text_23                          -0.004963
name_10                          -0.005062
name_5                           -0.008278
calculated_host_listings_count   -0.027459

[65 rows x 1 columns]
```

[67]:
```
splits=5
gkf = GroupKFold(n_splits = splits)
groups = X_train['host_id']
# test_preds = np.zeros((len(df_test), 1))
holdout_preds = np.zeros((len(X_valid), 1))
final_preds = []
for fold_ind, (trn_idx, val_idx) in enumerate(gkf.split(X_train, y_train,␣
↪groups)):
    print("Fold {}".format(fold_ind))
    X_trn,y_trn = X_train.drop(['host_id','postcode'],axis=1).
↪iloc[trn_idx],y_train.iloc[trn_idx]
    X_val,y_val = X_train.drop(['host_id','postcode'],axis=1).
↪iloc[val_idx],y_train.iloc[val_idx]
```

```
    clf = RandomForestRegressor(n_estimators= 300,max_depth=10, max_features=␣
↪'sqrt',bootstrap= True)
    clf.fit(X_trn, y_trn)
    final_preds.append(mean_absolute_error(y_pred=clf.
↪predict(X_val),y_true=y_val))
    holdout_preds += clf.predict(X_valid.drop(['host_id','postcode'],axis=1)).
↪reshape(-1,1)
print("Groupkfold CV MAE with RandomForest: ",sum(final_preds)/splits)
print("The test set score: ",mean_absolute_error(y_pred=holdout_preds/
↪5,y_true=y_valid))
```

```
Fold 0
Fold 1
Fold 2
Fold 3
Fold 4
Groupkfold CV MAE with RandomForest:  63.51048935877702
The test set score:  60.23143474267427
```

[68]:
```
imp = importances(clf,X_valid.drop(["host_id",'postcode'],axis=1),y_valid)
```

[69]:
```
imp
```

[69]:
```
                              Importance
Feature
room_type_Entire home/apt       0.063132
longitude                       0.030865
availability_365                0.023967
neighbourhood_group_Manhattan   0.018602
room_type_Private room          0.016021
…                                    …
text_2                         -0.001597
name_5                         -0.001705
name_0                         -0.001766
text_17                        -0.006884
text_3                         -0.009343

[65 rows x 1 columns]
```

Using straifiedkfold as cv method in regression problem, so basically we are dividing the data into bins to create folds and then test the model on it, now in this case it makes sense also we get improved MAE scores. We select the bins down below with Sturge's rule which helps us select optimal bins.

[70]:
```
optimal_bins = np.log2(len(df_rmsle)) + 1
```

[71]:
```
n_splits=5
skf = StratifiedKFold(n_splits = n_splits,random_state=22,shuffle=True)
for fold_ind, (trn_idx, val_idx) in enumerate(skf.split(df_rmsle, pd.
↪cut(y_temp,int(np.floor(optimal_bins)),labels=False))):
    X_train,y_train = df_rmsle.iloc[trn_idx],y_temp.iloc[trn_idx]
    X_valid,y_valid = df_rmsle.iloc[val_idx],y_temp.iloc[val_idx]
    break
```

```
C:\Users\samsu\anaconda3\lib\site-
packages\sklearn\model_selection\_split.py:676: UserWarning: The least populated
class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
```

[72]:
```
optimal_bins = np.log2(len(X_train)) + 1
```

[73]:
```
splits=5
skf = StratifiedKFold(n_splits = splits,random_state=22,shuffle=True)
# test_preds = np.zeros((len(df_test), 1))
holdout_preds = np.zeros((len(X_valid), 1))
final_preds = []
for fold_ind, (trn_idx, val_idx) in enumerate(skf.split(X_train, pd.
↪cut(y_train,int(np.floor(optimal_bins)),labels=False))):
    print("Fold {}".format(fold_ind))
    X_trn,y_trn = X_train.drop(['host_id','postcode'],axis=1).
↪iloc[trn_idx],y_train.iloc[trn_idx]
    X_val,y_val = X_train.drop(['host_id','postcode'],axis=1).
↪iloc[val_idx],y_train.iloc[val_idx]
    clf = RandomForestRegressor(n_estimators= 300,max_depth=10, max_features=␣
↪'sqrt',bootstrap= True)
    clf.fit(X_trn, y_trn)
    final_preds.append(mean_absolute_error(y_pred=clf.
↪predict(X_val),y_true=y_val))
    holdout_preds += clf.predict(X_valid.drop(['host_id','postcode'],axis=1)).
↪reshape(-1,1)
print("StratifiedKfold score Random forest: ", sum(final_preds)/splits)
print("Test set score :" , mean_absolute_error(y_pred=holdout_preds/
↪5,y_true=y_valid))
```

```
C:\Users\samsu\anaconda3\lib\site-
packages\sklearn\model_selection\_split.py:676: UserWarning: The least populated
class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
```

```
Fold 0
Fold 1
Fold 2
Fold 3
Fold 4
```

```
StratifiedKfold score Random forest:  62.339677508276466
Test set score : 61.438303182887076
```

[74]:
```python
imp = importances(clf,X_valid.drop(["host_id",'postcode'],axis=1),y_valid)
```

[82]:
```python
imp[:10]
```

[82]:

Feature	Importance
room_type_Entire home/apt	0.107978
availability_365	0.047057
longitude	0.035434
latitude	0.019595
minimum_nights	0.017915
calculated_host_listings_count	0.005684
name_2	0.003351
name_21	0.003297
name_6	0.003260
text_2	0.002763

[76]:
```python
splits=5
skf = StratifiedKFold(n_splits = splits,random_state=22,shuffle=True)
# test_preds = np.zeros((len(df_test), 1))
holdout_preds = np.zeros((len(X_valid), 1))
final_preds = []
for fold_ind, (trn_idx, val_idx) in enumerate(skf.split(X_train, pd.
 ↪cut(y_train,15,labels=False))):
    print("Fold {}".format(fold_ind))
    X_trn,y_trn = X_train.drop(['host_id','postcode'],axis=1).
 ↪iloc[trn_idx],y_train.iloc[trn_idx]
    X_val,y_val = X_train.drop(['host_id','postcode'],axis=1).
 ↪iloc[val_idx],y_train.iloc[val_idx]
    clf = lgb.LGBMRegressor(n_estimators=100,max_depth=10,random_state=22)
    clf.fit(X_trn, y_trn)
    final_preds.append(mean_absolute_error(y_pred=clf.
 ↪predict(X_val),y_true=y_val))
    holdout_preds += clf.predict(X_valid.drop(['host_id','postcode'],axis=1)).
 ↪reshape(-1,1)

print("The Skf lgbm cv score :", sum(final_preds)/splits)
print("The test set score :", mean_absolute_error(y_pred=holdout_preds/
 ↪5,y_true=y_valid))
```

```
C:\Users\samsu\anaconda3\lib\site-
packages\sklearn\model_selection\_split.py:676: UserWarning: The least populated
class in y has only 1 members, which is less than n_splits=5.
  warnings.warn(
```

```
Fold 0
Fold 1
Fold 2
Fold 3
Fold 4
The Skf lgbm cv score : 62.39228896749056
The test set score : 58.76379636663698
```

[77]:
```python
imp = importances(clf,X_valid.drop(["host_id",'postcode'],axis=1),y_valid)
```

[81]:
```python
imp[:10]
```

[81]:

Feature	Importance
room_type_Entire home/apt	0.107978
availability_365	0.047057
longitude	0.035434
latitude	0.019595
minimum_nights	0.017915
calculated_host_listings_count	0.005684
name_2	0.003351
name_21	0.003297
name_6	0.003260
text_2	0.002763

The cv strategy we use is stratifiedkfold since it gives better performance.

[ ]:
```python

```