Airbnb Listings Pricing in London

Project Aims

An efficient marketing strategy, especially the pricing strategy, should be set up on basis of the proficient familiarity to 3 main elements: **Product, Customers & the Market Environment**. Under current rapid developing and evolutionary market environment, regional strategies tailoring for local characteristics are called upon with paramount importance. In addition, good data visualisations will be of complementary benefit to the story telling of data analysis when presenting your marketing strategies to stakeholders.

In this session, we will introduce multiple data visualisation techniques utilising London Airbnb Listings, the **Product**; apply the exploratory spatial data analytics skills on to the **environment**; build up prelimenary Airbnb listings pricing regression model incorporating geographical weights (**Product, Customers' reviews and Market Environment**), and most of important, aiming for better marketing strategies for optimal location selection of Airbnb Listing and the corresponding pricing strategy. Last but not least, taking City of Westminster (the borough) as a case study, to "mine out" what the customers really care about when choosing their Airbnb accommodations.

Prepare the data

Airbnb Data

This dataset contains information on Airbnb properties for London. It is originally provided by Inside Airbnb. Same as the source, the dataset is released under a CC0 1.0 Universal License. Data utilised in this practical is dated by 10 September 2022.

Source: Inside Airbnb's extract of Airbnb locations in London.

London LSOA Boundaries

Boundaries for LSOAs in London. The original source is provided by Office for National Statistics (ONS). The dataset is released under UK Open Government Licence (OGL v2).

Source: open data from London datastore at HERE

In [1]:
```
# load necessary libraries
import os
import urllib
import zipfile

import seaborn as sns
sns.set(style="whitegrid")
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pysal as ps
```

```
import geopandas as gpd
from sklearn import cluster
from sklearn.preprocessing import StandardScaler
from shapely.geometry import Point

import warnings
warnings.simplefilter('ignore')
%matplotlib inline

# one type of interactive map visualisation
import folium
from folium.plugins import FastMarkerCluster
from branca.colormap import LinearColormap

#to make the plotly graphs
from plotly.offline import iplot, init_notebook_mode
from plotly import graph_objs as go
import plotly.express as px
from plotly.graph_objs import *

# Bokeh
from bokeh.io import output_notebook
from bokeh.layouts import gridplot,row,column
from bokeh.plotting import figure,show
output_notebook()
rainbow = sns.color_palette(palette='rainbow',n_colors=8)
```

BokehJS 2.4.3 successfully loaded.

In [2]:
```
# Create the data dir if it doesn't exist
if os.path.isdir('data') is not True:
    print("Creating 'data' directory...")
    os.mkdir('data')

# Configure the download
url  = 'https://github.com/cusp-london/Spatial-Data-Analysis/blob/master/LSOA_london.
path = os.path.join("data","LSOA_london.zip")

# Download
r    = urllib.request.urlretrieve(url, path)

# Unzip it into the data folder
z    = zipfile.ZipFile(path)
m    = z.extractall("data")
```

```
Creating 'data' directory...
```

In [3]:
```
# Configure the next download -- notice
# that we will need to visit InsideAirBnB
# in order to check that the data hasn't been
# updated from 2022/09/10.
url  = 'http://data.insideairbnb.com/united-kingdom/england/london/2022-09-10/data/li
path = os.path.join("data","listings.csv.gz")
# Download but don't unzip it
r    = urllib.request.urlretrieve(url, path)
```

In [4]:
```
# Configure the next download -- Airbnb reviews
url  = 'http://data.insideairbnb.com/united-kingdom/england/london/2022-09-10/data/re
path = os.path.join("data","reviews.csv.gz")
# Download but don't bother to unzip it by now
r    = urllib.request.urlretrieve(url, path)
```

Have a glance of the data

In [5]:
```
ablist=pd.read_csv('data/listings.csv.gz',compression='gzip')
ablist.head()
```

Out[5]:

	id	listing_url	scrape_id	last_scraped	source	name	de
0	13913	https://www.airbnb.com/rooms/13913	20220910194334	2022-09-11	city scrape	Holiday London DB Room Let-on going	wi wi
1	15400	https://www.airbnb.com/rooms/15400	20220910194334	2022-09-11	city scrape	Bright Chelsea Apartment. Chelsea!	an G
2	284532	https://www.airbnb.com/rooms/284532	20220910194334	2022-09-11	city scrape	COSY STUDIO-FLAT WITH A GREAT VIEW	R REI loc
3	106332	https://www.airbnb.com/rooms/106332	20220910194334	2022-09-11	city scrape	Lovely large room, Bethnal Green	cha bel
4	17402	https://www.airbnb.com/rooms/17402	20220910194334	2022-09-11	city scrape	Superb 3-Bed/2 Bath & Wifi: Trendy W1	You v st sup

5 rows × 75 columns

In [6]:
```
geometry = [Point(xy) for xy in zip(ablist.longitude, ablist.latitude)] # define the
crs      = {'init': 'epsg:4326'} # The projection for latitude and longitude
ablist       = gpd.GeoDataFrame(ablist, crs=crs, geometry=geometry) # add these geolo
ablist       = ablist.to_crs({'init': 'epsg:27700'}) # Reproject the data into Britis
```

In [7]:
```
ablist.columns
```

Out[7]:
```
Index(['id', 'listing_url', 'scrape_id', 'last_scraped', 'source', 'name',
       'description', 'neighborhood_overview', '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', 'neighbourhood',
       'neighbourhood_cleansed', 'neighbourhood_group_cleansed', 'latitude',
       'longitude', 'property_type', 'room_type', 'accommodates', 'bathrooms',
       'bathrooms_text', 'bedrooms', 'beds', 'amenities', 'price',
       'minimum_nights', 'maximum_nights', 'minimum_minimum_nights',
       'maximum_minimum_nights', 'minimum_maximum_nights',
       'maximum_maximum_nights', 'minimum_nights_avg_ntm',
       'maximum_nights_avg_ntm', 'calendar_updated', 'has_availability',
       'availability_30', 'availability_60', 'availability_90',
       'availability_365', 'calendar_last_scraped', 'number_of_reviews',
       'number_of_reviews_ltm', 'number_of_reviews_l30d', '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', 'license', 'instant_bookable',
       'calculated_host_listings_count',
       'calculated_host_listings_count_entire_homes',
       'calculated_host_listings_count_private_rooms',
       'calculated_host_listings_count_shared_rooms', 'reviews_per_month',
       'geometry'],
      dtype='object')
```

Explore the data

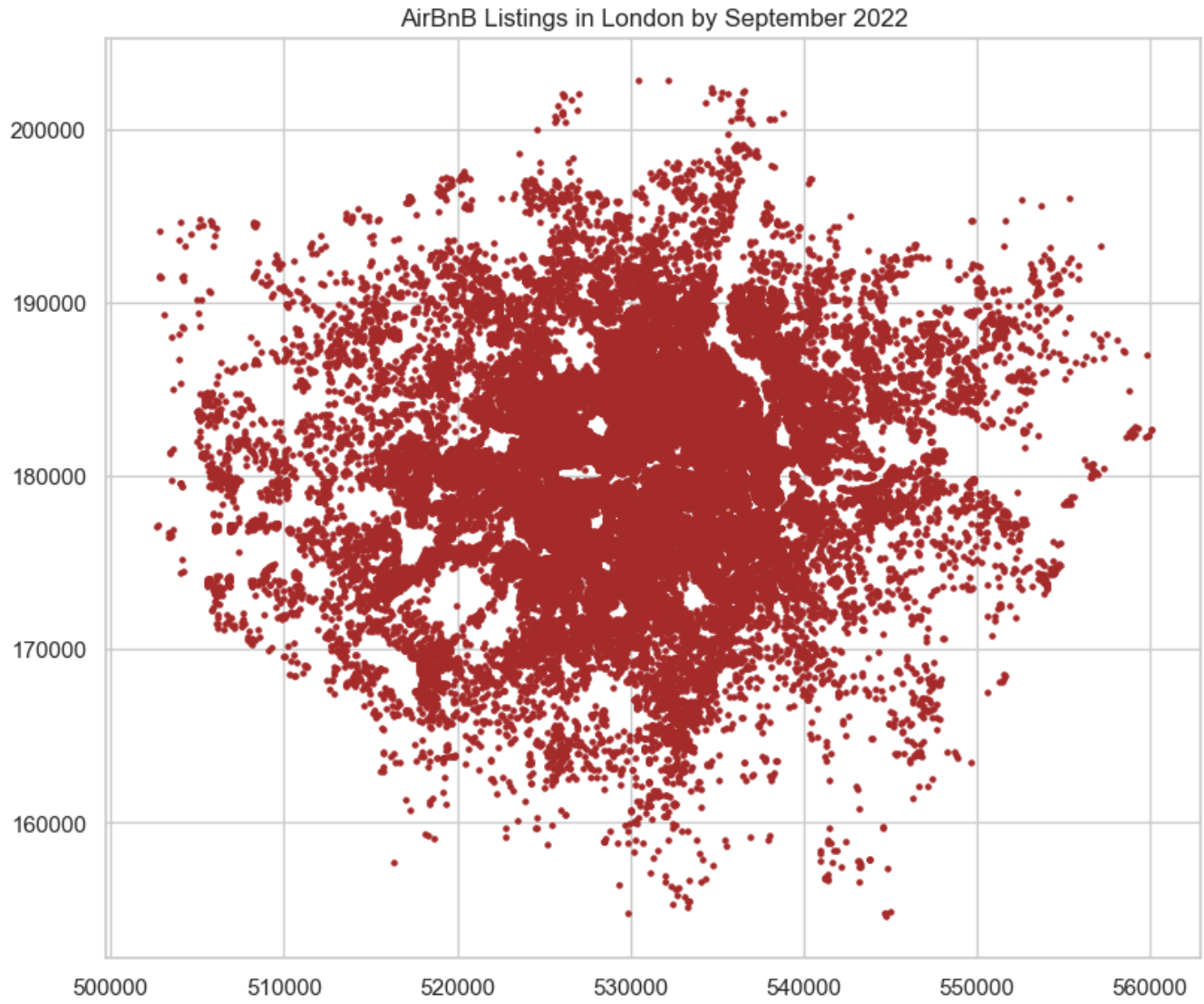
Since we are familiar with using maps to locate places, it is also possible to map out the locations of each Airbnb listings for our dataset **ablist**. There are several packages could facilitate our objectives, for example:

- *matplotlib*: commonly utilised, static figure
- *folium*: interactive map
- plotly&mapbox: interactive map

The most familiar -- Matplotlib (plt)

In [8]:
```
# set up the plot configuration
fig, ax = plt.subplots(1, figsize=(12, 8))

ablist.plot(ax=ax, marker='o', color='brown', markersize=5)

plt.title("AirBnB Listings in London by September 2022")
```

Out[8]:
```
Text(0.5, 1.0, 'AirBnB Listings in London by September 2022')
```



Folium

An interactive map visualisation package built on top of Open Street Map services.

In [9]:
```
lats = ablist['latitude'].tolist()
lons = ablist['longitude'].tolist()
locations = list(zip(lats, lons))

map1 = folium.Map(location=[51.5080, -0.1262], zoom_start=12) # take central location
FastMarkerCluster(data=locations).add_to(map1)
map1.save('airbnb_listings.html')
```

Plotly + Mapbox (Optional)

Plotly is a "newly" joined package to Python world, but was found very handy and powerful. Some of its visualisation functions should be realised through Mapbox, with the latter requiring registration. Details could be found at MAPBOX. Upon your successful registration with your email address, please log into your account at Mapbox, and find the 'Default public token' under your account.

Besides, there are also configurations refering to future map color schemes upon your preferences, please help to reference HERE.

In [10]:
```
mapbox_access_token = 'pk.eyJ1IjoiYW9saWZvZGFpc3kiLCJhIjoiY2tteXo5ejk1MDh4aDJ3cGtma2F
```

In [11]:
```
fig = px.scatter_mapbox(ablist, lat="latitude", lon="longitude", hover_name="name",
                        hover_data=["neighbourhood", "room_type","price","review_scor
                        color_discrete_sequence=["goldenrod"], zoom=10, height=300)
fig.update_layout(mapbox_accesstoken=mapbox_access_token)
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```

Exploratory Data Analysis and Visualisation

A targeted marketing strategy is customised by the product itself; similarly, a future Airbnb listing pricing strategy is built upon the familiarity of the Airbnb listings. So the starting point is to explore the features of **room_type** and **property_type**.

Room and Property

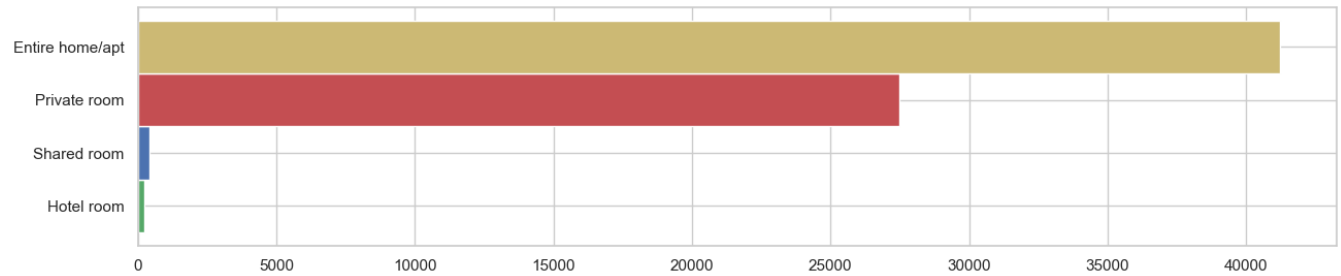
In order to get the general picture of room types and property types, we will call the function **unique** to recognise different categories firstly; then apply function **value_counts** with the specific feature for counting purpose, followed by its bar chart for vivid visualisation.

In [12]:
```
# recognise the categories of room type
ablist.room_type.unique()
```

Out[12]:
```
array(['Private room', 'Entire home/apt', 'Hotel room', 'Shared room'],
      dtype=object)
```

Since the output returns with 4 types of room as 'Private room', 'Entire home/apt', 'Hotel room' and 'Shared room', we could then differentiate them by color in the bar chart.

In [13]:
```
rt = ablist['room_type']. value_counts().sort_values(ascending=True)
rt.plot.barh(figsize=(15, 3), width=1, color = ["g","b","r", 'y'])
plt.show()
```



Findings: Majority of current Airbnb listings are "Entire home or apartment" and "Private room", with relatively smaller proportion of "Shared room" and "Hotel room". It may relate to the property itself, the geolocations, the customers' profiling (e.g., group tourists), the market niche of Airbnbs, etc.

Besides of room types, we will further investigate the types of the properties, following the similar procedure to check out categories of property type firstly.

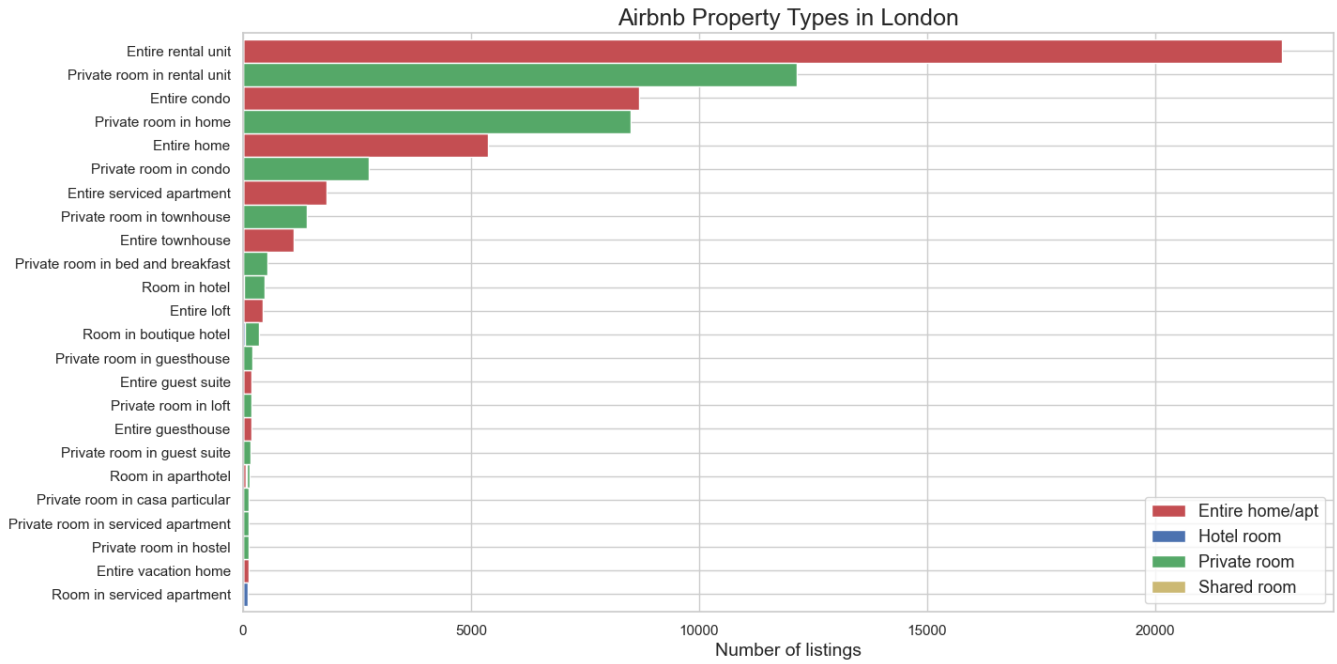
In [14]:
```
ablist.property_type.unique()
```

Out[14]:
```
array(['Private room in rental unit', 'Entire rental unit',
       'Private room in home', 'Private room in loft', 'Houseboat',
       'Entire townhouse', 'Private room in townhouse', 'Entire home',
       'Private room in condo', 'Entire condo',
       'Room in serviced apartment', 'Room in aparthotel', 'Entire loft',
       'Entire serviced apartment', 'Shared room in rental unit',
       'Tiny home', 'Private room in bed and breakfast',
       'Entire guest suite', 'Private room in serviced apartment',
       'Private room in guesthouse', 'Shared room in home',
       'Private room in bungalow', 'Entire guesthouse', 'Entire cabin',
       'Private room in cottage', 'Private room',
       'Private room in nature lodge', 'Room in bed and breakfast',
       'Private room in yurt', 'Shared room in condo',
       'Shared room in townhouse', 'Entire cottage', 'Boat',
       'Private room in guest suite', 'Private room in parking space',
       'Entire place', 'Private room in villa',
       'Private room in houseboat', 'Shared room in loft',
       'Private room in tipi', 'Private room in casa particular', 'Yurt',
       'Shared room', 'Private room in vacation home',
       'Room in boutique hotel', 'Shared room in bed and breakfast',
       'Camper/RV', 'Room in hotel', 'Private room in cabin',
       'Entire home/apt', 'Entire vacation home', 'Entire bungalow',
       'Private room in tiny home', 'Private room in hut', 'Entire villa',
       'Private room in chalet', 'Private room in camper/rv', 'Floor',
       'Entire chalet', 'Shared room in hostel', 'Earthen home',
       'Private room in hostel', 'Shared room in serviced apartment',
       'Room in hostel', 'Private room in floor',
       'Private room in lighthouse', 'Shared room in villa',
       'Private room in island', 'Private room in earthen home', 'Barn',
       'Private room in treehouse', 'Shared room in farm stay',
       'Shared room in bus', 'Shared room in guesthouse',
       'Shared room in bungalow', 'Castle', 'Campsite',
       'Private room in farm stay', 'Shared room in hotel',
       'Shared room in guest suite', 'Shared room in boutique hotel',
       'Hut', 'Room in rental unit', 'Private room in boat',
       'Shared room in earthen home', "Private room in shepherd's hut",
       'Casa particular', 'Private room in castle', 'Dome',
       'Shepherd’s hut', 'Tent', 'Private room in religious building',
       'Minsu', 'Shared room in vacation home', 'Riad', 'Island', 'Tower',
       'Religious building', 'Treehouse', 'Bus'], dtype=object)
```

It seems a bit chunky ... and if we still want to differentiate them by color in a bar chart, it may be challenging for insufficient colors. In order to simplify the data exploration, we will only look at those main types of properties with over 100 listings, and it is worthwhile to consider grouping property types with room types.

In [15]:
```
prt = ablist.groupby(['property_type','room_type']).room_type.count()
prt = prt.unstack()
# sum up all 4 room types for each property type
prt['total'] = prt.iloc[:,0:3].sum(axis = 1)
# list the number of properties by type in an ascending order
prt = prt.sort_values(by=['total'])
```

```
prt = prt[prt['total']>=100]    # property listings over 100
prt = prt.drop(columns=['total'])

# plot the bar chart
prt.plot(kind='barh',stacked=True, color = ["r","b","g", "y"],
              linewidth = 1, grid=True, figsize=(15,8), width=1)
plt.title('Airbnb Property Types in London', fontsize=18)
plt.xlabel('Number of listings', fontsize=14)
plt.ylabel("")
plt.legend(loc = 4,prop = {"size" : 13})
plt.rc('ytick', labelsize=13)
plt.show()
```



It could be observed that majorities of the property are with rooms either `Entire home/apt` or `Private room`, with only a small slice of `hotel room` in serviced apartment. However, are all these rooms listed by private hosts in a C2C manner, or there are also Airbnb hosts list multiple properties/rooms, or there are companies listing their rooms on Airbnb platform? In the latter cases, how could we be able to identify the crucial information out from the massive data? For example, we could check out the hosts' id:

Check out Hosts ID

In [16]:
```
freq = ablist.groupby(['host_id']).size().reset_index(name='num_host_listings')
host_prop = freq.groupby(['num_host_listings']).size().reset_index(name='count').tran
host_prop.columns = host_prop.iloc[0]
host_prop = host_prop.drop(host_prop.index[0])
host_prop
```

Out[16]:

num_host_listings	1	2	3	4	5	6	7	8	9	10	...	121	127	148	157	165
count	37505	4556	1338	572	318	217	136	85	63	52	...	2	2	1	1	1

1 rows × 77 columns

Among all over 76,000 Airbnb listings, there are only 37,500 (Approx.) single-host listings, whilst all the remainings are multiple listings from "big" hosts, there is even a host (highly possible a company) with 285 listings! This requires us to mine the data out following a "zoom-in" workflow, by scoping the data with defined features. Let's take `Entire home/apt` as an example,

In [17]:
```
# the listings labelled as "Entire home/apt"
entire = ablist[ablist['room_type'] == "Entire home/apt"]
entire.shape
```

Out[17]:
```
(41224, 76)
```

It returns in total 41,224 Airbnb listings labelled as Entire home/apt, but we'd like to further investigate the multiple listings issue. So the features on "host_id", "host_name" and property's "neighbourhood" had been used to group up listings as below:

In [18]:
```
host_entire = entire.groupby(['host_id', 'host_name', 'neighbourhood']).size().reset_
host_entire.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18008 entries, 16380 to 18007
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   host_id          18008 non-null  int64 
 1   host_name        18008 non-null  object
 2   neighbourhood    18008 non-null  object
 3   Entire home/apt  18008 non-null  int64 
dtypes: int64(2), object(2)
memory usage: 703.4+ KB
```

Now the total entries have halved (approx.) at 18,008, which indicated the multiple listing issue's existance. So how about another main room type `Private room`?

In [19]:
```
private = ablist[ablist['room_type'] == "Private room"]
private.shape
```

Out[19]:
```
(27479, 76)
```

In [20]:
```
host_private = private.groupby(['host_id', 'host_name', 'neighbourhood']).size().rese
host_private.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11551 entries, 8527 to 11550
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   host_id        11551 non-null  int64 
 1   host_name      11551 non-null  object
 2   neighbourhood  11551 non-null  object
 3   Private room   11551 non-null  int64 
dtypes: int64(2), object(2)
memory usage: 451.2+ KB
```

You are supposed to receive results on Private rooms' listings from 27,479 down to 11,551.

Market Pricing

The product profiling (room type, property type, location, etc.) will directly decide its pricing, but the latter had also been largely influenced by the market environment, so we need also to "draw the full picture" of the Airbnb market pricing:

In [21]:
```
ablist['price'].describe
```

Out[21]:
```
<bound method NDFrame.describe of 0         $50.00
1         $75.00
2         $90.00
3         $55.00
4        $379.00
          ...
69346     $55.00
69347    $201.00
69348    $246.00
69349    $250.00
69350    $134.00
Name: price, Length: 69351, dtype: object>
```

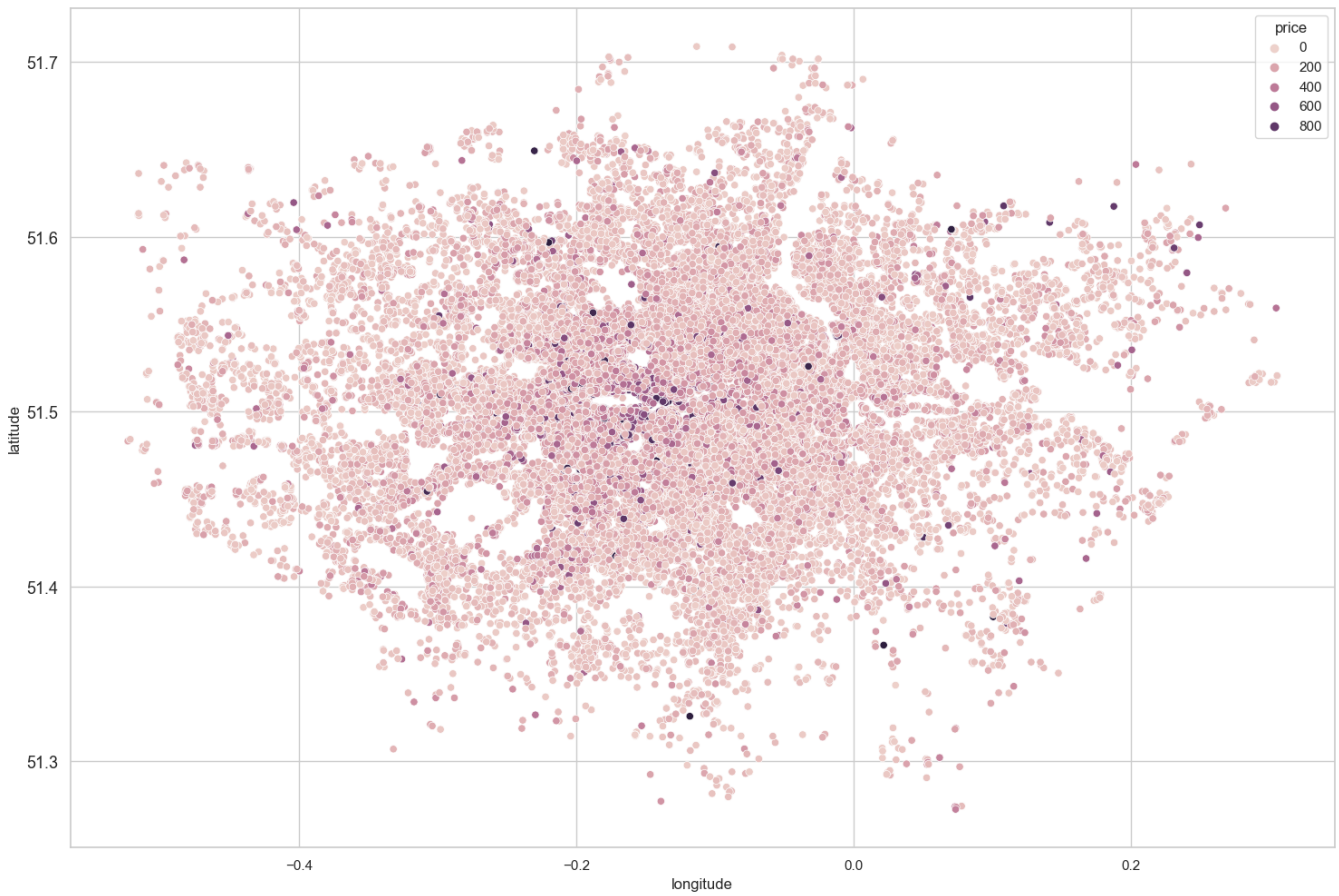
Not surprisingly the **price** column is stored as an object. The '$' is dead giveaways that the **price** column is not a numeric column. we want to clean up the string to remove the extra characters and convert to a float:

In [22]:
```
a=ablist['price'].str.replace('$', '', regex=True)
ablist['price']=pd.to_numeric(a, errors='coerce')
```

Now the price values are numerical and ready for further analysis. We will start with the identification of price distribution pattern, then try to explore the average price or price range for each type of room. A brief scatterplot on the price distribution in London by pricing.

In [23]:
```
plt.figure(figsize=(18,12))
sns.scatterplot(ablist.longitude,ablist.latitude,hue=ablist.price)
plt.ioff()
```

Out[23]:
```
<contextlib.ExitStack at 0x1561117c0>
```



Statistics on pricing by room type & varied ranges of price.

In [24]:
```
ablist.groupby('room_type')['price'].describe().reset_index()
```

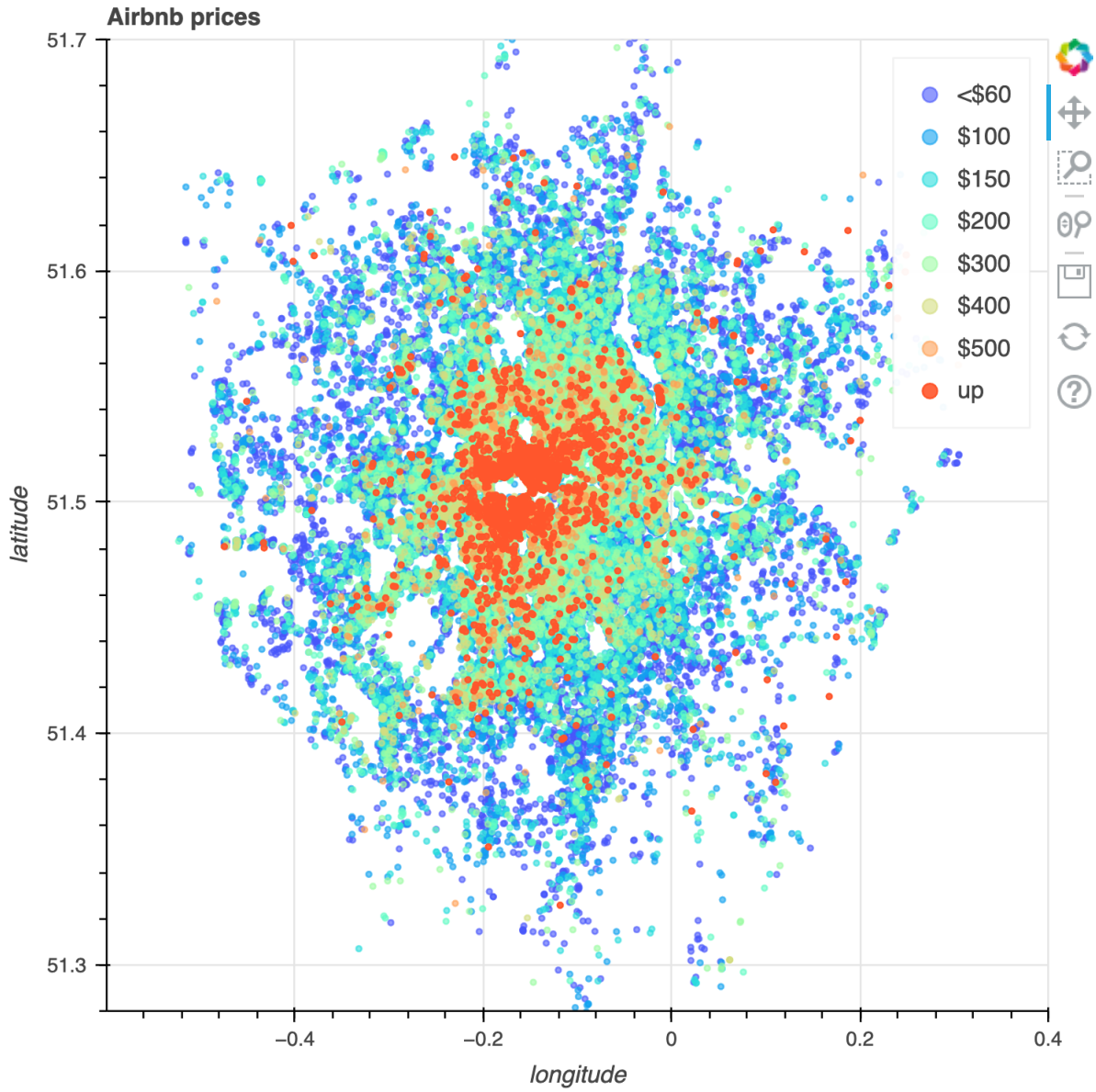
Out[24]:

	room_type	count	mean	std	min	25%	50%	75%	max
0	Entire home/apt	40346.0	185.994565	141.339823	8.0	96.00	145.0	224.0	999.0
1	Hotel room	240.0	225.495833	172.704260	0.0	99.75	200.0	271.0	820.0
2	Private room	26941.0	68.362766	74.438053	0.0	35.00	50.0	75.0	999.0
3	Shared room	403.0	63.918114	81.811948	7.0	29.50	39.0	65.0	850.0

OK, some unbelievable prices (e.g., $8 for entire home/apt) pop up, and we want to get rid of these outliers for further analysis. To get a general idea of the spatial distribution among defined price ranges, say, below 60 dollars, 60-100 dollars, 100-150 dollars, 150-200 dollars, 200-300 dollars, 300-400 dollars, 400-500 dollars and above 500 dollars. We will try `Bokeh` package this time to plot the outputs.

In [25]:
```
# define your points' coordinates for each price range
Lat60=ablist['latitude'][ablist['price']<60]
Long60=ablist['longitude'][ablist['price']<60]

Lat100=ablist['latitude'][(ablist['price']<100)&(ablist['price']>=60)]
Long100=ablist['longitude'][(ablist['price']<100)&(ablist['price']>=60)]

Lat150=ablist['latitude'][(ablist['price']<150)&(ablist['price']>=100)]
Long150=ablist['longitude'][(ablist['price']<150)&(ablist['price']>=100)]

Lat200=ablist['latitude'][(ablist['price']<200)&(ablist['price']>=150)]
Long200=ablist['longitude'][(ablist['price']<200)&(ablist['price']>=150)]

Lat300=ablist['latitude'][(ablist['price']<300)&(ablist['price']>=200)]
Long300=ablist['longitude'][(ablist['price']<300)&(ablist['price']>=200)]

Lat400=ablist['latitude'][(ablist['price']<400)&(ablist['price']>=300)]
Long400=ablist['longitude'][(ablist['price']<400)&(ablist['price']>=300)]

Lat500=ablist['latitude'][(ablist['price']<500)&(ablist['price']>=400)]
Long500=ablist['longitude'][(ablist['price']<500)&(ablist['price']>=400)]

Latup=ablist['latitude'][(ablist['price']>=500)]
Longup=ablist['longitude'][(ablist['price']>=500)]
```

In [26]:
```
p = figure(title="Airbnb prices",x_range=(-0.6,0.4),y_range=(51.28,51.7))
p.xaxis.axis_label = 'longitude'
p.yaxis.axis_label = 'latitude'

# plot points in each price range with different colors
p.circle(Long60, Lat60, size=3,color=rainbow.as_hex()[0],fill_alpha=0.6,line_alpha=0.
p.circle(Long100, Lat100, size=3,color=rainbow.as_hex()[1],fill_alpha=0.6,line_alpha=
p.circle(Long150, Lat150, size=3,color=rainbow.as_hex()[2],fill_alpha=0.6,line_alpha=
p.circle(Long200, Lat200, size=3,color=rainbow.as_hex()[3],fill_alpha=0.6,line_alpha=
p.circle(Long300, Lat300, size=3,color=rainbow.as_hex()[4],fill_alpha=0.6,line_alpha=
p.circle(Long400, Lat400, size=3,color=rainbow.as_hex()[5],fill_alpha=0.6,line_alpha=
p.circle(Long500, Lat500, size=3,color=rainbow.as_hex()[6],fill_alpha=0.6,line_alpha=
p.circle(Longup, Latup, size=3,color=rainbow.as_hex()[7],fill_alpha=0.9,line_alpha=1,
p.legend.location = 'top_right'
show(p, notebook_handle=True)
```



Out[26]: <Bokeh Notebook handle for **In[26]**>

You may already noticed that some buttons generated together with the plot on the right column. Try click on the buttons, and get the right way to zoom in certain areas in London. Of course, you also need to find your way back with another button. Once you zoomed in, you may find that areas e.g. near East Village, Chelsea, Hell's Kitchen, and Upper East Side, look brighter than the neighbors, indicating higher interest or more requests from Airbnb "hunters". In order to figure out which specific type of Airbnb really contribute to the distribution, we can plot them out by type separately.

As you may find, there are mainly 4 types of room types, Entire home/apt, Private room, Shared room and Hotel room, so we will try to plot them by different colors below.

In [27]:
```
# set up the plot frame,
# define x axis and y axis scales by the upper and lower limits of corresponding valu
p = figure(title="London airbnb room types", x_range=(-0.6,0.4), y_range=(51.28,51.7)
p.xaxis.axis_label = 'longitude'
p.yaxis.axis_label = 'latitude'
# set up latitude and longitude variables for type of Private room
privateLat=ablist['latitude'][ablist['room_type']=='Private room']
privateLong=ablist['longitude'][ablist['room_type']=='Private room']

# set up latitude and longitude variables for type of Entire home/apt
entireLat=ablist['latitude'][ablist['room_type']=='Entire home/apt']
```

```
entireLong=ablist['longitude'][ablist['room_type']=='Entire home/apt']

# plot private room
p1 = figure(width=500, height=400, title='Private airbnb rooms distribution in London
p1.circle(privateLong,privateLat,size=3,color=rainbow.as_hex()[1],fill_alpha=0.2,line

# plot entire home/apt
p3 = figure(width=500, height=400, title='Entire airbnb homes/apartments distribution
p3.circle(entireLong,entireLat,size=3,color=rainbow.as_hex()[3],fill_alpha=0.2,line_a

# get plots show
show(column(p1,p3), notebook_handle=True)
```

Out[27]: <Bokeh Notebook handle for **In[27]**>

Pricing by Geographical Regions

We will join the Airbnb listings data onto London geographical boundary data (LSOA), where the latter bears geometry information with it. It requests to call `sjoin` (spatial join) function from **geopandas (gpd)** package, and join the point-based Airbnb listings to its corresponding LSOA if falls within it.

In [28]:
```
lsoa=gpd.read_file('data/LSOA_london/LSOA_london.shp')
```

In [29]:
```
ab_lsoa = gpd.sjoin(ablist, lsoa, how="inner", op='within')
# check the columns upon spatial join
ab_lsoa.columns.to_list()
```

Out[29]:
```
['id',
 'listing_url',
 'scrape_id',
 'last_scraped',
 'source',
 'name',
 'description',
 'neighborhood_overview',
 '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',
 'neighbourhood',
 'neighbourhood_cleansed',
 'neighbourhood_group_cleansed',
 'latitude',
 'longitude',
 'property_type',
 'room_type',
 'accommodates',
 'bathrooms',
 'bathrooms_text',
 'bedrooms',
 'beds',
 'amenities',
 'price',
 'minimum_nights',
 'maximum_nights',
 'minimum_minimum_nights',
 'maximum_minimum_nights',
 'minimum_maximum_nights',
 'maximum_maximum_nights',
 'minimum_nights_avg_ntm',
 'maximum_nights_avg_ntm',
 'calendar_updated',
 'has_availability',
 'availability_30',
 'availability_60',
 'availability_90',
 'availability_365',
 'calendar_last_scraped',
 'number_of_reviews',
 'number_of_reviews_ltm',
 'number_of_reviews_l30d',
 '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',
'license',
'instant_bookable',
'calculated_host_listings_count',
'calculated_host_listings_count_entire_homes',
'calculated_host_listings_count_private_rooms',
'calculated_host_listings_count_shared_rooms',
'reviews_per_month',
'geometry',
'index_right',
'LSOA11CD',
'LSOA11NM',
'Value',
'USUALRES',
'HHOLDRES',
'COMESTRES',
'POPDEN',
'HHOLDS',
'AVHHOLDSZ',
'IMDScore',
'Borough']
```

In [30]:
```
print("airbnb has {0} rows, {1} columns".format(ab_lsoa.shape[0], ab_lsoa.shape[1]))
ab_lsoa.sample(3)[['name', 'room_type', 'accommodates', 'neighbourhood_cleansed', 'pr
```

```
airbnb has 68588 rows, 88 columns
```

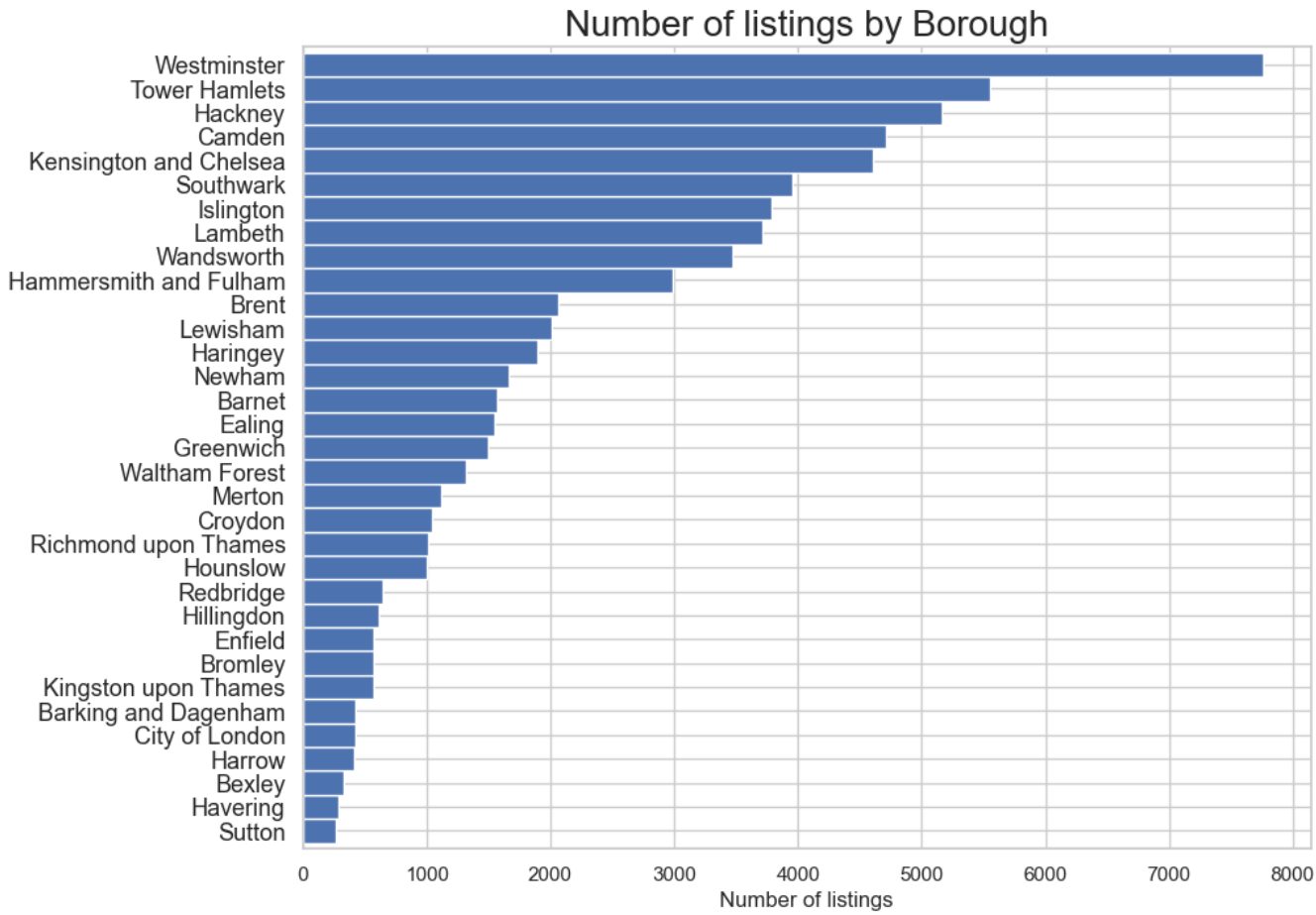
Out[30]:

	name	room_type	accommodates	neighbourhood_cleansed	price
11265	Complete 3 Bedroom House- great London location	Entire home/apt	6	Barnet	146.0
67089	Apartment with large patio in central Marylebone	Entire home/apt	2	Westminster	275.0
26681	Private garden tent pitches for All Points East!	Entire home/apt	2	Tower Hamlets	50.0

There are in total 68, 588 entries still, but the number of features/columns had increased to 88. We know that London has 32 boroughs plus City of London, so we may be able to work out the statistics on Airbnbs in each borough since the **neighbourhood_cleansed (Borough)** column has included now.

By Borough

In [31]:
```
list_bor=ab_lsoa['neighbourhood_cleansed'].value_counts().sort_values(ascending=True)
list_bor.plot.barh(figsize=(10, 8), color='b', width=1)
plt.title("Number of listings by Borough", fontsize=20)
plt.xlabel('Number of listings', fontsize=12)
plt.show()
```



The output echoes with previous findings that, Airbnb listings tend to accumulate in boroughs in Central London (all the top 5 boroughs in this bar chart are around central London, especially Westminster).

This is already good enough for information; however, is it possible to improve further, i.e., enabling customers who are not that familiar with London to be able to explore it from an interactive map at borough level? It reminds us the package **folium**, by:

- do the statistics on each borough's average price. Just to make the column label more readable, we can *rename* it to Borough.
- map out the spatial differences.

In [32]:
```
# average price for each borough
ab_bor = ab_lsoa.groupby('neighbourhood_cleansed')['price'].mean().sort_values(ascend
```

In [33]:
```
# calculate the value on mean price as target variable to map with
bor_price = pd.merge(lsoa, ab_bor, how='left', left_on='Borough', right_on='neighbour
bor_price.rename(columns={'price': 'average_price'}, inplace=True)
bor_price.average_price = bor_price.average_price.round(decimals=0)
```

In [34]:
```
# set up the mapping scale by target variable's value range
map_dict = bor_price.set_index('Borough')['average_price'].to_dict()
color_scale = LinearColormap(['blue','red'], vmin = min(map_dict.values()), vmax = ma

def get_color(feature):
    value = map_dict.get(feature['properties']['Borough'])
    return color_scale(value)

# draw the map
map_bor = folium.Map(location=[51.5080, -0.1262], zoom_start=11)
folium.GeoJson(data=bor_price,
               name='London Boroughs',
```

```
                tooltip=folium.features.GeoJsonTooltip(fields=['Borough', 'average_pri
                                                       labels=True,
                                                       sticky=False),
                style_function= lambda feature: {
                    'fillColor': get_color(feature),
                    'color': 'black',
                    'weight': 1,
                    'dashArray': '5, 5',
                    'fillOpacity':0.5
                    },
                highlight_function=lambda feature: {'weight':3, 'fillColor': get_color
map_bor.save('London Boroughs Airbnb average price.html')
```

You may feel free to explore the webpage based map from your local repository, and think of future marketing visualisation application tailoring to your own datasets.

By LSOAs

Besides of borough level, researchers normally scale down to finer geographical scale, LSOA, and we could "mine" the data for finer information at LSOA scale. For example, the proportions of varied types of properties in each LSOA.

In [35]:
```
# calculate the percentage of each property type in respective LSOA
types = pd.get_dummies(ab_lsoa['property_type'])
prop_types = types.join(ab_lsoa['LSOA11CD']).groupby('LSOA11CD').sum()
prop_types_pct = (prop_types * 100.).div(prop_types.sum(axis=1), axis=0)
prop_types_pct.head()
```

Out[35]:

LSOA11CD	Barn	Boat	Camper/RV	Campsite	Casa particular	Castle	Dome	Earthen home	Entire bungalow	Entire cabin
E01000001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
E01000002	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
E01000003	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
E01000005	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
E01000006	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 98 columns

We'd now like to "piece up" this newly created data with features like No. of bedrooms, accommodates, the review scores, the average days of availability in 90 days, and the average price per night for each LSOA. This will help to recap on 2 functions: **groupby** and **join**.

In [36]:
```
# work with selected features
selection = ab_lsoa.groupby('LSOA11CD')['bedrooms', 'accommodates', 'review_scores_ra

# bring these data together
sel_props = selection.join(prop_types_pct)
```

You may feel a bit confused on why we need to join these features together, and what is the aim for?

They are actually features describing the Airbnb listing's "look and feel" profile, and we'd like to use them "class up" 4835 LSOAs by shared characteristics, which is exactly the Clustering/Classification machine learning technique such as, `KMeans`.

To start with, we are going to standardise each column (such data could be checked through histogram chart by calling **hist** function from **Seaborn**.

In [37]:
```
# standardise columns to prepare for later clustering algorithm
sel_props.iloc[:, :] = StandardScaler().fit_transform(sel_props.iloc[:, :])
sel_props.head()
```

Out[37]:

LSOA11CD	bedrooms	accommodates	review_scores_rating	availability_90	price	Barn	
E01000001	-0.374923	-0.048235	0.171341	0.218555	1.050248	-0.017169	-0.0
E01000002	0.045967	-0.177045	-0.873558	-0.949974	0.528919	-0.017169	-0.0
E01000003	-0.853685	-0.254524	0.236864	-0.414559	0.258457	-0.017169	-0.0
E01000005	-0.663654	-0.126363	0.188288	0.609513	2.544250	-0.017169	-0.0
E01000006	-0.964169	-1.097111	0.225217	-0.010693	-1.022245	-0.017169	-0.0

5 rows × 103 columns

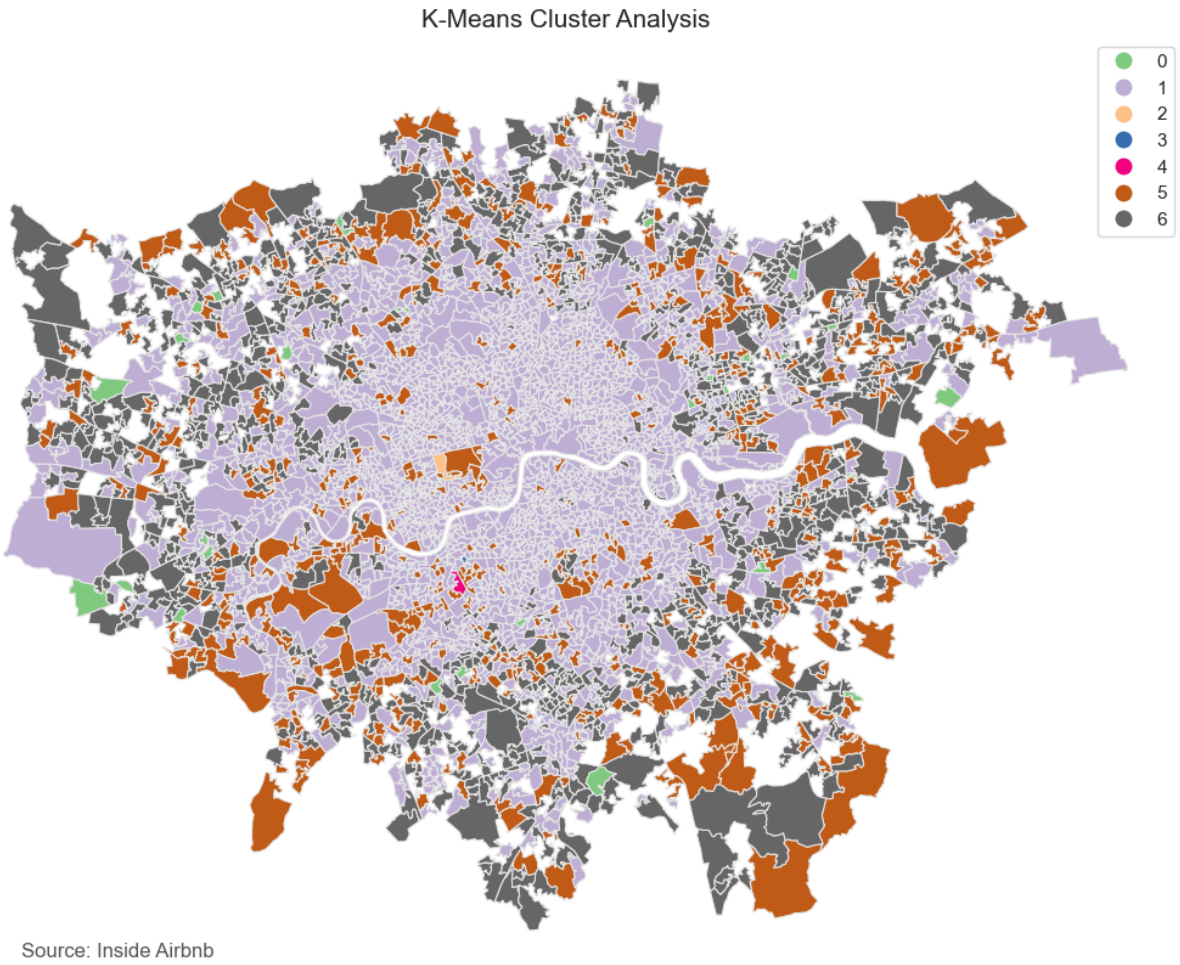
Since we'd like to map out the clustering result later, we would like also include the geographical feature "geometry" into our data, which need to resort back to "lsoa" geodataframe by joining the shared column "LSOA11CD".

In [38]:
```
cluster_lsoa = lsoa[['geometry', 'LSOA11CD']].join(sel_props, on='LSOA11CD').dropna()
```

It is now the time to call `KMeans` method from `Sciki-learn`'s `cluster` function.

In [44]:
```
km7 = cluster.KMeans(n_clusters=7)

km7cls = km7.fit(cluster_lsoa.drop(['geometry', 'LSOA11CD'], axis=1).values)

k_var  = 'KMeans' # Variable name
# Add it to the data frame
cluster_lsoa[k_var] = pd.Series(km7cls.labels_, index=cluster_lsoa.index)
```

In [52]:
```
# set up the plot configuration
fig, ax = plt.subplots(1, figsize=(14, 10))
# get the map
cluster_lsoa.assign(cl=km7cls.labels_).plot(column='cl', cmap='Accent', linewidth=0.8
# map title
ax.set_title('K-Means Cluster Analysis', fontdict={'fontsize': '15', 'fontweight': '2
# create an annotation for the data source
ax.annotate('Source: Inside Airbnb',xy=(0.1,.08), xycoords='figure fraction', horizon
# turn off the axis
ax.axis('off')
# save your figure into local directory
# fig.savefig('K-Means.png', dpi=150)
plt.show()
```



Clustering outputs are normally informative when working out regional marketing strategies, rather than purely considering "nearby neighbourhood" or simply "city centre-outskirt" principles.

Does house price matters more than Facilities?

In [53]:
```
# identify those Airbnb hosts with multiple listings i.e. >=10
ab_lsoa['Multiple Location Host'] = ab_lsoa.calculated_host_listings_count >=10
# subset of data would like to keep in further analysis, e.g., describing the "look a
variables=['bedrooms', 'accommodates', 'review_scores_rating','availability_90', 'pri
# aggregate these numerical data by obtaining the average (of bedrooms, etc.) per LSO
df = ab_lsoa.groupby('LSOA11CD')[variables].mean()
lsoa_2 = pd.merge(lsoa, df, how='left', left_on='LSOA11CD', right_index=True)
```

In [54]:
```
df = ab_lsoa[['LSOA11CD','Multiple Location Host']].groupby(['LSOA11CD','Multiple Loc
df = df.pivot(index='LSOA11CD', columns='Multiple Location Host', values='counts').fi
# a new variable called "property count"
lsoa_new = pd.merge(lsoa_2, df, how='left', left_on='LSOA11CD', right_index=True)
lsoa_new ['Property Count'] = lsoa_new ['Multiple Location Host'] + lsoa_new['Small H
```

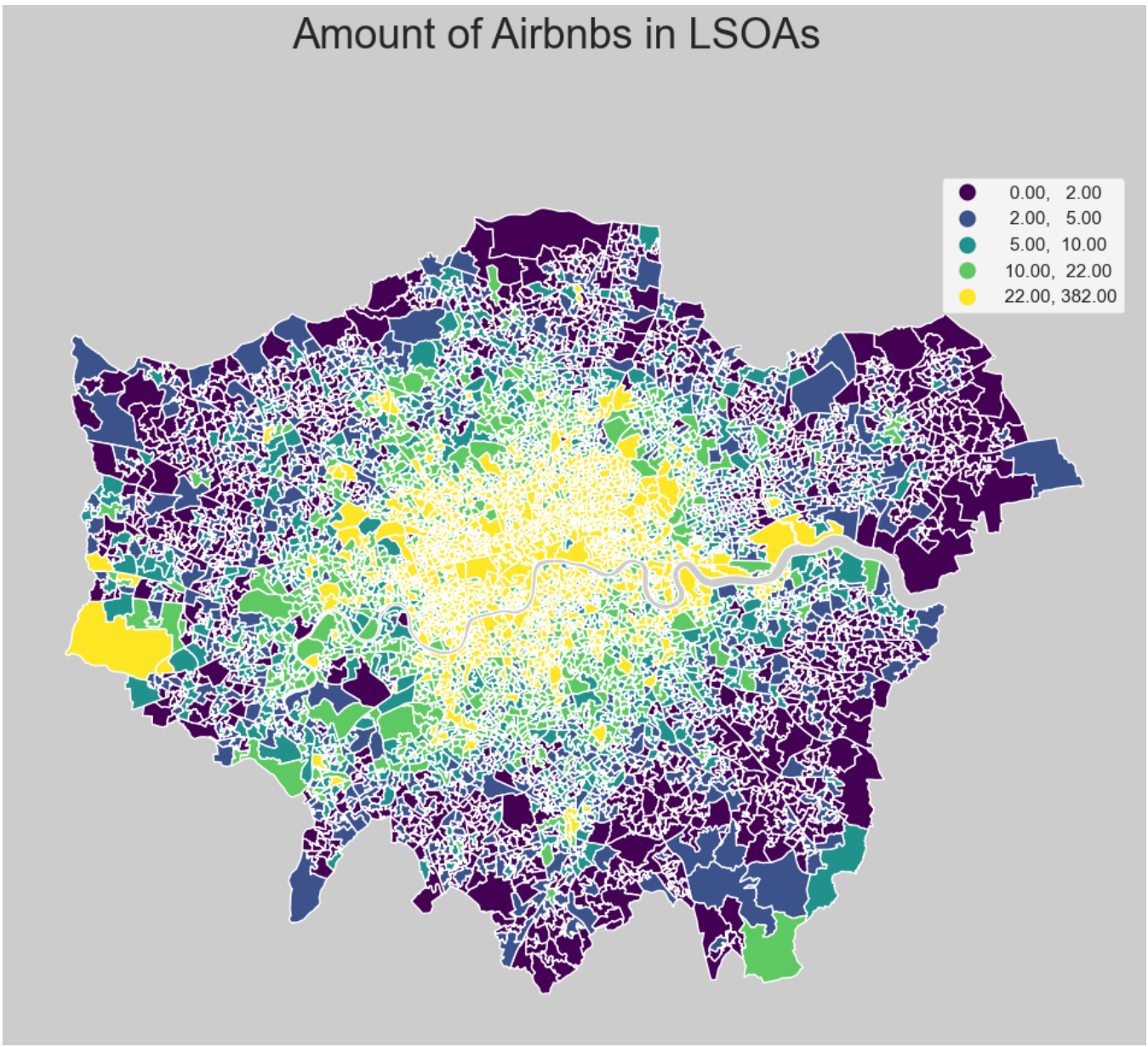
In [55]:
```
# missing data filling up
db_new=lsoa_new.fillna(0)
# save the data as geodataframe and further save it physically into your computer dis
db_geo = gpd.GeoDataFrame(db_new, geometry='geometry')
db_geo.to_file(driver='ESRI Shapefile', filename='data/lsoa_airbnb_housing.shp')
```

Now the airbnbs have been grouped by LSOAs, with some newly created variables on: the mean price of Airbnbs in each LSOA, the total number of properties in each LSOA, etc.

check the information on **lsoa**, you may find varied number of entries for columns. For example, variables for number of households have 4835 rows, whilst variables from Airbnb summarization have 4479 rows.

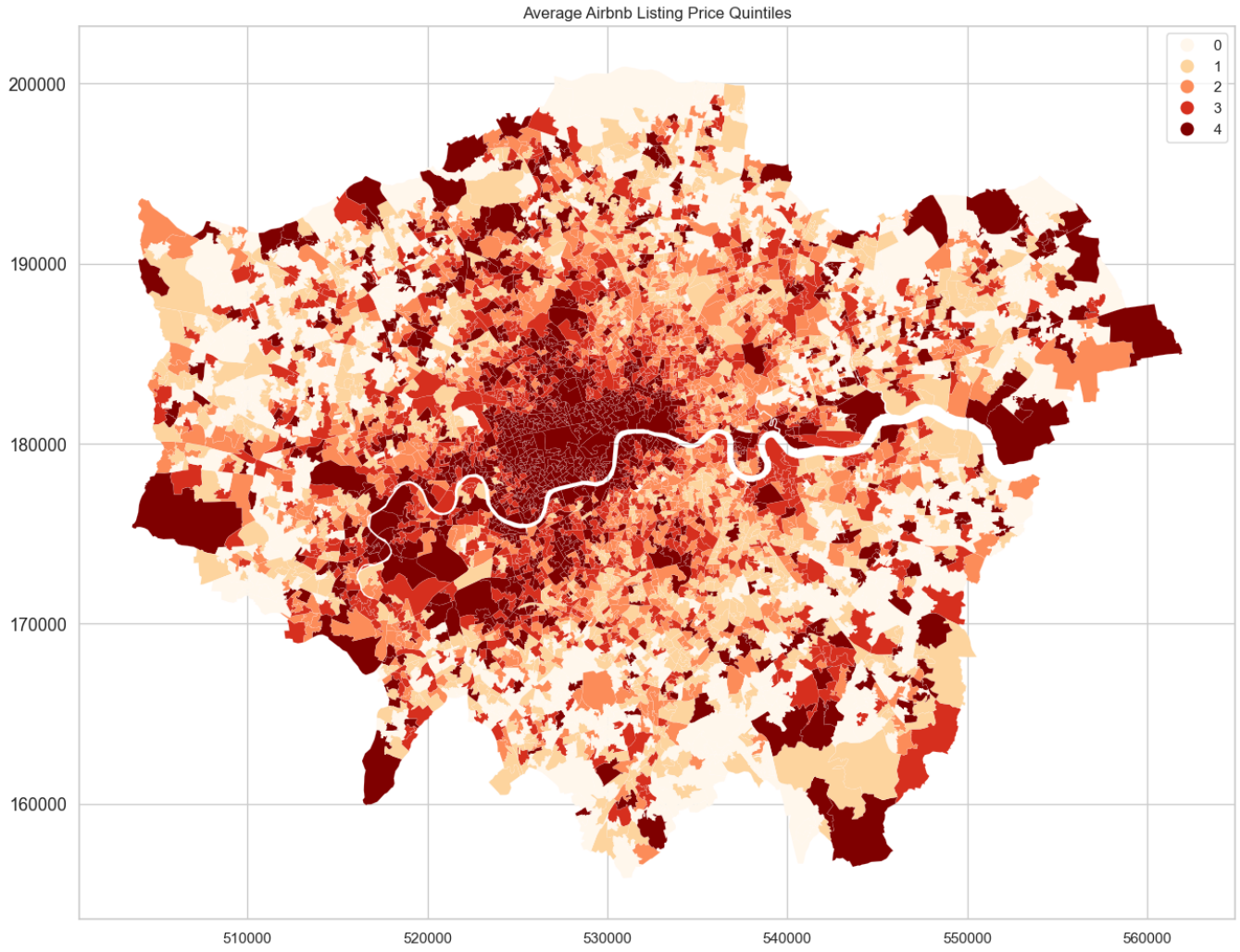
We could now visualise the distribution of Airbnb listings among LSOAs and their prices in quantile maps. The following maps will help to lead our way exploring "how many" and "how much" Airbnbs in London's LSOAs.

In [56]:
```
# Set up figure and axis
f, ax = plt.subplots(1, figsize=(12,10))
# Plot Number of Airbnbs
# Quickly plot
db_geo.plot(column='Property Count', scheme='Quantiles', legend=True, ax=ax)
# Remove axis frame
ax.set_axis_off()
# Change background color of the figure
f.set_facecolor('0.8')
# set up the title
f.suptitle('Amount of Airbnbs in LSOAs', size=25)

plt.show()
```



In [57]:
```
import mapclassify
pr = mapclassify.Quantiles(db_geo['price'], k=5)
f, ax = plt.subplots(1, figsize=(20, 12))
```

```
db_geo.assign(cl_pr=pr.yb).plot(column='cl_pr', categorical=True, k=5, cmap='OrRd',
                                linewidth=0.1, ax=ax, edgecolor='white', legend

plt.title('Average Airbnb Listing Price Quintiles')
plt.show()
```



Multivariate Regression

We had successfully present the spatial distribution of Airbnb listings and the corresponding prices among 4835 LSOAs as above. To better tailor our future marketing strategies towards local characteristics, it will be insightful to run regression model(s) consider spatial effects. We take 8 independant variables as an exmaple, to analyze the relationship between `Airbnb listing price` and selected independent variables, for example, `deprivation indices` , `house price` , `population density` , etc., which is recommended to use package `PySAL` .

In [58]:
```
# read in the data in geodataframe
gdf=gpd.read_file('data/lsoa_airbnb_housing.shp')
# First, prepare the dataset so it can be processed in pysal
price = gdf['price']
# Read in the listing_price (dependent variable) into an array y
y = np.array(price)
# PySAL requires your dependent variable to be nx1 numpy array
y.shape = (len(price),1)

# value for independent variables into a one dimmensional array X.
# You can feel free to change the independant variables
X= []
X.append(gdf['Value']) # average house price
X.append(gdf['HHOLDS']) # number of households
X.append(gdf['POPDEN']) # population density
```

```
X.append(gdf['USUALRES']) # usual residents
X.append(gdf['bedrooms']) # rooms
X.append(gdf['accommodat']) # accommodates
X.append(gdf['review_sco']) # review score values
X.append(gdf['Property C']) # property count
X = np.array(X).T
```

In [59]:
```
import libpysal as lps
W_queen = lps.weights.Queen.from_shapefile('data/lsoa_airbnb_housing.shp')
```

Spatial Lag model

In a similar way to how we have included the spatial lag, we could try the spatial lag model estimation with maximum likelihood. `Spatial lag` is the product of the spatial weights matrix and a given variable and that, if W is row-standardized, the result amounts to the average value of the variable in the neighborhood of each observation, utilising `ML_Lag` class in `pysal.model.spreg` to estimate this model.

In [60]:
```
from pysal.model.spreg import ML_Lag

# Read in the listing_price (dependent variable) into an array y
y = np.array(price)
# PySAL requires your dependent variable to be nx1 numpy array
y.shape = (len(price),1)

# value for independent variables into a one dimmensional array X.
# You can feel free to change the independant variables
X= []
X.append(gdf['Value']) # average house price
X.append(gdf['HHOLDS']) # number of households
X.append(gdf['POPDEN']) # population density
X.append(gdf['USUALRES']) # usual residents
X.append(gdf['bedrooms']) # rooms
X.append(gdf['accommodat']) # accommodates
X.append(gdf['review_sco']) # review score values
X.append(gdf['Property C']) # property count
X = np.array(X).T
```

In [61]:
```
spat_lag = ML_Lag(y, X, W_queen, name_y = 'price',
                  name_w='W_queen', name_x = ['Value', 'HHOLDS', 'POPDEN', 'USUALRES'
print(spat_lag.summary)
```

```
REGRESSION
----------
SUMMARY OF OUTPUT: MAXIMUM LIKELIHOOD SPATIAL LAG (METHOD = FULL)
-----------------------------------------------------------------
Data set            :lsoa_airbnb_housing
Weights matrix      :     W_queen
Dependent Variable  :       price                Number of Observations:        4835
Mean dependent var  :    100.4320                Number of Variables   :          10
S.D. dependent var  :     67.9558                Degrees of Freedom    :        4825
Pseudo R-squared    :      0.5816
Spatial Pseudo R-squared:  0.5744
Sigma-square ML     :    1931.821                Log likelihood        :  -25156.599
S.E of regression   :      43.952                Akaike info criterion :   50333.199
                                                 Schwarz criterion     :   50398.035

------------------------------------------------------------------------------------
            Variable     Coefficient       Std.Error     z-Statistic     Probability
------------------------------------------------------------------------------------
            CONSTANT       1.2871800       4.6709680       0.2755703       0.7828781
               Value       0.0000282       0.0000016      17.9110649       0.0000000
              HHOLDS       0.0434959       0.0070352       6.1826489       0.0000000
              POPDEN       0.0746180       0.0110157       6.7737598       0.0000000
            USUALRES      -0.0271210       0.0032541      -8.3343584       0.0000000
            bedrooms      16.2155833       1.9090911       8.4938760       0.0000000
          accommodat      21.5427863       0.9333360      23.0814898       0.0000000
          review_sco      -2.1635420       0.4715693      -4.5879619       0.0000045
          Property C       0.5369463       0.0383544      13.9995843       0.0000000
             W_price       0.0178611       0.0024406       7.3182954       0.0000000
------------------------------------------------------------------------------------
================================ END OF REPORT =====================================
```

Since we've analysed the product and market environment already, it is the time to analyse customers' feedback on the product, Airbnb listings, by analysing customers' **Reviews and Responses**.

Reviews and Responses

On the Airbnb platform, customers (or so-called guests) are normally been invited to submit their text reviews for the property/room afterwards, and also an overall star rating and a set of category star ratings on:

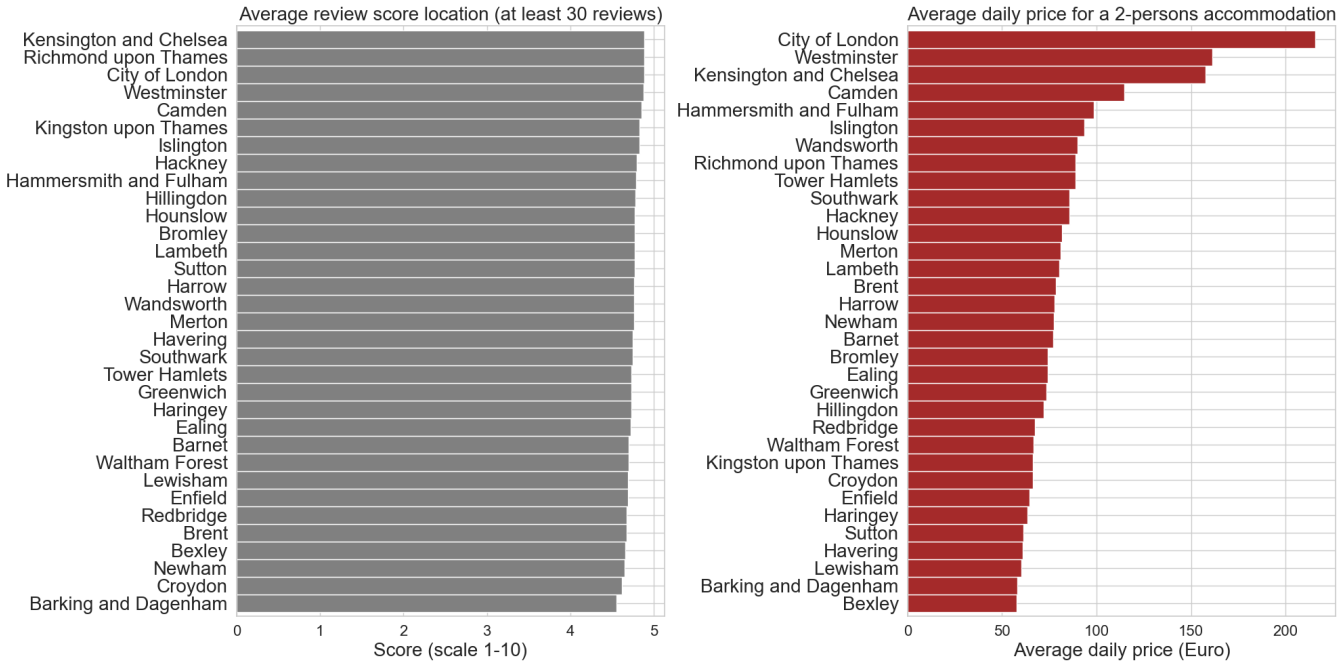
- Overall Experience (What was your overall experience?)
- Cleanliness (Did you feel that your space was clean and tidy?)
- Accuracy (How accurately did your listing page represent your space?)
- Value (Did you feel your listing provided good value for the price?)
- Communication (How well did you communicate with your host before and during their stay?)
- Arrival (How smoothly did their check-in go?)
- Location (How did you feel about the neighborhood?)

We'd like to know more about the customers' opinions on regular Airbnb listings (say with more than 10 reviews recorded, as the number of "10 averages" is reasonably small, which may make the indicator a bit more "distinguishable" than other indicators), and explore their comparisons among boroughs. For example, from a customer's stand, when we are going to choose the Airbnb property in London for accommodation, we may want to select the borough with good comments on its location whilst keeping a good value of my money.

In [62]:
```
ratings = [i for i in ab_lsoa if 'review_scores_' in i]
ratings
```

Out[62]:
```
['review_scores_rating',
 'review_scores_accuracy',
 'review_scores_cleanliness',
 'review_scores_checkin',
 'review_scores_communication',
 'review_scores_location',
 'review_scores_value']
```

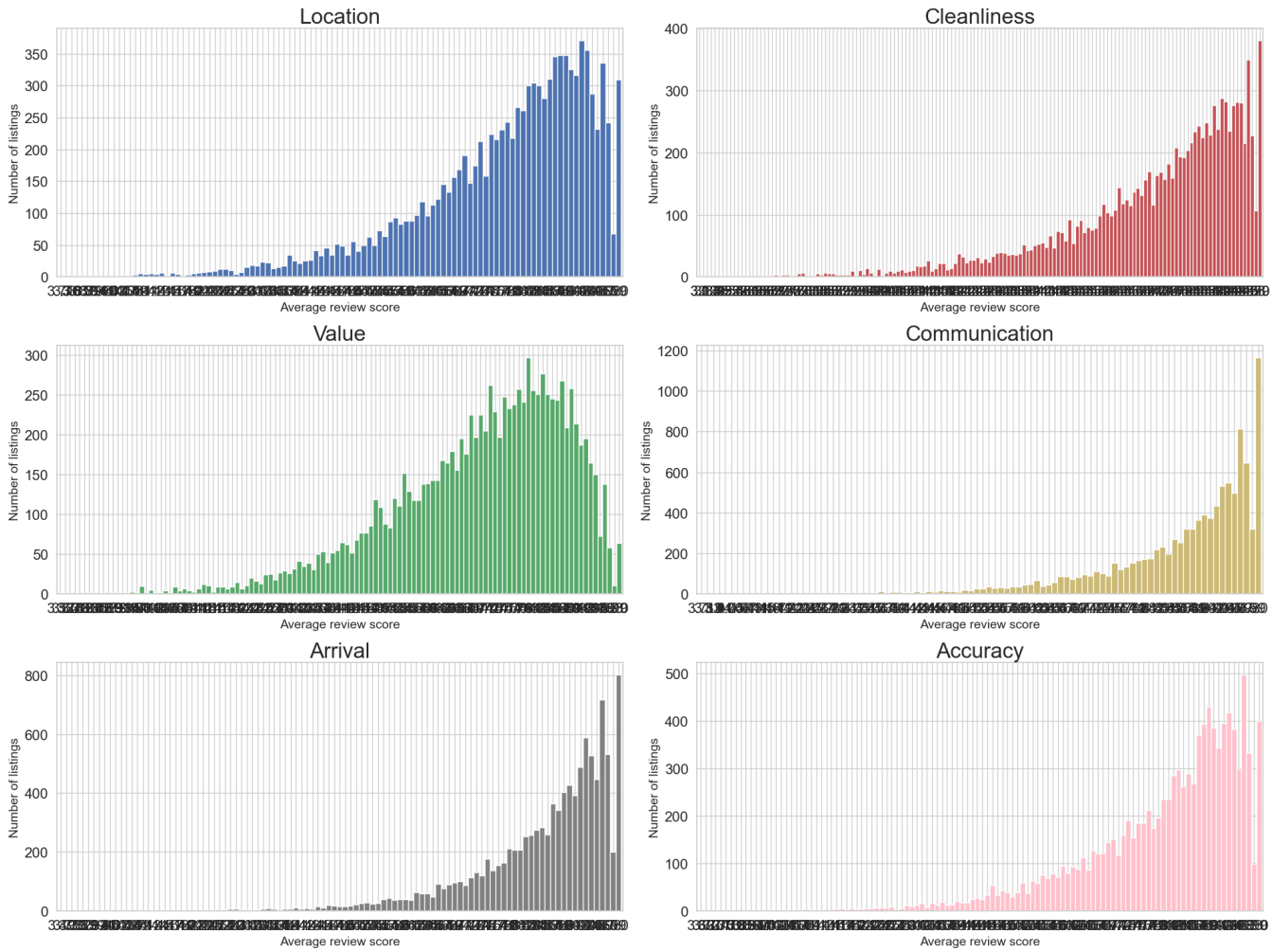
In [63]:
```
# only look at those propoerties with more than 30 reviews
review = ab_lsoa[ab_lsoa['number_of_reviews']>=30]

fig = plt.figure(figsize=(20,10))
plt.rc('xtick', labelsize=16)
plt.rc('ytick', labelsize=20)

# left plot for reviews on location
ax1 = fig.add_subplot(121)
review_bor = review.groupby('Borough')['review_scores_location'].mean().sort_values(a
ax1=review_bor.plot.barh(color='grey', width=1)
plt.title("Average review score location (at least 30 reviews)", fontsize=20)
plt.xlabel('Score (scale 1-10)', fontsize=20)
plt.ylabel("")

# right plot for average pricing
ax2 = fig.add_subplot(122)
accom = ab_lsoa[ab_lsoa['accommodates']==2]
accom_bor = accom.groupby('Borough')['price'].mean().sort_values(ascending=True)
ax2=accom_bor.plot.barh(color='brown', width=1)
plt.title("Average daily price for a 2-persons accommodation", fontsize=20)
plt.xlabel('Average daily price (Euro)', fontsize=20)
plt.ylabel("")

plt.tight_layout()
plt.show()
```



What caught my eye immediately from the above is that, review scores on location seem really high across the board! A quick internet search told me that this seems common across Airbnb. It is explained well that 95% of Airbnb listings rated 4.5 to 5 stars.

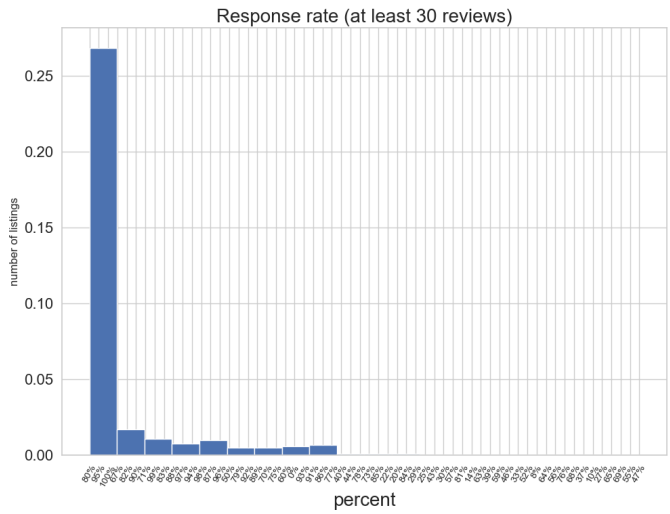
If I were a customer travelling to London soon, I would personally consider any score of 8 or lower to be not a good score, hence use any of the detailed scores in a search for accomodation. So I'd like to explore further on detailed review scores' distribution of all those categories.

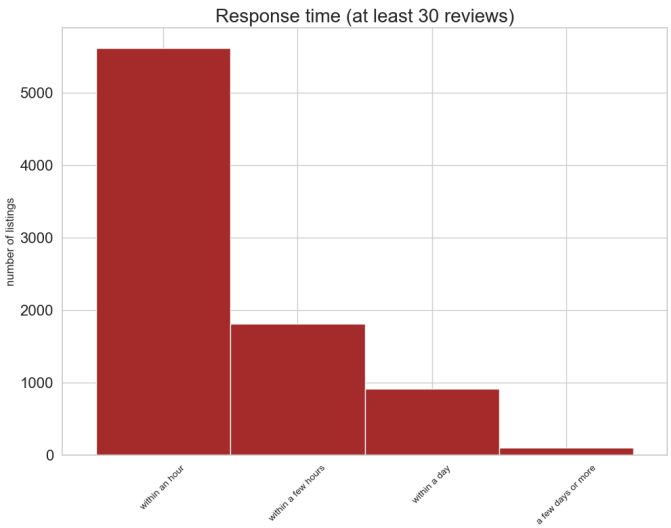
In [64]:
```
fig = plt.figure(figsize=(20,15))
plt.rc('xtick', labelsize=16)
plt.rc('ytick', labelsize=16)

ax1 = fig.add_subplot(321)
review_loc=review['review_scores_location'].value_counts().sort_index()
ax1=review_loc.plot.bar(color='b', width=1, rot=0)
#ax1.tick_params(axis = 'both', labelsize = 16)
plt.title("Location", fontsize=24)
plt.ylabel('Number of listings', fontsize=14)
plt.xlabel('Average review score', fontsize=14)

ax2 = fig.add_subplot(322)
review_cle=review['review_scores_cleanliness'].value_counts().sort_index()
ax2=review_cle.plot.bar(color='r', width=1, rot=0)
plt.title("Cleanliness", fontsize=24)
plt.ylabel('Number of listings', fontsize=14)
plt.xlabel('Average review score', fontsize=14)

ax3 = fig.add_subplot(323)
review_val=review['review_scores_value'].value_counts().sort_index()
ax3=review_val.plot.bar(color='g', width=1, rot=0)
plt.title("Value", fontsize=24)
plt.ylabel('Number of listings', fontsize=14)
plt.xlabel('Average review score', fontsize=14)

ax4 = fig.add_subplot(324)
review_comm=review['review_scores_communication'].value_counts().sort_index()
ax4=review_comm.plot.bar(color='y', width=1, rot=0)
plt.title("Communication", fontsize=24)
plt.ylabel('Number of listings', fontsize=14)
plt.xlabel('Average review score', fontsize=14)

ax5 = fig.add_subplot(325)
review_che=review['review_scores_checkin'].value_counts().sort_index()
ax5=review_che.plot.bar(color='grey', width=1, rot=0) # the bar color is grey
plt.title("Arrival", fontsize=24)
plt.ylabel('Number of listings', fontsize=14)
plt.xlabel('Average review score', fontsize=14)

# plot the bar chart in pink on subplot 326, on variable review_scores_accuracy
ax6 = fig.add_subplot(326)
review_accu=review['review_scores_accuracy'].value_counts().sort_index()
ax6=review_accu.plot.bar(color='pink', width=1, rot=0) # the bar color is grey
plt.title("Accuracy", fontsize=24)
plt.ylabel('Number of listings', fontsize=14)
plt.xlabel('Average review score', fontsize=14)

plt.tight_layout()
plt.show()
```



As we can see, majority of the listings have at least 30 reviews and respond to at least 90% of the new messages. So the host's responses to reviews and comments will also be considered into proving them being good responders on time, which will influence my making the choice from their response rate and speed.

In [65]:
```
fig = plt.figure(figsize=(20,8))
plt.rc('xtick', labelsize=10)
plt.rc('ytick', labelsize=16)

ax1 = fig.add_subplot(121)
res_rate = review['host_response_rate'].dropna()
ax1= plt.hist(res_rate, bins=20, density=True)
plt.title("Response rate (at least 30 reviews)", fontsize=20)
plt.ylabel("number of listings")
plt.xlabel("percent", fontsize=20)
plt.xticks(rotation=60)

ax2 = fig.add_subplot(122)
res_dur = review['host_response_time'].value_counts()
ax2=res_dur.plot.bar(color='brown', width=1, rot=45)
plt.title("Response time (at least 30 reviews)", fontsize=20)
plt.ylabel("number of listings")

plt.tight_layout()
plt.show()
```





Westminster Borough as an example

In [66]:
```
dwest = ab_lsoa[ab_lsoa['Borough']=='Westminster']
```

In [67]:
```
#Encode the input Variables
def Encode(airbnb):
    for column in airbnb.columns[airbnb.columns.isin(['neighbourhood_group', 'room_ty
        airbnb[column] = airbnb[column].factorize()[0]
    return airbnb

airbnb_en = Encode(dwest.drop(['host_id','latitude','longitude','neighbourhood','numb
airbnb_en.head(5)
```

Out[67]:

	id	listing_url	scrape_id	last_scraped	source	nam
4	17402	https://www.airbnb.com/rooms/17402	20220910194334	2022-09-11	city scrape	Superb 3 Bed Bath Wi Trendy W
1451	1291630	https://www.airbnb.com/rooms/1291630	20220910194334	2022-09-11	city scrape	Centr Londo Larg Stud Fl
1679	1780542	https://www.airbnb.com/rooms/1780542	20220910194334	2022-09-11	previous scrape	STUDIO CENTRA LONDO - JUL
3169	3990981	https://www.airbnb.com/rooms/3990981	20220910194334	2022-09-11	previous scrape	Moder spaciou 2 be Fitzrov Apartme
3202	4206679	https://www.airbnb.com/rooms/4206679	20220910194334	2022-09-11	previous scrape	A sma cosy roo for or person Oxfo Circu

5 rows × 83 columns

In [68]:

```
#Get Correlation between different variables
corr = airbnb_en.corr(method='kendall')
plt.figure(figsize=(18,12))
sns.heatmap(corr, annot=True)
airbnb_en.columns
```

Out[68]:
```
Index(['id', 'listing_url', 'scrape_id', 'last_scraped', 'source', 'name',
       'description', 'neighborhood_overview', 'picture_url', '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',
       'neighbourhood_cleansed', 'neighbourhood_group_cleansed',
       'property_type', 'room_type', 'accommodates', 'bathrooms',
       'bathrooms_text', 'bedrooms', 'beds', 'amenities', 'price',
       'minimum_nights', 'maximum_nights', 'minimum_minimum_nights',
       'maximum_minimum_nights', 'minimum_maximum_nights',
       'maximum_maximum_nights', 'minimum_nights_avg_ntm',
       'maximum_nights_avg_ntm', 'calendar_updated', 'has_availability',
       'availability_30', 'availability_60', 'availability_90',
       'availability_365', 'calendar_last_scraped', 'number_of_reviews_ltm',
       'number_of_reviews_l30d', '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', 'license', 'instant_bookable',
       'calculated_host_listings_count',
       'calculated_host_listings_count_entire_homes',
       'calculated_host_listings_count_private_rooms',
       'calculated_host_listings_count_shared_rooms', 'geometry',
       'index_right', 'LSOA11CD', 'LSOA11NM', 'Value', 'USUALRES', 'HHOLDRES',
       'COMESTRES', 'POPDEN', 'HHOLDS', 'AVHHOLDSZ', 'IMDScore', 'Borough',
       'Multiple Location Host'],
      dtype='object')
```

In [69]:
```
from collections import Counter
import re
from nltk.tokenize import word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.stem.porter import PorterStemmer
import string
from nltk.corpus import stopwords
from wordcloud import WordCloud, ImageColorGenerator
from textblob import TextBlob
```

In [70]:
```
import nltk
nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
```

Out[70]:
```
True
```

In [71]:
```
df = pd.read_csv('data/reviews.csv.gz', compression='gzip', header=0, sep=',', quotec
df_1 = pd.read_csv('data/listings.csv.gz', compression='gzip')
df_1 = df_1[df_1['neighbourhood_cleansed']=='Westminster']
id_1 = set(df['listing_id'])
id_2 = set(df_1['id'])
idlist = id_1&id_2
df = df[df['listing_id'].apply(lambda x : x in idlist)]
df.shape
```

Out[71]:
```
(156027, 6)
```

In [72]:
```
data = df.drop(['id','reviewer_name'],axis=1)

def remove_spaces(text):
    text=text.strip()
    while '  ' in text:
        text = text.replace('  ',' ')
    return text
```

```
contraction = {'cause':'because',
               'aint': 'am not',
               'aren\'t': 'are not'}

def mapping_replacer(x,dic):
    for words in dic.keys():
        if ' ' + words + ' ' in x:
            x=x.replace(' '+ words +' ' ,' '+dic[words]+' ' )
    return x

ps = PorterStemmer()
lem = WordNetLemmatizer()
def lexicon_normalization(text):
    words = word_tokenize(text)
    # 1- Stemming
    words_stem = [ps.stem(w) for w in words]
    # 2- Lemmatization
    words_lem = [lem.lemmatize(w) for w in words_stem]
    return words_lem

def clean_data(text):
    '''Make text lowercase, remove text in square brackets,remove links,remove punctu
    and remove words containing numbers.'''
    text = str(text).lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<\\*.*?>+', '', text)
    text = re.sub(f'[{re.escape(string.punctuation)}]', '', text)
    text = re.sub('\n', '', text)
    # text = re.sub('\w*\d\w*', '', text)
    text = re.sub('\'','', text)
    return text

def remove_stopword(text):
    stop_words = stopwords.words('english')
    stopwords_dict = Counter(stop_words)
    text = ' '.join([word for word in text.split() if word not in stopwords_dict])
    return text

useless_word = ['http','https','www','com', 'ev','u','ly','pic','would','br','locat',
def clean_useless_word(x):
    new = []
    for item in x:
        if item not in useless_word:
            new.append(item)
    return new
```

In [73]:
```
data['comments'] = data['comments'].map(lambda x: re.sub(r'\W+', ' ', str(x))).replac
data['comments']
```

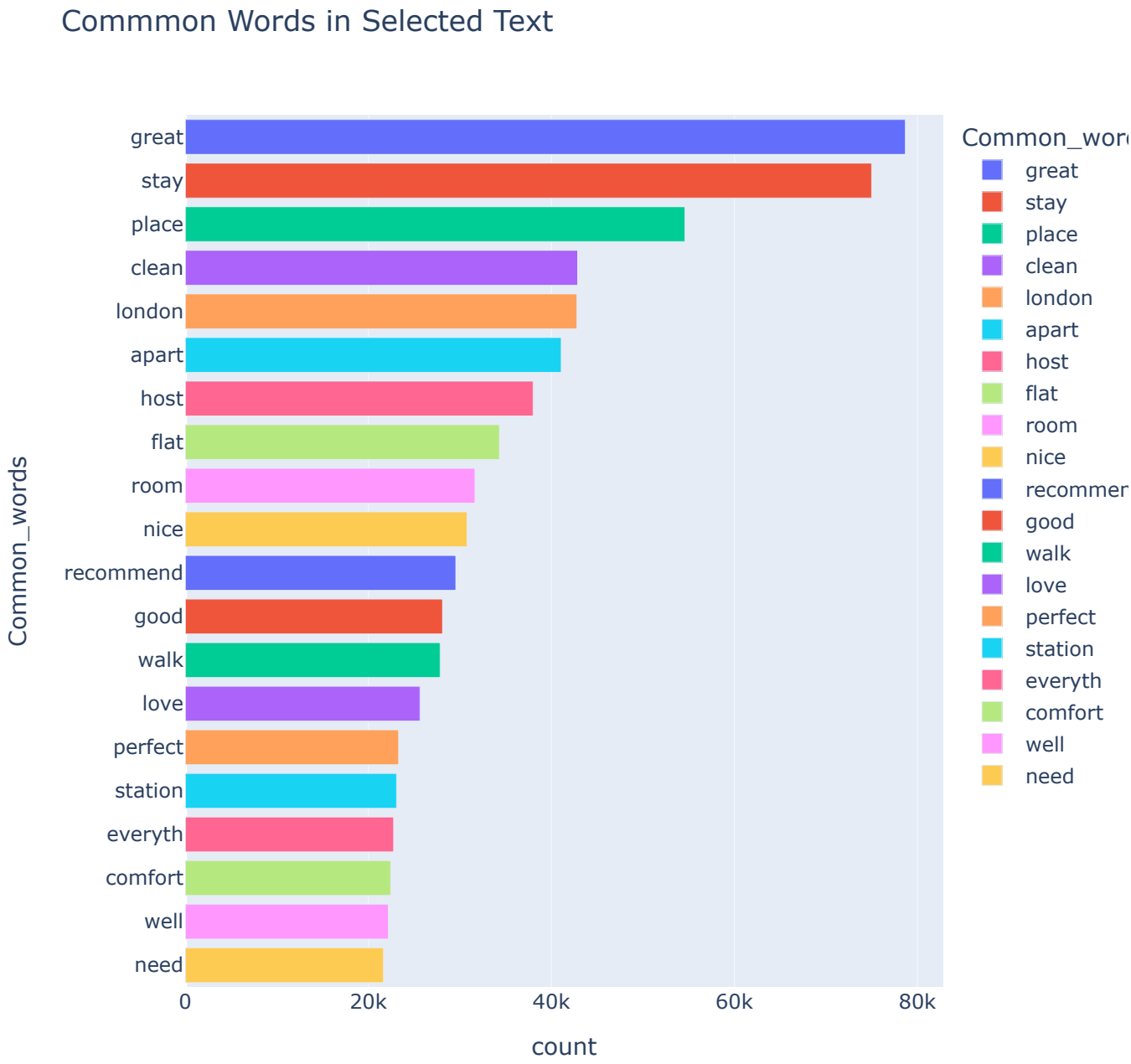
Out[73]:
```
19          [perfect, opposit, pimlico, tube, station, qui...
55          [hous, fantast, superb, quiet, yet, still, rig...
56          [zero, defeito, lugar, maravilhoso, completo, ...
58          [beauti, hous, tuck, away, hustl, bustl, victo...
59          [amaz, place, famili, headquart, visit, london...
                                  ...
1216207     [干净舒适, 适合家庭入住, 地段优越, 景点全部很多都可以步行到达, 节省了交通费, 房间...
1216208     [excellent, situat, au, plein, centr, londr, i...
1216209     [todo, perfecto, ubicación, comodidad, rachel,...
1216210     [tout, abord, même, si, n, avon, pa, eu, plais...
1216211                                        [rachel人很好]
Name: comments, Length: 156027, dtype: object
```

In [74]:
```
# Finding the most Common words in our Text
top = Counter([item for sublist in data['comments'] for item in sublist])
temp = pd.DataFrame(top.most_common(20))
temp.columns = ['Common_words','count']
temp.style.background_gradient(cmap='Blues')
```

Out[74]:

	Common_words	count
0	great	78677
1	stay	74988
2	place	54593
3	clean	42865
4	london	42787
5	apart	41071
6	host	37999
7	flat	34326
8	room	31647
9	nice	30790
10	recommend	29556
11	good	28105
12	walk	27836
13	love	25635
14	perfect	23301
15	station	23075
16	everyth	22755
17	comfort	22433
18	well	22184
19	need	21645

In [75]:
```
fig = px.bar(temp, x="count", y="Common_words", title='Commmon Words in Selected Text
             width=700, height=700,color='Common_words')
fig.show()
```



In [76]:

```
def get_reviews_sentiment(text):
    '''
    Utility function to classify sentiment of passed reviews
    using textblob's sentiment method
    '''
    # create TextBlob object of passed reviews text
    analysis = TextBlob(text)

    # set sentiment
    if analysis.sentiment.polarity > 0:
        return 'positive'
    elif analysis.sentiment.polarity == 0:
        return 'neutral'
    else:
        return 'negative'

data['sentiment']=data['comments'].apply(lambda x: get_reviews_sentiment(' '.join(x)))
data['sentiment']
```

Out[76]:
```
19          positive
55          positive
56          positive
58          positive
59           neutral
             ...
1216207     positive
1216208     positive
1216209      neutral
1216210      neutral
1216211      neutral
Name: sentiment, Length: 156027, dtype: object
```

In [77]:
```
Positive_sent = data[data['sentiment']=='positive']
Negative_sent = data[data['sentiment']=='negative']
Neutral_sent = data[data['sentiment']=='neutral']

print('Number of reviews with positive sentiment is ', Positive_sent['sentiment'].sha
print('Number of reviews with negative sentiment is ', Negative_sent['sentiment'].sha
print('Number of reviews with neutral sentiment is ', Neutral_sent['sentiment'].shape
```

```
Number of reviews with positive sentiment is  130578
Number of reviews with negative sentiment is  3544
Number of reviews with neutral sentiment is  21905
```

In [78]:
```
top = Counter([item for lst in Positive_sent['comments'] for item in lst])
temp_positive = pd.DataFrame(top.most_common(20))
temp_positive.columns = ['Common_words','count']
temp_positive.style.background_gradient(cmap='PuBu')
```

Out[78]:

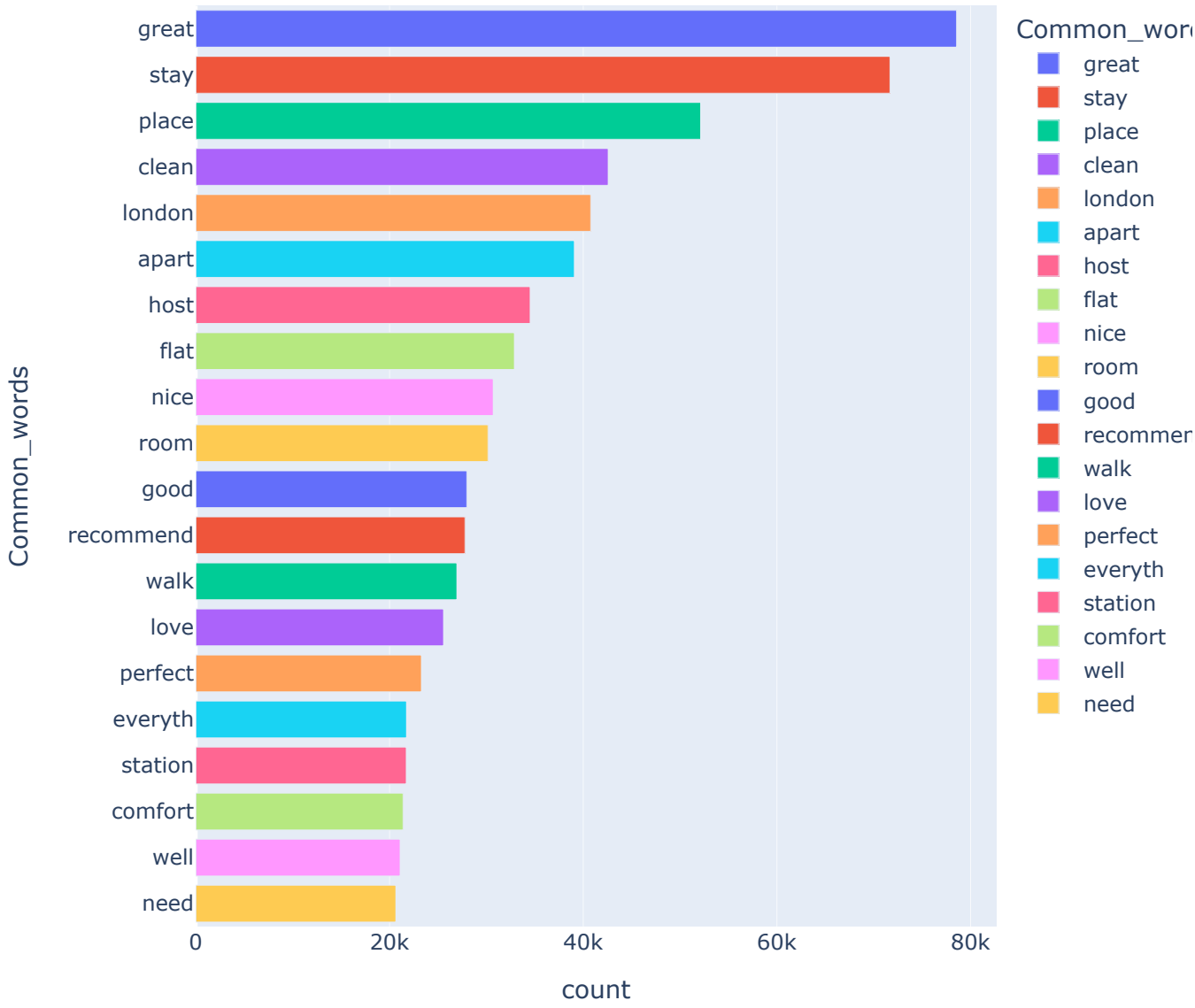
	Common_words	count
0	great	78553
1	stay	71671
2	place	52122
3	clean	42579
4	london	40771
5	apart	39081
6	host	34519
7	flat	32891
8	nice	30706
9	room	30165
10	good	27976
11	recommend	27788
12	walk	26954
13	love	25577
14	perfect	23289
15	everyth	21752
16	station	21726
17	comfort	21392
18	well	21094
19	need	20651

In [79]:
```
top = Counter([item for sublist in Positive_sent['comments'] for item in sublist])
temp_positive = pd.DataFrame(top.most_common(20))
temp_positive.columns = ['Common_words','count']
temp_positive['Common_words'] = temp_positive['Common_words'].map(lambda x: re.sub(r'
temp_positive['Common_words'] = temp_positive['Common_words'].replace(r'\W+', '', reg
temp_positive['Common_words'] = temp_positive['Common_words'].apply(lambda x:remove_s
temp_positive=temp_positive[~temp_positive['Common_words'].isin(['s','gre','“',' * ']
mask1 = temp_positive.Common_words.str.contains('[a-zA-Z]')
mask2 = temp_positive.Common_words.notna()
temp_positive = temp_positive[mask1 | mask2]
temp_positive.Common_words =  temp_positive.Common_words.str.replace(r"\s+", "").repl
temp_positive=temp_positive.dropna()
temp_positive.style.background_gradient(cmap='Greens')
```

Out[79]:

	Common_words	count
0	great	78553
1	stay	71671
2	place	52122
3	clean	42579
4	london	40771
5	apart	39081
6	host	34519
7	flat	32891
8	nice	30706
9	room	30165
10	good	27976
11	recommend	27788
12	walk	26954
13	love	25577
14	perfect	23289
15	everyth	21752
16	station	21726
17	comfort	21392
18	well	21094
19	need	20651

In [80]:
```
fig = px.bar(temp_positive, x="count", y="Common_words", title='Most Commmon Words in
             width=700, height=700,color='Common_words')
fig.show()
```



In [81]:
```
#MosT common negative words
top = Counter([item for sublist in Negative_sent['comments'] for item in sublist])
temp_negative = pd.DataFrame(top.most_common(20))
temp_negative = temp_negative.iloc[1:,:]
temp_negative.columns = ['Common_words','count']
temp_negative.style.background_gradient(cmap='Reds')
```

Out[81]:

	Common_words	count
1	stay	1401
2	room	945
3	place	862
4	host	845
5	apart	843
6	london	800
7	small	701
8	check	664
9	recommend	568
10	airbnb	566
11	bad	529
12	und	525
13	well	498
14	bed	486
15	time	482
16	need	474
17	night	472
18	one	454
19	get	450

In [82]:

```
temp_negative.Common_words =  temp_negative.Common_words.replace("", np.nan)
temp_negative = temp_negative.dropna(subset=['Common_words'])

temp_negative.style.background_gradient(cmap='Reds')
```

Out[82]:

	Common_words	count
1	stay	1401
2	room	945
3	place	862
4	host	845
5	apart	843
6	london	800
7	small	701
8	check	664
9	recommend	568
10	airbnb	566
11	bad	529
12	und	525
13	well	498
14	bed	486
15	time	482
16	need	474
17	night	472
18	one	454
19	get	450

In [83]:

```
fig = px.treemap(temp_negative, path=['Common_words'], values='count',title='Tree Of
fig.show()
```

Tree Of Most Common Words in Negative Comments

In [84]:
```
def plot_wordcloud(text, mask=None, max_words=100, max_font_size=100, figure_size=(10
                   title = None, title_size=40, image_color=False):

    wordcloud = WordCloud(background_color=color,
                    max_words = max_words,
                    font_path='chinese.msyh.ttf',
                    max_font_size = max_font_size,
                    random_state = 42,
                    width=400,
                    height=200,
                    mask = mask)
    wordcloud.generate(str(text))

    plt.figure(figsize=figure_size)
    if image_color:
        image_colors = ImageColorGenerator(mask)
        plt.imshow(wordcloud.recolor(color_func=image_colors), interpolation="bilinea
        plt.title(title, fontdict={'size': title_size,
                                   'verticalalignment': 'bottom'})
    else:
        plt.imshow(wordcloud)
        plt.title(title, fontdict={'size': title_size, 'color': 'black',
                                   'verticalalignment': 'bottom'})
    plt.axis('off')
    plt.tight_layout()
```

In [85]:
```
Neutral_sent
```

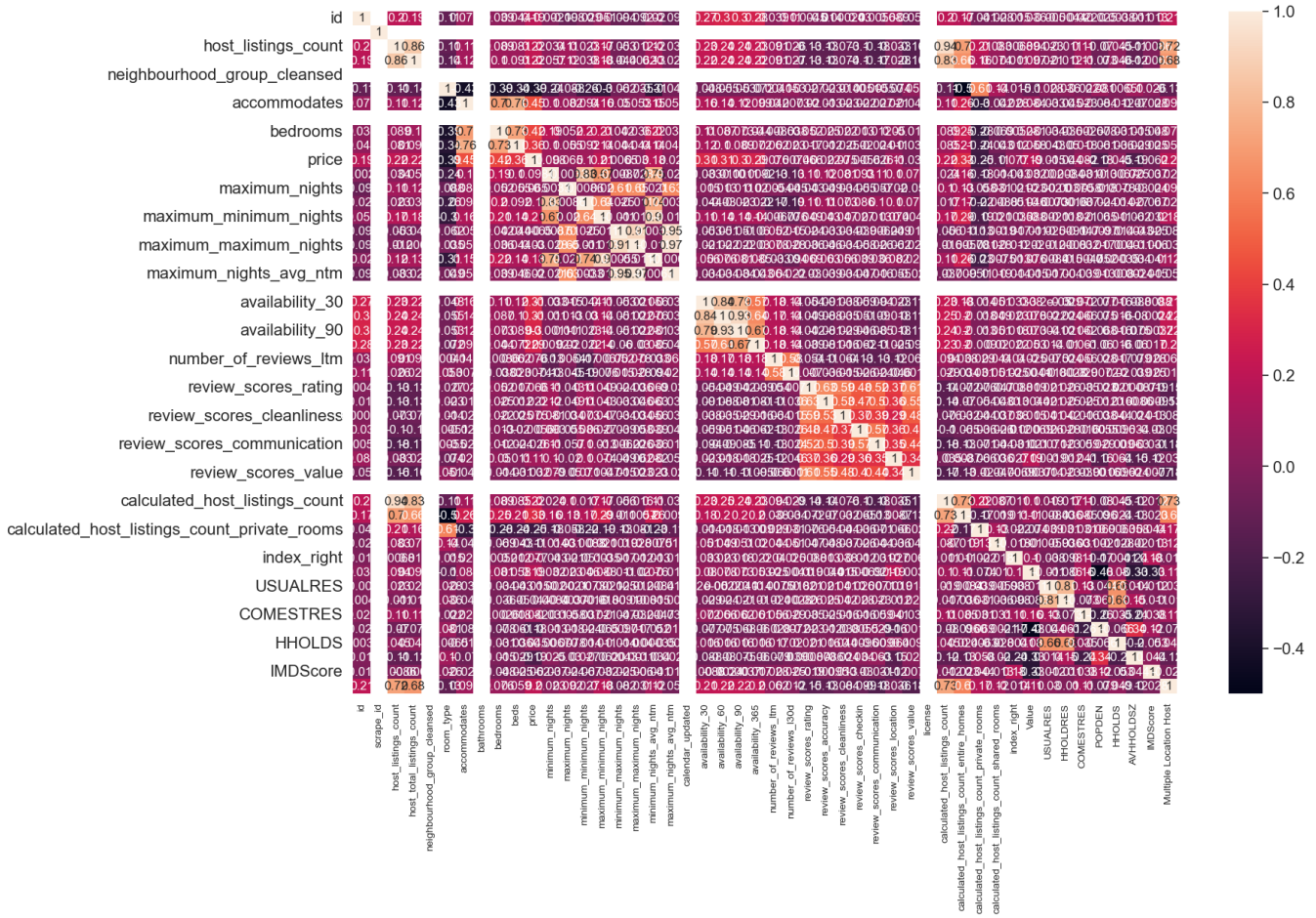
Out[85]:

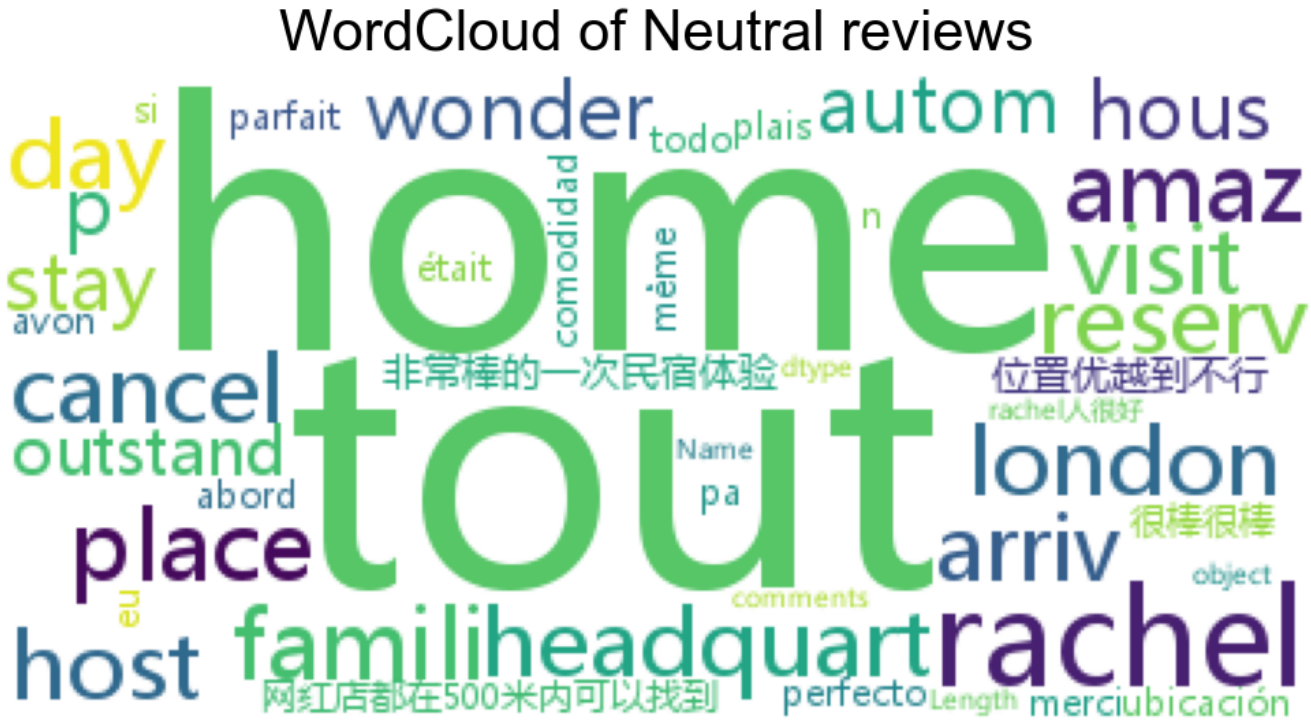
	listing_id	date	reviewer_id	comments	sentiment
59	21526298	2022-08-15	180350733	[amaz, place, famili, headquart, visit, london...	neutral
66	21526298	2018-04-15	93354715	[host, cancel, reserv, 7, day, arriv, autom, p...	neutral
70	21526298	2021-06-06	133405220	[home, home, stay]	neutral
74	21526298	2021-04-14	133405220	[wonder, hous, outstand]	neutral
92	21526298	2022-07-21	215666012	[]	neutral
...	...	...	...	...	...
1216205	14832630	2018-10-03	59897810	[非常棒的一次民宿体验, 位置优越到不行, 网红店都在500米内可以找到, 很棒很棒]	neutral
1216206	14832630	2018-01-01	64551062	[tout, était, parfait, merci, rachel]	neutral
1216209	14832630	2017-05-20	119296298	[todo, perfecto, ubicación, comodidad, rachel,...	neutral
1216210	14832630	2017-04-18	16394435	[tout, abord, même, si, n, avon, pa, eu, plais...	neutral
1216211	14832630	2016-09-23	65608870	[rachel人很好]	neutral

21905 rows × 5 columns

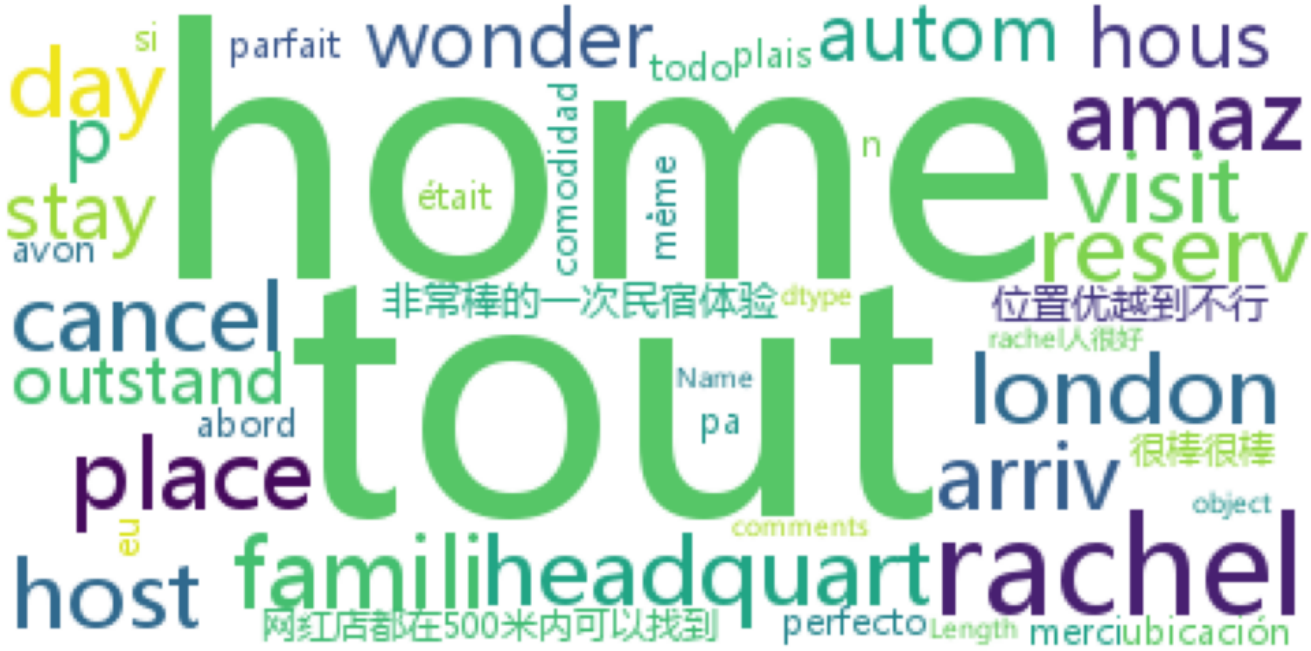
In [87]:

```
plot_wordcloud(Neutral_sent.comments,color='white',
               max_font_size=100,title_size=30,title="WordCloud of Neutral reviews")
plt.show()
```





WordCloud of Neutral reviews



In [88]:
```
plot_wordcloud(Positive_sent.comments,
               title="Word Cloud Of Positive reviews",title_size=30)
plt.show()
```

In [89]:
```
plot_wordcloud(Negative_sent.comments,
               title="Word Cloud of Negative reviews",color='white',title_size=30)
plt.show()
```

Credits!

Contributors:

The following individual(s) have contributed to these teaching materials: Yijing Li (yijing.li@kcl.ac.uk).

License