Data sources: https://www.kaggle.com/dgomonov/new-york-city-airbnb-open-data

data descriptions:

- id: listing id
- review_scores_location: 0-5 stars converted into a 0-10 scale
- name: listing name
- host_id: host id
- host_name: host name
- neighbourhood_group: NYC borough
- neighbourhood: NYC neighborhood
- latitude: listing latitude
- longitude: listing longitude
- room_type: type of listing (Entire home/apt, Private room, Shared room)
- price: listing price
- minimum_nights: required minimum nights stay
- number_of_reviews: total number of reviews
- last_review: date of last review
- reviews per month: average number of reviews per month
- calculated_host_listings_count: total number of listings for this host
- availability_365: number of days listing is available out of 365

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import numpy as np
import matplotlib.image as mpimg
import geopandas
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
```

In [2]:
```
ori_data=pd.read_csv("AB_NYC_2019.csv")
```

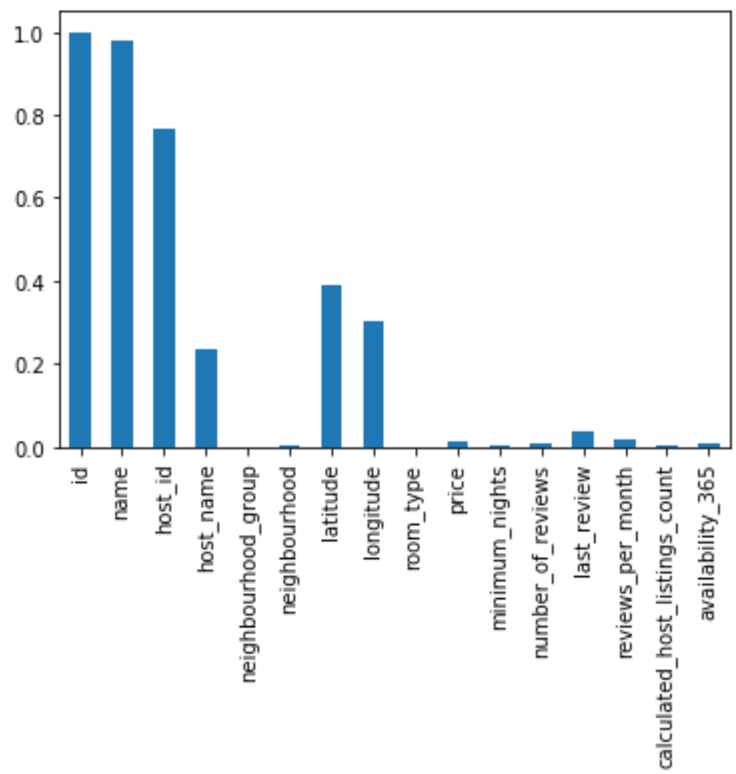
In [3]:
```
print(f'total rows:{ori_data.shape[0]}')
print(f'total columns:{ori_data.shape[1]}')
print(f'column name:{ori_data.columns.values}')
```

```
total rows:48895
total columns:16
column name:['id' 'name' 'host_id' 'host_name' 'neighbourhood_group' 'nei
ghbourhood'
 'latitude' 'longitude' 'room_type' 'price' 'minimum_nights'
 'number_of_reviews' 'last_review' 'reviews_per_month'
 'calculated_host_listings_count' 'availability_365']
```

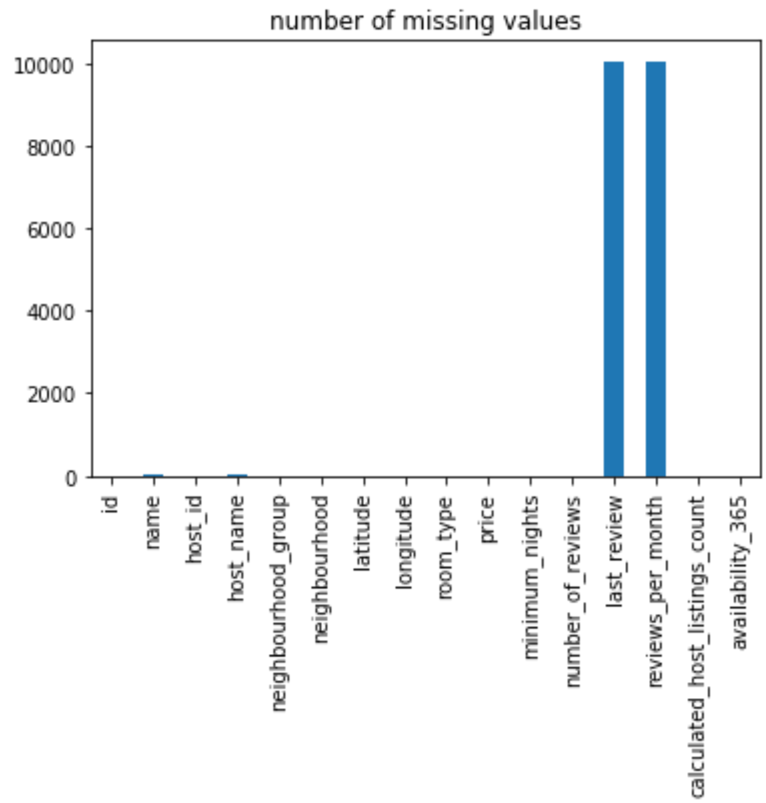
In [4]:
```
## which columns has duplicated values
ori_data.apply(lambda x:x.unique().shape[0],axis=0)
```

Out[4]:
```
id                                48895
name                              47906
host_id                           37457
host_name                         11453
neighbourhood_group                   5
neighbourhood                       221
latitude                          19048
longitude                         14718
room_type                             3
price                               674
minimum_nights                      109
number_of_reviews                   394
last_review                        1765
reviews_per_month                   938
calculated_host_listings_count       47
availability_365                    366
dtype: int64
```

In [5]:
```
## The lower y ,the highly repeat for x (may be categorical variable)
## With high y,x may be continuous variable or index
(ori_data.apply(lambda x:x.unique().shape[0],axis=0)/ori_data.shape[0]).plc
plt.show()
```

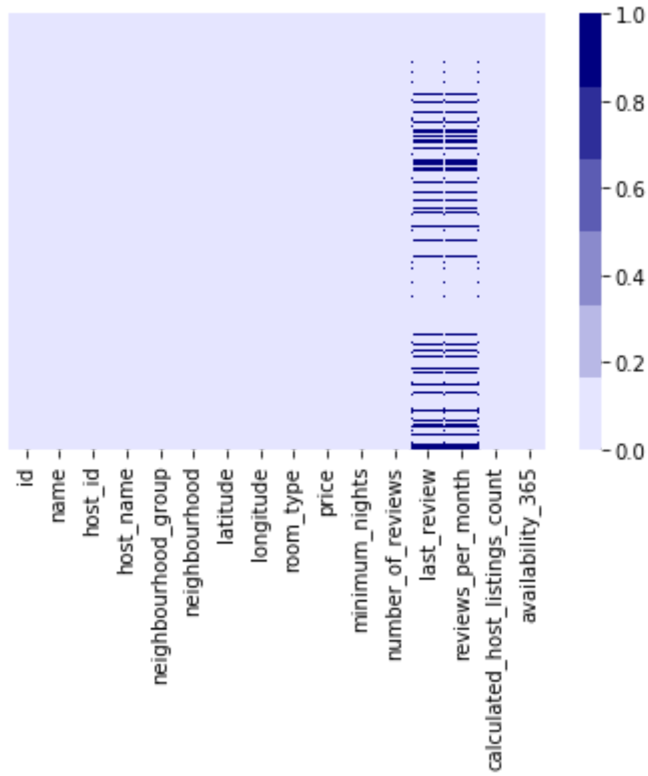


In [6]:
```
##which column contains NaN
ori_data.isna().sum().plot(kind='bar',title='number of missing values')
plt.show()
ori_data.isna().sum()
```



Out[6]:
```
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
dtype: int64
```

In [7]:
```
cmap=sns.light_palette('navy',reverse=False)
sns.heatmap(ori_data.isnull().astype(np.int8),yticklabels=False,cmap=cmap)
plt.show()
```



The goal is to predict the price by the geographical and accommodation_related features. So, we should not take into account host-name and place-name. However , we also can find out that there are many missing data in 'last_review' and 'reviews_per_month'. If the place is a new post ,it would not have any review before. Thus, we can add a feature determining whether it is a new post or not.

In [8]:
```
ori_data.describe()[['price','minimum_nights','number_of_reviews','reviews_
```

Out[8]:

	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_host_listing
count	48895.000000	48895.000000	48895.000000	38843.000000	4889
mean	152.720687	7.029962	23.274466	1.373221	
std	240.154170	20.510550	44.550582	1.680442	3
min	0.000000	1.000000	0.000000	0.010000	
25%	69.000000	1.000000	1.000000	0.190000	
50%	106.000000	3.000000	5.000000	0.720000	
75%	175.000000	5.000000	24.000000	2.020000	
max	10000.000000	1250.000000	629.000000	58.500000	32

In [9]:
```
for col in ['price','minimum_nights','number_of_reviews','reviews_per_month
    print(f'{col}:{ori_data[col].quantile(0.95) }')
```

```
price:355.0
minimum_nights:30.0
number_of_reviews:114.0
reviews_per_month:4.64
calculated_host_listings_count:15.0
availability_365:359.0
```

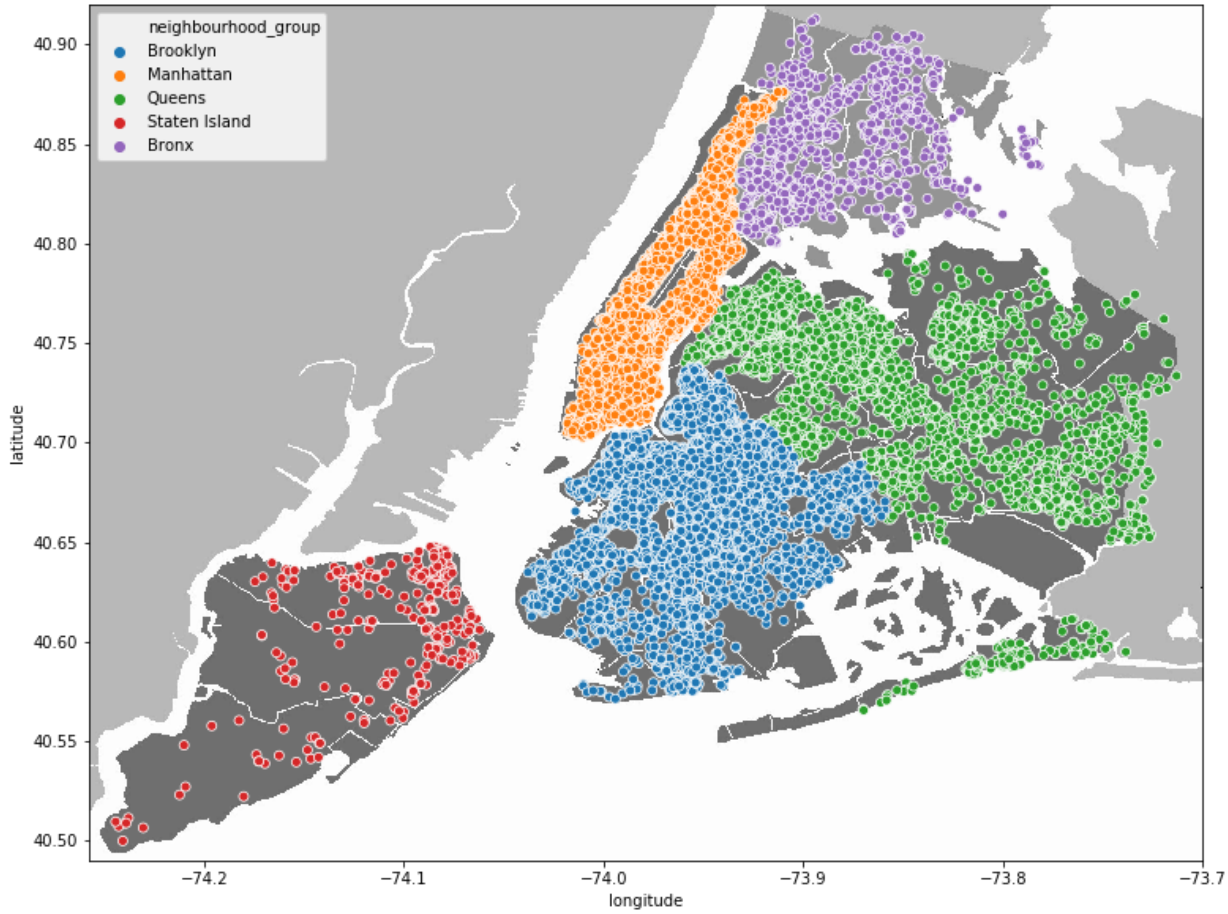
The outliers in "price" and "minimum_nights" are unreasonable.Thus, we can filter out top 5% outlier in this two columns.

In [10]:
```
price_filter=(ori_data['price']<ori_data['price'].quantile(0.95))&(ori_data
minimum_nights_filter=ori_data['minimum_nights']<ori_data['minimum_nights']
filter_data=ori_data[price_filter&minimum_nights_filter].copy()
```

In [11]:
```
print(f'original_data:{ori_data.shape[0]}')
print(f'filtered_data:{filter_data.shape[0]}')
```

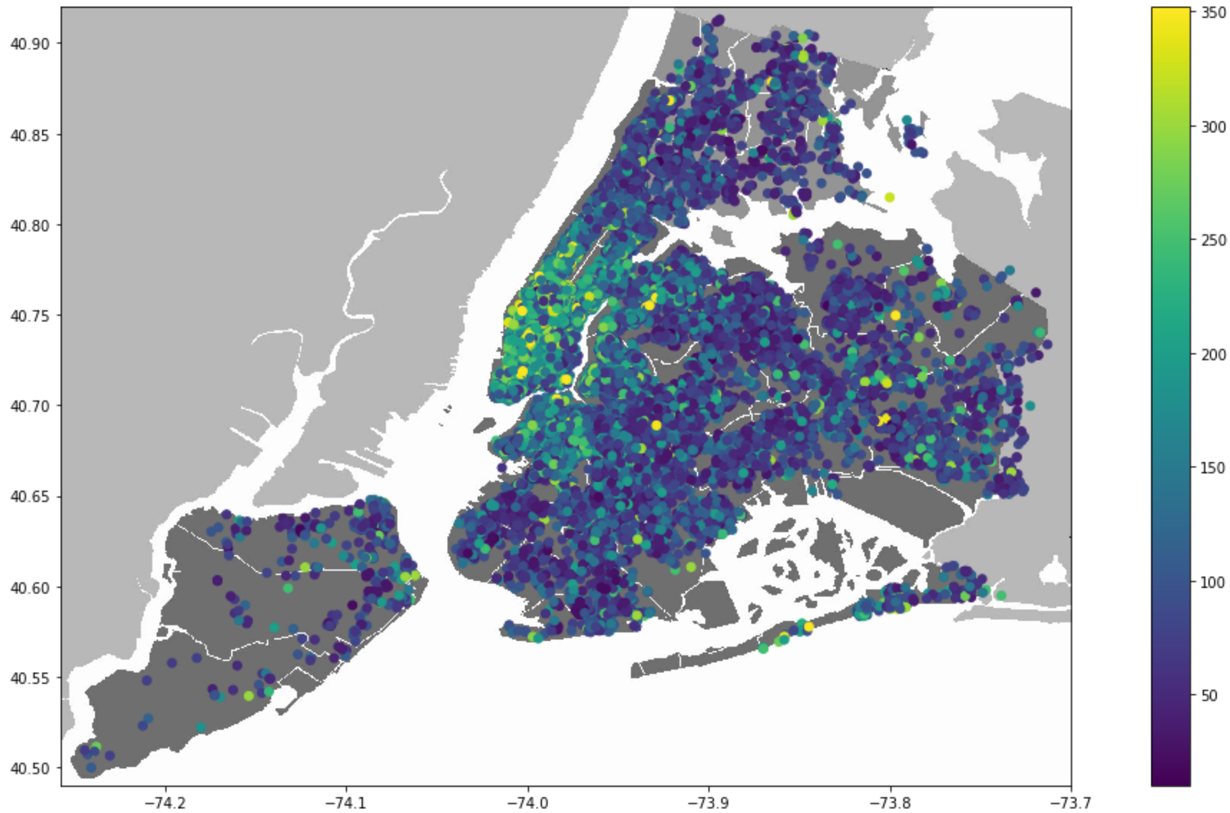
```
original_data:48895
filtered_data:42201
```

In [12]:
```
plt.figure(figsize=(20,10))
nyc_img = plt.imread("./New_York_City.png",0)
plt.imshow(nyc_img,zorder=0,extent=[-74.258, -73.7, 40.49,40.92])
ax=plt.gca()
sns.scatterplot(x=filter_data.longitude,y=filter_data.latitude,hue=ori_data
plt.show()
```



We compare price based on location on the map. Manhattan gathers most high-priced places.

In [13]:
```
plt.figure(figsize=(20,10))
nyc_img = plt.imread("./New_York_City.png",0)
ax=plt.gca()
##filter outlier
plt.imshow(nyc_img,zorder=0,extent=[-74.258, -73.7, 40.49,40.92])
sc=plt.scatter(filter_data.longitude, filter_data.latitude, c=filter_data.p
plt.colorbar(sc)
plt.show()
```

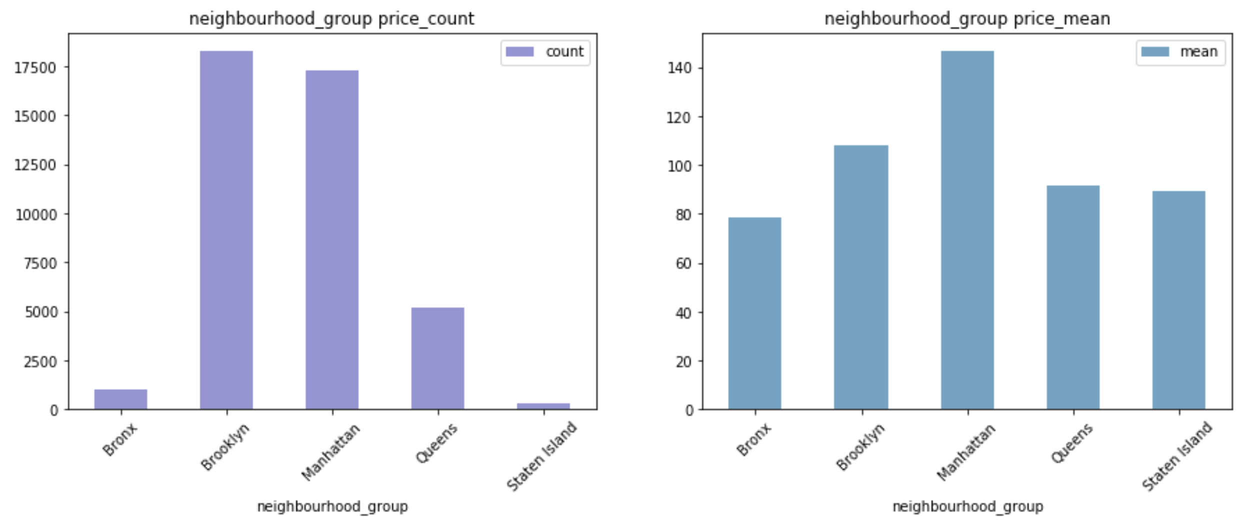


In [14]:
```
##only 5 unique values in neighbourhood_group
filter_data.neighbourhood_group.unique()
```

Out[14]:
```
array(['Brooklyn', 'Manhattan', 'Queens', 'Staten Island', 'Bronx'],
      dtype=object)
```

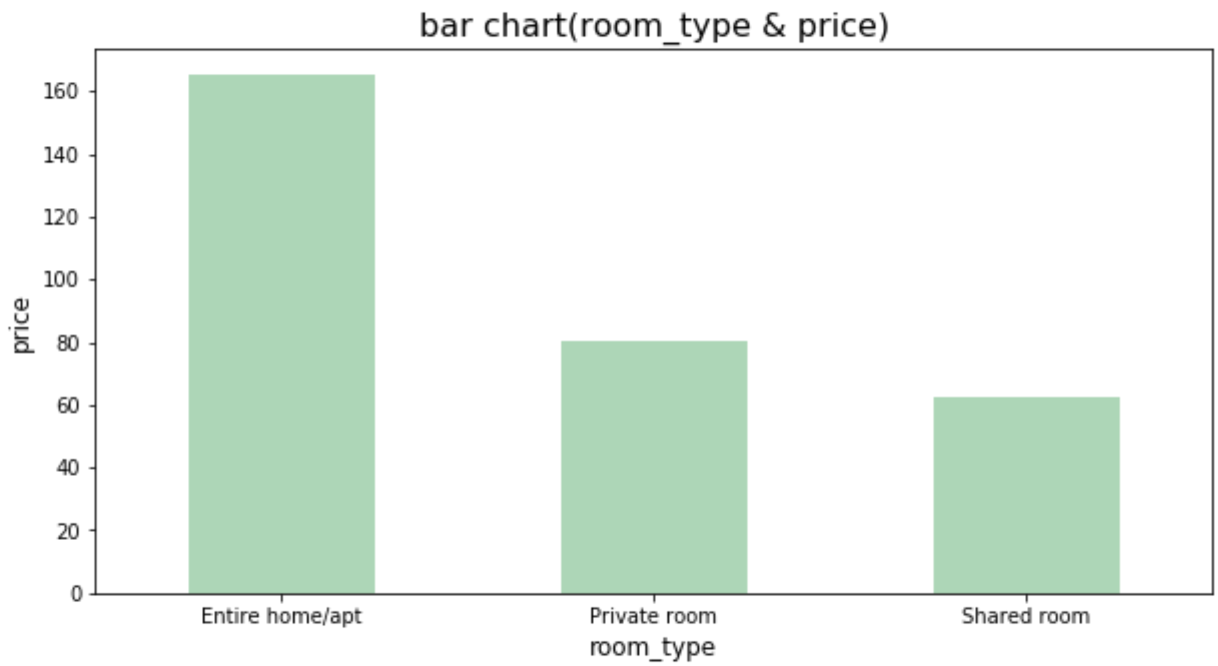
In [15]:
```
neighbourhood_group=filter_data.groupby("neighbourhood_group")['price'].des
fig, axes = plt.subplots(nrows=1, ncols=2,figsize=(15,5))
neighbourhood_group.plot(x="neighbourhood_group", y=["count"], kind="bar",r
neighbourhood_group.plot(x="neighbourhood_group", y=["mean"], kind="bar",ro

plt.show()
```



- More than 17000 posts in Manhattan and Brooklyn.
- The averge price in Manhattan is most expensive and less in Bronx.

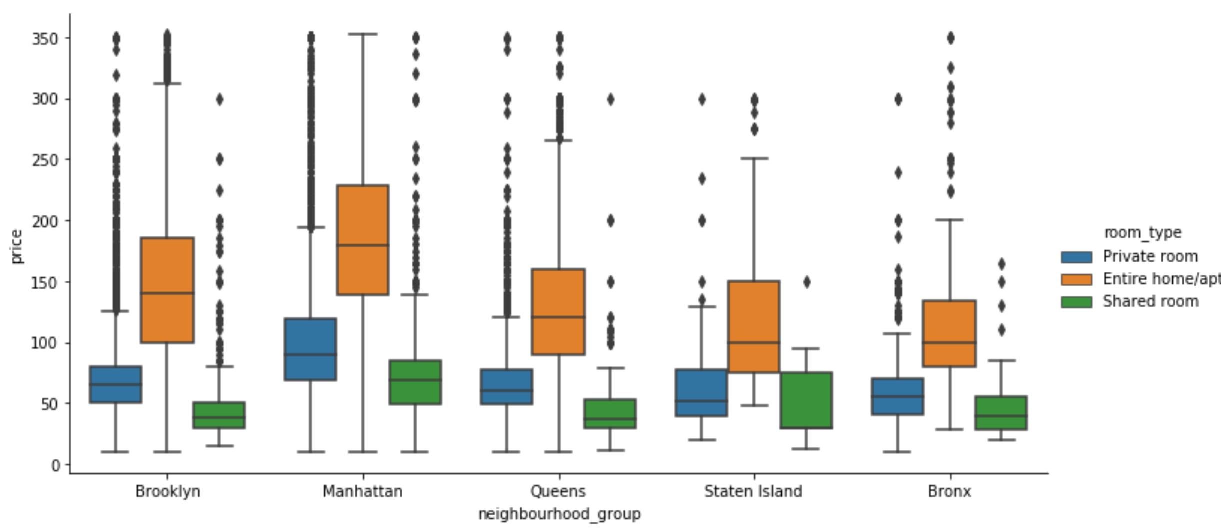
In [16]:
```
plt.figure(figsize=(10,5))
filter_data.groupby('room_type')['price'].mean().plot( kind="bar",rot=0,col
plt.xlabel('room_type',fontsize=12)
plt.ylabel('price',fontsize=12)
plt.title("bar chart(room_type & price)",fontsize=16)
plt.show()
```



- The averge price for Entire home/apt is higher than Private room and Shared room.

In [17]:
```
plt.figure(figsize=(10,10))
sns.catplot(x='neighbourhood_group',y='price',data=filter_data,hue='room_ty
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```

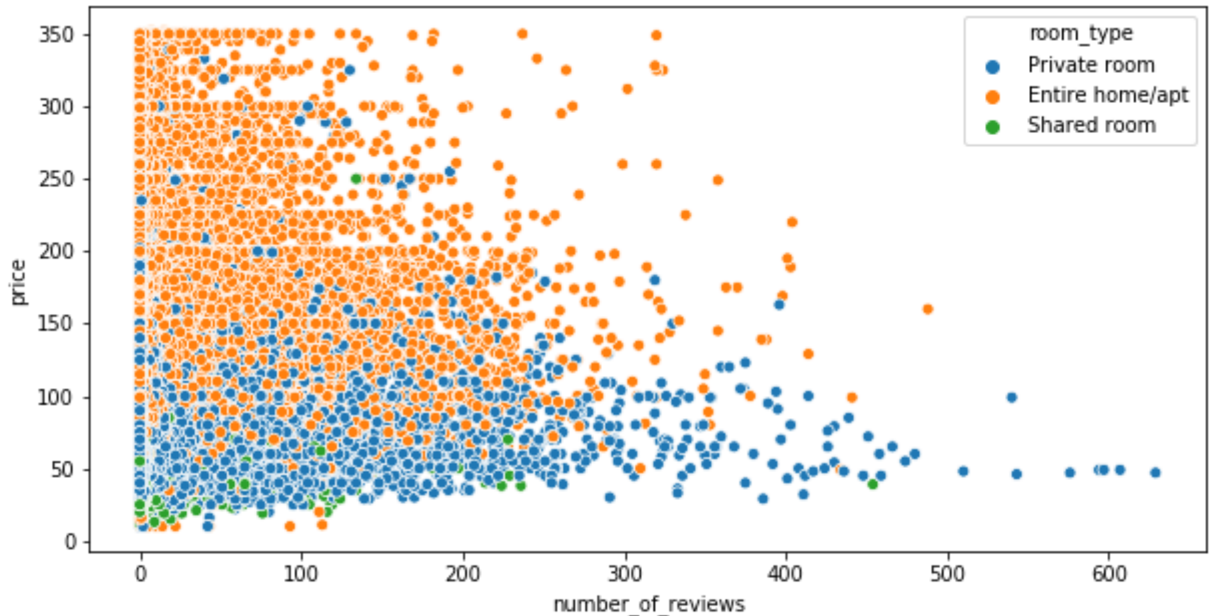


- Most area in New York, the average price of Shared room is far lower than other room_type. However, the price in Staten Island is close to Private room.

In [18]:
```
#number_of_reviews*reviews_per_month=post_month(how long did the place be p
filter_data['posted_month']=round(filter_data['number_of_reviews']/filter_d
```

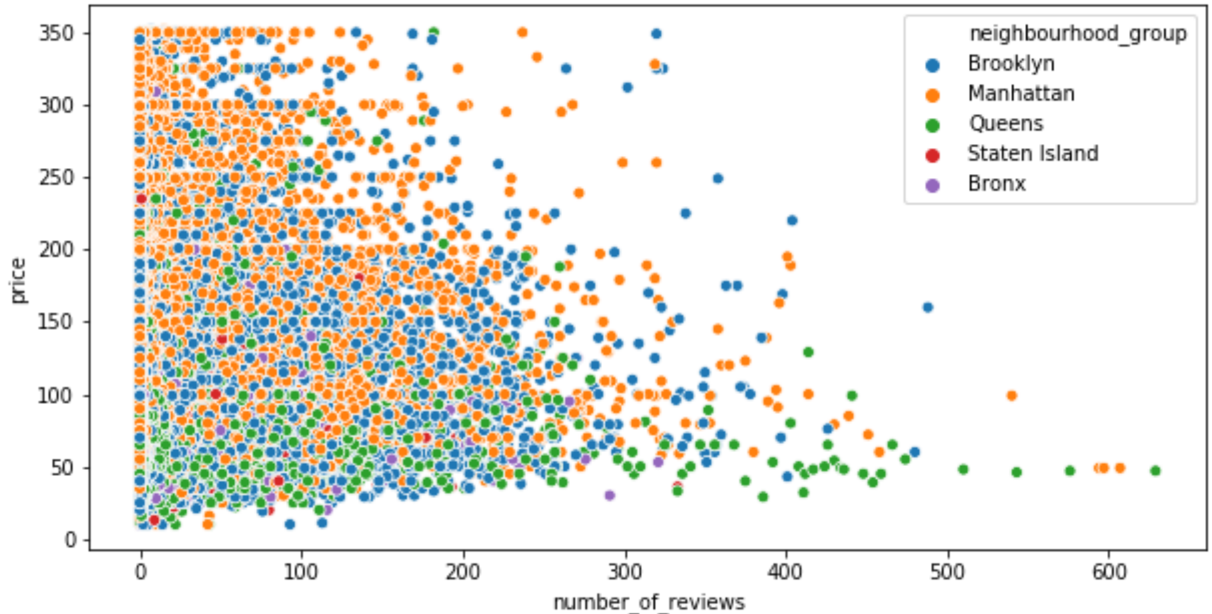
In [19]:
```
filter_data['is_New']=filter_data['reviews_per_month'].apply(lambda x:1 if
```

In [20]:
```
plt.figure(figsize=(10,5))
sns.scatterplot(x=filter_data.number_of_reviews,y=filter_data.price,hue=fil
plt.show()
```



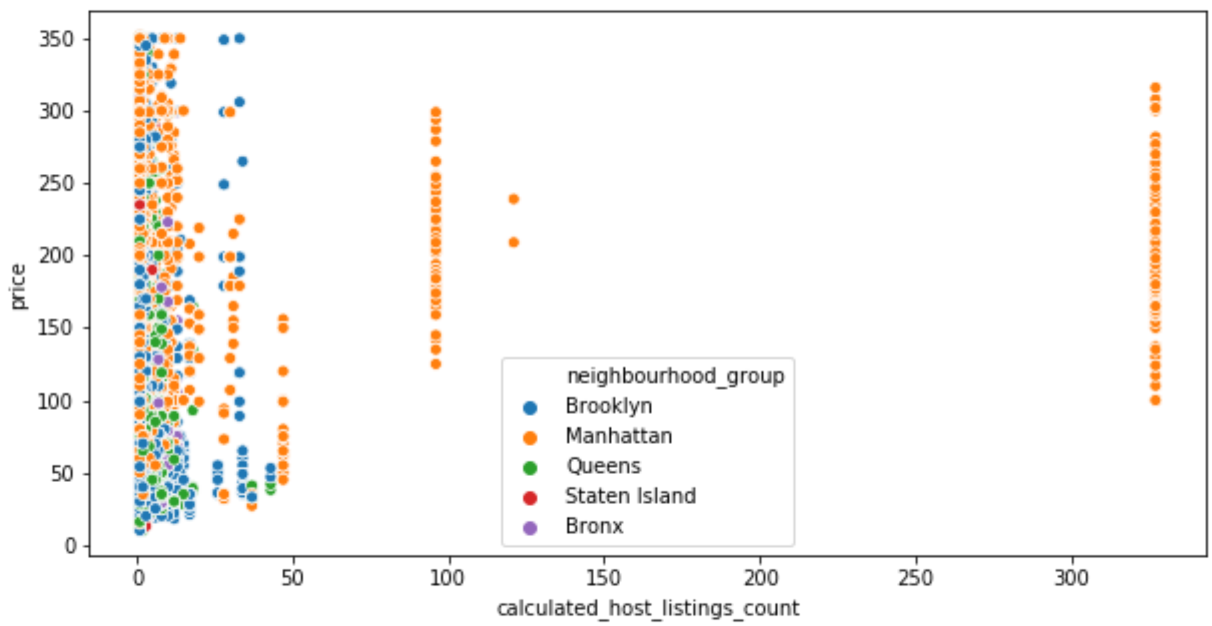
- Reviews more than 400 are gather in Private room.
- Though the number of places in Queens is not highest but several places in Queens had more than 400 reviews.

In [21]:
```
plt.figure(figsize=(10,5))
sns.scatterplot(x=filter_data.number_of_reviews,y=filter_data.price,hue=fil
plt.show()
```



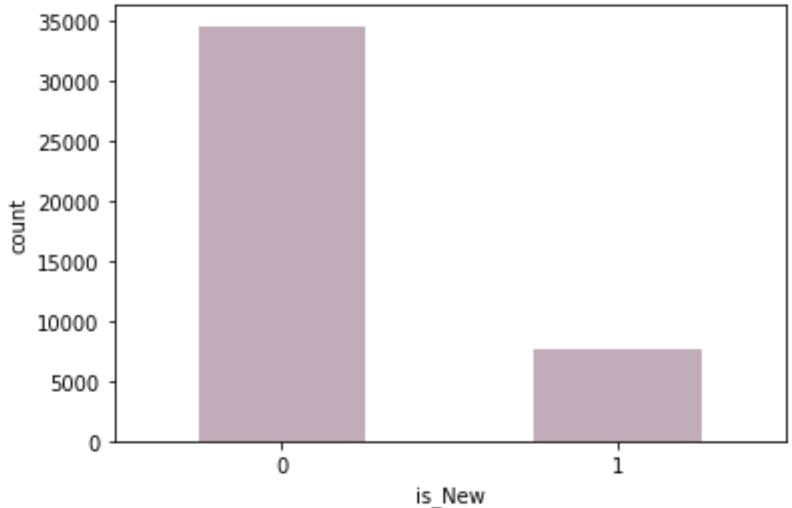
- Most of hosts had less than 50 host listings and few hosts in Manhattan had more than 50 host listings.

In [22]:
```
plt.figure(figsize=(10,5))
sns.scatterplot(x=filter_data.calculated_host_listings_count,y=filter_data.
plt.show()
```



In [112]:
```
filter_data.groupby('is_New')['price'].count().plot( kind="bar",rot=0,color
plt.ylabel("count")
```

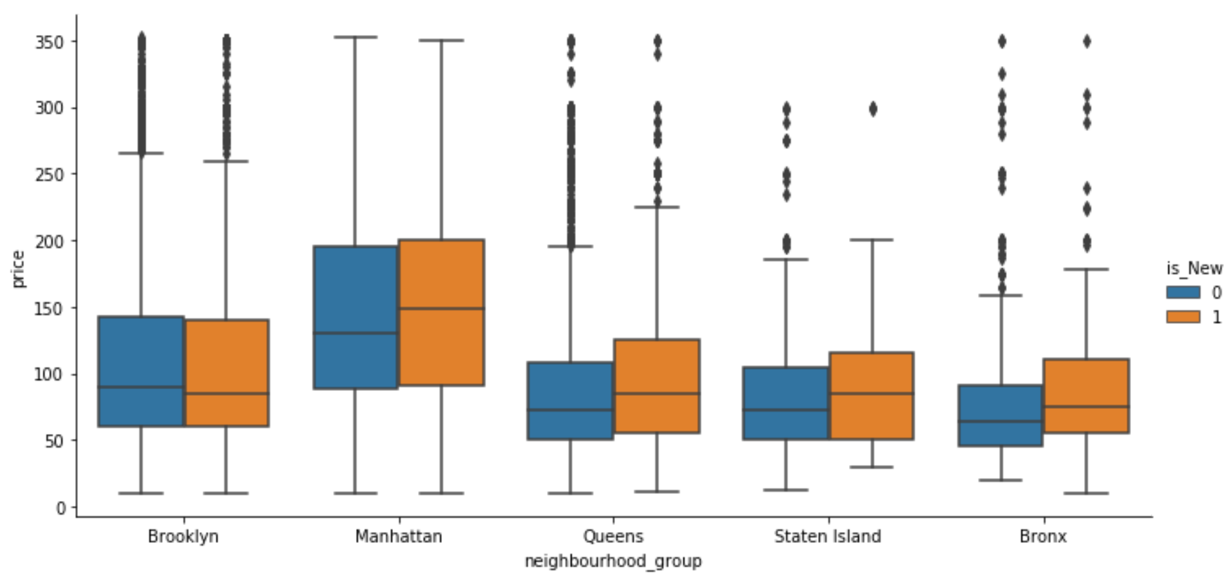
Out[112]: Text(0, 0.5, 'count')



In Brooklyn ,the average price of new-posted places is less than that of old places but in contrast with other areas.

In [24]:
```
plt.figure(figsize=(10,10))
sns.catplot(x='neighbourhood_group',y='price',data=filter_data,hue='is_New'
plt.show()
```

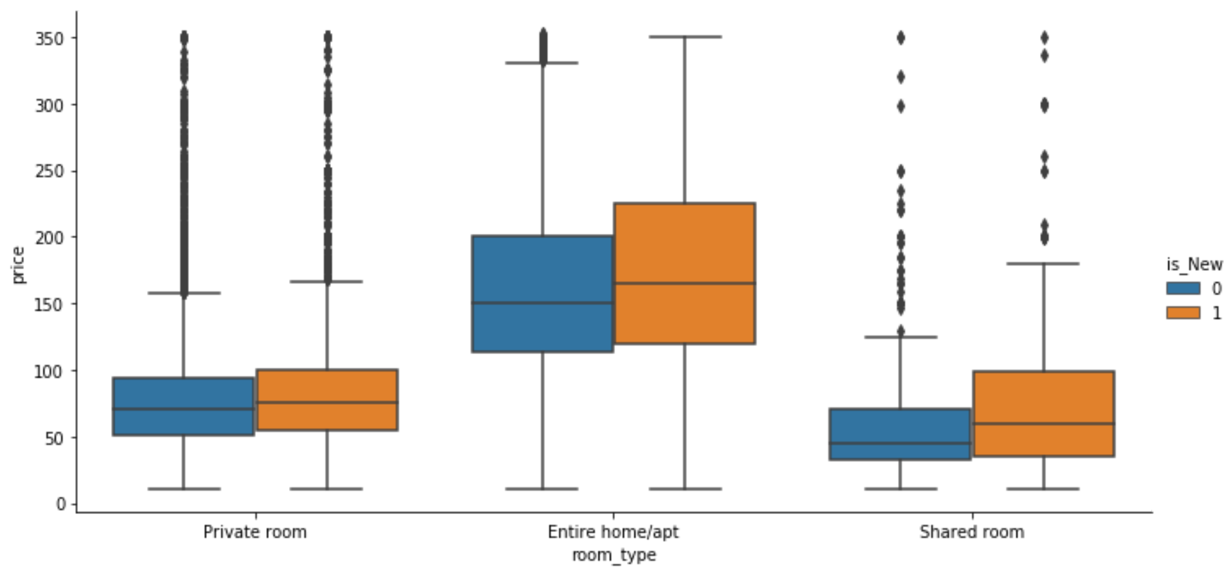
<Figure size 720x720 with 0 Axes>



- For room-type, we can see the average price for all type of new-posted places had higher price than before.

In [25]:
```
plt.figure(figsize=(10,10))
sns.catplot(x='room_type',y='price',data=filter_data,hue='is_New', kind="bc
plt.show()
```

<Figure size 720x720 with 0 Axes>

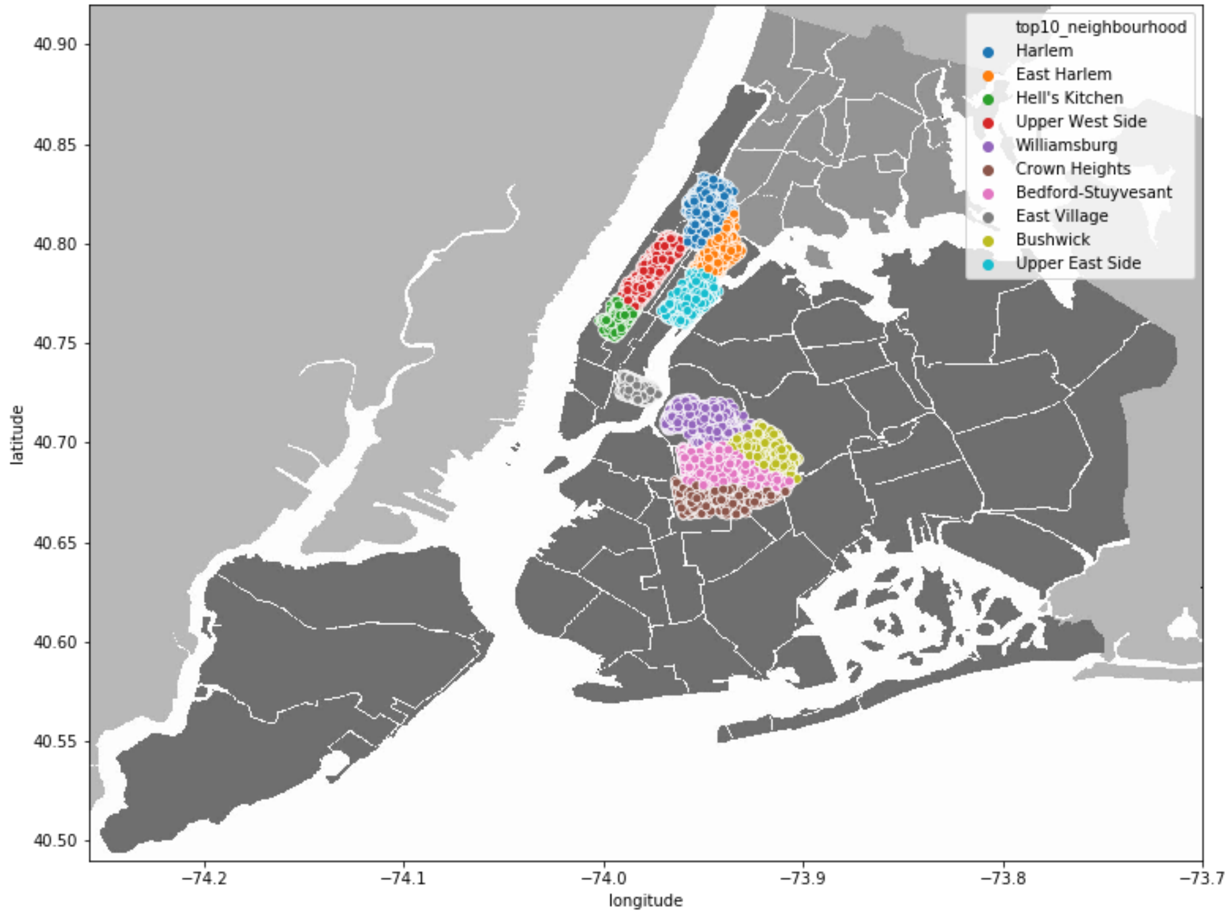


the list of top 10 neighbourhood in NYC

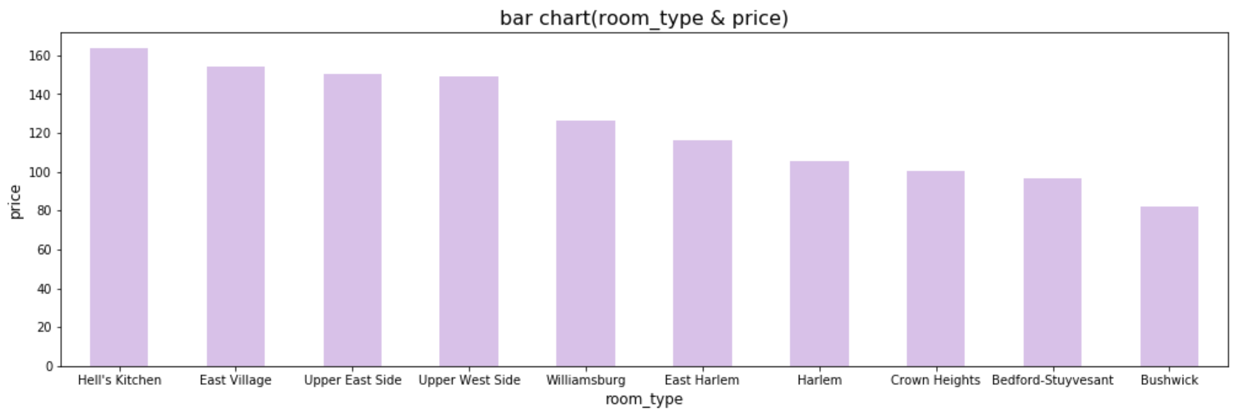
In [26]:
```
#neighbourhood value_counts
top10_list=filter_data.neighbourhood.value_counts()[:10].index
def top10(x):
    if(x in top10_list):
        return x
filter_data['top10_neighbourhood']=filter_data['neighbourhood'].apply(top10
```

- Top-10-favorited neighbourhood are gathered in Manhattan and Brooklyn.

In [27]:
```
plt.figure(figsize=(20,10))
nyc_img = plt.imread("./New_York_City.png",0)
plt.imshow(nyc_img,zorder=0,extent=[-74.258, -73.7, 40.49,40.92])
ax=plt.gca()
sns.scatterplot(x=filter_data.longitude,y=filter_data.latitude,hue=filter_d
plt.show()
```



In [28]:
```
plt.figure(figsize=(17,5))
filter_data.groupby('top10_neighbourhood')['price'].mean().sort_values(asce
plt.xlabel('room_type',fontsize=12)
plt.ylabel('price',fontsize=12)
plt.title("bar chart(room_type & price)",fontsize=16)
plt.show()
```



Look into "top10_neighbourhood", the chart above shows the average price of places near by most popular neighbourhood. The top-4 leaders are all in Manhattan.

In [29]:
```
filter_data=filter_data.drop(columns=['neighbourhood','id','name','host_id'
```

In [30]:
```
##fillna with zero
filter_data=filter_data.fillna('0')
filter_data=filter_data[['neighbourhood_group', 'latitude', 'longitude', 'r
       'minimum_nights', 'number_of_reviews', 'reviews_per_month',
       'calculated_host_listings_count', 'availability_365', 'posted_month'
       'is_New', 'top10_neighbourhood','price']]
```

In [31]:
```
backup=filter_data.copy()
#filter_data=backup.copy()
```

In [32]:
```
ori_price=backup[['price']]
```

In [33]:
```
filter_data.columns
```

Out[33]:
```
Index(['neighbourhood_group', 'latitude', 'longitude', 'room_type',
       'minimum_nights', 'number_of_reviews', 'reviews_per_month',
       'calculated_host_listings_count', 'availability_365', 'posted_mont
h',
       'is_New', 'top10_neighbourhood', 'price'],
      dtype='object')
```

In [34]:
```
##label encoding for categorical features
##StandardScaler for continuous features
continous_columns=['latitude', 'longitude', 'minimum_nights', 'number_of_re
       'reviews_per_month', 'calculated_host_listings_count','availability_
categorical_columns=['neighbourhood_group','room_type','is_New', 'top10_nei
labelencoder = LabelEncoder()
for col in categorical_columns:
    filter_data[col] = labelencoder.fit_transform(filter_data[col])
```

In [35]:
```
scaler = StandardScaler()
scaler.fit(filter_data[continous_columns])
filter_data[continous_columns] = scaler.transform(filter_data[continous_col
```

In [36]:
```
##train_test_split
y=filter_data[['price']]
X=filter_data.drop(columns=['price'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

In [37]:
```
regr = RandomForestRegressor()
regr.fit(X_train, y_train)
y_predict=regr.predict(X_test)
mse = np.mean((regr.predict(X_test)-y_test['price']) ** 2)
r_squared=regr.score(X_test, y_test)
adj_r_squared = r_squared - (1 - r_squared) * (X_test.shape[1] / (X_test.sh
print(f"MSE:{mse}")
print(f"r_squared:{r_squared}")
print(f"adj_r_squared:{adj_r_squared}")
```

```
/usr/local/lib/python3.7/site-packages/sklearn/ensemble/forest.py:245: Fu
tureWarning: The default value of n_estimators will change from 10 in ver
sion 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:2: DataConve
rsionWarning: A column-vector y was passed when a 1d array was expected.
Please change the shape of y to (n_samples,), for example using ravel().


MSE:0.48889020985651976
r_squared:0.5201393449254608
adj_r_squared:0.5194561071631335
```

Tuning Parameters

In [40]:
```
# Number of trees in random forest
n_estimators = [10,20,40,50,100,200,400,800,1000]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [2,5,8,10,12,20,50,80]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
tuned_parameters = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(tuned_parameters)
```

```
{'n_estimators': [10, 20, 40, 50, 100, 200, 400, 800, 1000], 'max_feature
s': ['auto', 'sqrt'], 'max_depth': [2, 5, 8, 10, 12, 20, 50, 80, None],
'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4], 'bootstra
p': [True, False]}
```

In [41]:
```
rf = RandomForestRegressor()
clf = GridSearchCV(rf, tuned_parameters, n_jobs=-1, verbose=1)
clf.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 2916 candidates, totalling 8748 fits

/home/jim/.local/lib/python3.6/site-packages/sklearn/model_selection/_spl
it.py:1978: FutureWarning: The default value of cv will change from 3 to
5 in version 0.22. Specify it explicitly to silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent worke
rs.
[Parallel(n_jobs=-1)]: Done  18 tasks      | elapsed:    3.1s
[Parallel(n_jobs=-1)]: Done 168 tasks      | elapsed:   49.1s
[Parallel(n_jobs=-1)]: Done 418 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done 768 tasks      | elapsed:  4.8min
[Parallel(n_jobs=-1)]: Done 1218 tasks      | elapsed:  9.8min
[Parallel(n_jobs=-1)]: Done 1768 tasks      | elapsed: 17.4min
[Parallel(n_jobs=-1)]: Done 2418 tasks      | elapsed: 26.8min
[Parallel(n_jobs=-1)]: Done 3168 tasks      | elapsed: 46.1min
[Parallel(n_jobs=-1)]: Done 4018 tasks      | elapsed: 64.9min
[Parallel(n_jobs=-1)]: Done 4968 tasks      | elapsed: 75.6min
[Parallel(n_jobs=-1)]: Done 6018 tasks      | elapsed: 93.9min
[Parallel(n_jobs=-1)]: Done 7168 tasks      | elapsed: 125.5min
[Parallel(n_jobs=-1)]: Done 8418 tasks      | elapsed: 171.3min
[Parallel(n_jobs=-1)]: Done 8748 out of 8748 | elapsed: 180.8min finished
/home/jim/.local/lib/python3.6/site-packages/sklearn/model_selection/_sea
rch.py:715: DataConversionWarning: A column-vector y was passed when a 1d
array was expected. Please change the shape of y to (n_samples,), for exa
mple using ravel().
  self.best_estimator_.fit(X, y, **fit_params)
```

Out[41]:
```
GridSearchCV(cv='warn', error_score='raise-deprecating',
             estimator=RandomForestRegressor(bootstrap=True, criterion='m
se',
                                             max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.
0,
                                             n_estimators='warn', n_jobs=
None,
                                             oob_score=False, random_sta
t...ne,
                                             verbose=0, warm_start=Fals
e),
             iid='warn', n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'max_depth': [2, 5, 8, 10, 12, 20, 50, 80, Non
e],
                         'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
```

```
                                    'n_estimators': [10, 20, 40, 50, 100, 200, 400,
800,
                                                     1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fals
e,
             scoring=None, verbose=1)
```

In [42]:
```
print(clf.best_params_)
```

```
{'bootstrap': True, 'max_depth': 20, 'max_features': 'sqrt', 'min_samples
_leaf': 1, 'min_samples_split': 10, 'n_estimators': 1000}
```

In [39]:
```
best_params_={'bootstrap': True, 'max_depth': 20, 'max_features': 'sqrt', '
```

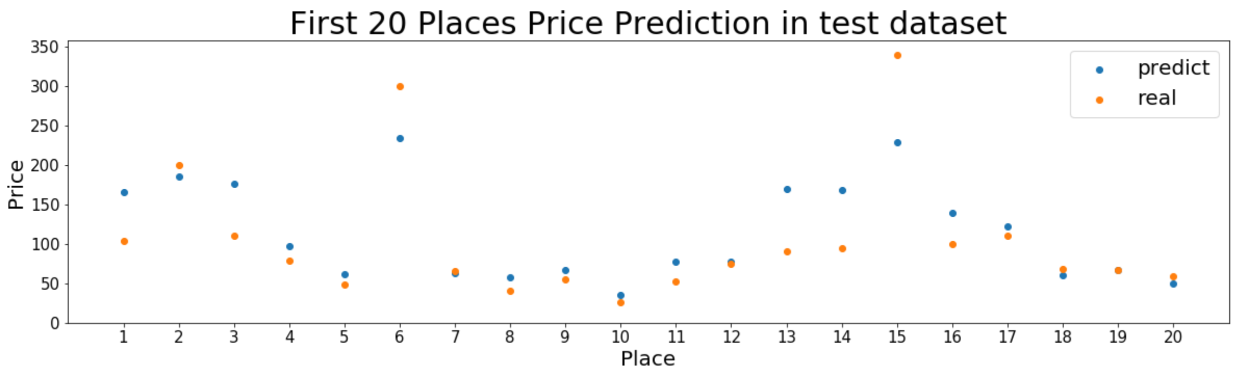
In [40]:
```
regr = RandomForestRegressor(**best_params_)
regr.fit(X_train, y_train)
y_predict=regr.predict(X_test)
mse = np.mean((y_predict-y_test['price']) ** 2)
r_squared=regr.score(X_test, y_test)
adj_r_squared = r_squared - (1 - r_squared) * (X_test.shape[1] / (X_test.sh
print(f"MSE:{mse}")
print(f"r_squared:{r_squared}")
print(f"adj_r_squared:{adj_r_squared}")
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:2: DataConve
rsionWarning: A column-vector y was passed when a 1d array was expected.
Please change the shape of y to (n_samples,), for example using ravel().


MSE:0.43945942074606587
r_squared:0.5686571723746036
adj_r_squared:0.5680430155246385
```

In [82]:
```
test_df=X_test.copy()
test_df['price']=y_predict
test_df[continous_columns]=scaler.inverse_transform(test_df[continous_colum
test_df=test_df.reset_index()
real_df=ori_price.reset_index().rename(columns={'price':'real_price'})
test_comparsion=pd.merge(test_df,real_df,how='left')[['price','real_price']
test_comparsion=test_comparsion.sort_values('price').reset_index(drop=True)
```

In [106]:
```
x=list(test_comparsion.index[1:21])
y=list(test_comparsion['price'].values[1:21])
y2=list(test_comparsion['real_price'].values[1:21])
fig, ax = plt.subplots(figsize=(20, 5))
ax.scatter(x, y,label='predict')
ax.scatter(x, y2,label='real')
plt.title('First 20 Places Price Prediction in test dataset',fontsize=30)
plt.yticks(np.linspace(0,350,8),fontsize=15)
plt.xticks(np.linspace(1,20,20),fontsize=15)
plt.xlabel('Place',fontsize=20)
plt.ylabel('Price',fontsize=20)
plt.legend(loc='best', fontsize = 20)
plt.show()
```



In [ ]: