In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

```
from sklearn.preprocessing import  LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score, KFold
from sklearn import metrics
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
import lightgbm as lgbm
from catboost import CatBoostRegressor
from xgboost import XGBRegressor
```

In [3]:

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

In [4]:

```
f = pd.read_csv("trainc.csv")
f.head()
```

Out[4]:

	id	log_price	property_type	room_type	amenities	accommodates	bathrooms	bed_type	cance
0	6901257	5.010635	Apartment	Entire home/apt	{"Wireless Internet","Air conditioning",Kitche...	3	1.0	Real Bed	
1	6304928	5.129899	Apartment	Entire home/apt	{"Wireless Internet","Air conditioning",Kitche...	7	1.0	Real Bed	
2	7919400	4.976734	Apartment	Entire home/apt	{TV,"Cable TV","Wireless Internet","Air condit...	5	1.0	Real Bed	
3	13418779	6.620073	House	Entire home/apt	{TV,"Cable TV",Internet,"Wireless Internet",Ki...	4	1.0	Real Bed	

In [5]:

```
f.shape
```

Out[5]:

```
(74111, 29)
```

In [6]:

```
f.columns
```

Out[6]:

```
Index(['id', 'log_price', 'property_type', 'room_type', 'amenities',
       'accommodates', 'bathrooms', 'bed_type', 'cancellation_policy',
       'cleaning_fee', 'city', 'description', 'first_review',
       'host_has_profile_pic', 'host_identity_verified', 'host_response_rate',
       'host_since', 'instant_bookable', 'last_review', 'latitude',
       'longitude', 'name', 'neighbourhood', 'number_of_reviews',
       'review_scores_rating', 'thumbnail_url', 'zipcode', 'bedrooms', 'beds'],
      dtype='object')
```

In [7]:

```
f.describe()
```

Out[7]:

	id	log_price	accommodates	bathrooms	latitude	longitude	number_
count	7.411100e+04	74111.000000	74111.000000	73911.000000	74111.000000	74111.000000	74
mean	1.126662e+07	4.782069	3.155146	1.235263	38.445958	-92.397525	
std	6.081735e+06	0.717394	2.153589	0.582044	3.080167	21.705322	
min	3.440000e+02	0.000000	1.000000	0.000000	33.338905	-122.511500	
25%	6.261964e+06	4.317488	2.000000	1.000000	34.127908	-118.342374	
50%	1.225415e+07	4.709530	2.000000	1.000000	40.662138	-76.996965	
75%	1.640226e+07	5.220356	4.000000	1.000000	40.746096	-73.954660	
max	2.123090e+07	7.600402	16.000000	8.000000	42.390437	-70.985047	(

In [8]:

```
#types of data
f.dtypes
```

Out[8]:

```
id                          int64
log_price                 float64
property_type              object
room_type                  object
amenities                  object
accommodates                int64
bathrooms                 float64
bed_type                   object
cancellation_policy        object
cleaning_fee                 bool
city                       object
description                object
first_review               object
host_has_profile_pic       object
host_identity_verified     object
host_response_rate         object
host_since                 object
instant_bookable           object
last_review                object
latitude                  float64
longitude                 float64
name                       object
neighbourhood              object
number_of_reviews           int64
review_scores_rating      float64
thumbnail_url              object
zipcode                    object
bedrooms                  float64
beds                      float64
dtype: object
```

In [9]:

```
#processing and analysis
f.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74111 entries, 0 to 74110
Data columns (total 29 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   id                      74111 non-null  int64  
 1   log_price               74111 non-null  float64
 2   property_type           74111 non-null  object 
 3   room_type               74111 non-null  object 
 4   amenities               74111 non-null  object 
 5   accommodates            74111 non-null  int64  
 6   bathrooms               73911 non-null  float64
 7   bed_type                74111 non-null  object 
 8   cancellation_policy     74111 non-null  object 
 9   cleaning_fee            74111 non-null  bool   
 10  city                    74111 non-null  object 
 11  description             74111 non-null  object 
 12  first_review            58247 non-null  object 
 13  host_has_profile_pic    73923 non-null  object 
 14  host_identity_verified  73923 non-null  object 
 15  host_response_rate      55812 non-null  object 
 16  host_since              73923 non-null  object 
 17  instant_bookable        74111 non-null  object 
 18  last_review             58284 non-null  object 
 19  latitude                74111 non-null  float64
 20  longitude               74111 non-null  float64
 21  name                    74111 non-null  object 
 22  neighbourhood           67239 non-null  object 
 23  number_of_reviews       74111 non-null  int64  
 24  review_scores_rating    57389 non-null  float64
 25  thumbnail_url           65895 non-null  object 
 26  zipcode                 73145 non-null  object 
 27  bedrooms                74020 non-null  float64
 28  beds                    73980 non-null  float64
dtypes: bool(1), float64(7), int64(3), object(18)
memory usage: 15.9+ MB
```

In [10]:

```
#counting unique values
ix= ["host_response_rate","property_type", "room_type","accommodates","bathrooms","bed_type", "ca
       "city","instant_bookable", "beds", "bedrooms", "neighbourhood","first_review", "last_revi
       "name","host_since","thumbnail_url", "latitude", "longitude",
        "host_has_profile_pic", "host_identity_verified"]
for i in ix:
    print(f[i].value_counts(), "\n")
    print("==============================================================")
```

```
==============================================================
Apartment            49003
House                16511
Condominium           2658
Townhouse             1692
Loft                  1244
Other                  607
Guesthouse             498
Bed & Breakfast        462
Bungalow               366
Villa                  179
Dorm                   142
Guest suite            123
Camper/RV               94
Timeshare               77
Cabin                   72
In-law                  71
Hostel                  70
Boutique hotel          69
```

In [11]:

```
#counting missing values
for c in f.columns:
    if f[c].isnull().sum() != 0:
        print("-----------------------------------")
        print("\n{} : {},  dtypes : {}".format(c,f[c].isnull().sum(),f[c].dtypes))
```

```
-----------------------------------

bathrooms : 200,  dtypes : float64
-----------------------------------

first_review : 15864,  dtypes : object
-----------------------------------

host_has_profile_pic : 188,  dtypes : object
-----------------------------------

host_identity_verified : 188,  dtypes : object
-----------------------------------

host_response_rate : 18299,  dtypes : object
-----------------------------------

host_since : 188,  dtypes : object
-----------------------------------

last_review : 15827,  dtypes : object
-----------------------------------

neighbourhood : 6872,  dtypes : object
-----------------------------------

review_scores_rating : 16722,  dtypes : float64
-----------------------------------

thumbnail_url : 8216,  dtypes : object
-----------------------------------

zipcode : 966,  dtypes : object
-----------------------------------

bedrooms : 91,  dtypes : float64
-----------------------------------

beds : 131,  dtypes : float64
```

In [12]:

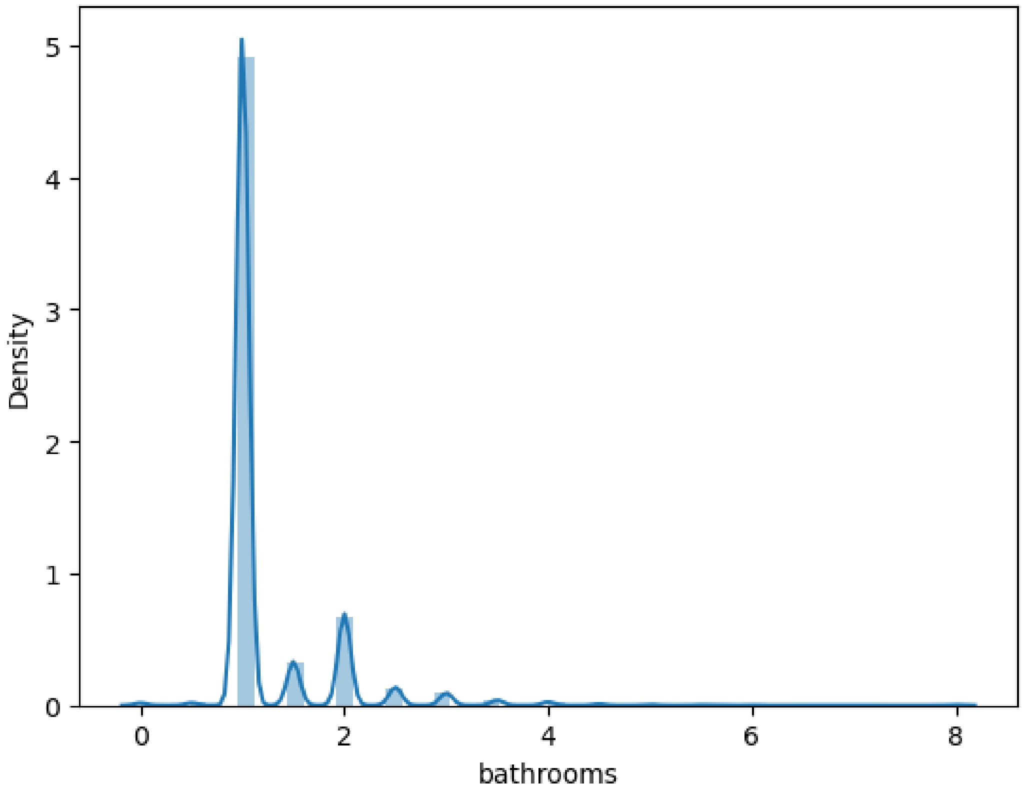
```
#filling missing values
f.last_review.fillna(method="ffill",inplace=True)
f.first_review.fillna(method="ffill",inplace=True)
f.host_since.fillna(method="ffill",inplace=True)
f.host_has_profile_pic.fillna(method="ffill",inplace=True)
f.host_identity_verified.fillna(method="ffill",inplace=True)
f.host_response_rate.fillna(method="ffill",inplace=True)

f["bathrooms"] =f['bathrooms'].fillna(round(f["bathrooms"].median()))
```

In [13]:

```
sns.distplot(f["bathrooms"])
```

Out[13]:

<Axes: xlabel='bathrooms', ylabel='Density'>

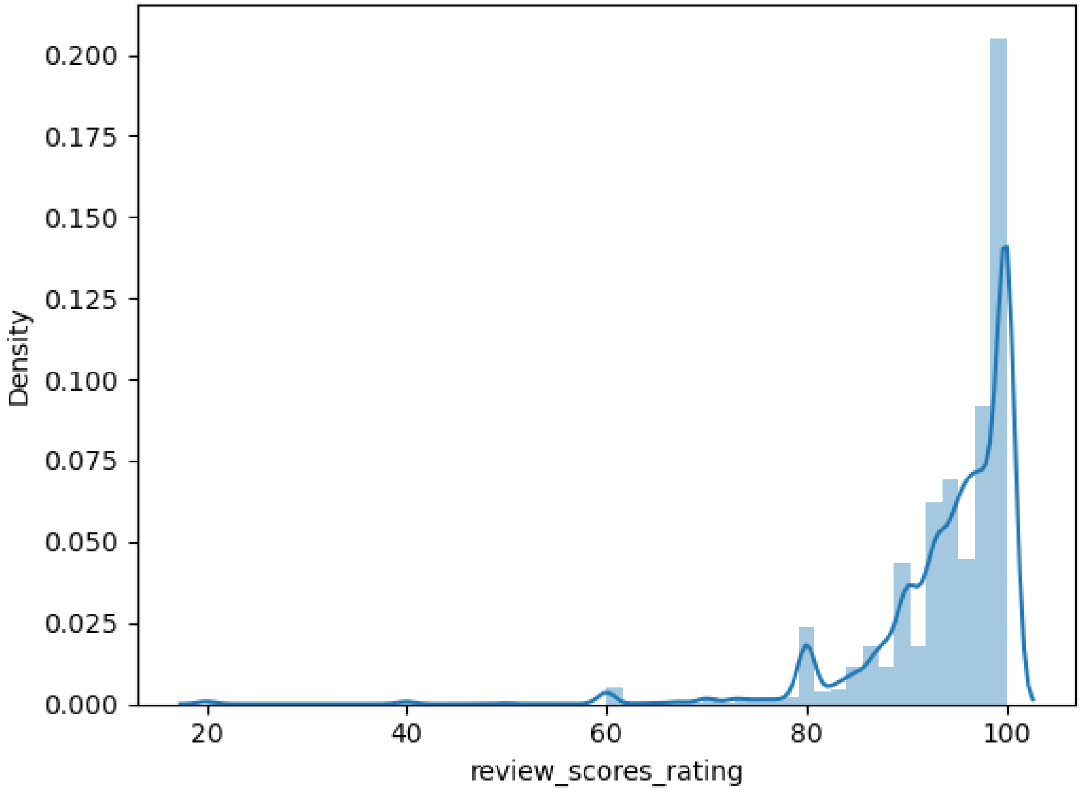


In [14]:

```
sns.distplot(f["review_scores_rating"])
```

Out[14]:

<Axes: xlabel='review_scores_rating', ylabel='Density'>

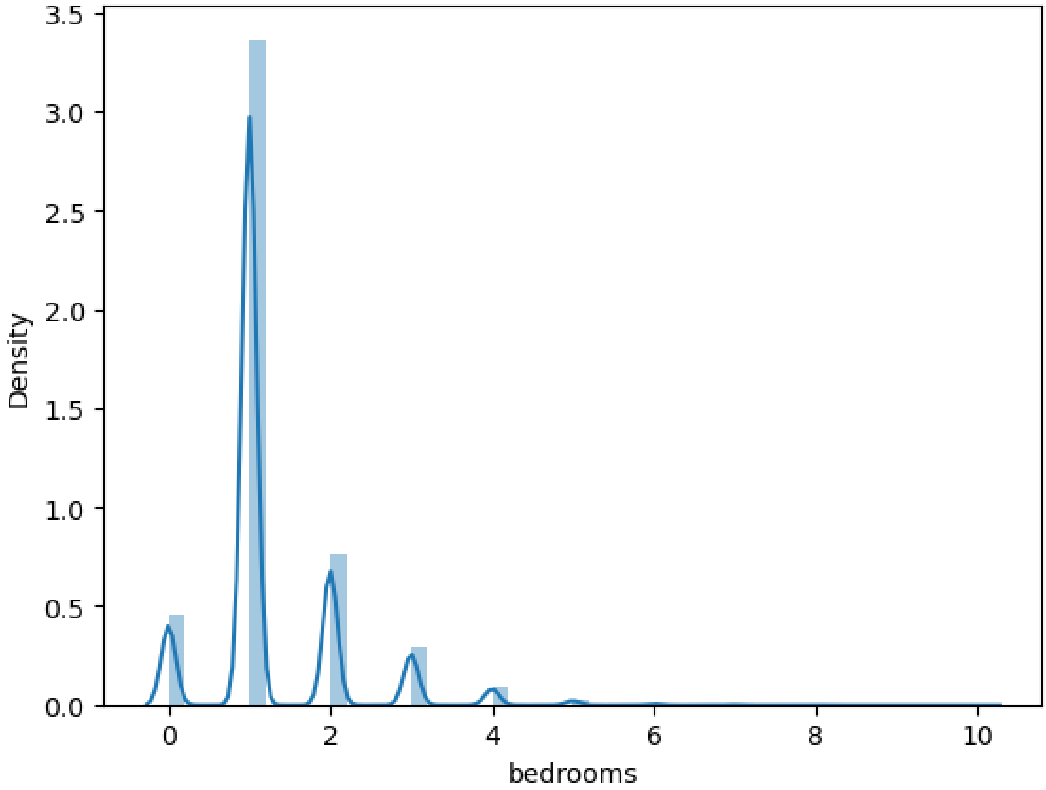


In [15]:

```
f["review_scores_rating"] = f["review_scores_rating"].fillna(0)
```

In [16]:

```
f["bedrooms"] = f['bedrooms'].fillna((f["bedrooms"].median()))
f["beds"] = f["beds"].fillna((f["beds"].median()))
sns.distplot(f["bedrooms"])
plt.show()
```



In [17]:

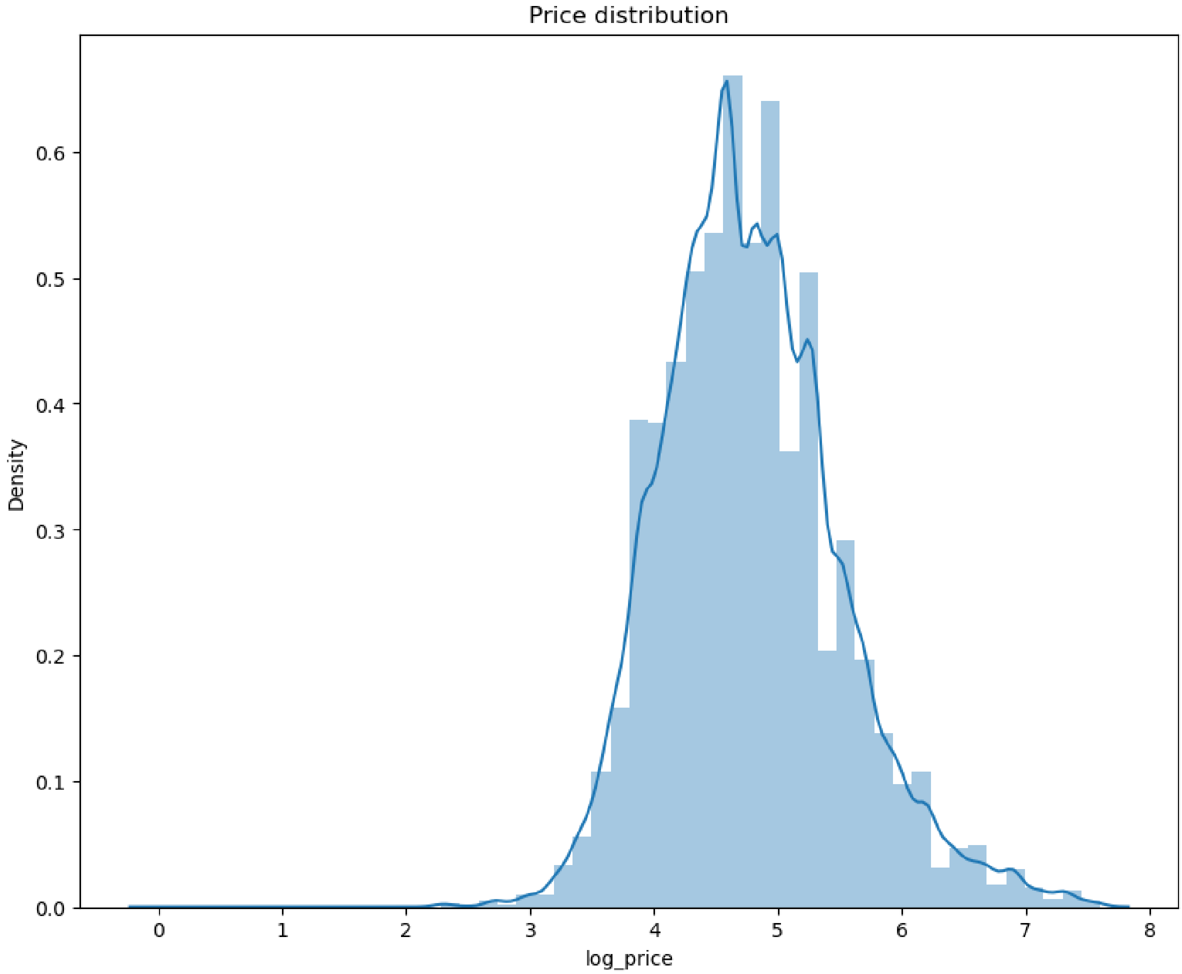
```
amenities_count = []
for i in f["amenities"]:
    amenities_count.append(len(i))

f["amenities"] = amenities_count
```

In [18]:

```
# Function to plot catplot graphs
def plot_catplot(h,v,he,a):
    sns.set(font_scale=1.5)
    sns.catplot(x=h,kind=v,data=f,height=he, aspect = a)

# Function to plot catplot graphs
def plot_piechart(h):
    sns.set(font_scale=1.5)
    fig = plt.figure(figsize=(5,5))
    ax = fig.add_axes([0,0,1,1])
    langs = list(f[h].unique())
    ax.axis('equal')
    std =list(f[h].value_counts())
    ax.pie(std, labels = langs,autopct='%1.2f%%')
    plt.show()
```

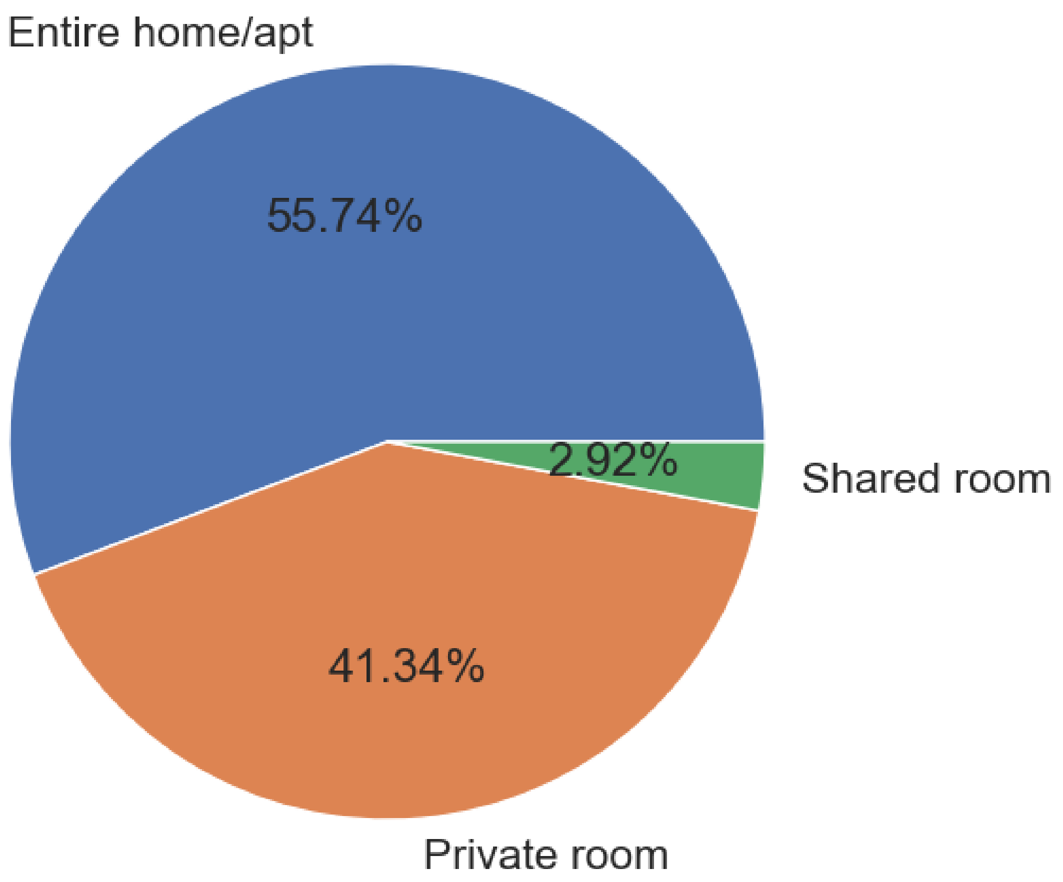
In [19]:

```
plt.figure(figsize = (10, 8))
sns.distplot(f["log_price"])
plt.title('Price distribution')
plt.show()
```



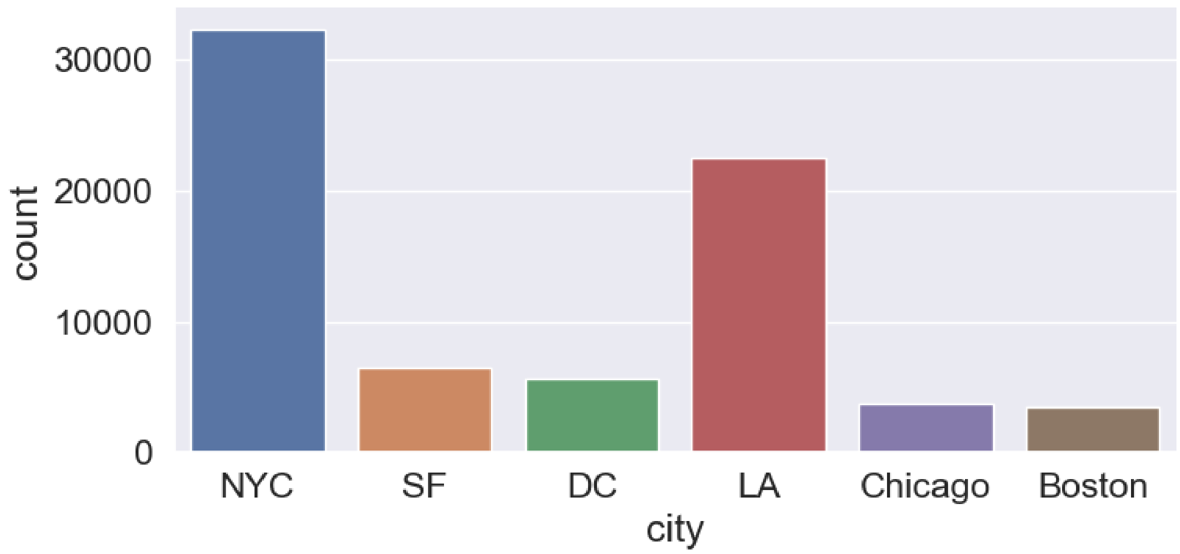
In [20]:

```
plot_piechart("room_type")
```



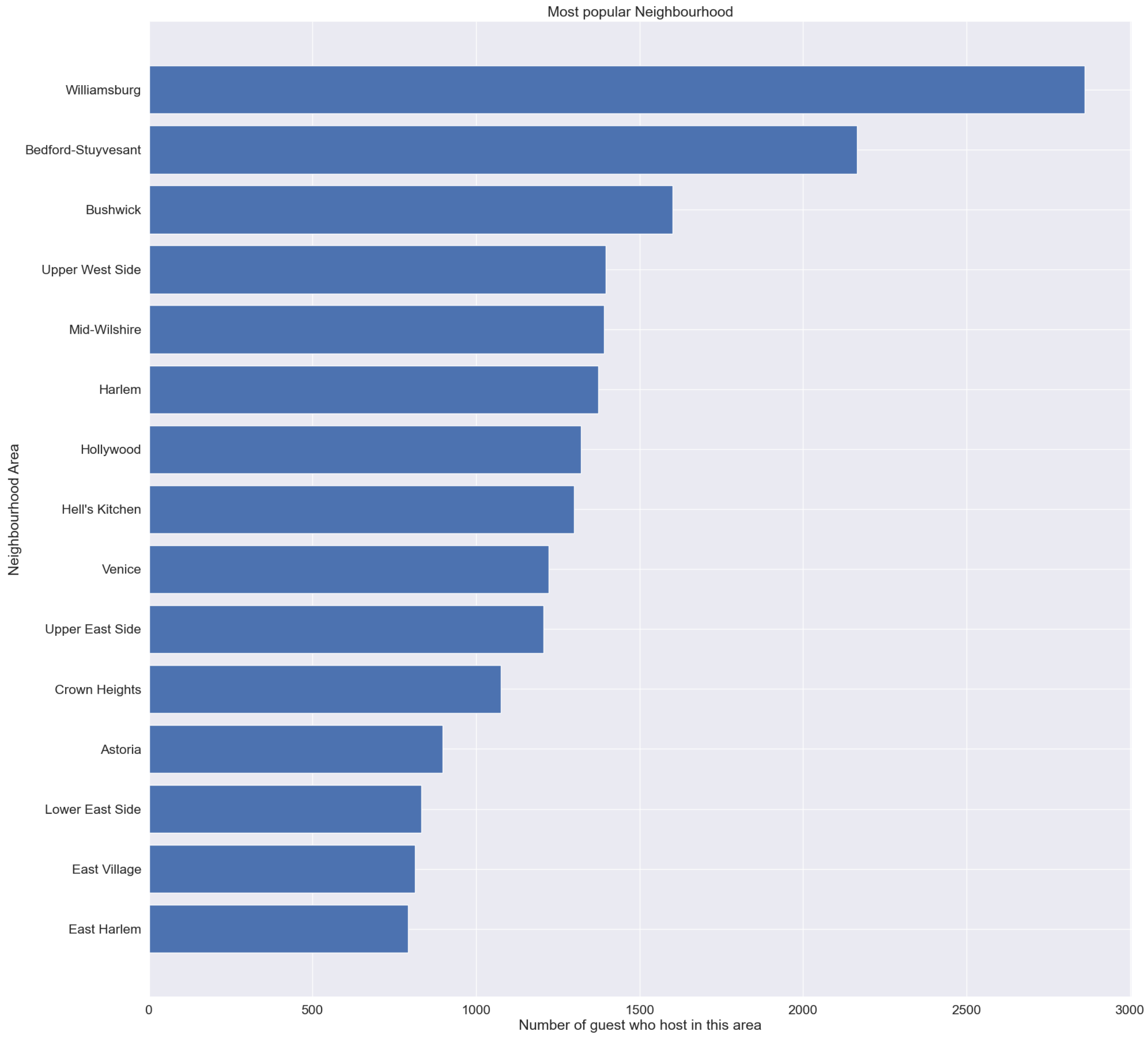
In [21]:

```
plot_catplot("city","count", 4, 2)
```



In [22]:

```
data = f.neighbourhood.value_counts()[:15]
plt.figure(figsize=(22,22))
x = list(data.index)
y = list(data.values)
x.reverse()
y.reverse()
plt.title("Most popular Neighbourhood")
plt.ylabel("Neighbourhood Area")
plt.xlabel("Number of guest who host in this area")
plt.barh(x,y)
plt.show()
```



In [23]:

```
#getting column with categorical data
categorical_col = []
for c in f.columns:
    if f[c].dtypes != "float64" and f[c].dtypes != "int64":
        categorical_col.append(c)
categorical_col
```

Out[23]:

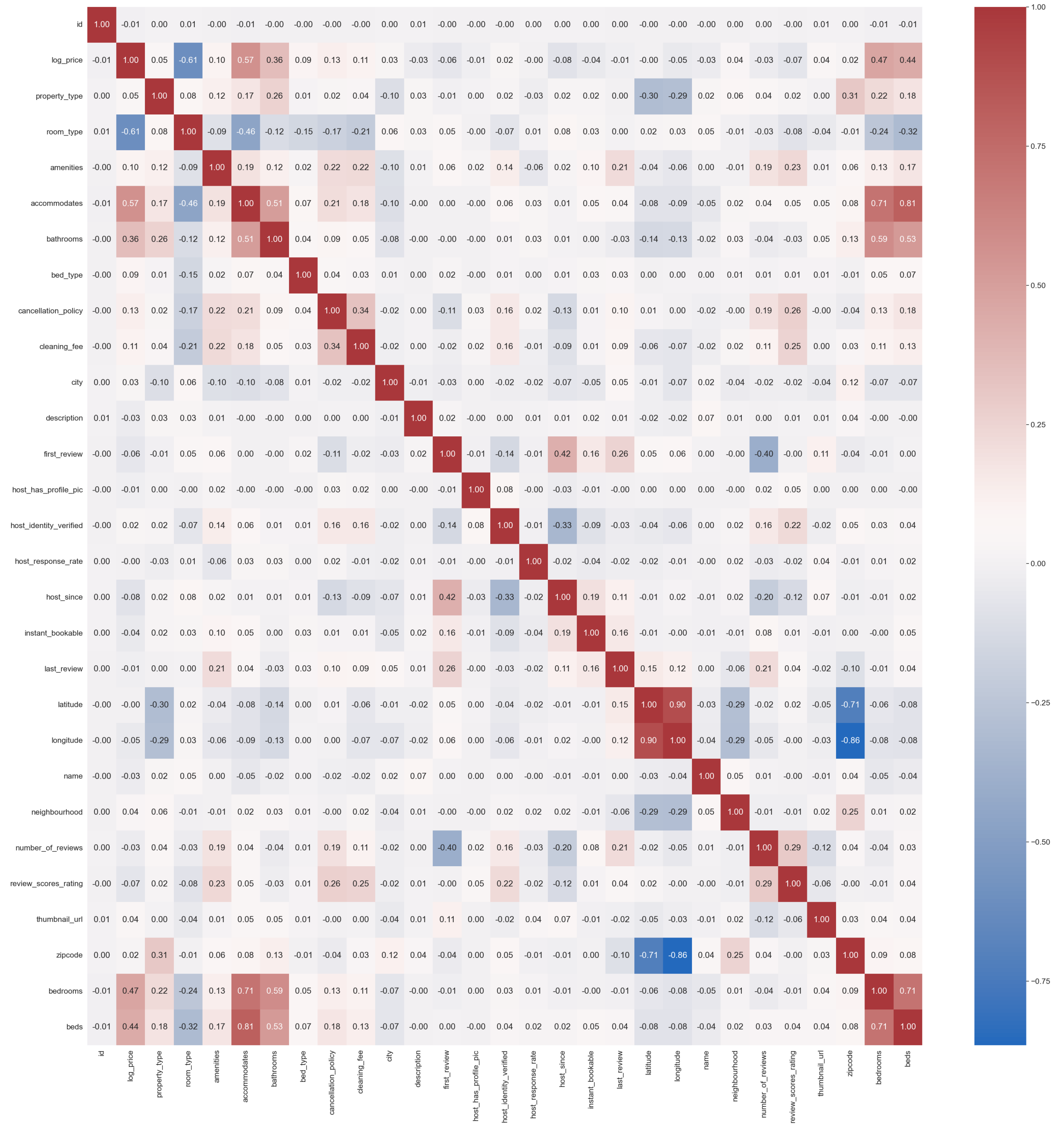
```
['property_type',
 'room_type',
 'bed_type',
 'cancellation_policy',
 'cleaning_fee',
 'city',
 'description',
 'first_review',
 'host_has_profile_pic',
 'host_identity_verified',
 'host_response_rate',
 'host_since',
 'instant_bookable',
 'last_review',
 'name',
 'neighbourhood',
 'thumbnail_url',
 'zipcode']
```

In [24]:

```
#converting non integer values into int
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for c in categorical_col:
     f[c] = le.fit_transform(f[c])
```

In [25]:

```
#correlation matrix of all categorical values
plt.figure(figsize = (40,40))
sns.heatmap(f.corr(), annot=True, fmt=".2f", cmap="vlag")
plt.show()
```



In [26]:

```
#assigning data for price prediction
x = f.drop(["id","name","log_price","description","first_review","host_since","last_review","nei
          "thumbnail_url", "zipcode"],axis = 1)
y = f.log_price
x
```

Out[26]:

	property_type	room_type	amenities	accommodates	bathrooms	bed_type	cancellation_policy
0	0	0	152	3	1.0	4	2
1	0	0	218	7	1.0	4	2
2	0	0	311	5	1.0	4	1
3	17	0	210	4	1.0	4	0
4	0	0	174	2	1.0	4	1
...	...	...	...	...	...	...	...
74106	0	1	2	1	1.0	4	0
74107	0	0	224	4	2.0	4	1
74108	0	0	402	5	1.0	4	1
74109	0	0	189	2	1.0	4	2
74110	2	0	279	4	1.0	4	1

74111 rows × 19 columns

In [27]:

```
#splitting into training and testing data
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=101)
x_train
```

Out[27]:

	property_type	room_type	amenities	accommodates	bathrooms	bed_type	cancellation_policy
16037	0	2	203	2	1.0	4	2
55761	0	0	236	6	1.0	4	0
23948	0	0	285	3	1.0	4	0
16709	17	1	253	2	1.0	4	2
55009	0	0	256	4	2.0	4	1
...	...	...	...	...	...	...	...
55293	0	1	278	2	1.0	4	2
49751	0	0	201	4	1.0	4	0
5695	0	1	264	2	1.0	4	2
73542	0	0	197	2	1.0	4	2
45919	0	1	268	2	1.0	4	2

59288 rows × 19 columns

In [28]:

```
#using Linear Regression Model
from sklearn.metrics import accuracy_score
lr = LinearRegression()
lr.fit(x_train,y_train)
y_pred_lr = lr.predict(x_test)

mae_lr = metrics.mean_absolute_error(y_test, y_pred_lr)
mse_lr = metrics.mean_squared_error(y_test, y_pred_lr)
rmse_lr = np.sqrt(metrics.mean_squared_error(y_test, y_pred_lr))
r2_lr   = metrics.r2_score(y_test, y_pred_lr)
vs_lr = metrics.explained_variance_score(y_test, y_pred_lr)
sd_lr = np.sqrt(vs_lr)

print('\nMean Absolute Error of Linear Regression     : ', mae_lr)
print('\nMean Squarred Error of Linear Regression     : ', mse_lr)
print('\nRoot Mean Squarred Error of Linear Regression: ', rmse_lr)
print('\nR2 Score of Linear Regression                : ', r2_lr)
print('\nVariance Score of Linear Regression          : ', vs_lr)
print('\nStandard Deviation of Linear Regression      : ', sd_lr)
```

```
Mean Absolute Error of Linear Regression     :  0.3686387280760641

Mean Squarred Error of Linear Regression     :  0.23491424456600318

Root Mean Squarred Error of Linear Regression:  0.48467952769433453

R2 Score of Linear Regression                :  0.5463886507190256

Variance Score of Linear Regression          :  0.5464128027519581

Standard Deviation of Linear Regression      :  0.7391974044542893
```

In [29]:

```
#using random forest
rf = RandomForestRegressor()
rf.fit(x_train,y_train)
y_pred_rf = rf.predict(x_test)

mae_rf  = metrics.mean_absolute_error(y_test, y_pred_rf)
mse_rf  = metrics.mean_squared_error(y_test, y_pred_rf)
rmse_rf = np.sqrt(metrics.mean_squared_error(y_test, y_pred_rf))
r2_rf   = metrics.r2_score(y_test, y_pred_rf)
vs_rf   = metrics.explained_variance_score(y_test, y_pred_rf)
sd_rf   = np.sqrt(vs_rf)

print('\nMean Absolute Error of Random Forest Regressor     : ', mae_rf)
print('\nMean Squarred Error of Random Forest Regressor     : ', mse_rf)
print('\nRoot Mean Squarred Error of Random Forest Regressor: ', rmse_rf)
print('\nR2 Score of Random Forest Regressor                : ', r2_rf)
print('\nVariance Score of Random Forest Regressor          : ', vs_rf)
print('\nStandard Deviation of Random Forest Regressor      : ', sd_rf)
```

```

Mean Absolute Error of Random Forest Regressor     :  0.28472732110885185

Mean Squarred Error of Random Forest Regressor     :  0.15631659354104516

Root Mean Squarred Error of Random Forest Regressor:  0.3953689334546218

R2 Score of Random Forest Regressor                :  0.6981580191437196

Variance Score of Random Forest Regressor          :  0.6982562012654179

Standard Deviation of Random Forest Regressor      :  0.8356172576397749
```

In [30]:

```
#polynomial regression model
from sklearn.linear_model import Ridge
model = Pipeline([('poly', PolynomialFeatures()),('ridge', Ridge(fit_intercept=True))])
param_grid = {'poly__degree': [1, 2, 3],'ridge__alpha': [0.1, 0.5, 1.0, 2.0]}

poly_tuned = GridSearchCV(model, param_grid, cv=5)

poly_tuned.fit(x_train, y_train)

y_pred_poly = poly_tuned.predict(x_test)

mae_poly  = metrics.mean_absolute_error(y_test, y_pred_poly)
mse_poly  = metrics.mean_squared_error(y_test, y_pred_poly)
rmse_poly = np.sqrt(metrics.mean_squared_error(y_test, y_pred_poly))
r2_poly   = metrics.r2_score(y_test, y_pred_poly)
vs_poly   = metrics.explained_variance_score(y_test, y_pred_poly)
sd_poly   = np.sqrt(vs_poly)

print('\nMean Absolute Error of Polynomial Regression     : ', mae_poly)
print('\nMean Squarred Error of Polynomial Regression     : ', mse_poly)
print('\nRoot Mean Squarred Error of Polynomial Regression: ', rmse_poly)
print('\nR2 Score of Polynomial Regression                : ', r2_poly)
print('\nVariance Score of Polynomial Regression          : ', vs_poly)
print('\nStandard Deviation of Polynomial Regression      : ', sd_poly)
```

```

Mean Absolute Error of Polynomial Regression     :  0.3420595709472461

Mean Squarred Error of Polynomial Regression     :  0.2037209304790029

Root Mean Squarred Error of Polynomial Regression:  0.4513545507458664

R2 Score of Polynomial Regression                :  0.6066218703677126

Variance Score of Polynomial Regression          :  0.6066413961063583

Standard Deviation of Polynomial Regression      :  0.7788718739987716
```

In [31]:

```
#polynomial regression model using standard scaler
from sklearn.linear_model import Ridge
models = Pipeline([('poly', PolynomialFeatures()),('ridge', Ridge(fit_intercept=True))])
param_grids = {'poly__degree': [1, 2, 3],'ridge__alpha': [0.1, 0.5, 1.0, 2.0]}

poly_tuneds = GridSearchCV(models, param_grids, cv=5)

scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

poly_tuneds.fit(x_train_scaled, y_train)

y_pred_polys = poly_tuneds.predict(x_test_scaled)

mae_polys  = metrics.mean_absolute_error(y_test, y_pred_polys)
mse_polys  = metrics.mean_squared_error(y_test, y_pred_polys)
rmse_polys = np.sqrt(metrics.mean_squared_error(y_test, y_pred_polys))
r2_polys   = metrics.r2_score(y_test, y_pred_polys)
vs_polys   = metrics.explained_variance_score(y_test, y_pred_polys)
sd_polys   = np.sqrt(vs_polys)

print('\nMean Absolute Error of Polynomial Regression     : ', mae_polys)
print('\nMean Squarred Error of Polynomial Regression     : ', mse_polys)
print('\nRoot Mean Squarred Error of Polynomial Regression: ', rmse_polys)
print('\nR2 Score of Polynomial Regression                : ', r2_polys)
print('\nVariance Score of Polynomial Regression          : ', vs_polys)
print('\nStandard Deviation of Polynomial Regression      : ', sd_polys)
```

```

Mean Absolute Error of Polynomial Regression     :  0.34174445569151973

Mean Squarred Error of Polynomial Regression     :  0.2032825029040481

Root Mean Squarred Error of Polynomial Regression:  0.4508686093575911

R2 Score of Polynomial Regression                :  0.6074684589779719

Variance Score of Polynomial Regression          :  0.6074877392915878

Standard Deviation of Polynomial Regression      :  0.7794149981181963
```

In [32]:

```
#polynomial regression model using minmax scaler
from sklearn.linear_model import Ridge
modelm = Pipeline([('poly', PolynomialFeatures()),('ridge', Ridge(fit_intercept=True))])
param_gridm = {'poly__degree': [1, 2, 3],'ridge__alpha': [0.1, 0.5, 1.0, 2.0]}

poly_tunedm = GridSearchCV(modelm, param_gridm, cv=5)

mscaler = MinMaxScaler()
x_train_mscaled = mscaler.fit_transform(x_train)
x_test_mscaled = mscaler.transform(x_test)

poly_tunedm.fit(x_train_mscaled, y_train)

y_pred_polym = poly_tunedm.predict(x_test_mscaled)

mae_polym  = metrics.mean_absolute_error(y_test, y_pred_polym)
mse_polym  = metrics.mean_squared_error(y_test, y_pred_polym)
rmse_polym = np.sqrt(metrics.mean_squared_error(y_test, y_pred_polym))
r2_polym   = metrics.r2_score(y_test, y_pred_polym)
vs_polym   = metrics.explained_variance_score(y_test, y_pred_polym)
sd_polym   = np.sqrt(vs_polym)

print('\nMean Absolute Error of Polynomial Regression     : ', mae_polym)
print('\nMean Squarred Error of Polynomial Regression     : ', mse_polym)
print('\nRoot Mean Squarred Error of Polynomial Regression: ', rmse_polym)
print('\nR2 Score of Polynomial Regression                : ', r2_polym)
print('\nVariance Score of Polynomial Regression          : ', vs_polym)
print('\nStandard Deviation of Polynomial Regression      : ', sd_polym)
```

```

Mean Absolute Error of Polynomial Regression     :  0.3366638218154728

Mean Squarred Error of Polynomial Regression     :  0.198799636085604

Root Mean Squarred Error of Polynomial Regression:  0.4458695280971823

R2 Score of Polynomial Regression                :  0.6161247210531735

Variance Score of Polynomial Regression          :  0.6161524903064721

Standard Deviation of Polynomial Regression      :  0.7849538141231446
```

In [33]:

```
gb = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3)

gb.fit(x_train, y_train)

y_pred_gb = gb.predict(x_test)

mae_gb  = metrics.mean_absolute_error(y_test, y_pred_gb)
mse_gb  = metrics.mean_squared_error(y_test, y_pred_gb)
rmse_gb = np.sqrt(metrics.mean_squared_error(y_test, y_pred_gb))
r2_gb   = metrics.r2_score(y_test, y_pred_gb)
vs_gb = metrics.explained_variance_score(y_test, y_pred_gb)
sd_gb = np.sqrt(vs_gb)

print('\nMean Absolute Error of Gradient Boosting     : ', mae_gb)
print('\nMean Squarred Error of Gradient Boosting     : ', mse_gb)
print('\nRoot Mean Squarred Error of Gradient Boosting: ', rmse_gb)
print('\nR2 Score of Gradient Boosting                : ', r2_gb)
print('\nVariance Score of Gradient Boosting          : ', vs_gb)
print('\nStandard Deviation of Gradient Boosting      : ', sd_gb)
```

```

Mean Absolute Error of Gradient Boosting     :  0.3044776810309421

Mean Squarred Error of Gradient Boosting     :  0.16924208684550437

Root Mean Squarred Error of Gradient Boosting:  0.41139043115452306

R2 Score of Gradient Boosting                :  0.6731993348851726

Variance Score of Gradient Boosting          :  0.6732069126731021

Standard Deviation of Gradient Boosting      :  0.8204918723991738
```

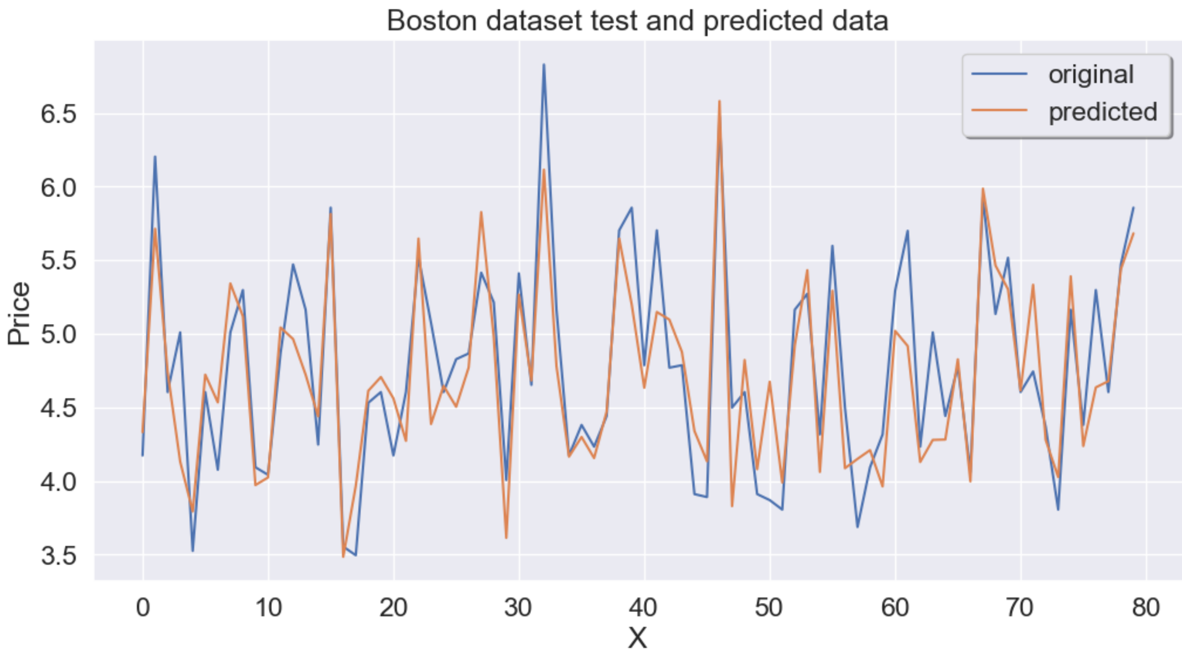
In [50]:

```
#using XGBRegressor
xgb = XGBRegressor(objective='reg:squarederror')
xgb.fit(x_train, y_train)
y_pred_xgb = xgb.predict(x_test)

mae_xgb  = metrics.mean_absolute_error(y_test, y_pred_xgb)
mse_xgb  = metrics.mean_squared_error(y_test, y_pred_xgb)
rmse_xgb = np.sqrt(metrics.mean_squared_error(y_test, y_pred_xgb))
r2_xgb   = metrics.r2_score(y_test, y_pred_xgb)
vs_xgb = metrics.explained_variance_score(y_test, y_pred_xgb)
sd_xgb = np.sqrt(vs_xgb)

print('\nMean Absolute Error of XGBoost Regressor     : ', mae_xgb)
print('\nMean Squarred Error of XGBoost Regressor     : ', mse_xgb)
print('\nRoot Mean Squarred Error of XGBoost Regressor: ', rmse_xgb)
print('\nR2 Score of XGBoost Regressor                : ', r2_xgb)
print('\nVariance Score of XGBoost Regressor          : ', vs_xgb)
print('\nStandard Deviation of XGBoost Regressor      : ', sd_xgb)
```

```
Mean Absolute Error of XGBoost Regressor     :  0.28333366551958417

Mean Squarred Error of XGBoost Regressor     :  0.15133932457667712

Root Mean Squarred Error of XGBoost Regressor:  0.3890235527274372

R2 Score of XGBoost Regressor                :  0.7077689548059328

Variance Score of XGBoost Regressor          :  0.7077725884367306

Standard Deviation of XGBoost Regressor      :  0.8412922134649355
```

In [52]:

```
x_ax = range(len(y_test))
plt.figure(figsize=(12, 6))
plt.plot(x_ax[0:80], y_test[0:80], label="original")
plt.plot(x_ax[0:80], y_pred_xgb[0:80], label="predicted")
plt.title("Boston dataset test and predicted data")
plt.xlabel('X')
plt.ylabel('Price')
plt.legend(loc='best',fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```



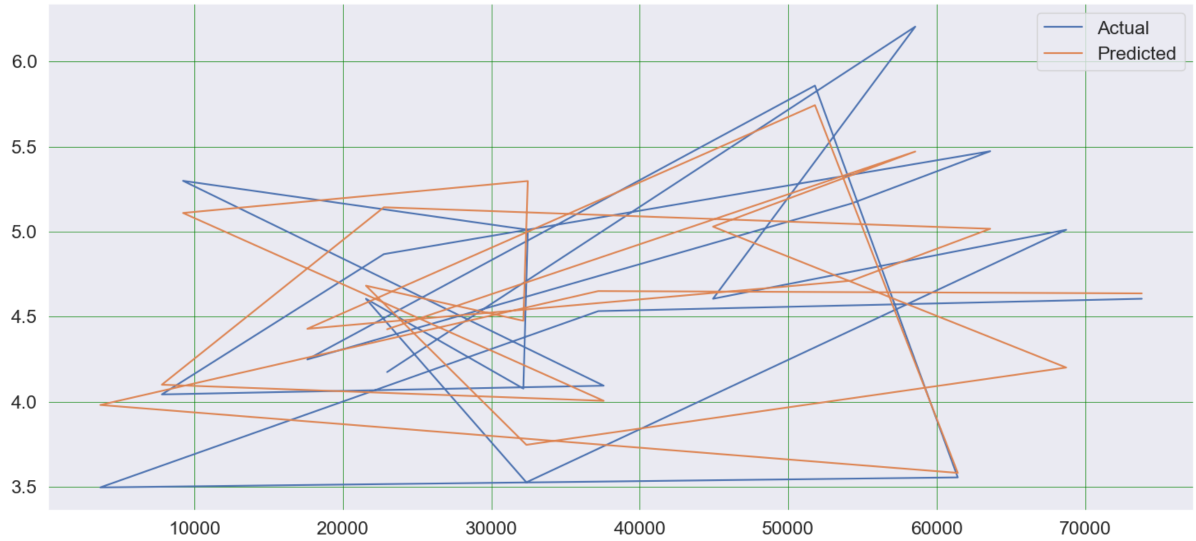
In [40]:

```
#using CatboostRegressor
model_CBR = CatBoostRegressor()
model_CBR.fit(x_train, y_train)
cross_val_score(model_CBR, x_train, y_train,scoring='r2',cv=KFold(n_splits=5,shuffle=True,random_

y_pred_cbr = model_CBR.predict(x_test)

mae_cbr  = metrics.mean_absolute_error(y_test, y_pred_cbr)
mse_cbr  = metrics.mean_squared_error(y_test, y_pred_cbr)
rmse_cbr = np.sqrt(metrics.mean_squared_error(y_test, y_pred_cbr))
r2_cbr   = metrics.r2_score(y_test, y_pred_cbr)
vs_cbr = metrics.explained_variance_score(y_test, y_pred_cbr)
sd_cbr = np.sqrt(vs_cbr)

print('\nMean Absolute Error of CatBoost Regressor     : ', mae_cbr)
print('\nMean Squarred Error of CatBoost Regressor     : ', mse_cbr)
print('\nRoot Mean Squarred Error of CatBoost Regressor: ', rmse_cbr)
print('\nR2 Score of CatBoost Regressor                : ', r2_cbr)
print('\nVariance Score of CatBoost Regressor          : ', vs_cbr)
print('\nStandard Deviation of CatBoost Regressor      : ', sd_cbr)
```

```
993:	learn: 0.3564552	total: 8.64s	remaining: 52.1ms
994:	learn: 0.3564283	total: 8.64s	remaining: 43.4ms
995:	learn: 0.3564011	total: 8.65s	remaining: 34.7ms
996:	learn: 0.3563933	total: 8.66s	remaining: 26.1ms
997:	learn: 0.3563838	total: 8.67s	remaining: 17.4ms
998:	learn: 0.3563518	total: 8.68s	remaining: 8.69ms
999:	learn: 0.3563376	total: 8.69s	remaining: 0us

Mean Absolute Error of CatBoost Regressor     :  0.2780911445736927

Mean Squarred Error of CatBoost Regressor     :  0.14572259903144336

Root Mean Squarred Error of CatBoost Regressor:  0.3817362951455407

R2 Score of CatBoost Regressor                :  0.718614659194023

Variance Score of CatBoost Regressor          :  0.7186237127405559

Standard Deviation of CatBoost Regressor      :  0.8477167644564757
```

In [37]:

```
df = pd.DataFrame({'Actual': y_test[0:20], 'Predicted': y_pred_cbr[0:20]})

df.plot(kind='line',figsize=(18,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



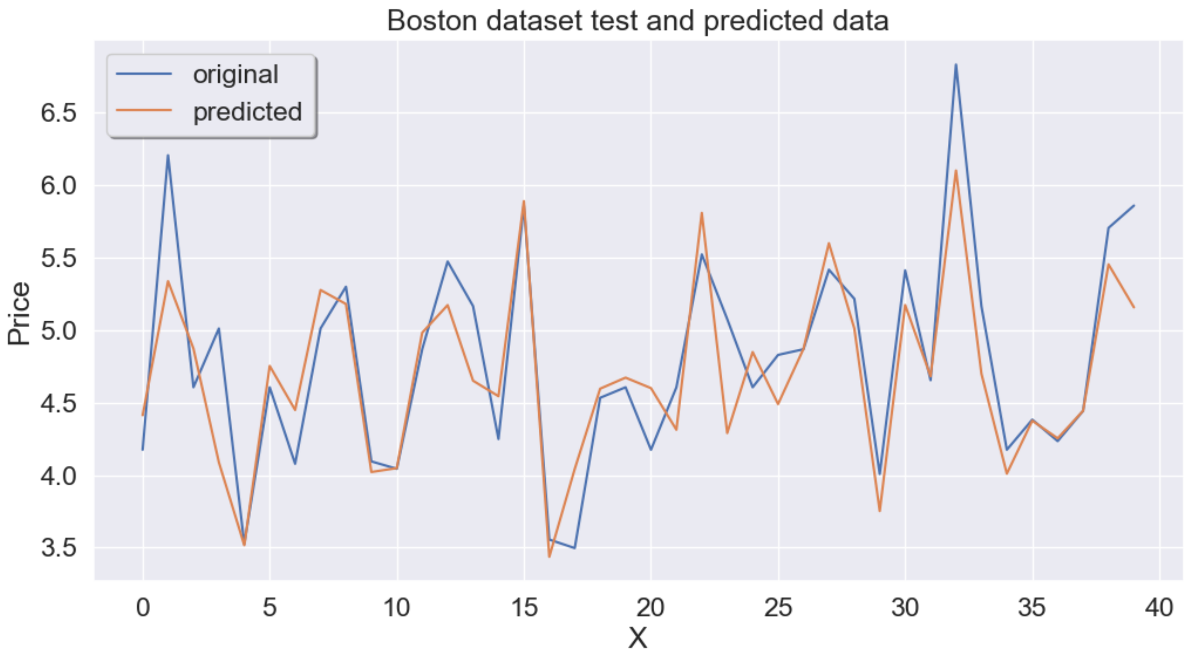
In [41]:

```
#using LGBMRegression

lgb_train = lgbm.Dataset(x_train, y_train)
lgb_eval = lgbm.Dataset(x_test, y_test, reference=lgb_train)
params = {'task': 'predict', 'boosting': 'gbdt',
          'objective': 'mean_absolute_error','num_leaves': 512,
          'learnnig_rate': 0.05,
          'metric': {'l2','l1'},'verbose': -1}

model_lgb = lgbm.train(params,train_set=lgb_train,valid_sets=lgb_eval)

y_pred_lgb = model_lgb.predict(x_test)

mae_lgb  = metrics.mean_absolute_error(y_test, y_pred_lgb)
mse_lgb  = metrics.mean_squared_error(y_test, y_pred_lgb)
rmse_lgb = np.sqrt(metrics.mean_squared_error(y_test, y_pred_lgb))
r2_lgb   = metrics.r2_score(y_test, y_pred_lgb)
vs_lgb = metrics.explained_variance_score(y_test, y_pred_lgb)
sd_lgb = np.sqrt(vs_xgb)

print('\nMean Absolute Error of LGBMRegressor     : ', mae_lgb)
print('\nMean Squarred Error of LGBMRegressor     : ', mse_lgb)
print('\nRoot Mean Squarred Error of LGBMRegressor: ', rmse_lgb)
print('\nR2 Score of LGBMRegressor                : ', r2_lgb)
print('\nVariance Score of LGBMRegressor          : ', vs_lgb)
print('\nStandard Deviation of LGBMRegressor      : ', sd_lgb)
```

```

Mean Absolute Error of LGBMRegressor     :  0.27813377949520196

Mean Squarred Error of LGBMRegressor     :  0.1497512894747939

Root Mean Squarred Error of LGBMRegressor:  0.3869771175079916

R2 Score of LGBMRegressor                :  0.7108353961220037

Variance Score of LGBMRegressor          :  0.7112074709763658

Standard Deviation of LGBMRegressor      :  0.8412922134649355
```

In [39]:

```
x_ax = range(len(y_test))
plt.figure(figsize=(12, 6))
plt.plot(x_ax[0:40], y_test[0:40], label="original")
plt.plot(x_ax[0:40], y_pred_lgb[0:40], label="predicted")
plt.title("Boston dataset test and predicted data")
plt.xlabel('X')
plt.ylabel('Price')
plt.legend(loc='best',fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```



In [42]:

```
r2_list = {"Linear Regression": r2_lr,
           "Random Forest": r2_rf ,
           "Polynomial Regression": r2_poly,
           "Polynomial Regression Scaled": r2_polys,
           "Polynomial Regression MinMax": r2_polym,
           "CatBoost": r2_cbr,
           "Gradient Boosting":r2_gb ,
           "XGBoost": r2_xgb,
           "LGBMRegression": r2_lgb}

mae_list = {"Linear Regression": mae_lr,
           "Random Forest": mae_rf ,
           "Polynomial Regression": mae_poly,
           "Polynomial Regression Scaled": mae_polys,
           "Polynomial Regression MinMax": mae_polym,
           "CatBoost": mae_cbr,
           "Gradient Boosting":mae_gb ,
           "XGBoost": mae_xgb,
            "LGBMRegression": mae_lgb}

mse_list = {"Linear Regression": mse_lr,
           "Random Forest": mse_rf ,
           "Polynomial Regression": mse_poly,
           "Polynomial Regression Scaled": mse_polys,
           "Polynomial Regression MinMax": mse_polym,
           "CatBoost": mse_cbr,
           "Gradient Boosting":mse_gb ,
           "XGBoost": mse_xgb,
            "LGBMRegression": mse_lgb}

rmse_list = {"Linear Regression": rmse_lr,
           "Random Forest": rmse_rf ,
           "Polynomial Regression": rmse_poly,
           "Polynomial Regression Scaled": rmse_polys,
           "Polynomial Regression MinMax": rmse_polym,
           "CatBoost": rmse_cbr,
           "Gradient Boosting":rmse_gb ,
           "XGBoost": rmse_xgb,
            "LGBMRegression": rmse_lgb}
vs_list = {"Linear Regression": vs_lr,
           "Random Forest": vs_rf ,
           "Polynomial Regression":vs_poly,
           "Polynomial Regression Scaled": vs_polys,
           "Polynomial Regression MinMax": vs_polym,
           "CatBoost": vs_cbr,
           "Gradient Boosting":vs_gb ,
           "XGBoost": vs_xgb,
            "LGBMRegression": vs_lgb}
sd_list = {"Linear Regression": sd_lr,
           "Random Forest": sd_rf ,
           "Polynomial Regression": sd_poly,
           "Polynomial Regression Scaled": sd_polys,
           "Polynomial Regression MinMax":sd_polym,
           "CatBoost":sd_cbr,
           "Gradient Boosting":sd_gb ,
           "XGBoost": sd_xgb,
            "LGBMRegression": sd_lgb}
```

In [44]:

```
a1 =  pd.DataFrame.from_dict(r2_list, orient = 'index', columns = ["R2 SCORE"])
a2 =  pd.DataFrame.from_dict(mae_list, orient = 'index', columns = ["MEAN ABSOLUTE ERROR"])
a3 =  pd.DataFrame.from_dict(mse_list, orient = 'index', columns = ["MEAN SQUARRED ERROR"])
a4 =  pd.DataFrame.from_dict(rmse_list, orient = 'index', columns = ["ROOT MEAN SQUARRED ERROR"]
a5 =  pd.DataFrame.from_dict(vs_list, orient = 'index', columns = ["VARIANCE"])
a6 =  pd.DataFrame.from_dict(sd_list, orient = 'index', columns = ["STANDARD DEVIATION"])
```

In [47]:

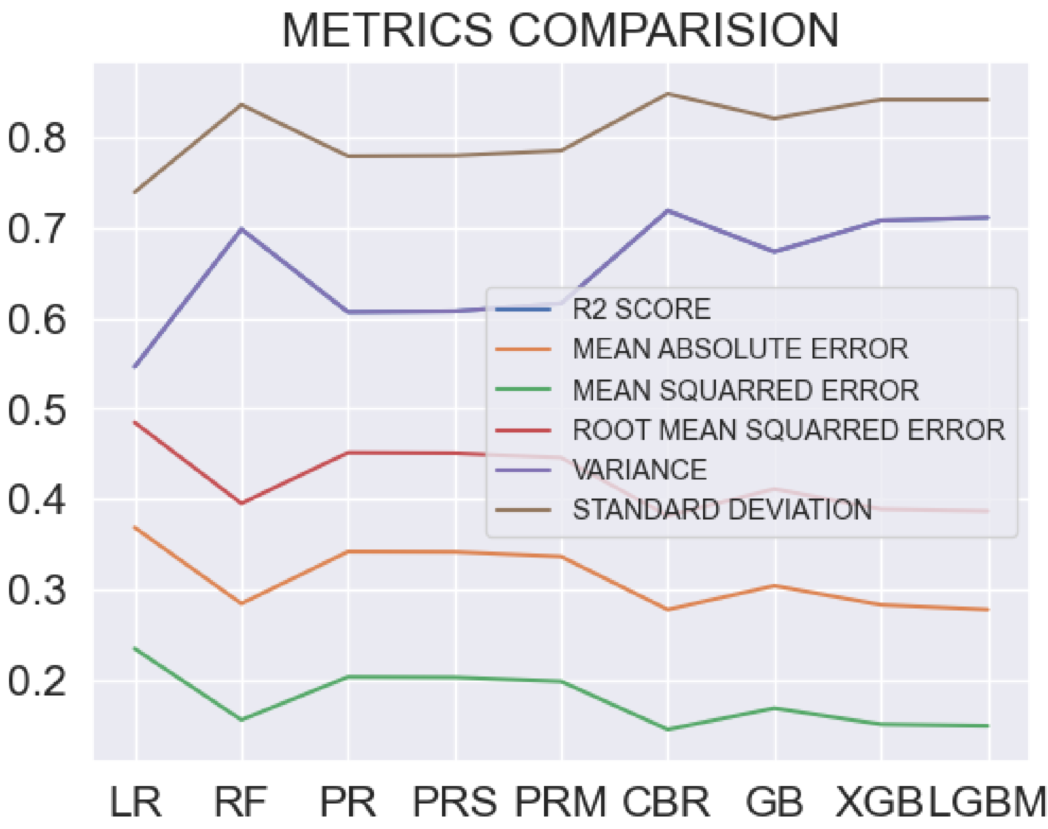
```
org = pd.concat([a1, a2, a3, a4,a5,a6], axis = 1)
org
```

Out[47]:

	R2 SCORE	MEAN ABSOLUTE ERROR	MEAN SQUARRED ERROR	ROOT MEAN SQUARRED ERROR	VARIANCE	STANDARD DEVIATION
Linear Regression	0.546389	0.368639	0.234914	0.484680	0.546413	0.739197
Random Forest	0.698158	0.284727	0.156317	0.395369	0.698256	0.835617
Polynomial Regression	0.606622	0.342060	0.203721	0.451355	0.606641	0.778872
Polynomial Regression Scaled	0.607468	0.341744	0.203283	0.450869	0.607488	0.779415
Polynomial Regression MinMax	0.616125	0.336664	0.198800	0.445870	0.616152	0.784954
CatBoost	0.718615	0.278091	0.145723	0.381736	0.718624	0.847717
Gradient Boosting	0.673199	0.304478	0.169242	0.411390	0.673207	0.820492
XGBoost	0.707769	0.283334	0.151339	0.389024	0.707773	0.841292
LGBMRegression	0.710835	0.278134	0.149751	0.386977	0.711207	0.841292

In [49]:

```
alg = ['LR','RF','PR','PRS','PRM','CBR','GB','XGB','LGBM']
plt.plot(alg,a1)
plt.plot(alg,a2)
plt.plot(alg,a3)
plt.plot(alg,a4)
plt.plot(alg,a5)
plt.plot(alg,a6)
legend = ["R2 SCORE", "MEAN ABSOLUTE ERROR", "MEAN SQUARRED ERROR", "ROOT MEAN SQUARRED ERROR","\

plt.title("METRICS COMPARISION")
plt.legend(legend, loc= 'right', fontsize='xx-small')
plt.show()
```



In [ ]: