house_price_prediction

September 9, 2024

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In this project, we will create a model for predicting house prices in India.
This project was suggested to me by the DataScience training provided by Codebasics (https://codebasics.io/). The dataset comes from the Kaggle website (https://www.kaggle.com/datasets/amitabhajoy/bengaluru-house-price-data/data) and can be viewed using the CSV file in the current directory.

This project enabled me to put forward certain skills in data exploration and fine-tuning models.

[2]:
```python
df = pd.read_csv('bengaluru_house_prices.csv')
df.head()
```

[2]:
```
              area_type   availability                  location       size  \
0  Super built-up  Area         19-Dec  Electronic City Phase II      2 BHK
1            Plot  Area  Ready To Move          Chikka Tirupathi  4 Bedroom
2        Built-up  Area  Ready To Move               Uttarahalli      3 BHK
3  Super built-up  Area  Ready To Move        Lingadheeranahalli      3 BHK
4  Super built-up  Area  Ready To Move                  Kothanur      2 BHK

  society total_sqft  bath  balcony   price
0 Coomee        1056   2.0      1.0   39.07
1 Theanmp       2600   5.0      3.0  120.00
2     NaN       1440   2.0      3.0   62.00
3 Soiewre       1521   3.0      1.0   95.00
4     NaN       1200   2.0      1.0   51.00
```

[3]:
```python
df.shape
```

[3]:
```
(13320, 9)
```

Data Cleaning

Arbitrarily, I choose to remove the columns 'availability' and 'society' which I don't find relevant for our prediction problem.

[4]:
```python
df1 = df.drop(['availability', 'society'], axis = 'columns')
df1.head()
```

[4]:
```
                area_type                 location       size total_sqft  bath  \
0  Super built-up  Area  Electronic City Phase II      2 BHK       1056   2.0
1            Plot  Area          Chikka Tirupathi  4 Bedroom       2600   5.0
2        Built-up  Area               Uttarahalli      3 BHK       1440   2.0
3  Super built-up  Area        Lingadheeranahalli      3 BHK       1521   3.0
4  Super built-up  Area                  Kothanur      2 BHK       1200   2.0

   balcony   price
0      1.0   39.07
1      3.0  120.00
2      3.0   62.00
3      1.0   95.00
4      1.0   51.00
```

[5]:
```python
df1.isna().sum()
```

[5]:
```
area_type       0
location        1
size           16
total_sqft      0
bath           73
balcony       609
price           0
dtype: int64
```

Since the dataset is big enough, I choose to get rid of the NaN values.

[6]:
```python
df2 = df1.dropna()
df2.isna().sum()
```

[6]:
```
area_type     0
location      0
size          0
total_sqft    0
bath          0
balcony       0
price         0
dtype: int64
```

Now, we will study some columns which may contain wrong encoded features.

[7]:
```python
df2['size'].unique()
```

[7]:
```
array(['2 BHK', '4 Bedroom', '3 BHK', '3 Bedroom', '1 BHK', '1 RK',
       '4 BHK', '1 Bedroom', '2 Bedroom', '6 Bedroom', '8 Bedroom',
       '7 Bedroom', '5 BHK', '7 BHK', '6 BHK', '5 Bedroom', '11 BHK',
       '9 BHK', '9 Bedroom', '27 BHK', '11 Bedroom', '43 Bedroom',
       '14 BHK', '8 BHK', '12 Bedroom', '10 Bedroom', '13 BHK'],
      dtype=object)
```

Here we can notice that the bedroom counting is not homogeneous. For instance, a house with 3 bedrooms can be written '3 BHK' or '3 Bedroom'. We need to fix this in order to allow the predictions.

[8]:
```
def extract_nb_bedrooms(size : str) -> int :
    return int(size.split(' ')[0])
```

[9]:
```
df3 = df2.copy()
df3['bedroom_formated'] = df2['size'].apply(extract_nb_bedrooms)
```

[10]:
```
df3 = df3.drop('size', axis = 'columns')
```

[11]:
```
df3['total_sqft'].unique()
```

[11]:
```
array(['1056', '2600', '1440', …, '1133 - 1384', '774', '4689'],
      dtype=object)
```

Some houses are listed with a range for the surface area instead of a unique value or with a different unit. We will replace these ranges with the mean value, and convert in the correct unit using Regex. At this stage, we choose to remove certain conversion values that seem particularly low/high, in order to discard potential calculation errors. The interval we'll keep is [50, 20000] sqft.

[12]:
```
import re
```

[13]:
```
df3['total_sqft'] = df3['total_sqft'].apply(str)
def format_surf_area(area : str) -> int:
    if '-' in area:
        min = float(area.split(' ')[0])
        max = float(area.split(' ')[-1])
        return (min + max)/2
    else:
        try:
            return float(area)
        except:
            # return None
            decomp = re.match(r'([\d.]+)\s*(.*)', area)
            value = float(decomp.group(1))
            unity = decomp.group(2)
            if unity=='Sq. Meter':
                conversion = value*10.764 # 1 m² = 10.764 sqft
            elif unity=='Perch':
                conversion = value*272.3 # 1 perch = 272.3 sqft
            elif unity=='Acres':
                conversion = value*43560 # 1 Acre = 43560 sqft
            elif unity=='Guntha':
                conversion = value*1089 # 1 Gunta = 1089 sqft
            elif unity=='Sq. Yards':
                conversion = value*9 # 1 Yard = 9 sqft
```

```
            elif unity=='Cents':
                conversion = value*431 # 1 Cent = 431 sqft
            elif unity=='Grounds':
                conversion = value*2400 # 1 Cent = 2400 sqft
        if conversion < 50 or conversion > 20000:
            return None
        else:
            return conversion
```

[14]:
```
df4 = df3.copy()
df4['surf_formated'] = df3['total_sqft'].apply(format_surf_area)
```

[15]:
```
df4 = df4.dropna() # To remove the outliers with surface not included in [50,␣
 ↪20000] sqft.
df4 = df4.drop('total_sqft', axis = 'columns')
df4.head()
```

[15]:
```
              area_type                  location  bath  balcony   price  \
0  Super built-up  Area  Electronic City Phase II   2.0      1.0   39.07
1            Plot  Area            Chikka Tirupathi   5.0      3.0  120.00
2        Built-up  Area               Uttarahalli   2.0      3.0   62.00
3  Super built-up  Area        Lingadheeranahalli   3.0      1.0   95.00
4  Super built-up  Area                  Kothanur   2.0      1.0   51.00

   bedroom_formated  surf_formated
0                 2         1056.0
1                 4         2600.0
2                 3         1440.0
3                 3         1521.0
4                 2         1200.0
```

[16]:
```
df4.isna().sum()
```

[16]:
```
area_type           0
location            0
bath                0
balcony             0
price               0
bedroom_formated    0
surf_formated       0
dtype: int64
```

At this moment, the values in this dataframe must be well-formated. We still have to One-Hot encode the categorical features.

[17]:
```
oh_features = pd.get_dummies(df4[['area_type', 'location']])
```

[18]:
```
oh_features.shape
```

[18]: (12697, 1264)

We encounter an issue here : there are too much different locations, OH encoding would create 1260+ features. To tackle this issue, we will group the locations with less than 50 occurrences in a category 'Other'.

[19]:
```
df_tmp = df4.groupby('location')['location'].agg('count')
```

[20]:
```
list_loc_ok = df_tmp[df_tmp>50].sort_values(ascending=False)
```

[21]:
```
def format_location(loc : str) -> str:
    if loc in list_loc_ok:
        return loc
    else:
        return 'Other'
```

[22]:
```
df5 = df4.copy()
df5['loc_formated'] = df4['location'].apply(format_location)
df5 = df5.drop('location', axis = 'columns')
```

[23]:
```
df5.groupby('loc_formated')['loc_formated'].agg('count')
```

[23]:
```
loc_formated
7th Phase JP Nagar          147
8th Phase JP Nagar           56
Akshaya Nagar                58
Banashankari                 74
Bannerghatta Road           144
Begur Road                   83
Bellandur                    91
Bisuvanahalli                51
Budigere                     54
Chandapura                   98
Electronic City             300
Electronic City Phase II    130
Electronics City Phase 1     86
Haralur Road                135
Harlur                       76
Hebbal                      173
Hennur                       51
Hennur Road                 142
Hoodi                        86
Hormavu                      71
Hosa Road                    72
JP Nagar                     64
Jakkur                       67
KR Puram                     85
Kaggadasapura                61
```

```
Kanakpura Road              259
Kasavanhalli                 77
Kengeri                      71
Koramangala                  69
Kothanur                     59
Malleshwaram                 52
Marathahalli                164
Nagarbhavi                   62
Old Madras Road              69
Other                      7060
Panathur                     51
Rachenahalli                 54
Raja Rajeshwari Nagar       168
Rajaji Nagar                 99
Ramamurthy Nagar             72
Sarjapur                     81
Sarjapur  Road              372
TC Palaya                    60
Thanisandra                 231
Thigalarapalya               60
Uttarahalli                 186
Varthur                      68
Whitefield                  514
Yelahanka                   206
Yeshwanthpur                 78
Name: loc_formated, dtype: int64
```

[24]:
```
df5.groupby('area_type')['area_type'].agg('count')
```

[24]:
```
area_type
Built-up  Area          2308
Carpet  Area              82
Plot  Area              1826
Super built-up  Area    8481
Name: area_type, dtype: int64
```

No need to do the same operation with the 'area_type' feature since there are only 4 different values.

Now, we have 50 different locations. We will keep the columns 'loc_formated' and 'area_type' for the moment because it is more convenient for outliers removing.

[25]:
```
df6 = df5.copy()
```

The dataset is well-formated. We still have to remove the outliers in order to improve the future model we will use. To do so, I will introduce two temporary features : price per sqft, number of sqft per bedroom.

- We will get rid of the rows with a Z-score between -1 and 1 for the price_per_sqft (there are not too many values with a Z-score higher than 1, hence the choice of the restrictive filter).

- We discard the rows with an area_per_bedroom lower than 300 sqft.
- A final filter will be applied on the bath number. Arbitrarily, we choose to discard the rows with more bathrooms than bedrooms.

[26]:
```
df6['price_per_sqft'] = df6.price / df6.surf_formated
df6['area_per_bedroom'] = df6.surf_formated / df6.bedroom_formated
```

[27]:
```
df6.shape
```

[27]:
```
(12697, 9)
```

[28]:
```
from scipy import stats
```

[29]:
```
df_outliers_removing = df6.copy()
df_outliers_removing =␣
↪df_outliers_removing[~(df_outliers_removing['area_per_bedroom']<300)]
```

[30]:
```
df_outliers_removing.shape
```

[30]:
```
(12038, 9)
```

[31]:
```
df_outliers_removing['zscore_price_per_sqft'] = df_outliers_removing.
↪groupby('loc_formated')['price_per_sqft'].transform(lambda x: stats.
↪zscore(x))
```

[32]:
```
df_outliers_removing.head()
```

[32]:
```
              area_type  bath  balcony   price  bedroom_formated  \
0  Super built-up  Area   2.0      1.0   39.07                 2
1            Plot  Area   5.0      3.0  120.00                 4
2        Built-up  Area   2.0      3.0   62.00                 3
3  Super built-up  Area   3.0      1.0   95.00                 3
4  Super built-up  Area   2.0      1.0   51.00                 2

   surf_formated              loc_formated  price_per_sqft  area_per_bedroom  \
0         1056.0  Electronic City Phase II        0.036998             528.0
1         2600.0                     Other        0.046154             650.0
2         1440.0               Uttarahalli        0.043056             480.0
3         1521.0                     Other        0.062459             507.0
4         1200.0                  Kothanur        0.042500             600.0

   zscore_price_per_sqft
0              -0.057460
1              -0.416340
2               0.010446
3              -0.070597
4              -0.725487
```

[33]:
```
df_outliers_removing =␣
↪df_outliers_removing[abs(df_outliers_removing['zscore_price_per_sqft'])<1] #␣
↪Outliers price per square ft
print(df_outliers_removing.shape)
```

(10373, 10)

[34]:
```
df_outliers_removing =␣
↪df_outliers_removing[df_outliers_removing['bath']<=df_outliers_removing['bedroom_formated']]
↪# Outliers bath numbers
print(df_outliers_removing.shape)
```

(9858, 10)

At this point, all that remains is to One-Hot encode the categorical features so that they can be used by the models.

[35]:
```
dummies_var = pd.get_dummies(df_outliers_removing[['area_type',␣
↪'loc_formated']], drop_first = True)
```

[36]:
```
df7 = pd.concat([df_outliers_removing, dummies_var], axis = 'columns')
```

[37]:
```
df7.shape
```

[37]: (9858, 62)

[38]:
```
print(f'After removing the outliers, {df7.shape[0]} houses remain, out of the␣
↪original {df.shape[0]}.')
```

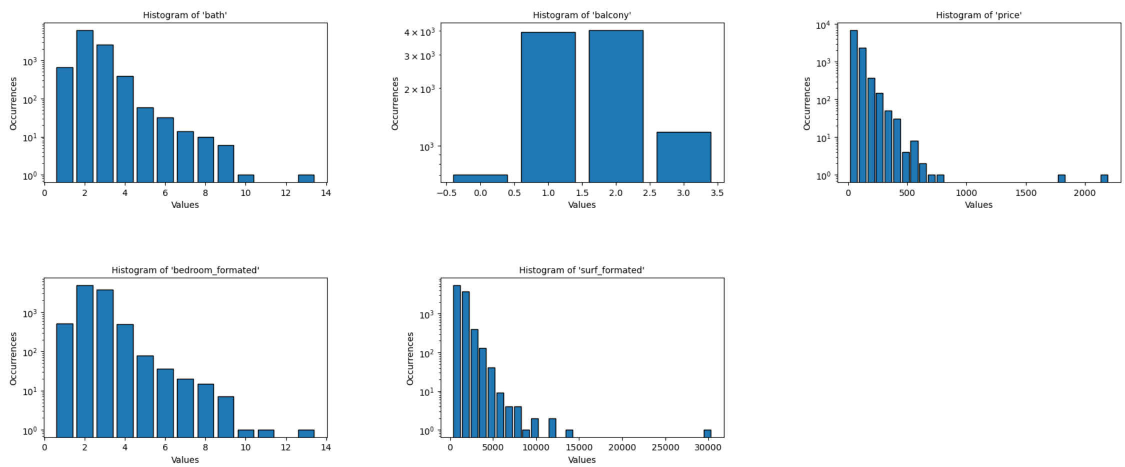
After removing the outliers, 9858 houses remain, out of the original 13320.

Data Visualization

[39]:
```
list_columns_to_visualize = ['bath', 'balcony', 'price', 'bedroom_formated',␣
↪'surf_formated']
```

[40]:
```
discrete_columns = ['bath', 'balcony', 'bedroom_formated']
continuous_columns = ['price', 'surf_formated']

k = 0
plt.figure(figsize=(18, 12))

for col_name in list_columns_to_visualize:
    k += 1
    plt.subplot(3, 3, k)
    if col_name in discrete_columns:
        min_value = df7[col_name].min()
        max_value = df7[col_name].max()
        bins = np.arange(min_value - 0.5, max_value + 1.5, 1)
    else:
```

```
        bins = 30  # Arbitrarily chosen

    plt.hist(df7[col_name], rwidth=0.8, bins=bins, edgecolor='black')
    plt.xlabel('Values')
    plt.ylabel('Occurrences')
    plt.yscale('log')
    plt.title(f"Histogram of '{col_name}'", fontsize=10)

plt.tight_layout()
plt.subplots_adjust(top=0.9, hspace=0.6, wspace=0.4)
plt.show()
```



Model Conception

Now that the data has been properly prepared, we can build the model we'll use for our price prediction. To do this, we'll compare several models: Linear regression, SVR, Random Forest Regressor and Lasso.

To do this, we'll be using RandomizedSearchCV rather than GridSearchCV (to avoid time-consuming calculations on my personal computer).

I've also looked into other options, such as standardizing features with the following code:

 # from sklearn.preprocessing import StandardScaler
 # ss = StandardScaler()
 # X[['bath','balcony','bedroom_formated','surf_formated']] = ss.fit_transform(X[['bath','balcony','bedroom_forma

Or PCA dimension reduction with this code:
 # from sklearn.decomposition import PCA
 # pca = PCA(0.95) # X_pca = pca.fit_transform(X)

Standardizing values doesn't increase performance, and using PCA slightly reduces training time, but also model accuracy.

[41]:
```
from sklearn.model_selection import train_test_split, GridSearchCV,␣
↪RandomizedSearchCV
from sklearn.linear_model import Lasso, LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
```

[42]:
```
X = df7.drop(['price', 'area_type', 'loc_formated', 'price_per_sqft',␣
↪'area_per_bedroom', 'zscore_price_per_sqft'], axis = 'columns')
y = df7.price
```

[43]:
```
X.head()
```

[43]:
```
   bath  balcony  bedroom_formated  surf_formated  area_type_Carpet  Area  \
0   2.0      1.0                 2         1056.0                    False
2   2.0      3.0                 3         1440.0                    False
3   3.0      1.0                 3         1521.0                    False
4   2.0      1.0                 2         1200.0                    False
5   2.0      1.0                 2         1170.0                    False

   area_type_Plot  Area  area_type_Super built-up  Area  \
0                 False                             True
2                 False                            False
3                 False                             True
4                 False                             True
5                 False                             True

   loc_formated_8th Phase JP Nagar  loc_formated_Akshaya Nagar  \
0                            False                       False
2                            False                       False
3                            False                       False
4                            False                       False
5                            False                       False

   loc_formated_Banashankari  …  loc_formated_Sarjapur  \
0                      False  …                  False
2                      False  …                  False
3                      False  …                  False
4                      False  …                  False
5                      False  …                  False

   loc_formated_Sarjapur  Road  loc_formated_TC Palaya  \
0                        False                   False
2                        False                   False
3                        False                   False
4                        False                   False
5                        False                   False
```

```
   loc_formated_Thanisandra  loc_formated_Thigalarapalya  \
0                     False                        False
2                     False                        False
3                     False                        False
4                     False                        False
5                     False                        False

   loc_formated_Uttarahalli  loc_formated_Varthur  loc_formated_Whitefield  \
0                     False                 False                    False
2                      True                 False                    False
3                     False                 False                    False
4                     False                 False                    False
5                     False                 False                     True

   loc_formated_Yelahanka  loc_formated_Yeshwanthpur
0                   False                      False
2                   False                      False
3                   False                      False
4                   False                      False
5                   False                      False

[5 rows x 56 columns]
```

[44]:

```python
parameters = {
    'svr' : {
    'model' : SVR(),
    'params' : {
        'kernel':['rbf', 'linear'],
        'gamma':['auto','scale']
        }
    },
    'rf' : {
    'model' : RandomForestRegressor(),
    'params' : {
        'n_estimators': [50, 100],
        'criterion' :['squared_error','absolute_error'],
        'max_depth': [None, 10, 100],
        'max_features': ['log2', 'sqrt']
        }
    },
    'lr' : {
    'model' : LinearRegression(),
    'params' : {
    'fit_intercept': [True, False]
        }
    },
```

```
    'lasso' : {
    'model' : Lasso(max_iter=1000),
    'params' : {
        'alpha': [0.001, 0.01, 0.1, 1],
        'fit_intercept': [True, False]
        }
    },
    'knn': {
        'model': KNeighborsRegressor(),
        'params': {
            'n_neighbors': [3, 5, 7, 10],
            'weights': ['uniform', 'distance'],
            'algorithm': ['auto', 'ball_tree', 'kd_tree']
        }
    }
}
```

[45]:

```
%%time
for model_name, model_params in parameters.items():
    if model_name == 'svr':
        nb_max_iter = 4
    elif model_name == 'rf':
        nb_max_iter = 12
    elif model_name == 'lr':
        nb_max_iter = 2
    elif model_name == 'lasso':
        nb_max_iter = 8
    elif model_name == 'knn':
        nb_max_iter = 10
    rscv = RandomizedSearchCV(model_params['model'], model_params['params'], cv␣
 ↪= 5, verbose = 1, n_iter=nb_max_iter, n_jobs = -1)
    rscv.fit(X, y)
    print(f'Les meilleures performances de {model_name} sont de {rscv.
 ↪best_score_}, atteintes avec les paramètres {rscv.best_params_}. \n')
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Les meilleures performances de svr sont de 0.729494356069259, atteintes avec les
paramètres {'kernel': 'linear', 'gamma': 'auto'}.

Fitting 5 folds for each of 12 candidates, totalling 60 fits
Les meilleures performances de rf sont de 0.6987818459473016, atteintes avec les
paramètres {'n_estimators': 50, 'max_features': 'sqrt', 'max_depth': 100,
'criterion': 'absolute_error'}.

Fitting 5 folds for each of 2 candidates, totalling 10 fits
Les meilleures performances de lr sont de 0.7590223296027843, atteintes avec les
paramètres {'fit_intercept': False}.
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Les meilleures performances de lasso sont de 0.7590303997949256, atteintes avec
les paramètres {'fit_intercept': False, 'alpha': 0.001}.

Fitting 5 folds for each of 10 candidates, totalling 50 fits
Les meilleures performances de knn sont de 0.6708521962575168, atteintes avec
les paramètres {'weights': 'uniform', 'n_neighbors': 3, 'algorithm': 'kd_tree'}.

CPU times: user 34min 50s, sys: 1.88 s, total: 34min 52s
Wall time: 1h 56min 50s
```

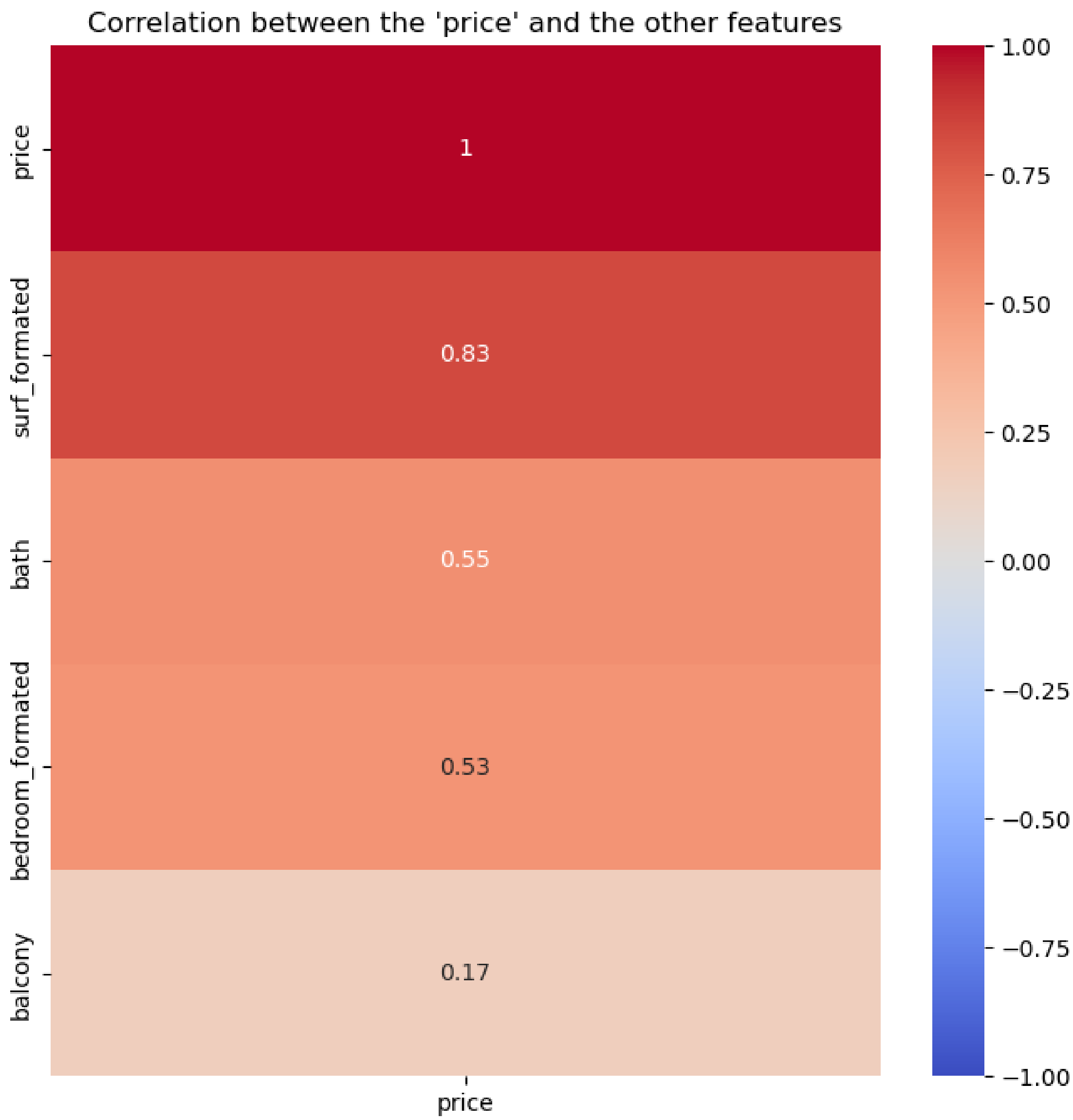
Here, we see that the best model for our regression is linear regression, especially Lasso, with parameters {'fit_intercept': False, 'alpha': 0.001}. This model gives a test accuracy of about 76%.

[66]:
```
from sklearn.ensemble import BaggingRegressor
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)
bag_model = BaggingRegressor(
    estimator = Lasso(fit_intercept = False, alpha = 0.001),
    n_estimators = 100,
    max_samples = 0.8,
    oob_score = True,
    random_state = 53)
bag_model.fit(X_train, y_train)
bag_model.score(X_test, y_test)
```

[66]:
```
0.7451721761619813
```

As this model is stable, and the data have a fairly low variance (due to outliers removing), bagging does not significantly improve performance.

[156]:
```
import seaborn as sn
df_corr = X[['bath',␣
 ↪'balcony',        'bedroom_formated',        'surf_formated']]
df_corr = pd.concat([df_corr, y], axis = 'columns')

corr_matrix = df_corr.corr()

corr_price = corr_matrix[['price']].sort_values(by='price', ascending=False)

plt.figure(figsize=(8, 8))
sn.heatmap(corr_price, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title("Correlation between the 'price' and the other features")
plt.show()
```



Here, we can notice that the price mostly depends on the area of the house. However, the balcony number doesn't seem to impact much the price.

[80]:
```
import joblib
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪random_state=53)

model = Lasso(fit_intercept = False, alpha = 0.001)
model.fit(X_train, y_train)
print(model.score(X_test, y_test))
joblib.dump(model, 'model_regression.pkl')
```

0.8463855336013805

[80]: ['model_regression.pkl']

Here, the model performs better. This is solely due to the distribution chosen between the training set and the test set. We save this model as 'model_regression.pkl'. We then load this model and try to make some predictions

Price predictions

[81]:
```
loaded_model = joblib.load('model_regression.pkl')
```

[154]:
```
def predict_price(bath : int, balcony : int, bedroom : int, area : int,␣
↪location : str, area_type : str) -> int:
    area_type = area_type.replace(' Area', '  Area')
    loc_index = np.where(X.columns=='loc_formated_'+location)[0][0]
    area_type_index = np.where(X.columns=='area_type_'+area_type)[0][0]
    tbl = np.zeros(56)
    tbl[0] = bath
    tbl[1] = balcony
    tbl[2] = bedroom
    tbl[3] = area
    if loc_index >= 0:
        tbl[loc_index]=True
    if area_type_index >=0:
        tbl[area_type_index]=False
    df_tbl = pd.DataFrame([tbl], columns = X.columns)
    price_predicted = round(loaded_model.predict(df_tbl)[0], 2)
    print(f"""The house : \n
        of {area} sqft, \n
        located at {location}, \n
        with {bedroom} bedroom(s), {bath} bathroom(s), {balcony} balcony/
↪balconies, \n
costs about {price_predicted} lakh rupees (about {np.floor(price_predicted *␣
↪1078.21)}€)""")
```

[163]:
```
predict_price(2, 0, 2, 1000, 'Thanisandra','Carpet Area')
```

```
The house :

        of 1000 sqft,

        located at Thanisandra,

        with 2 bedroom(s), 2 bathroom(s), 0 balcony/balconies,

costs about 53.01 lakh rupees (about 57155.0€)
```

[164]:

```
predict_price(7, 2, 6, 7000, 'Sarjapur','Super built-up Area')
```

```
The house :

        of 7000 sqft,

        located at Sarjapur,

        with 6 bedroom(s), 7 bathroom(s), 2 balcony/balconies,

costs about 471.12 lakh rupees (about 507966.0€)
```

[166]:

```
predict_price(3, 2, 2, 5200, 'Sarjapur','Plot Area')
```

```
The house :

        of 5200 sqft,

        located at Sarjapur,

        with 2 bedroom(s), 3 bathroom(s), 2 balcony/balconies,

costs about 350.25 lakh rupees (about 377643.0€)
```

[ ]:

```
```