Harshith Narasimhamurthy

Project on predicting House Rent in Bangalore city

In [1]:

```
import pandas as pd
```

importing the dataframe using pandas

In [2]:

```
df = pd.read_csv("C:/Users/Lenovo/Downloads/ban.csv")
```

Printing the first 5 rows of dataframe

In [3]:

```
df.head()
```

Out[3]:

	bedroom	bathroom	layout_type	property_type	price	area	furnish_type
0	2	2	BHK	Independent Floor	20000.0	1140	Semi-Furnished
1	2	2	BHK	Independent House	8000.0	840	Semi-Furnished
2	2	2	BHK	Independent House	21000.0	1000	Semi-Furnished
3	1	1	BHK	Apartment	9000.0	550	Semi-Furnished
4	3	3	BHK	Apartment	17000.0	1230	Furnished

Printing the last 5 rows of dataframe

In [4]:

```
df.tail()
```

Out[4]:

	bedroom	bathroom	layout_type	property_type	price	area	furnish_type
22692	1	1	BHK	Independent Floor	6000.0	500	Semi-Furnished
22693	2	1	BHK	Independent House	10500.0	400	Semi-Furnished
22694	2	2	BHK	Independent House	16000.0	1100	Semi-Furnished
22695	2	2	BHK	Apartment	24000.0	1000	Semi-Furnished
22696	3	3	BHK	Independent Floor	19000.0	1200	Semi-Furnished

Checking for shape of our dataframe

In [5]:

```
df.shape
```

Out[5]:

```
(22697, 7)
```

Listing out the columns names of dataframe

In [6]:

```
df.columns
```

Out[6]:

```
Index(['bedroom', 'bathroom', 'layout_type', 'property_type', 'price', 'area',
       'furnish_type'],
      dtype='object')
```

```
Checking out for the missing values in dataframe
```

In [7]:

```
df.isnull().sum()
```

Out[7]:

```
bedroom          0
bathroom         0
layout_type      0
property_type    0
price            0
area             0
furnish_type     0
dtype: int64
```

Getting the information about data types in dataframe

In [8]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22697 entries, 0 to 22696
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   bedroom        22697 non-null  int64  
 1   bathroom       22697 non-null  int64  
 2   layout_type    22697 non-null  object 
 3   property_type  22697 non-null  object 
 4   price          22697 non-null  float64
 5   area           22697 non-null  int64  
 6   furnish_type   22697 non-null  object 
dtypes: float64(1), int64(3), object(3)
memory usage: 1.2+ MB
```

Getting the statistics regarding our dataframe

In [9]:

```
df.describe()
```

Out[9]:

	bedroom	bathroom	price	area
count	22697.000000	22697.000000	22697.000000	22697.000000
mean	1.923470	1.820152	19792.939596	1098.033088
std	0.777335	0.753264	12088.477756	541.019284
min	1.000000	1.000000	1500.000000	100.000000
25%	1.000000	1.000000	11500.000000	650.000000
50%	2.000000	2.000000	17000.000000	1100.000000
75%	2.000000	2.000000	25000.000000	1350.000000
max	6.000000	4.000000	99000.000000	5500.000000

Checking out for number of unique value present in each columns of dataframe

In [10]:

```
df.nunique()
```

Out[10]:

```
bedroom             6
bathroom            4
layout_type         2
property_type       6
price             547
area             1035
furnish_type        3
dtype: int64
```

In order to visualise the data, I am importing below libraries

In [11]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

checking out for unique value for layout type column

In [12]:

```
df['layout_type'].unique()
```

Out[12]:

```
array(['BHK', 'RK'], dtype=object)
```

getting the numbers for each unique characters

In [13]:

```
df['layout_type'].value_counts()
```

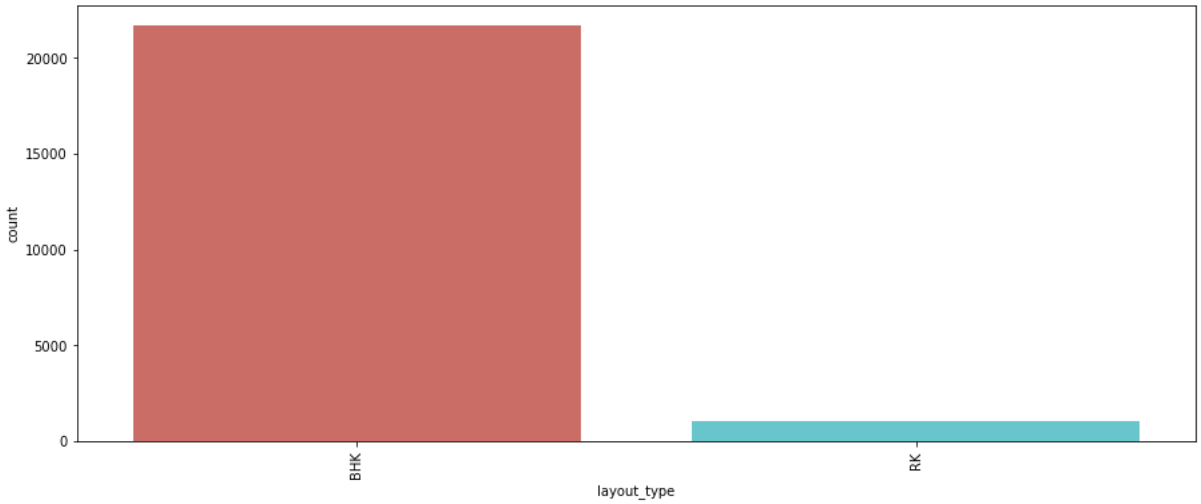
Out[13]:

```
BHK    21675
RK      1022
Name: layout_type, dtype: int64
```

Visualising through graphs

In [14]:

```
plt.figure(figsize=(15,6))
sns.countplot('layout_type', data = df, palette = 'hls')
plt.xticks(rotation = 90)
plt.show()
```



checking out for unique value for property type column

In [15]:

```
df['property_type'].unique()
```

Out[15]:

```
array(['Independent Floor', 'Independent House', 'Apartment',
       'Studio Apartment', 'Villa', 'Penthouse'], dtype=object)
```

getting the numbers for each unique characters

In [16]:

```
df['property_type'].value_counts()
```

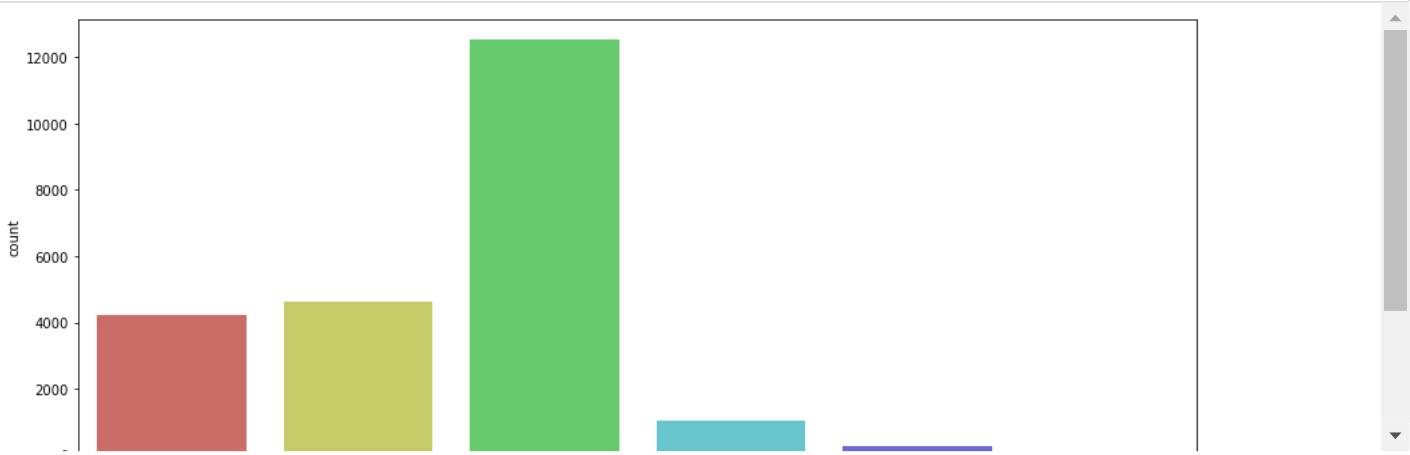
Out[16]:

```
Apartment            12533
Independent House     4627
Independent Floor     4237
Studio Apartment      1022
Villa                  257
Penthouse               21
Name: property_type, dtype: int64
```

Visualising through graphs

In [17]:

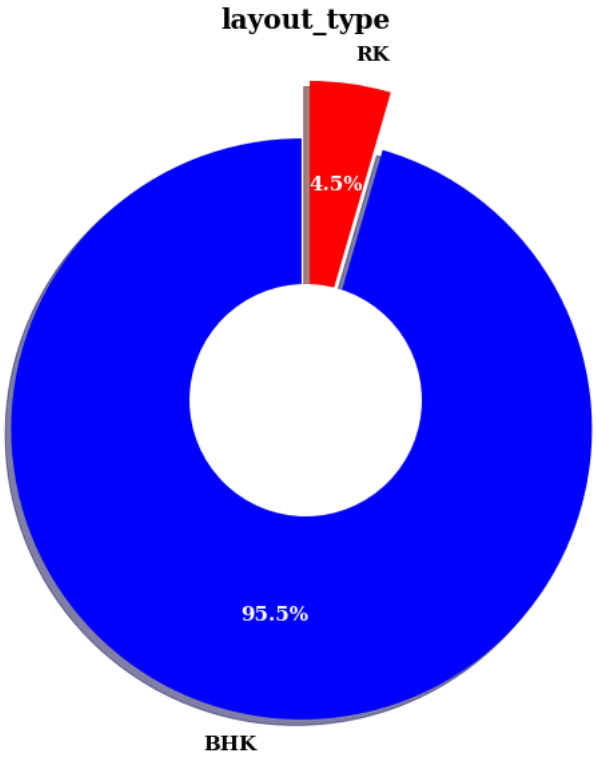
```
plt.figure(figsize=(15,6))
sns.countplot('property_type', data = df, palette = 'hls')
plt.xticks(rotation = 90)
plt.show()
```



Getting donut graph representing the percentage share of each characters in layout type column

In [18]:

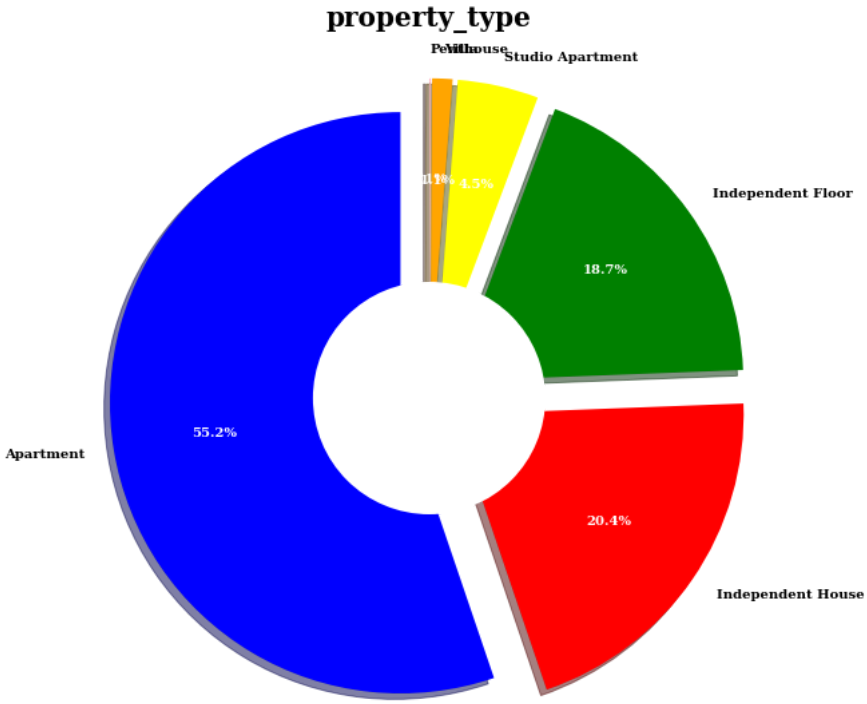
```
label_data = df['layout_type'].value_counts()
explode = (0.1, 0.1)
plt.figure(figsize=(14, 10))
patches, texts, pcts = plt.pie(label_data,
 labels = label_data.index,
 colors = ['blue', 'red'],
 pctdistance = 0.65,
 shadow = True,
 startangle = 90,
 explode = explode,
 autopct = '%1.1f%%',
 textprops={ 'fontsize': 15,
 'color': 'black',
 'weight': 'bold',
 'family': 'serif' })
plt.setp(pcts, color='white')
hfont = {'fontname':'serif', 'weight': 'bold'}
plt.title('layout_type', size=20, **hfont)
centre_circle = plt.Circle((0,0),0.40,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.show()
```



Getting donut graph representing the percentage share of each characters in property type column

In [19]:

```
label_data = df['property_type'].value_counts()
explode = (0.1, 0.1, 0.1, 0.1, 0.1, 0.1)
plt.figure(figsize=(14, 10))
patches, texts, pcts = plt.pie(label_data,
 labels = label_data.index,
 colors = ['blue', 'red', 'green', 'yellow', 'orange', 'pink'],
 pctdistance = 0.65,
 shadow = True,
 startangle = 90,
 explode = explode,
 autopct = '%1.1f%%',
 textprops={ 'fontsize': 10,
 'color': 'black',
 'weight': 'bold',
 'family': 'serif' })
plt.setp(pcts, color='white')
hfont = {'fontname':'serif', 'weight': 'bold'}
plt.title('property_type', size=20, **hfont)
centre_circle = plt.Circle((0,0),0.40,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.show()
```



checking out for unique value for furnish type column

In [20]:

```
df['furnish_type'].unique()
```

Out[20]:

```
array(['Semi-Furnished', 'Furnished', 'Unfurnished'], dtype=object)
```

getting the numbers for each unique characters

In [21]:

```
df['furnish_type'].value_counts()
```

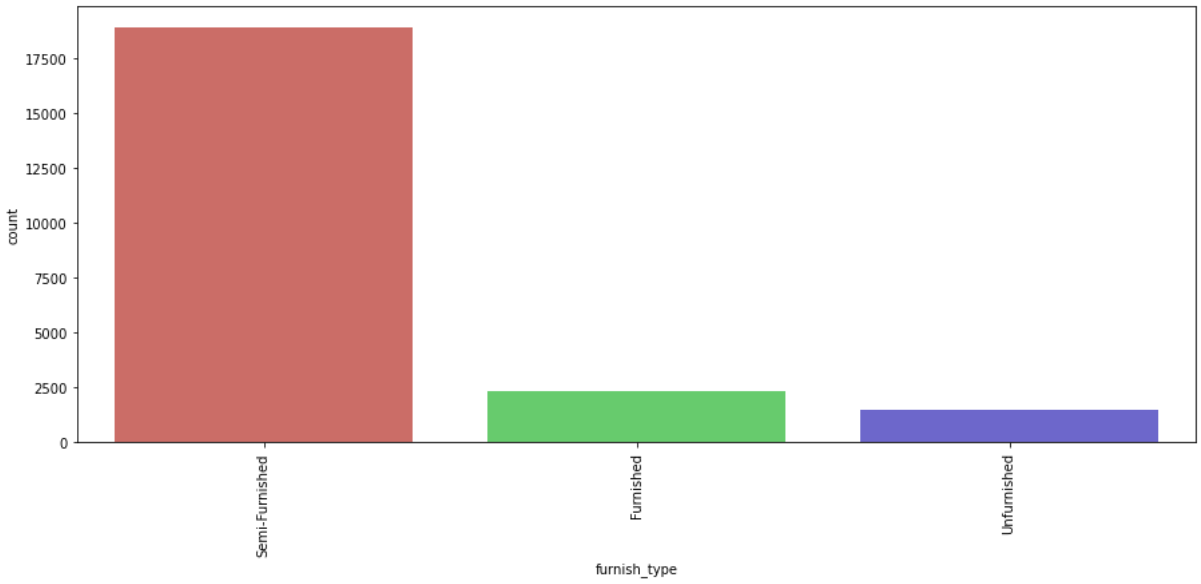
Out[21]:

```
Semi-Furnished    18900
Furnished          2309
Unfurnished        1488
Name: furnish_type, dtype: int64
```

Visualising through graphs

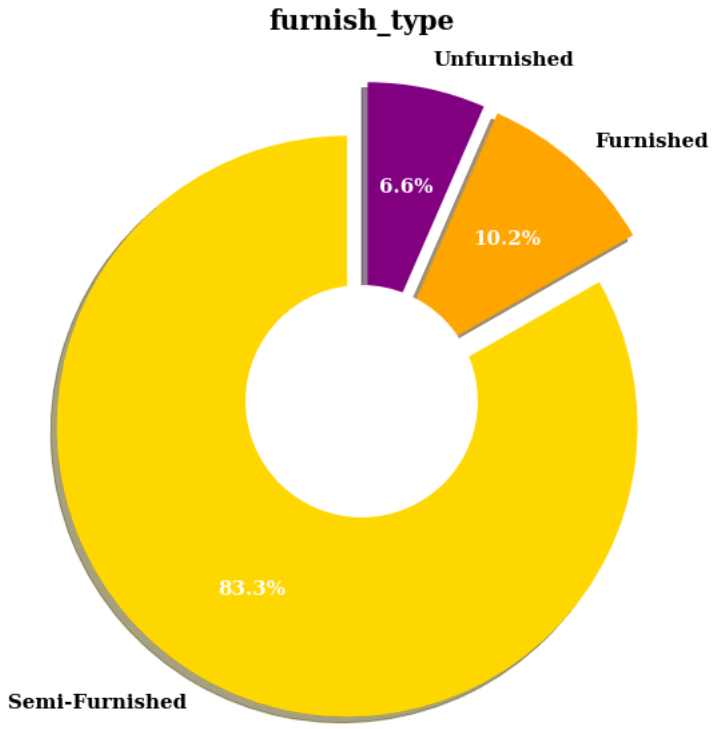
In [22]:

```
plt.figure(figsize=(15,6))
sns.countplot('furnish_type', data = df, palette = 'hls')
plt.xticks(rotation = 90)
plt.show()
```



In [23]:

```
label_data = df['furnish_type'].value_counts()
explode = (0.1, 0.1, 0.1)
plt.figure(figsize=(14, 10))
patches, texts, pcts = plt.pie(label_data,
 labels = label_data.index,
 colors = ['gold', 'orange','purple'],
 pctdistance = 0.65,
 shadow = True,
 startangle = 90,
 explode = explode,
 autopct = '%1.1f%%',
 textprops={ 'fontsize': 15,
 'color': 'black',
 'weight': 'bold',
 'family': 'serif' })
plt.setp(pcts, color='white')
hfont = {'fontname':'serif', 'weight': 'bold'}
plt.title('furnish_type', size=20, **hfont)
centre_circle = plt.Circle((0,0),0.40,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.show()
```



checking out for unique value for number of bedrooms column

In [24]:

```
df['bedroom'].unique()
```

Out[24]:

```
array([2, 1, 3, 4, 5, 6], dtype=int64)
```

getting the numbers for each unique characters

In [25]:

```
df['bedroom'].value_counts()
```

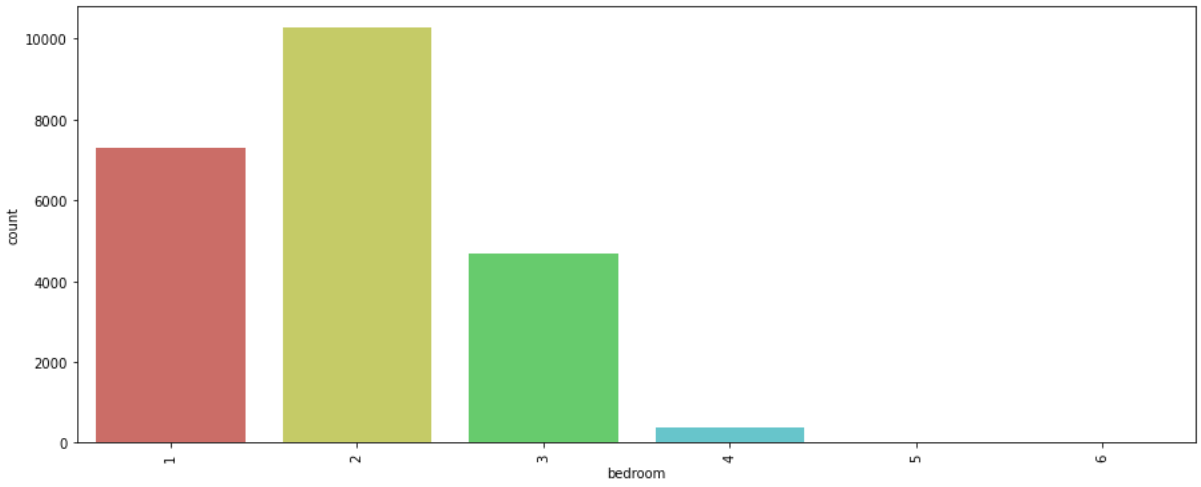
Out[25]:

```
2    10286
1     7299
3     4698
4      385
5       22
6        7
Name: bedroom, dtype: int64
```

Visualising through graphs

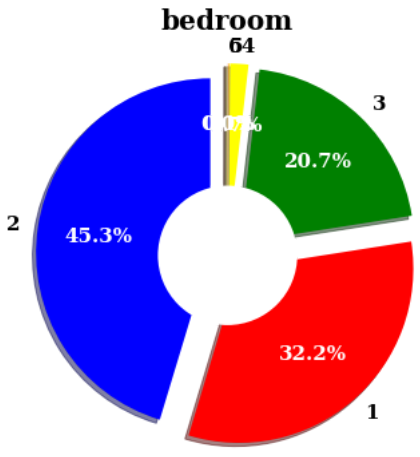
In [26]:

```
plt.figure(figsize=(15,6))
sns.countplot('bedroom', data = df, palette = 'hls')
plt.xticks(rotation = 90)
plt.show()
```



In [27]:

```
label_data = df['bedroom'].value_counts()
explode = (0.1, 0.1, 0.1, 0.1, 0.1, 0.1)
plt.figure(figsize=(14, 6))
patches, texts, pcts = plt.pie(label_data,
 labels = label_data.index,
 colors = ['blue', 'red', 'green', 'yellow', 'orange', 'pink'],
 pctdistance = 0.65,
 shadow = True,
 startangle = 90,
 explode = explode,
 autopct = '%1.1f%%',
 textprops={ 'fontsize': 15,
 'color': 'black',
 'weight': 'bold',
 'family': 'serif' })
plt.setp(pcts, color='white')
hfont = {'fontname':'serif', 'weight': 'bold'}
plt.title('bedroom', size=20, **hfont)
centre_circle = plt.Circle((0,0),0.40,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.show()
```



checking out for unique value for number of bathrooms column

In [28]:

```
df['bathroom'].unique()
```

Out[28]:

```
array([2, 1, 3, 4], dtype=int64)
```

getting the numbers for each unique characters

In [29]:

```
df['bathroom'].value_counts()
```

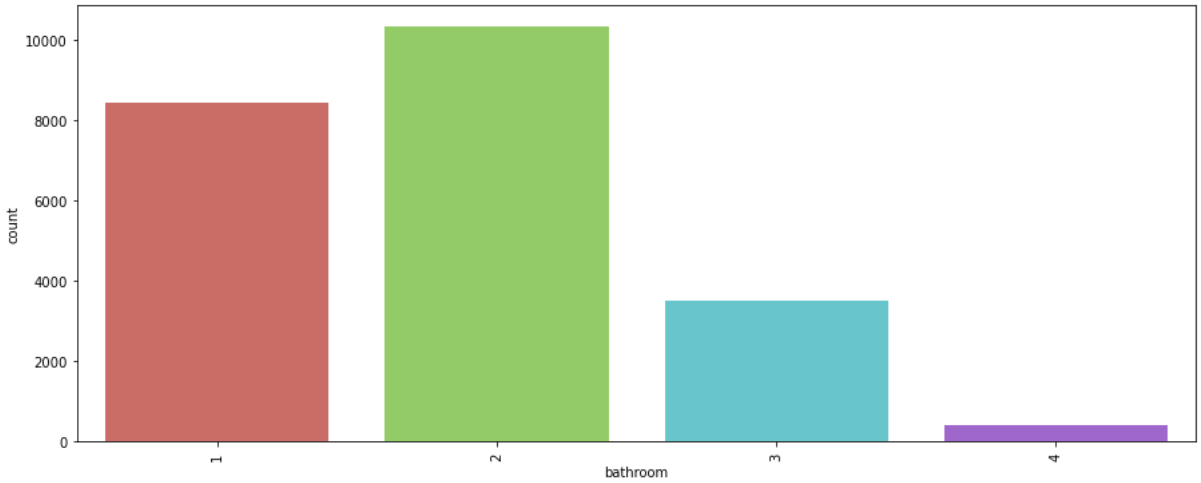
Out[29]:

```
2    10342
1     8428
3     3508
4      419
Name: bathroom, dtype: int64
```

Visualising through graphs

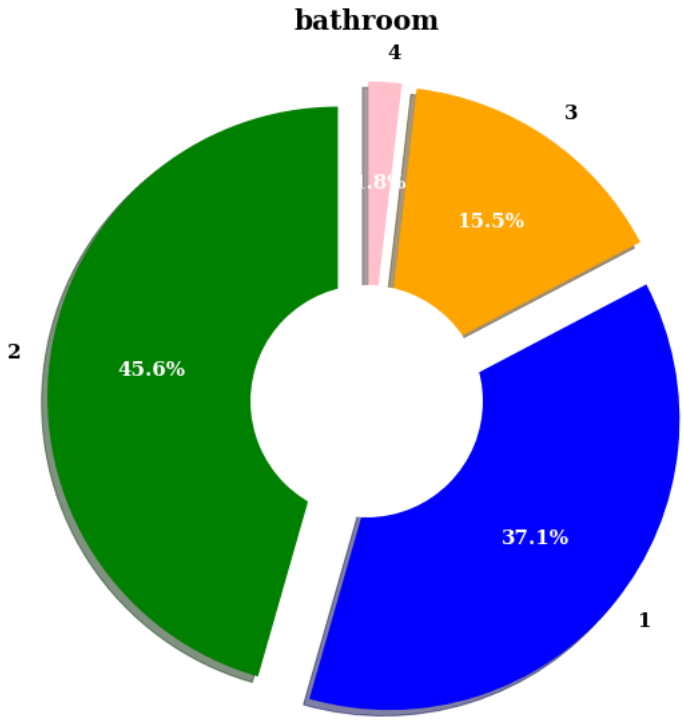
In [30]:

```
plt.figure(figsize=(15,6))
sns.countplot('bathroom', data = df, palette = 'hls')
plt.xticks(rotation = 90)
plt.show()
```



In [31]:

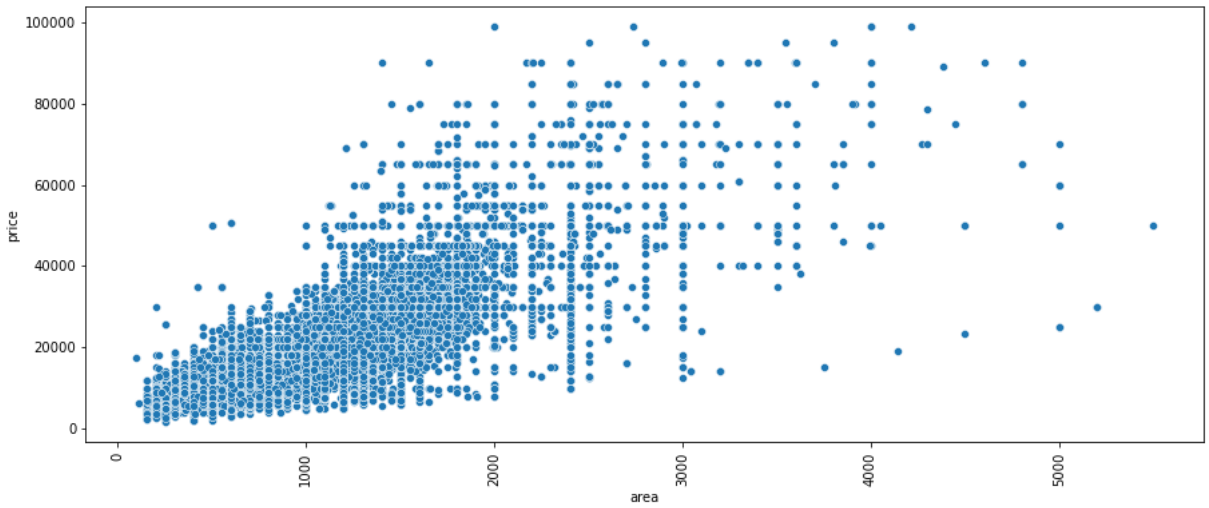
```
label_data = df['bathroom'].value_counts()
explode = (0.1, 0.1, 0.1, 0.1)
plt.figure(figsize=(14, 10))
patches, texts, pcts = plt.pie(label_data,
 labels = label_data.index,
 colors = ['green', 'blue', 'orange', 'pink'],
 pctdistance = 0.65,
 shadow = True,
 startangle = 90,
 explode = explode,
 autopct = '%1.1f%%',
 textprops={ 'fontsize': 15,
 'color': 'black',
 'weight': 'bold',
 'family': 'serif' })
plt.setp(pcts, color='white')
hfont = {'fontname':'serif', 'weight': 'bold'}
plt.title('bathroom', size=20, **hfont)
centre_circle = plt.Circle((0,0),0.40,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.show()
```



Visualising the relationship between area and price through scatterplot graph

In [32]:

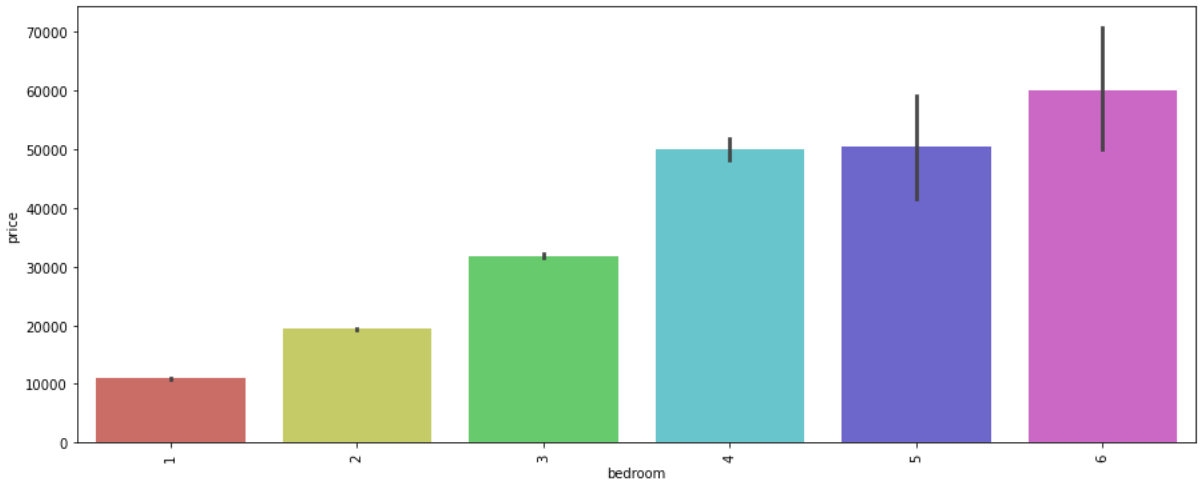
```
plt.figure(figsize=(15,6))
sns.scatterplot(x= df['area'], y = df['price'], data = df,
 palette = 'hls')
plt.xticks(rotation = 90)
plt.show()
```



Visualising the relationship between number of bedrooms and price through barplot graph

In [33]:

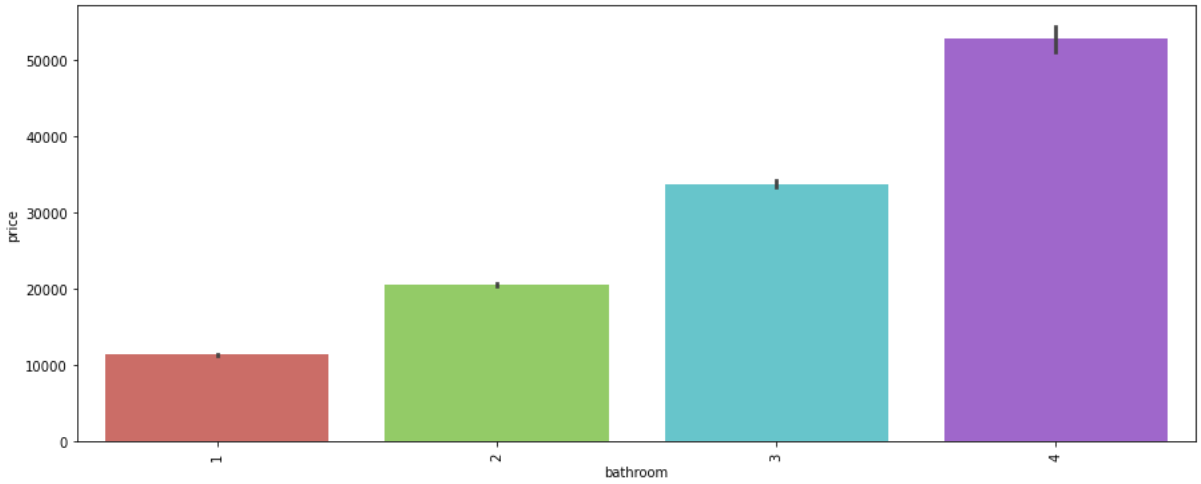
```
plt.figure(figsize=(15,6))
sns.barplot(y = df['price'], x = df['bedroom'], data = df,
 palette = 'hls')
plt.xticks(rotation = 90)
plt.show()
```



Visualising the relationship between number of bathrooms and price through barplot graph

In [34]:

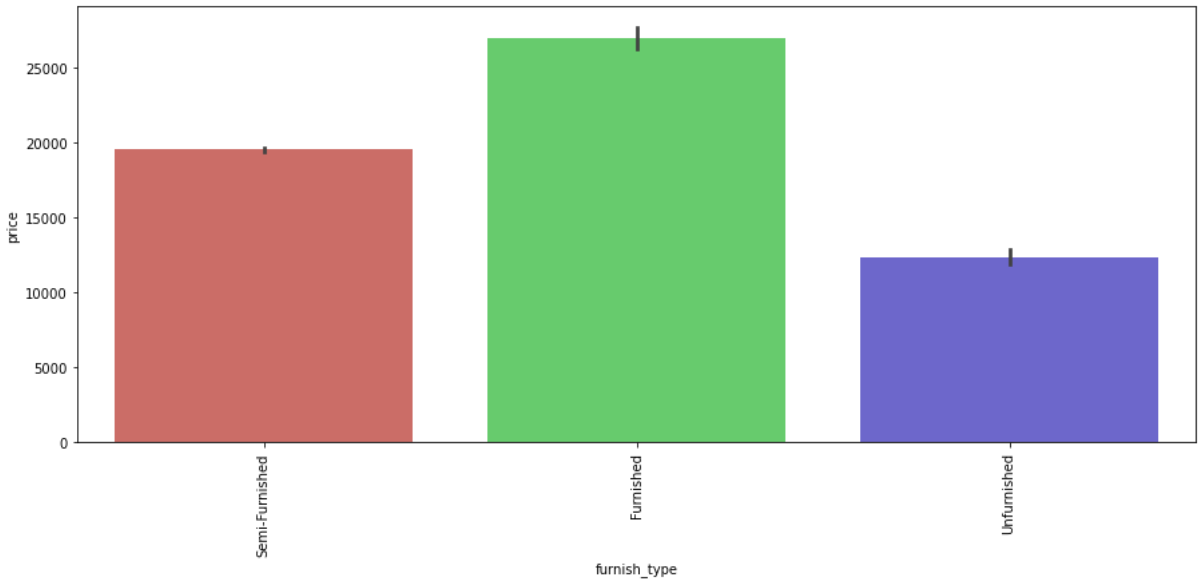
```
plt.figure(figsize=(15,6))
sns.barplot(y = df['price'], x = df['bathroom'], data = df,
 palette = 'hls')
plt.xticks(rotation = 90)
plt.show()
```



Visualising the relationship between furnished conditions and price through barplot graph

In [35]:

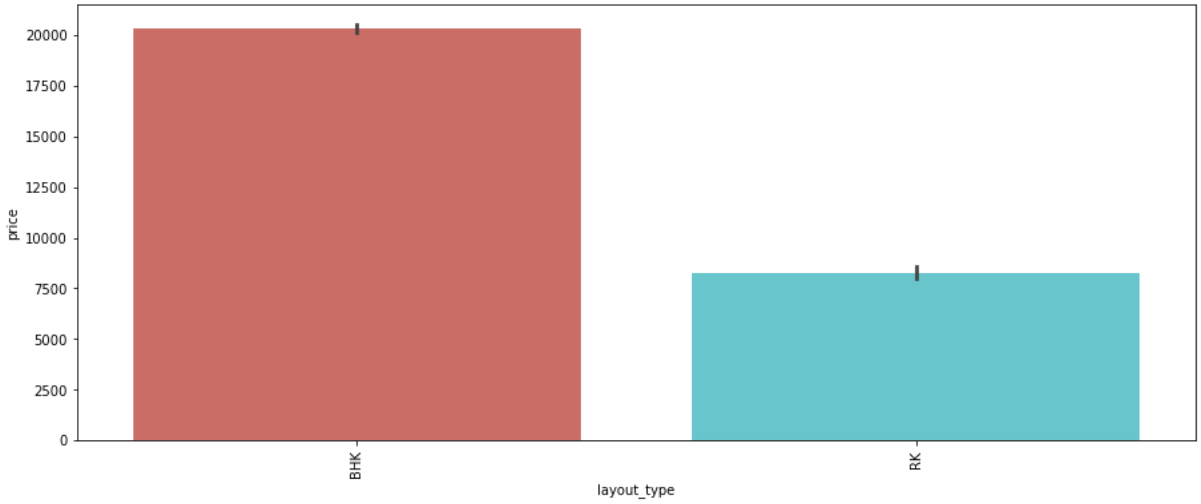
```
plt.figure(figsize=(15,6))
sns.barplot(y = df['price'], x = df['furnish_type'], data = df,
 palette = 'hls')
plt.xticks(rotation = 90)
plt.show()
```



Visualising the relationship between layout type and price through barplot graph

In [36]:

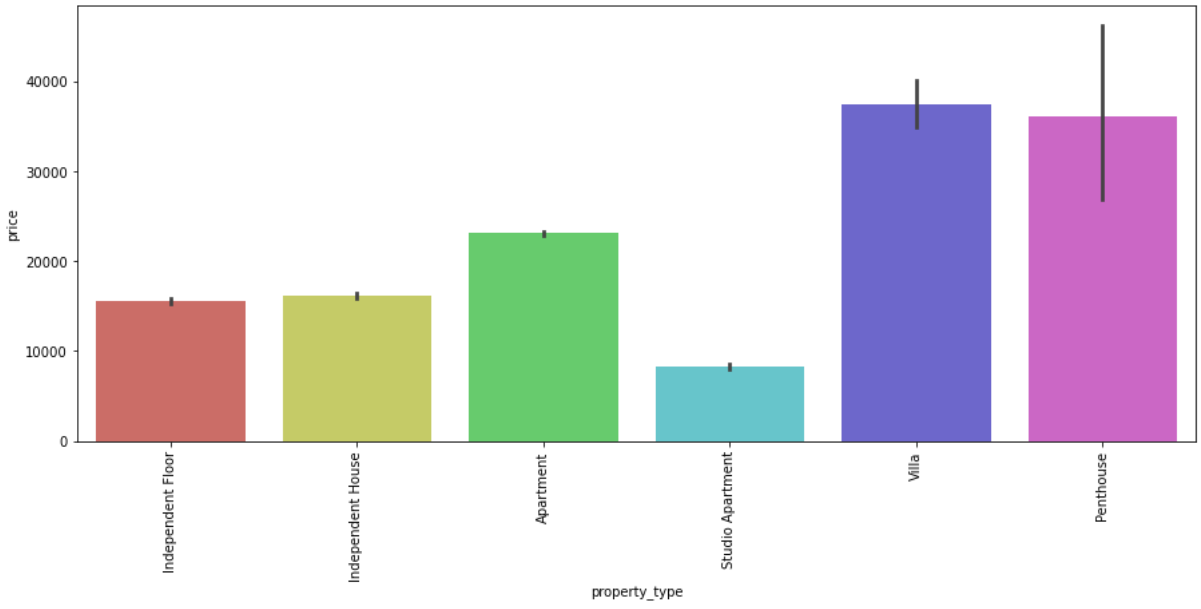
```
plt.figure(figsize=(15,6))
sns.barplot(y = df['price'], x = df['layout_type'], data = df,
 palette = 'hls')
plt.xticks(rotation = 90)
plt.show()
```



Visualising the relationship between property type and price through barplot graph

In [37]:

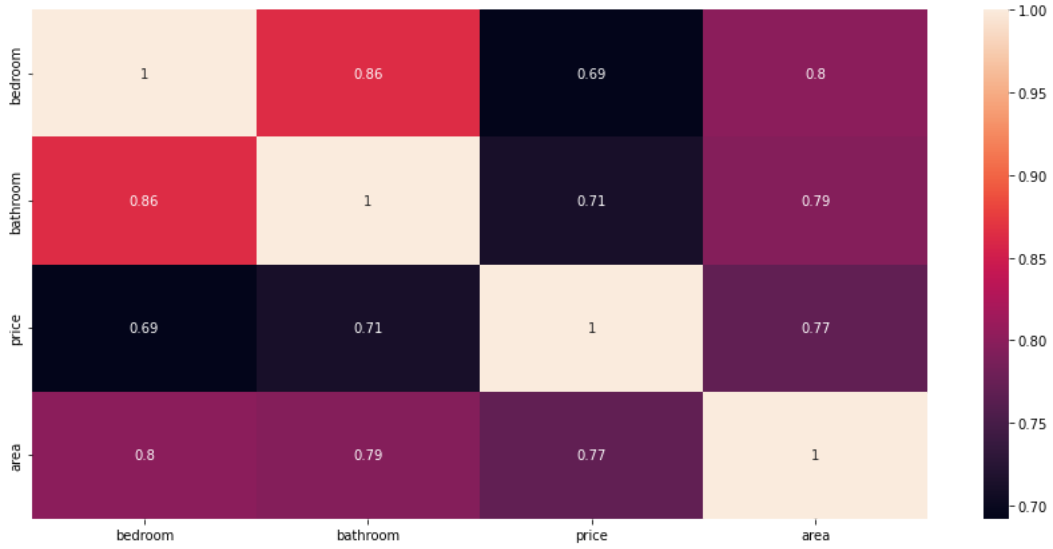
```
plt.figure(figsize=(15,6))
sns.barplot(y = df['price'], x = df['property_type'], data = df,
 palette = 'hls')
plt.xticks(rotation = 90)
plt.show()
```



Creating a heat map correlation matrix to analyse the relation between the variables

In [38]:

```
plt.figure(figsize=(15,7))
sns.heatmap(df.corr(), annot=True)
plt.show()
```



Naming the columns to get dummies and concating the dummies into main dataframe at once

In [39]:

```
cols = ['layout_type','property_type','furnish_type']
```

In [40]:

```
for col in cols:
 one = pd.get_dummies(df[col],prefix=col)
 df = pd.concat([df,one],axis=1).drop(col,axis=1)
```

In [41]:

```
df
```

Out[41]:

	bedroom	bathroom	price	area	layout_type_BHK	layout_type_RK	property_type_Apartment	property_type_Independent Floor	property_type_Independ Ho
0	2	2	20000.0	1140	1	0	0	1	
1	2	2	8000.0	840	1	0	0	0	
2	2	2	21000.0	1000	1	0	0	0	
3	1	1	9000.0	550	1	0	1	0	
4	3	3	17000.0	1230	1	0	1	0	
...	...	...	...	...	...	...	...	...	
22692	1	1	6000.0	500	1	0	0	1	
22693	2	1	10500.0	400	1	0	0	0	
22694	2	2	16000.0	1100	1	0	0	0	
22695	2	2	24000.0	1000	1	0	1	0	
22696	3	3	19000.0	1200	1	0	0	1	

22697 rows × 15 columns

Separating independent and dependent variable from dataframe

In [42]:

```
x = df.drop('price', axis=1)
y = df['price']
```

Importing sklearn model to split datasets into training and testing points

In [43]:

```
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=10)
```

Importing linear regression model and testing its accuracy by fitted the dataset

In [44]:

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train,y_train)
lr.score(X_test,y_test)
```

Out[44]:

```
0.6261528381647584
```

shuffling my datasets into 10 splits and testing with 20 percent of datasets and checking accuracy

In [45]:

```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score

cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)

cross_val_score(LinearRegression(), x, y, cv=cv)
```

Out[45]:

```
array([0.66477548, 0.65493519, 0.64575228, 0.65623674, 0.64553232,
       0.63915711, 0.648529  , 0.62897554, 0.6481302 , 0.64663484])
```

Finding for suitable best model for our dataset

In [46]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor

def find_best_model_using_gridsearchcv(X,y):
    algos = {
        'linear_regression' : {
            'model': LinearRegression(),
            'params': {
                'normalize': [True, False]
            }
        },
        'lasso': {
            'model': Lasso(),
            'params': {
                'alpha': [1,2],
                'selection': ['random', 'cyclic']
            }
        },
        'decision_tree': {
            'model': DecisionTreeRegressor(),
            'params': {
                'criterion' : ['mse','friedman_mse'],
                'splitter': ['best','random']
            }
        }
    }
    scores = []
    cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
    for algo_name, config in algos.items():
        gs =  GridSearchCV(config['model'], config['params'], cv=cv, return_train_score=False)
        gs.fit(X,y)
        scores.append({
            'model': algo_name,
            'best_score': gs.best_score_,
            'best_params': gs.best_params_
        })

    return pd.DataFrame(scores,columns=['model','best_score','best_params'])

find_best_model_using_gridsearchcv(x,y)
```

Out[46]:

	model	best_score	best_params
0	linear_regression	0.653491	{'normalize': True}
1	lasso	0.653440	{'alpha': 1, 'selection': 'cyclic'}
2	decision_tree	0.727870	{'criterion': 'friedman_mse', 'splitter': 'ran...

Importing decission tree regression, fitting the dataset and checking for its accuracy

In [47]:

```
from sklearn.tree import DecisionTreeRegressor
```

In [48]:

```
dt = DecisionTreeRegressor(random_state=0)
```

In [49]:

```
dt.fit(X_train, y_train)
```

Out[49]:

```
DecisionTreeRegressor(random_state=0)
```

In [50]:

```
y_pred = dt.predict(X_test)
```

In [51]:

```
y_pred
```

Out[51]:

```
array([24357.14285714, 22833.33333333, 21165.82352941, ...,
       11735.47599165, 22000.        , 23820.        ])
```

In [52]:

```
y_pred.shape
```

Out[52]:

```
(4540,)
```

In [53]:

```
from sklearn.metrics import mean_squared_error
```

In [54]:

```
RMSE_dt = mean_squared_error(y_test, y_pred, squared = False)
RMSE_dt
```

Out[54]:

```
6654.723497506857
```

In [55]:

```
cross_val_score(dt, X_train, y_train, cv=10)
```

Out[55]:

```
array([0.71695762, 0.72226477, 0.72846749, 0.69142886, 0.70544631,
       0.69447864, 0.7070609 , 0.6942133 , 0.6841565 , 0.71246434])
```

Fitting the entire dataset into decision tree regressor and checking for acuuracy

In [56]:

```
','bathroom','layout_type_BHK','layout_type_RK','property_type_Apartment','property_type_Independent Floor','property_type_Independent Hous
```

Out[56]:

```
DecisionTreeRegressor(random_state=0)
```

In [57]:

```
dt.score(df[['bedroom','area','bathroom','layout_type_BHK','layout_type_RK','property_type_Apartment','property_type_Independent Floor','p
```

Out[57]:

```
0.8221726071468105
```

Predicting the house rent price using above trained model. here I am considering independent house with 500 square feet area having 2 bedrooms, 1 bathroom and BHK type

In [58]:

```
dt.predict([[2,500,1,1,0,0,0,1,0,0,0,1,0,0]])
```

Out[58]:

```
array([16189.23809524])
```

Predicting the price by using user input prefrences

In [59]:

```
bedrooms = int(input('Enter a number of required bedrooms\n'))
area = float(input('Enter a required flat size in square feet\n'))
bathroom = int(input('Enter a number of required bathrooms\n'))
lt1 = int(input('If you wish to have layout type BHK press 1 if not press 0 \n'))
lt2 = int(input('If you wish to have layout type RK press 1 if not press 0 \n'))
pt1 = int(input('If you wish to have property_type_Apartment press 1 if not press 0 \n'))
pt2 = int(input('If you wish to have property_type_Independent Floor press 1 if not press 0 \n'))
pt3 = int(input('If you wish to have property_type_Independent House press 1 if not press 0 \n'))
pt4 = int(input('If you wish to have property_type_Penthouse press 1 if not press 0 \n'))
pt5 = int(input('If you wish to have property_type_Studio Apartment press 1 if not press 0 \n'))
pt6 = int(input('If you wish to have property_type_Villa press 1 if not press 0 \n'))
ft1 = int(input('If you wish to have furnish_type_Furnished press 1 if not press 0 \n'))
ft2 = int(input('If you wish to have furnish_type_Semi-Furnished press 1 if not press 0 \n'))
ft3 = int(input('If you wish to have furnish_type_Unfurnished press 1 if not press 0 \n'))

print(
    "The price amount according to your prefrence will be\n",
     dt.predict([[bedrooms,area,bathroom,lt1,lt2,pt1,pt2,pt3,pt4,pt5,pt6,ft1,ft2,ft3]]),"Rupees")
```

```
Enter a number of required bedrooms
2
Enter a required flat size in square feet
500
Enter a number of required bathrooms
1
If you wish to have layout type BHK press 1 if not press 0
1
If you wish to have layout type RK press 1 if not press 0
0
If you wish to have property_type_Apartment press 1 if not press 0
0
If you wish to have property_type_Independent Floor press 1 if not press 0
0
If you wish to have property_type_Independent House press 1 if not press 0
1
If you wish to have property_type_Penthouse press 1 if not press 0
0
If you wish to have property_type_Studio Apartment press 1 if not press 0
0
If you wish to have property_type_Villa press 1 if not press 0
0
If you wish to have furnish_type_Furnished press 1 if not press 0
1
If you wish to have furnish_type_Semi-Furnished press 1 if not press 0
0
If you wish to have furnish_type_Unfurnished press 1 if not press 0
0
The price amount according to your prefrence will be
 [16189.23809524] Rupees
```

In order to cross check the result obtained above I'm attaching the screenshot of excel sheet of dataframe.

I appreciate your reviews and suggestions in comments.

Thank you, hope you liked it.

	A	B	C	D	E	F	G	H
1	bedroom	bathroom	layout_type	property_type	price	area	furnish_type	
781	2	1	BHK	Independent House	15000	500	Semi-Furnished	
1028	2	1	BHK	Independent House	15000	500	Semi-Furnished	
15810	2	1	BHK	Independent House	10000	500	Unfurnished	
20812	2	1	BHK	Independent House	10000	500	Unfurnished	
22699								
22700								