Part I Introduction

About the data

1. To study on this project, we have collected the primary data of vallabh vidyanagar ,Anand . we firstly selected vallabh vidyanagar as prime location and then have sub location as nanabazar , motabazar ,bakrol gate & karamsad road.
2. Columns:

    - Price:Price of the to be sold
    - Society:Name of the society
    - location:In which area society is located
    - OverallCondition:Condition of the house. OverallCond: Rates the overall condition of the house

      10-- Very Excellent, 9-- Excellent, 8-- Very Good, 7-- Good, 6-- Above Average,
      5-- Average, 4-- Below Average,
      3-- Fair, 2-- Poor, 1-- Very Poor
    - YearBuilt:In which year house is built.
    - area sqrtft:Squarefoot of living in the area.
    - type of house:Which type of house is available(for e.g.2bhk,3bhk etc.)

Goal

- Here our goal is to predict the price of house(dependent variable) located in vallabh vidyanagar, with the help of other essential features(independent variable) available in our dataset. The predictive analysis using supervise learning is used for effective prediction and cost evaluated on the area type, locality, availability of facilities etc.

Part II Data Assessment

import libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
sns.set()
import matplotlib.pyplot as plt
%matplotlib inline
```

Check out the data

In [2]:
```
data=pd.read_excel("C:\\Users\\hp\\Desktop\\Roll no.20&30\\project.xls
x")
data.head()
```

Out[2]:

	Id	area type	society	location	OverallCond	YearBuilt	area sqrtft	type of house	price
0	maruti1	carpet area	maruti	arvind marg	9	2019	684	2 bhk	25.0
1	maruti2	carpet area	maruti	arvind marg	9	2019	450	1 bhk	17.0
2	maruti1	carpet area	maruti	arvind marg	9	2019	1364	3 bhk	40.0
3	maruti1	carpet area	maruti	arvind marg	9	2019	200	1 shop	25.0
4	ramkunj1	carpet area	ram kunj	arvind marg	8	2017	850	2 bhk	25.0

Check Missing Data

In [3]:
```
data.isnull().sum()
```

Out[3]:
```
Id                12
area type          0
society            0
location           0
OverallCond        0
YearBuilt          0
area sqrtft        0
type of house      0
price              0
dtype: int64
```

we can see here only 12 rows are nan in Id column.Now we will drop Id because Id column is not useful in modeling.

Calculating Statistics

We will start with calculating some descriptive statistics about this house data.

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 47 entries, 0 to 46
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             35 non-null     object
 1   area type      47 non-null     object
 2   society        47 non-null     object
 3   location       47 non-null     object
 4   OverallCond    47 non-null     int64
 5   YearBuilt      47 non-null     int64
 6   area sqrtft    47 non-null     int64
 7   type of house  47 non-null     object
 8   price          47 non-null     float64
```

```
dtypes: float64(1), int64(3), object(5)
memory usage: 2.4+ KB
```

In [5]:
```
data.columns
```

Out[5]:
```
Index(['Id', 'area type', 'society', 'location', 'OverallCond', 'YearBu
ilt',
       'area sqrtft', 'type of house', 'price'],
      dtype='object')
```

In [6]:
```
data.shape
```

Out[6]:
```
(47, 9)
```

In [7]:
```
data['location'].unique()
```

Out[7]:
```
array(['arvind marg', 'sahid chowk', 'harinagar road', 'kisan market',
       'bakrol road', 'bakrol gate', 'rajender marg', 'nagar palika roa
d'],
      dtype=object)
```

In [8]:
```
data['location'].value_counts()
```

Out[8]:
```
rajender marg       8
arvind marg         7
sahid chowk         6
kisan market        6
bakrol road         5
harinagar road      5
nagar palika road   5
bakrol gate         5
Name: location, dtype: int64
```

In [9]:
```
data.describe()
```

Out[9]:

	OverallCond	YearBuilt	area sqrtft	price

	OverallCond	YearBuilt	area sqrtft	price
count	47.000000	47.000000	47.000000	47.000000
mean	8.021277	2010.702128	782.595745	22.441702
std	0.820640	7.629790	344.923600	8.744207
min	6.000000	1995.000000	200.000000	8.000000
25%	8.000000	2005.000000	510.000000	15.000000
50%	8.000000	2010.000000	750.000000	22.000000
75%	9.000000	2018.000000	900.000000	28.000000
max	9.000000	2020.000000	1500.000000	45.000000

To get better understanding about this dataset, I summaried all importabt variables in terms of minimum, first quartile, median, mean, third quantile and maximum value.

In [10]:
```
data['price'].describe()
```

Out[10]:
```
count    47.000000
mean     22.441702
std       8.744207
min       8.000000
25%      15.000000
50%      22.000000
75%      28.000000
max      45.000000
Name: price, dtype: float64
```

In [11]:
```
data['area sqrtft'].describe()
```

Out[11]:
```
count      47.000000
mean      782.595745
std       344.923600
min       200.000000
25%       510.000000
50%       750.000000
```

```
75%        900.000000
max       1500.000000
Name: area sqrtft, dtype: float64
```

Data Cleaning

In [12]:
```
data['bhk']=data['type of house'].apply(lambda x: int(x.split(' ')[0]))
```

In [13]:
```
data.head()
```

Out[13]:

	Id	area type	society	location	OverallCond	YearBuilt	area sqrtft	type of house	price	bhk
0	maruti1	carpet area	maruti	arvind marg	9	2019	684	2 bhk	25.0	2
1	maruti2	carpet area	maruti	arvind marg	9	2019	450	1 bhk	17.0	1
2	maruti1	carpet area	maruti	arvind marg	9	2019	1364	3 bhk	40.0	3
3	maruti1	carpet area	maruti	arvind marg	9	2019	200	1 shop	25.0	1
4	ramkunj1	carpet area	ram kunj	arvind marg	8	2017	850	2 bhk	25.0	2

In [14]:
```
data['age of building']=2020-data['YearBuilt']
```

In [15]:
```
data.head()
```

Out[15]:

	Id	area type	society	location	OverallCond	YearBuilt	area sqrtft	type of house	price	bhk	age of building
0	maruti1	carpet area	maruti	arvind marg	9	2019	684	2 bhk	25.0	2	1

	Id	area type	society	location	OverallCond	YearBuilt	area sqrtft	type of house	price	bhk	age of building
1	maruti2	carpet area	maruti	arvind marg	9	2019	450	1 bhk	17.0	1	1
2	maruti1	carpet area	maruti	arvind marg	9	2019	1364	3 bhk	40.0	3	1
3	maruti1	carpet area	maruti	arvind marg	9	2019	200	1 shop	25.0	1	1
4	ramkunj1	carpet area	ram kunj	arvind marg	8	2017	850	2 bhk	25.0	2	3

In [16]:

```
df1=data.drop(['Id','area type','society','YearBuilt','type of house'],
axis=1)
df1.head()
```

Out[16]:

	location	OverallCond	area sqrtft	price	bhk	age of building
0	arvind marg	9	684	25.0	2	1
1	arvind marg	9	450	17.0	1	1
2	arvind marg	9	1364	40.0	3	1
3	arvind marg	9	200	25.0	1	1
4	arvind marg	8	850	25.0	2	3

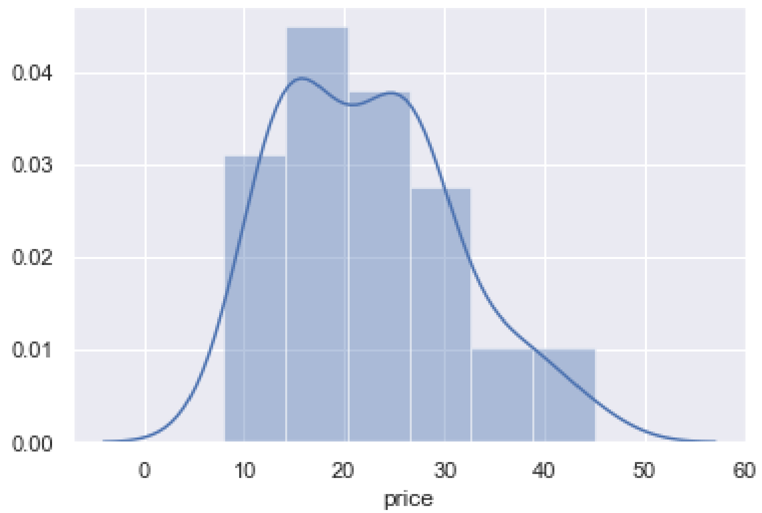
we droped columns(Id,areatype,society) because these are not useful in prediction and we also dropped YearBuilt and type of house because we already convert these two variable by bhk and age of building.

exploratory analysis of SalePrice

SalePrice is our target variable and also the dependent variable for prediction. According to the assumptions of Linear Regression, data should be normally distributed. By checking the distribution of SalePrice, we can decide if we need non-linear transformation, like log term, to make better prediction.

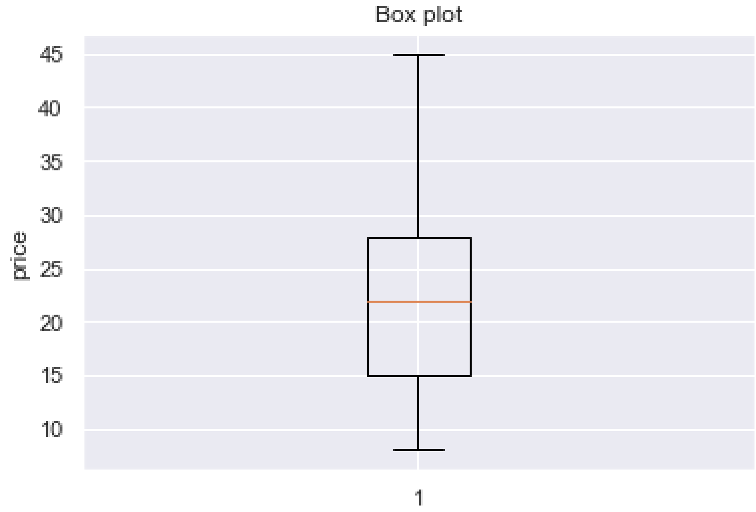
In [17]:
```python
### Lets check the outlier
import seaborn as sns
sns.distplot(df1['price'])
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0xe8950f0>



In [18]:
```python
plt.boxplot(df1.price)
plt.ylabel('price')
plt.title("Box plot")
```

Out[18]: Text(0.5, 1.0, 'Box plot')



From above it is clear that there is no outlier but this is not true lets see why

In [19]:
```
df2=df1.copy()
df2['price_per_srqft']=df2['price']*100000/df2['area sqrtft']
```

In [20]:
```
df2.head()
```

Out[20]:

	location	OverallCond	area sqrtft	price	bhk	age of building	price_per_srqft
0	arvind marg	9	684	25.0	2	1	3654.970760
1	arvind marg	9	450	17.0	1	1	3777.777778
2	arvind marg	9	1364	40.0	3	1	2932.551320
3	arvind marg	9	200	25.0	1	1	12500.000000
4	arvind marg	8	850	25.0	2	3	2941.176471

In [21]:
```
df2.describe()
```

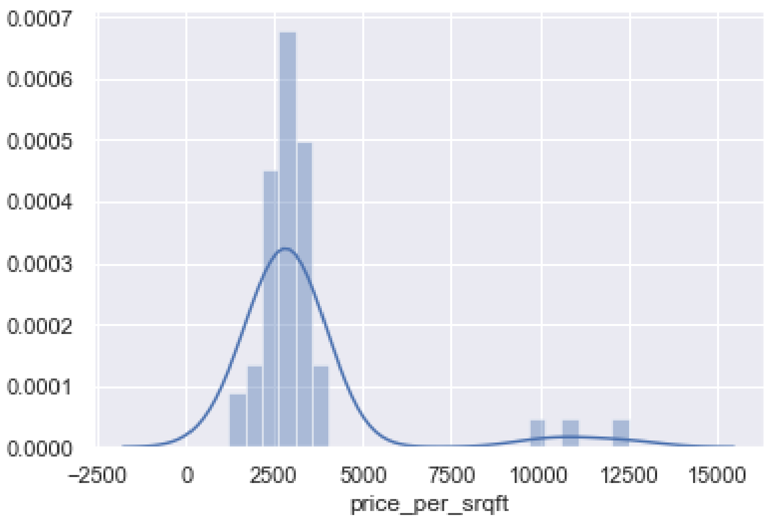
Out[21]:

	OverallCond	area sqrtft	price	bhk	age of building	price_per_srqft
count	47.000000	47.000000	47.000000	47.000000	47.000000	47.000000
mean	8.021277	782.595745	22.441702	1.978723	9.297872	3313.962848
std	0.820640	344.923600	8.744207	0.706780	7.629790	2163.663013
min	6.000000	200.000000	8.000000	1.000000	0.000000	1200.000000
25%	8.000000	510.000000	15.000000	1.500000	2.000000	2422.222222
50%	8.000000	750.000000	22.000000	2.000000	10.000000	2888.888889
75%	9.000000	900.000000	28.000000	2.000000	15.000000	3298.368298
max	9.000000	1500.000000	45.000000	3.000000	25.000000	12500.000000

In above if we see the table of descriptive statistics,in which price_per_srqft the difference betweenQ3 and max are huge.it seems to be a outlier.

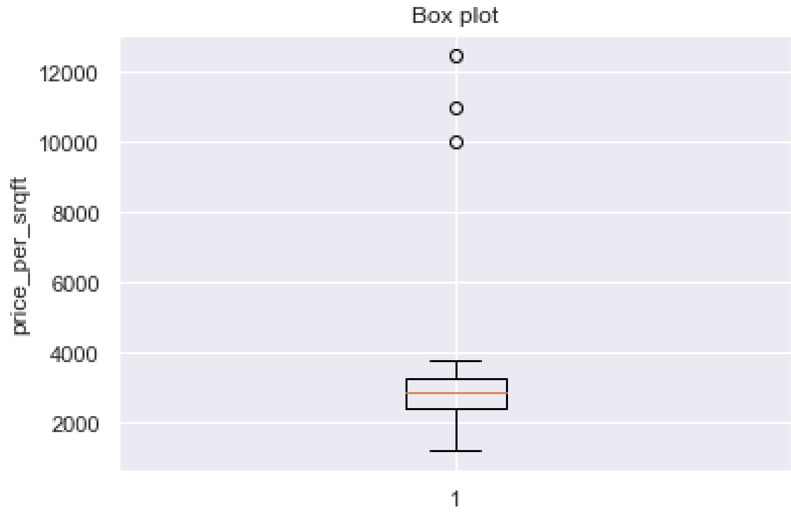
In [22]:
```
sns.distplot(df2['price_per_srqft'])
```

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0xe9605b0>



In [23]:
```
plt.boxplot(df2.price_per_srqft)
plt.ylabel('price_per_srqft')
plt.title("Box plot")
```

Out[23]: Text(0.5, 1.0, 'Box plot')



In above histogram some outlier is present

let's remove all those rows that price_per_srqft>4000

In [24]:
```
df2[df2['price_per_srqft']>=4000]
```

Out[24]:

	location	OverallCond	area sqrtft	price	bhk	age of building	price_per_srqft
3	arvind marg	9	200	25.0	1	1	12500.0
12	sahid chowk	8	200	22.0	1	3	11000.0
15	harinagar road	7	200	20.0	1	20	10000.0

In [25]:
```
df3=df2.drop([3,12,15],axis=0)
```

```python
df3.head()
```

Out[25]:

	location	OverallCond	area sqrtft	price	bhk	age of building	price_per_srqft
0	arvind marg	9	684	25.0	2	1	3654.970760
1	arvind marg	9	450	17.0	1	1	3777.777778
2	arvind marg	9	1364	40.0	3	1	2932.551320
4	arvind marg	8	850	25.0	2	3	2941.176471
5	arvind marg	8	900	30.0	2	3	3333.333333

In [26]:
```python
df3.shape
```

Out[26]: (44, 7)

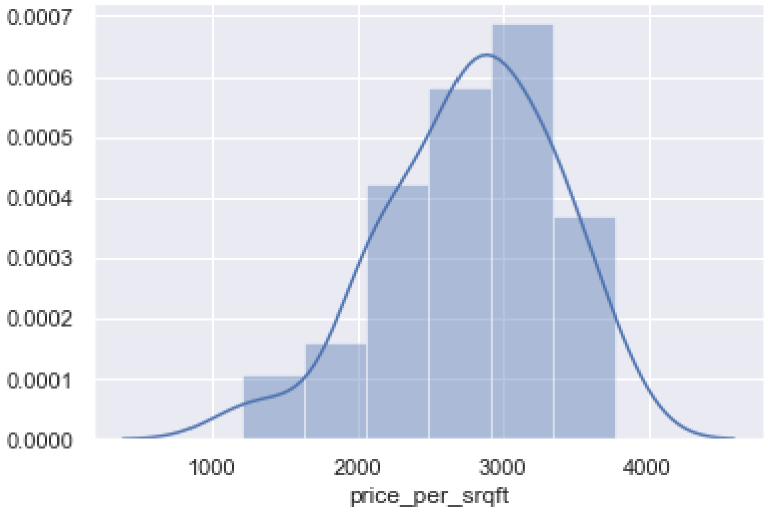
In [27]:
```python
df3.describe()
```

Out[27]:

	OverallCond	area sqrtft	price	bhk	age of building	price_per_srqft
count	44.000000	44.000000	44.000000	44.000000	44.000000	44.000000
mean	8.022727	822.318182	22.449091	2.045455	9.386364	2778.551224
std	0.820908	319.339636	9.027746	0.680443	7.555118	582.123337
min	6.000000	400.000000	8.000000	1.000000	0.000000	1200.000000
25%	8.000000	587.500000	15.000000	2.000000	2.000000	2396.153846
50%	8.000000	750.000000	22.000000	2.000000	10.000000	2817.444877
75%	9.000000	925.000000	28.000000	2.250000	15.000000	3238.970588
max	9.000000	1500.000000	45.000000	3.000000	25.000000	3777.777778

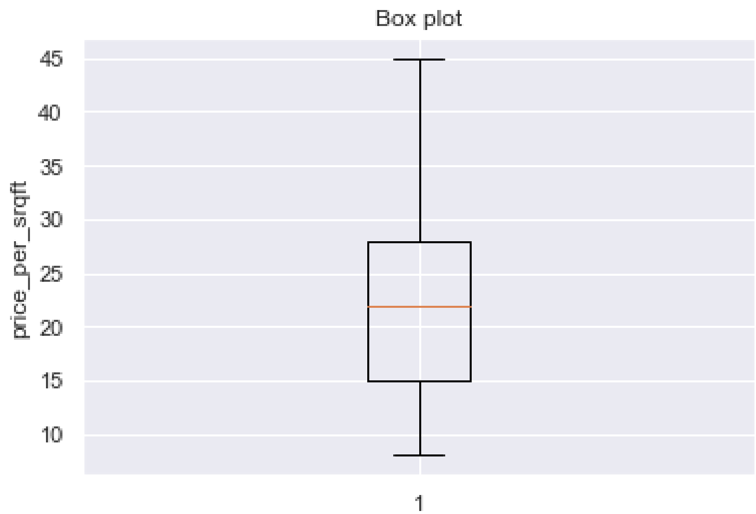
In [28]:
```python
sns.distplot(df3['price_per_srqft'])
```

Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0xea22130>



In [29]:
```
plt.boxplot(df3.price)
plt.ylabel('price_per_srqft')
plt.title("Box plot")
```

Out[29]: Text(0.5, 1.0, 'Box plot')



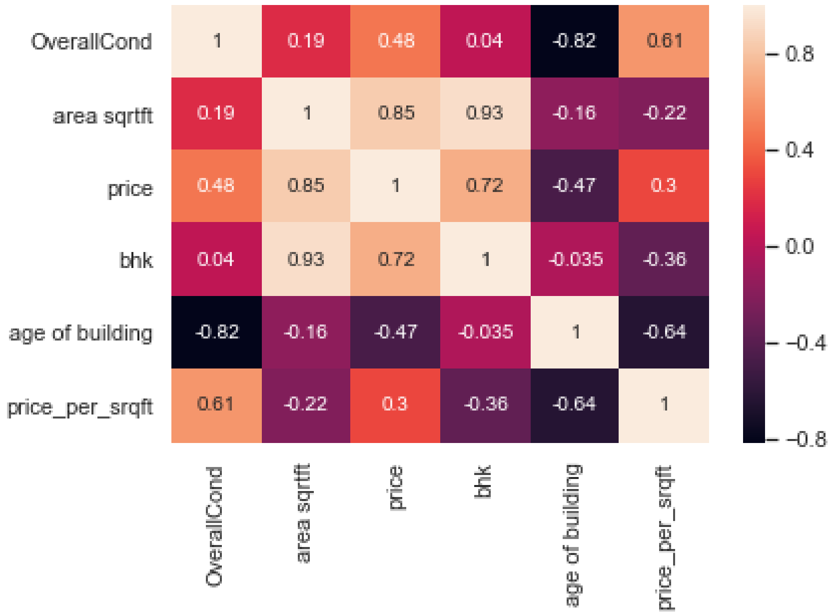
In [30]:
```
df3['price_per_srqft'].describe()
```

Out[30]:
```
count      44.000000
mean     2778.551224
std       582.123337
min      1200.000000
25%      2396.153846
50%      2817.444877
75%      3238.970588
max      3777.777778
Name: price_per_srqft, dtype: float64
```

Correlation matrix

In [31]:
```
sns.heatmap(df3.corr(),annot=True)
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0xea95810>



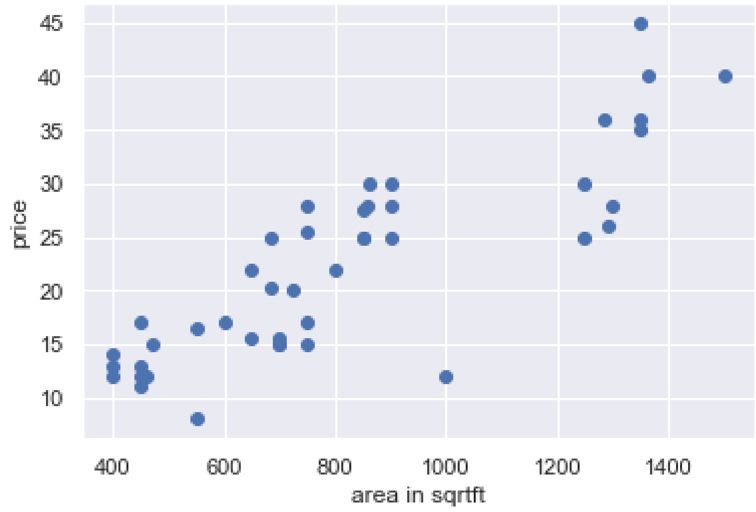
In above correlation matrix price is inversely correlated with age of building and directly correlated with remaing feature.

1.Scatter plot

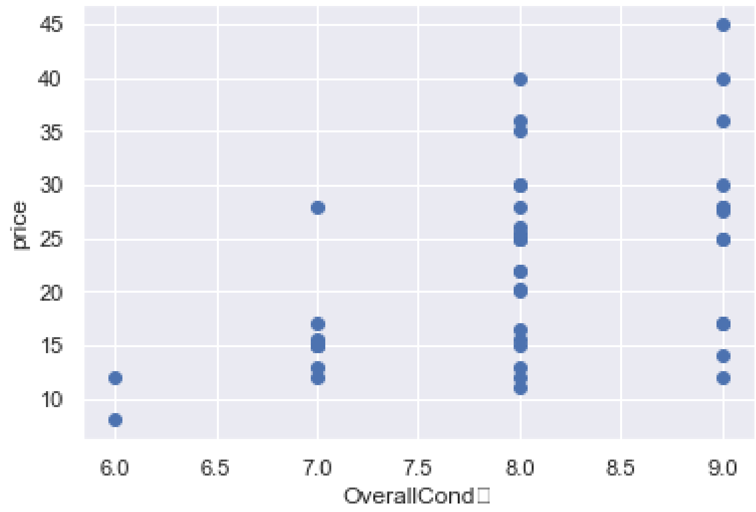
In [32]:
```
from matplotlib import pyplot as plt

plt.scatter(df3['age of building'],df3['price'])
plt.xlabel("age of building")
plt.ylabel("price");
```

In [33]:
```
plt.scatter(df3['area sqrtft'],df3['price'])
plt.xlabel("area in sqrtft")
plt.ylabel("price");
```



In [34]:
```
plt.scatter(df3['OverallCond'],df3['price'])
plt.xlabel("OverallCond ")
plt.ylabel("price");
```



In [35]:
```
df4=df3.groupby('bhk',as_index=False).agg({"price": "mean"})
```

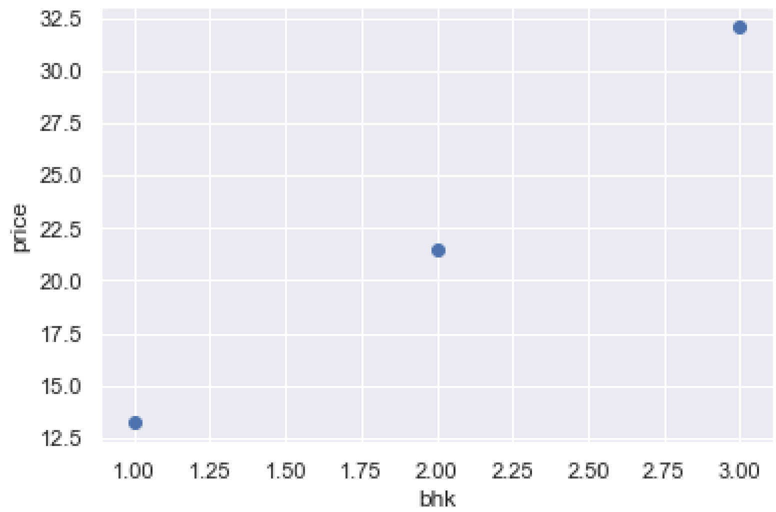
```
df4
```

Out[35]:

	bhk	price
0	1	13.222222
1	2	21.490000
2	3	32.090909

In [36]:

```
plt.scatter(df4['bhk'],df4['price'])
plt.xlabel("bhk")
plt.ylabel("price");
```



From above two it is clear that data is linear.

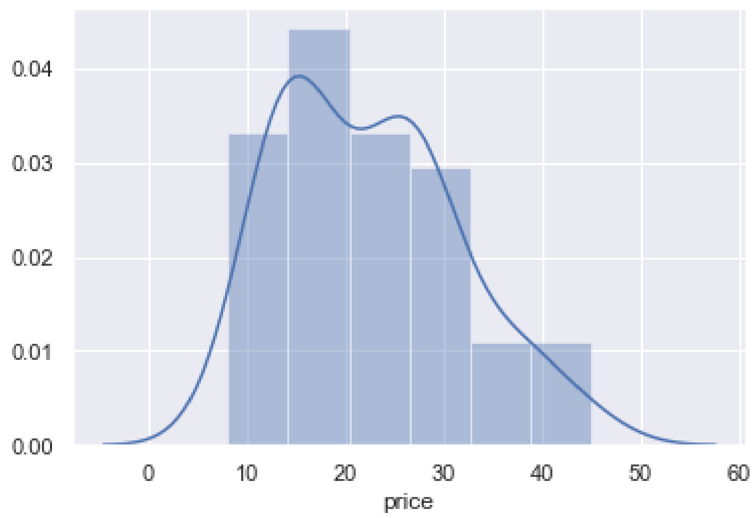
By seeing above scatter plot we can say that all independent variables have some relation between dependent variable

2.Histogram or Distribution plot

Histogram to check the normality of data by graphically

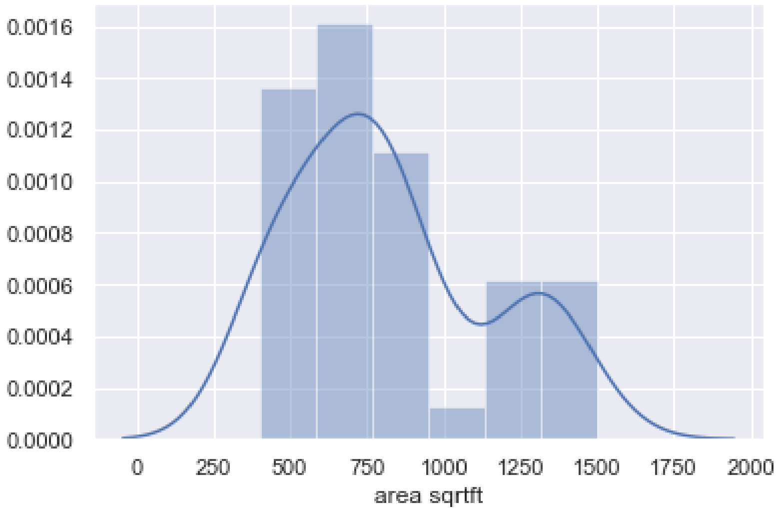
In [37]:
```python
sns.distplot(df3['price'])
```

Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0xec71470>



In [38]:
```python
sns.distplot(df3['area sqrtft'])
```

Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0xec2d770>

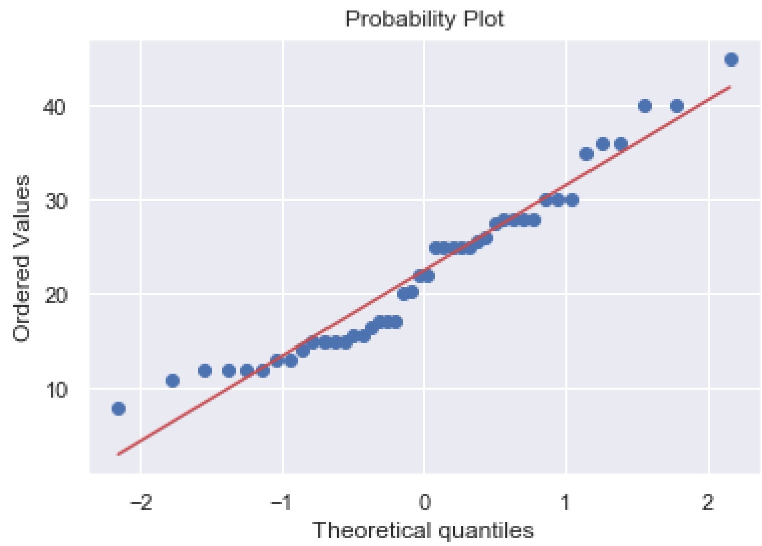


We can also check normality by using NPP or Q-Q plot

In [39]:
```
from scipy import stats
import scipy as scipy
import numpy as np
import matplotlib.pyplot as plt
```

In [40]:
```
stats.probplot(df3.price,plot=plt)
plt.figure()
```

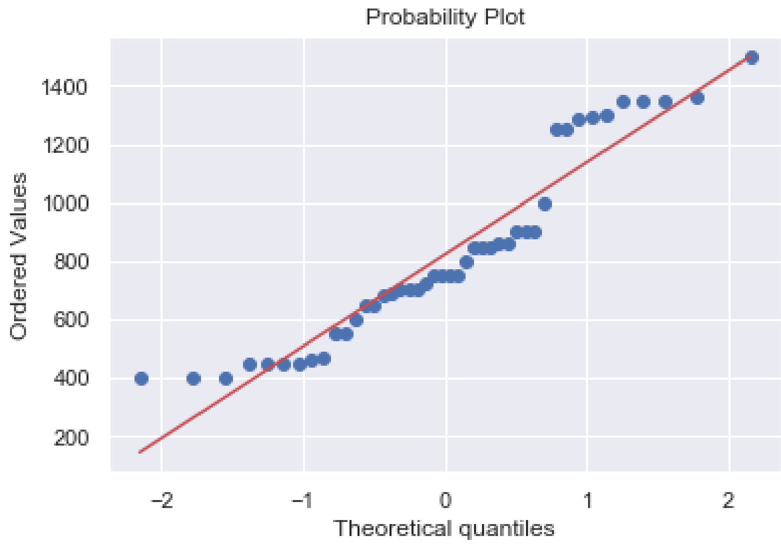
Out[40]: <Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>

In [41]:
```
stats.probplot(df3['area sqrtft'],plot=plt)
plt.figure()
```

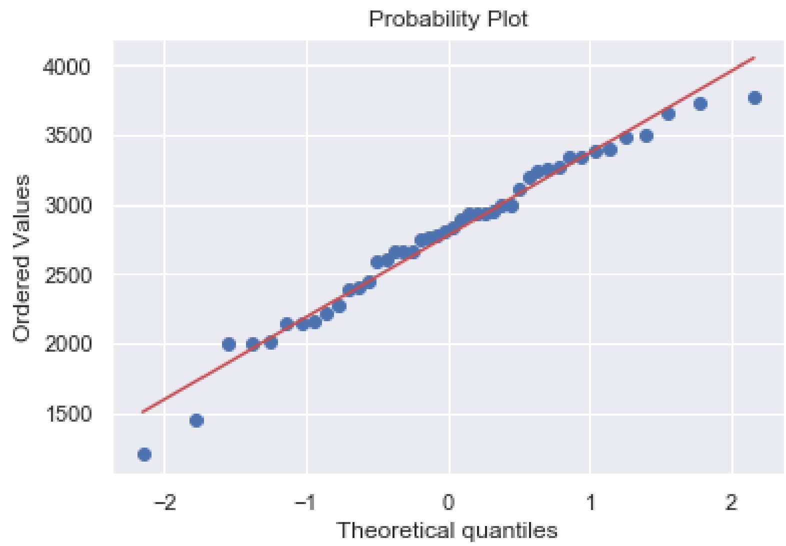
Out[41]: <Figure size 432x288 with 0 Axes>




```
<Figure size 432x288 with 0 Axes>
```

In [42]:
```
stats.probplot(df3.price_per_srqft,plot=plt)
plt.figure()
```

Out[42]:
```
<Figure size 432x288 with 0 Axes>
```




```
<Figure size 432x288 with 0 Axes>
```

By seeing above plot we can say that all independent variables and dependent variable are normal.

3.Multicolinearity¶

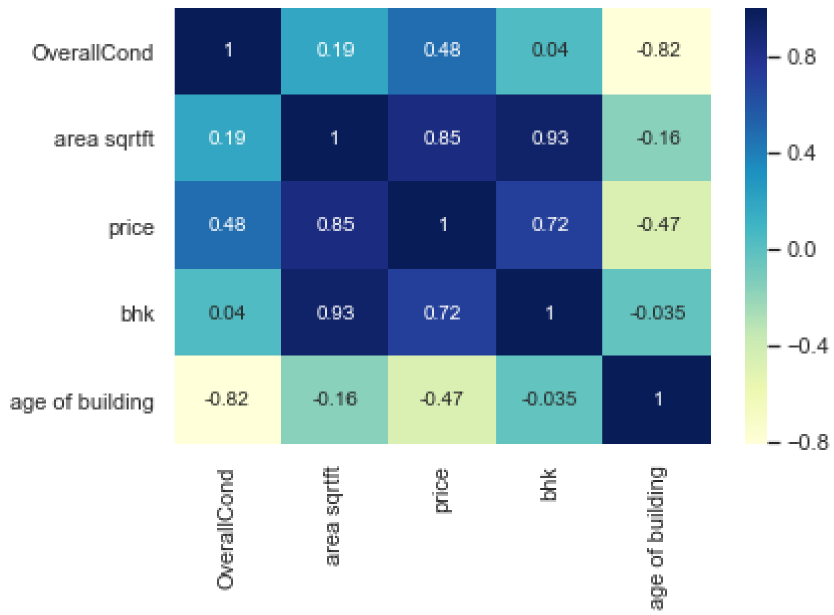
In [43]:
```
df4=df3.drop(['price_per_srqft'],axis=1)
df4.head()
```

Out[43]:

	location	OverallCond	area sqrtft	price	bhk	age of building
0	arvind marg	9	684	25.0	2	1
1	arvind marg	9	450	17.0	1	1
2	arvind marg	9	1364	40.0	3	1
4	arvind marg	8	850	25.0	2	3
5	arvind marg	8	900	30.0	2	3

```
In [44]: #1.correlation matrix
         ## Data visualization and building the correlation matrix
         sns.heatmap(df4.corr(),cmap="YlGnBu",annot=True)
```

```
Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0xec1c1b0>
```



In above correlation matrix price is inversely correlated with age of building and directly correlated with remaining feature.

Part III Handling categorical feature

```
In [45]: d=pd.get_dummies(df4.location)
```

```
In [46]: d.head()
```

```
Out[46]:
```

	arvind marg	bakrol gate	bakrol road	harinagar road	kisan market	nagar palika road	rajender marg	sahid chowk
0	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0

In [47]:

```
df5=pd.concat([df4,d.drop("sahid chowk",axis=1)],axis=1)
df5.head()
```

Out[47]:

	location	OverallCond	area sqrtft	price	bhk	age of building	arvind marg	bakrol gate	bakrol road	harinagar road	kisan market
0	arvind marg	9	684	25.0	2	1	1	0	0	0	0
1	arvind marg	9	450	17.0	1	1	1	0	0	0	0
2	arvind marg	9	1364	40.0	3	1	1	0	0	0	0
4	arvind marg	8	850	25.0	2	3	1	0	0	0	0
5	arvind marg	8	900	30.0	2	3	1	0	0	0	0

In [48]:

```
df6=df5.drop(['location'],axis=1)
df6.head()
```

Out[48]:

	OverallCond	area sqrtft	price	bhk	age of building	arvind marg	bakrol gate	bakrol road	harinagar road	kisan market	nagar palika road	ra

	OverallCond	area sqrtft	price	bhk	age of building	arvind marg	bakrol gate	bakrol road	harinagar road	kisan market	nagar palika road	ra
0	9	684	25.0	2	1	1	0	0	0	0	0	
1	9	450	17.0	1	1	1	0	0	0	0	0	
2	9	1364	40.0	3	1	1	0	0	0	0	0	
4	8	850	25.0	2	3	1	0	0	0	0	0	
5	8	900	30.0	2	3	1	0	0	0	0	0	

Feature Scaling

In [49]:
```
from sklearn.preprocessing import StandardScaler
```

In [50]:
```
scaler=StandardScaler()
df_scaler=scaler.fit_transform(df6)
df_scaler
```

Out[50]:
```
array([[ 1.20424087, -0.4381457 ,  0.28582997, -0.06757374, -1.1228572
4,
         2.51661148, -0.35805744, -0.35805744, -0.31622777, -0.3973597
1,
        -0.35805744, -0.47140452],
       [ 1.20424087, -1.17937938, -0.61057192, -1.55419597, -1.1228572
4,
         2.51661148, -0.35805744, -0.35805744, -0.31622777, -0.3973597
1,
        -0.35805744, -0.47140452],
       [ 1.20424087,  1.71586669,  1.96658351,  1.41904849, -1.1228572
4,
         2.51661148, -0.35805744, -0.35805744, -0.31622777, -0.3973597
1,
        -0.35805744, -0.47140452],
       [-0.0280056 ,  0.08768673,  0.28582997, -0.06757374, -0.8550755
6,
```

```
          2.51661148, -0.35805744, -0.35805744, -0.31622777, -0.3973597
1,
         -0.35805744, -0.47140452],
        [-0.0280056 ,  0.24607   ,  0.84608115, -0.06757374, -0.8550755
6,
          2.51661148, -0.35805744, -0.35805744, -0.31622777, -0.3973597
1,
         -0.35805744, -0.47140452],
        [ 1.20424087,  1.67151937,  2.52683468,  1.41904849, -1.2567480
7,
          2.51661148, -0.35805744, -0.35805744, -0.31622777, -0.3973597
1,
         -0.35805744, -0.47140452],
        [ 1.20424087, -1.14770272, -1.1708231 , -1.55419597, -1.1228572
4,
         -0.39735971, -0.35805744, -0.35805744, -0.31622777, -0.3973597
1,
         -0.35805744, -0.47140452],
        [ 1.20424087, -0.22907979, -0.61057192, -0.06757374, -1.1228572
4,
         -0.39735971, -0.35805744, -0.35805744, -0.31622777, -0.3973597
1,
         -0.35805744, -0.47140452],
        [ 1.20424087,  1.35475284,  0.28582997,  1.41904849, -1.1228572
4,
         -0.39735971, -0.35805744, -0.35805744, -0.31622777, -0.3973597
1,
         -0.35805744, -0.47140452],
        [-0.0280056 , -0.22907979,  0.34297559, -0.06757374, -0.8550755
6,
         -0.39735971, -0.35805744, -0.35805744, -0.31622777, -0.3973597
1,
         -0.35805744, -0.47140452],
        [-0.0280056 ,  2.14666916,  1.96658351,  1.41904849, -0.8550755
6,
         -0.39735971, -0.35805744, -0.35805744, -0.31622777, -0.3973597
1,
         -0.35805744, -0.47140452],
        [-0.0280056 , -1.17937938, -1.28287334, -1.55419597,  1.4210686
```

```
4,
        -0.39735971, -0.35805744, -0.35805744,  3.16227766, -0.3973597
1,
        -0.35805744, -0.47140452],
       [-1.26025208, -0.38746306, -0.83467239, -0.06757374,  1.4210686
4,
        -0.39735971, -0.35805744, -0.35805744,  3.16227766, -0.3973597
1,
        -0.35805744, -0.47140452],
       [-0.0280056 , -0.54584632, -0.77864728, -0.06757374,  0.3499419
6,
        -0.39735971, -0.35805744, -0.35805744,  3.16227766, -0.3973597
1,
        -0.35805744, -0.47140452],
       [-0.0280056 , -0.43181037, -0.24640866, -0.06757374,  1.5549594
8,
        -0.39735971, -0.35805744, -0.35805744,  3.16227766, -0.3973597
1,
        -0.35805744, -0.47140452],
       [-0.0280056 ,  1.48145946,  0.3978802 ,  1.41904849,  0.7516144
6,
        -0.39735971, -0.35805744, -0.35805744, -0.31622777,  2.5166114
8,
        -0.35805744, -0.47140452],
       [-0.0280056 , -1.11602607, -0.83467239, -1.55419597,  0.7516144
6,
        -0.39735971, -0.35805744, -0.35805744, -0.31622777,  2.5166114
8,
        -0.35805744, -0.47140452],
       [-0.0280056 , -0.30827143, -0.27442121, -0.06757374,  0.7516144
6,
        -0.39735971, -0.35805744, -0.35805744, -0.31622777,  2.5166114
8,
        -0.35805744, -0.47140452],
       [ 1.20424087, -0.22907979,  0.62198067, -0.06757374, -1.2567480
7,
        -0.39735971, -0.35805744, -0.35805744, -0.31622777,  2.5166114
8,
        -0.35805744, -0.47140452],
```

```
        [-0.0280056 , -0.54584632, -0.05032074, -0.06757374,  0.0821602
9,
         -0.39735971, -0.35805744, -0.35805744, -0.31622777,  2.5166114
8,
         -0.35805744, -0.47140452],
        [ 1.20424087,  0.08768673,  0.56595556, -0.06757374, -1.2567480
7,
         -0.39735971, -0.35805744, -0.35805744, -0.31622777,  2.5166114
8,
         -0.35805744, -0.47140452],
        [-1.26025208, -0.38746306, -0.77864728, -0.06757374,  1.1532869
7,
         -0.39735971, -0.35805744,  2.79284801, -0.31622777, -0.3973597
1,
         -0.35805744, -0.47140452],
        [-1.26025208, -0.22907979, -0.83467239, -0.06757374,  1.0193961
3,
         -0.39735971, -0.35805744,  2.79284801, -0.31622777, -0.3973597
1,
         -0.35805744, -0.47140452],
        [-1.26025208, -1.17937938, -1.1708231 , -1.55419597,  1.0193961
3,
         -0.39735971, -0.35805744,  2.79284801, -0.31622777, -0.3973597
1,
         -0.35805744, -0.47140452],
        [-2.49249855, -0.86261285, -1.61902405, -0.06757374,  2.0905228
2,
         -0.39735971, -0.35805744,  2.79284801, -0.31622777, -0.3973597
1,
         -0.35805744, -0.47140452],
        [-2.49249855,  0.56283653, -1.1708231 ,  1.41904849,  2.0905228
2,
         -0.39735971, -0.35805744,  2.79284801, -0.31622777, -0.3973597
1,
         -0.35805744, -0.47140452],
        [-0.0280056 , -0.07069653, -0.05032074, -0.06757374,  0.7516144
6,
         -0.39735971,  2.79284801, -0.35805744, -0.31622777, -0.3973597
1,
```

```
        -0.35805744, -0.47140452],
       [-0.0280056 ,  1.51313611,  0.62198067,  1.41904849,  0.0821602
9,
        -0.39735971,  2.79284801, -0.35805744, -0.31622777, -0.3973597
1,
        -0.35805744, -0.47140452],
       [-1.26025208,  0.24607   ,  0.62198067, -0.06757374,  1.1532869
7,
        -0.39735971, -0.35805744, -0.35805744, -0.31622777, -0.3973597
1,
        -0.35805744,  2.12132034],
       [-0.0280056 ,  1.67151937,  1.51838256,  1.41904849,  0.3499419
6,
        -0.39735971, -0.35805744, -0.35805744, -0.31622777, -0.3973597
1,
        -0.35805744,  2.12132034],
       [-1.26025208, -1.17937938, -1.05877287, -1.55419597,  0.0821602
9,
        -0.39735971, -0.35805744, -0.35805744, -0.31622777, -0.3973597
1,
         2.79284801, -0.47140452],
       [-1.26025208, -0.70422958, -0.61057192, -0.06757374,  0.0821602
9,
        -0.39735971, -0.35805744, -0.35805744, -0.31622777, -0.3973597
1,
         2.79284801, -0.47140452],
       [-0.0280056 , -0.86261285, -0.66659704, -0.06757374, -0.9889664
,
        -0.39735971, -0.35805744, -0.35805744, -0.31622777, -0.3973597
1,
         2.79284801, -0.47140452],
       [-0.0280056 , -1.33776264, -1.05877287, -1.55419597, -0.9889664
,
        -0.39735971, -0.35805744, -0.35805744, -0.31622777, -0.3973597
1,
         2.79284801, -0.47140452],
       [-1.26025208, -0.38746306, -0.83467239, -0.06757374,  1.4210686
4,
        -0.39735971, -0.35805744, -0.35805744, -0.31622777, -0.3973597
```

```
        1,
                 2.79284801, -0.47140452],
               [-0.0280056 ,  1.35475284,  0.84608115,  1.41904849,  0.0821602
        9,
                -0.39735971, -0.35805744, -0.35805744, -0.31622777, -0.3973597
        1,
                -0.35805744,  2.12132034],
               [-0.0280056 ,  0.08768673,  0.28582997, -0.06757374,  0.0821602
        9,
                -0.39735971, -0.35805744, -0.35805744, -0.31622777, -0.3973597
        1,
                -0.35805744,  2.12132034],
               [-0.0280056 , -1.33776264, -1.1708231 , -1.55419597,  0.0821602
        9,
                -0.39735971, -0.35805744, -0.35805744, -0.31622777, -0.3973597
        1,
                -0.35805744,  2.12132034],
               [ 1.20424087, -1.33776264, -0.94672263, -1.55419597, -0.5872938
        9,
                -0.39735971, -0.35805744, -0.35805744, -0.31622777, -0.3973597
        1,
                -0.35805744,  2.12132034],
               [ 1.20424087,  0.11302806,  0.62198067, -0.06757374, -0.5872938
        9,
                -0.39735971, -0.35805744, -0.35805744, -0.31622777, -0.3973597
        1,
                -0.35805744,  2.12132034],
               [ 1.20424087,  1.46562113,  1.51838256,  1.41904849, -0.5872938
        9,
                -0.39735971, -0.35805744, -0.35805744, -0.31622777, -0.3973597
        1,
                -0.35805744,  2.12132034],
               [ 1.20424087,  0.11936339,  0.84608115, -0.06757374, -1.1228572
        4,
                -0.39735971,  2.79284801, -0.35805744, -0.31622777, -0.3973597
        1,
                -0.35805744, -0.47140452],
               [-0.0280056 ,  0.24607   ,  0.28582997, -0.06757374,  0.0821602
        9,
```

```
        -0.39735971,  2.79284801, -0.35805744, -0.31622777, -0.3973597
1,
        -0.35805744, -0.47140452],
       [-0.0280056 ,  1.67151937,  1.40633233,  1.41904849,  0.0821602
9,
        -0.39735971,  2.79284801, -0.35805744, -0.31622777, -0.3973597
1,
        -0.35805744, -0.47140452]])
```

In [51]:
```
df6.columns
```

Out[51]:
```
Index(['OverallCond', 'area sqrtft', 'price', 'bhk', 'age of building',
       'arvind marg', 'bakrol gate', 'bakrol road', 'harinagar road',
       'kisan market', 'nagar palika road', 'rajender marg'],
      dtype='object')
```

In [52]:
```
df_scaler=pd.DataFrame(df_scaler,columns=['OverallCond','area sqrtft',
'price','bhk','age of building','arvind marg','bakrol gate','bakrol roa
d','harinagar road','kisan market','nagar palika road','rajender marg'
])
df_scaler
```

Out[52]:

	OverallCond	area sqrtft	price	bhk	age of building	arvind marg	bakrol gate	bakrol road	hari
0	1.204241	-0.438146	0.285830	-0.067574	-1.122857	2.516611	-0.358057	-0.358057	-0.3
1	1.204241	-1.179379	-0.610572	-1.554196	-1.122857	2.516611	-0.358057	-0.358057	-0.3
2	1.204241	1.715867	1.966584	1.419048	-1.122857	2.516611	-0.358057	-0.358057	-0.3
3	-0.028006	0.087687	0.285830	-0.067574	-0.855076	2.516611	-0.358057	-0.358057	-0.3
4	-0.028006	0.246070	0.846081	-0.067574	-0.855076	2.516611	-0.358057	-0.358057	-0.3
5	1.204241	1.671519	2.526835	1.419048	-1.256748	2.516611	-0.358057	-0.358057	-0.3
6	1.204241	-1.147703	-1.170823	-1.554196	-1.122857	-0.397360	-0.358057	-0.358057	-0.3
7	1.204241	-0.229080	-0.610572	-0.067574	-1.122857	-0.397360	-0.358057	-0.358057	-0.3
8	1.204241	1.354753	0.285830	1.419048	-1.122857	-0.397360	-0.358057	-0.358057	-0.3

	OverallCond	area sqrtft	price	bhk	age of building	arvind marg	bakrol gate	bakrol road	hari
9	-0.028006	-0.229080	0.342976	-0.067574	-0.855076	-0.397360	-0.358057	-0.358057	-0.3
10	-0.028006	2.146669	1.966584	1.419048	-0.855076	-0.397360	-0.358057	-0.358057	-0.3
11	-0.028006	-1.179379	-1.282873	-1.554196	1.421069	-0.397360	-0.358057	-0.358057	3.1
12	-1.260252	-0.387463	-0.834672	-0.067574	1.421069	-0.397360	-0.358057	-0.358057	3.1
13	-0.028006	-0.545846	-0.778647	-0.067574	0.349942	-0.397360	-0.358057	-0.358057	3.1
14	-0.028006	-0.431810	-0.246409	-0.067574	1.554959	-0.397360	-0.358057	-0.358057	3.1
15	-0.028006	1.481459	0.397880	1.419048	0.751614	-0.397360	-0.358057	-0.358057	-0.3
16	-0.028006	-1.116026	-0.834672	-1.554196	0.751614	-0.397360	-0.358057	-0.358057	-0.3
17	-0.028006	-0.308271	-0.274421	-0.067574	0.751614	-0.397360	-0.358057	-0.358057	-0.3
18	1.204241	-0.229080	0.621981	-0.067574	-1.256748	-0.397360	-0.358057	-0.358057	-0.3
19	-0.028006	-0.545846	-0.050321	-0.067574	0.082160	-0.397360	-0.358057	-0.358057	-0.3
20	1.204241	0.087687	0.565956	-0.067574	-1.256748	-0.397360	-0.358057	-0.358057	-0.3
21	-1.260252	-0.387463	-0.778647	-0.067574	1.153287	-0.397360	-0.358057	2.792848	-0.3
22	-1.260252	-0.229080	-0.834672	-0.067574	1.019396	-0.397360	-0.358057	2.792848	-0.3
23	-1.260252	-1.179379	-1.170823	-1.554196	1.019396	-0.397360	-0.358057	2.792848	-0.3
24	-2.492499	-0.862613	-1.619024	-0.067574	2.090523	-0.397360	-0.358057	2.792848	-0.3
25	-2.492499	0.562837	-1.170823	1.419048	2.090523	-0.397360	-0.358057	2.792848	-0.3
26	-0.028006	-0.070697	-0.050321	-0.067574	0.751614	-0.397360	2.792848	-0.358057	-0.3
27	-0.028006	1.513136	0.621981	1.419048	0.082160	-0.397360	2.792848	-0.358057	-0.3
28	-1.260252	0.246070	0.621981	-0.067574	1.153287	-0.397360	-0.358057	-0.358057	-0.3
29	-0.028006	1.671519	1.518383	1.419048	0.349942	-0.397360	-0.358057	-0.358057	-0.3
30	-1.260252	-1.179379	-1.058773	-1.554196	0.082160	-0.397360	-0.358057	-0.358057	-0.3
31	-1.260252	-0.704230	-0.610572	-0.067574	0.082160	-0.397360	-0.358057	-0.358057	-0.3

	OverallCond	area sqrtft	price	bhk	age of building	arvind marg	bakrol gate	bakrol road	hari
32	-0.028006	-0.862613	-0.666597	-0.067574	-0.988966	-0.397360	-0.358057	-0.358057	-0.3
33	-0.028006	-1.337763	-1.058773	-1.554196	-0.988966	-0.397360	-0.358057	-0.358057	-0.3
34	-1.260252	-0.387463	-0.834672	-0.067574	1.421069	-0.397360	-0.358057	-0.358057	-0.3
35	-0.028006	1.354753	0.846081	1.419048	0.082160	-0.397360	-0.358057	-0.358057	-0.3
36	-0.028006	0.087687	0.285830	-0.067574	0.082160	-0.397360	-0.358057	-0.358057	-0.3
37	-0.028006	-1.337763	-1.170823	-1.554196	0.082160	-0.397360	-0.358057	-0.358057	-0.3
38	1.204241	-1.337763	-0.946723	-1.554196	-0.587294	-0.397360	-0.358057	-0.358057	-0.3
39	1.204241	0.113028	0.621981	-0.067574	-0.587294	-0.397360	-0.358057	-0.358057	-0.3
40	1.204241	1.465621	1.518383	1.419048	-0.587294	-0.397360	-0.358057	-0.358057	-0.3
41	1.204241	0.119363	0.846081	-0.067574	-1.122857	-0.397360	2.792848	-0.358057	-0.3
42	-0.028006	0.246070	0.285830	-0.067574	0.082160	-0.397360	2.792848	-0.358057	-0.3
43	-0.028006	1.671519	1.406332	1.419048	0.082160	-0.397360	2.792848	-0.358057	-0.3

In [53]:
```
df_scaler.shape
```

Out[53]:
```
(44, 12)
```

Train Test Split

Now let's split the data into a training set and a testing set. We will train out model on the training set and then use the test set to evaluate the model. The training dataset and test dataset must be similar, usually have the same predictors or variables. They differ on the observations and specific values in the variables. If you fit the model on the training dataset, then you implicitly minimize error or find correct responses. The fitted model provides a good prediction on the training dataset. Then you test the model on the test dataset. If the model predicts good also on the test dataset, you have more confidence. You have more confidence since the test dataset is similar to the training dataset, but not the same nor seen by the model. It means the model transfers prediction or learning in real sense.

So,by splitting dataset into training and testing subset, we can efficiently measure our trained model since it never sees testing data before.Thus it's possible to prevent overfitting.

I am just splitting dataset into 20% of test data and remaining 80% will used for training the model.

In [54]:
```
y=df6['price']
x=df6.drop(['price'],axis=1)
```

In [55]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,rando
m_state=10)
```

Part IV Model Fitting

1.LinearRegression

Linear regression is been studied at great length, and there is a lot of literature on how your data must be structured to make best use of the model.

As such, there is a lot of sophistication when talking about these requirements and expectations which can be intimidating. In practice, you can uses these rules more as rules of thumb when using Ordinary Least Squares Regression, the most common implementation of linear regression.

Try different preparations of your data using these heuristics and see what works best for your problem.

- Linear Assumption. Linear regression assumes that the relationship between your input and output is linear. It does not support anything else. This may be obvious, but it is good to remember when you have a lot of attributes. You may need to transform data to make the relationship linear (e.g. log transform for an exponential relationship).

- Remove Noise. Linear regression assumes that your input and output variables are not noisy. Consider using data cleaning operations that let you better expose and clarify the signal in your data. This is most important for the output variable and you want to remove outliers in the output variable (y) if possible. Remove Collinearity. Linear regression will over-fit your data when you have highly correlated input variables. Consider calculating pairwise correlations for your input data and removing the most correlated.
- Gaussian Distributions. Linear regression will make more reliable predictions if your input and output variables have a Gaussian distribution. You may get some benefit using transforms (e.g. log or BoxCox) on you variables to make their distribution more Gaussian looking.
- Rescale Inputs: Linear regression will often make more reliable predictions if you rescale input variables using standardization or normalization.

In [56]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression(normalize=True)
lr.fit(x_train,y_train)
```

Out[56]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normaliz
e=True)
```

In [57]:
```
lr.intercept_
```

Out[57]:
```
17.700029037747235
```

In [58]:
```
pred=lr.predict(x_test)
pred
```

Out[58]:
```
array([21.67559385, 36.45983069, 34.46422629, 25.5179434 , 29.69351199,
       15.13993397, 18.84309747, 24.50958231, 40.06347606])
```

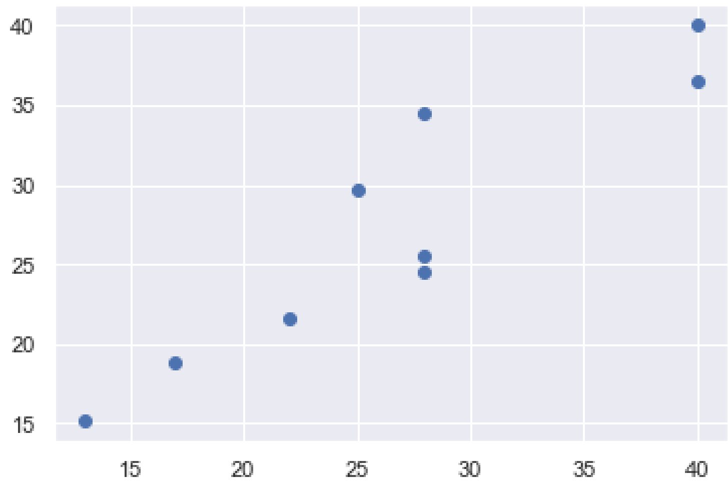
In [59]:
```
coeff_df = pd.DataFrame(lr.coef_, x.columns, columns=['Coefficient'])
coeff_df
```

Out[59]:

	Coefficient
OverallCond	-1.470874
area sqrtft	0.025623
bhk	-2.200436
age of building	-0.435506
arvind marg	7.688245
bakrol gate	6.177553
bakrol road	0.113007
harinagar road	5.010745
kisan market	5.230979
nagar palika road	2.029110
rajender marg	5.649578

In [60]:
```
plt.scatter(y_test, pred)
```

Out[60]:
```
<matplotlib.collections.PathCollection at 0xf1dd230>
```



Regression Evaluation Metrics

Here are three common evaluation metrics for regression problems:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors: n ∑(1/n)*|yi−y_pred| i=1 Mean Squared Error (MSE) is the mean of the squared errors:

n ∑(1/n)*(yi−y_pred)^2 i=1

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors: sqrt((1/n)*∑(yi−y_pred)^2)

Comparing these metrics:

- MAE is the easiest to understand, because it's the average error.
- MSE is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.
- RMSE is even more popular than MSE, because RMSE is interpretable in the "y" units.
- All of these are loss functions, because we want to minimize them.

In [61]:
```
from sklearn import metrics
from sklearn.model_selection import cross_val_score

def cross_val(model):
    pred = cross_val_score(model, x, y, cv=10)
    return pred.mean()

def print_evaluate(true, predicted):
    mae = metrics.mean_absolute_error(true, predicted)
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    r2_square = metrics.r2_score(true, predicted)
    print('MAE:', mae)
    print('MSE:', mse)
    print('RMSE:', rmse)
    print('R2 Square', r2_square)
```

```
def evaluate(true, predicted):
    mae = metrics.mean_absolute_error(true, predicted)
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    r2_square = metrics.r2_score(true, predicted)
    return mae, mse, rmse, r2_square
```

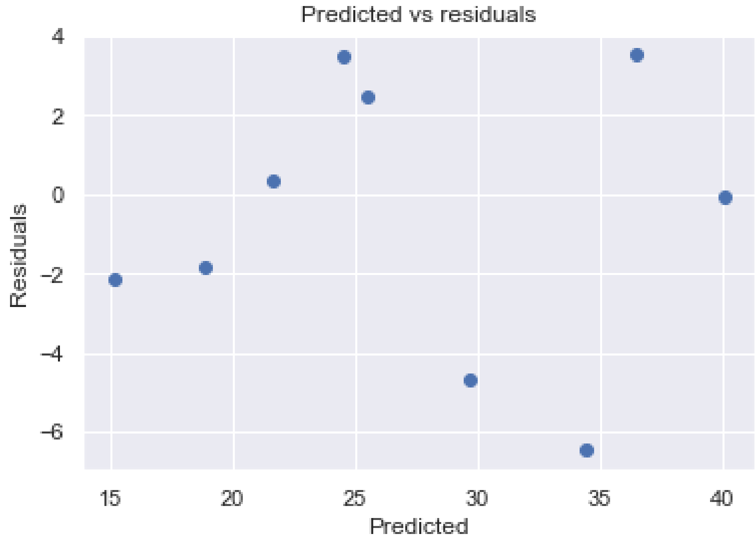
In [62]:
```
test_pred = lr.predict(x_test)
train_pred = lr.predict(x_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)
print('====================================')
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)
```

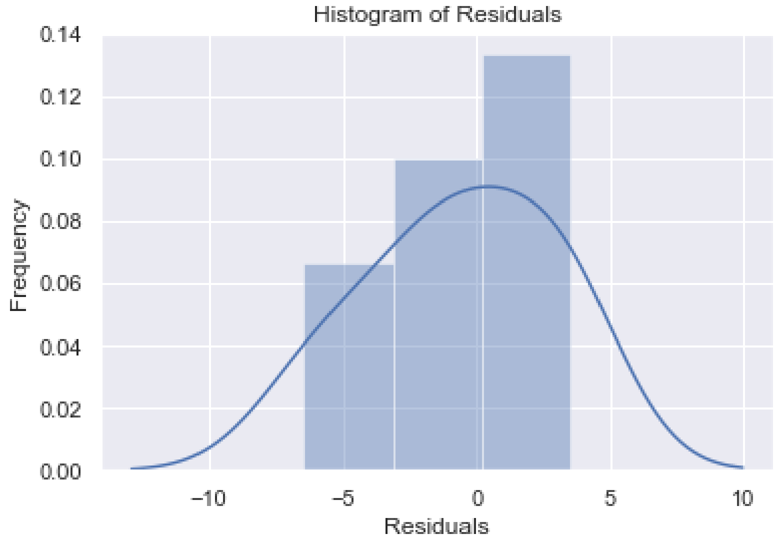
```
Test set evaluation:
_____________________________________
MAE: 2.7823661697765134
MSE: 11.419698882003706
RMSE: 3.3793044967868324
R2 Square 0.8455766929144741
====================================
Train set evaluation:
_____________________________________
MAE: 2.1905025706485444
MSE: 7.693746361007907
RMSE: 2.7737603286888195
R2 Square 0.8974929117316708
```

In [63]:
```
# Visualizing the differences between actual prices and predicted value
s
plt.scatter(y_test, pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```

In [64]:
```
# Checking residuals
plt.scatter(pred,y_test-pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```



In [65]:
```
# Checking Normality of errors
sns.distplot(y_test-pred)
plt.title("Histogram of Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()
```



In [66]:
```
def predict_price(OverallCond,area,bhk,ageofbuilding,location):
    loc_index = np.where(x.columns==location)[0][0]

    X = np.zeros(len(x.columns))
    X[0] = OverallCond
    X[1] = area
    X[2] = bhk
    X[3] = ageofbuilding
    if loc_index >= 0:
        X[loc_index] = 1

    return lr.predict([X])[0]
```

In [67]:
```
predict_price(9,684,2,0,'arvind marg')
```

Out[67]: 25.275725726034377

In [68]:
```
lr.predict([[9,684,2,0,1,0,0,0,0,0,0]])
```

Out[68]: array([25.27572573])

In [69]:
```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score

cv = ShuffleSplit(n_splits=3, test_size=0.2, random_state=0)

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

Out[69]:
```
array([0.79706516, 0.73700929, 0.75786989])
```

In [70]:
```
import numpy as np
np.mean(score)
```

Out[70]:
```
0.7639814427096079
```

In [71]:
```
results_df = pd.DataFrame(data=[["Linear Regression", *evaluate(y_test,
 test_pred)]],
                          columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 S
quare'])
results_df
```

Out[71]:

	Model	MAE	MSE	RMSE	R2 Square
0	Linear Regression	2.782366	11.419699	3.379304	0.845577

Here our data is working very well with respect to train and test.Train set evaluation:R2 Square 0.8974929117316708 and Test set evaluation:R2 Square 0.8455766929144741.we can see that the difference between train R2 test R2 is very low.

2.Support Vector Machine(SVM)

In [72]:
```
from sklearn.svm import SVR
svr=SVR(degree=2,kernel='linear')
svr.fit(x_train,y_train)
```

Out[72]:
```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=2, epsilon=0.1, gamma='sca
```

```
le',
    kernel='linear', max_iter=-1, shrinking=True, tol=0.001, verbose=Fa
lse)
```

In [73]:
```
svr.intercept_
```

Out[73]:
```
array([1.27940149])
```

In [74]:
```
svr.coef_
```

Out[74]:
```
array([[ 0.76436511,  0.02571232, -1.26954867, -0.31590877,  2.
,
         0.06212269, -1.00333261, -0.47679245,  1.        , -0.6715150
6,
         0.08951744]])
```

In [75]:
```
pred=svr.predict(x_test)
pred
```

Out[75]:
```
array([20.74857558, 41.20643647, 33.91473291, 26.19073795, 27.76297435,
       15.1063708 , 24.58792457, 25.90383334, 41.10574302])
```

In [76]:
```
test_pred = svr.predict(x_test)
train_pred = svr.predict(x_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)
print('====================================')
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)
```

```
Test set evaluation:
_____________________________________
MAE: 2.871226138883465
MSE: 12.949228150819616
RMSE: 3.5985035988337732
R2 Square 0.8248935759238082
====================================
```

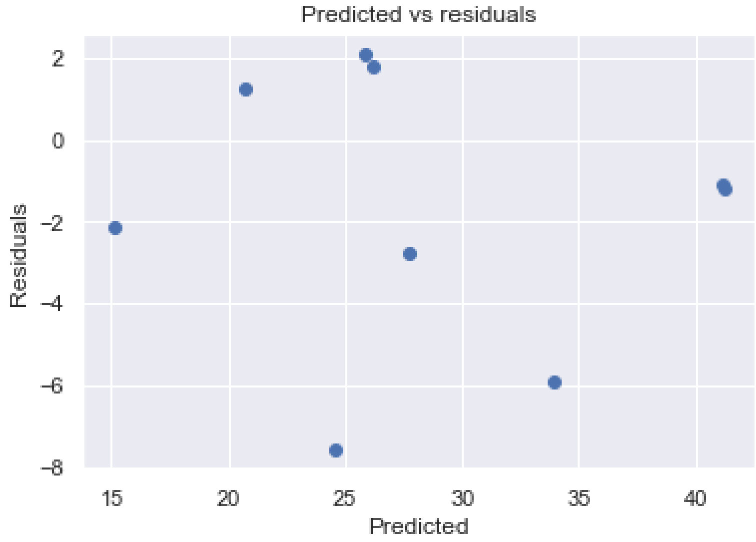
```
Train set evaluation:
_____________________________________
MAE: 2.224788885448431
MSE: 11.456404659795092
RMSE: 3.38473110598096
R2 Square 0.8473613986482564
```

In [77]:

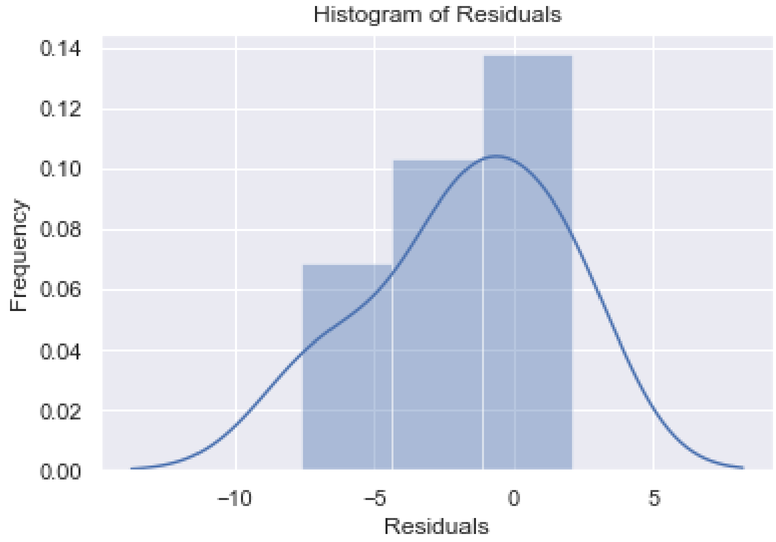
```
# Visualizing the differences between actual prices and predicted value
s
plt.scatter(y_test, pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```

In [78]:

```
# Checking residuals
plt.scatter(pred,y_test-pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```



In [79]:
```
# Checking Normality of errors
sns.distplot(y_test-pred)
plt.title("Histogram of Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()
```



In [80]:
```
def predict_price(OverallCond,area,bhk,ageofbuilding,location):
    loc_index = np.where(x.columns==location)[0][0]

    X = np.zeros(len(x.columns))
    X[0] = OverallCond
    X[1] = area
    X[2] = bhk
    X[3] = ageofbuilding
    if loc_index >= 0:
        X[loc_index] = 1

    return svr.predict([X])[0]
```

In [81]:
```
predict_price(9,684,2,0,'arvind marg')
```

Out[81]: 25.206819933320983

In [82]:
```
results_df_2 = pd.DataFrame(data=[["Support Vector Machine", *evaluate(
y_test, test_pred) ]],
                            columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2
 Square'])
```

```
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[82]:

	Model	MAE	MSE	RMSE	R2 Square
0	Linear Regression	2.782366	11.419699	3.379304	0.845577
1	Support Vector Machine	2.871226	12.949228	3.598504	0.824894

Hyperparameter tunning

In [83]:

```
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, re
turn_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_para
ms'])

find_best_model_using_gridsearchcv(x,y)
```

Out[83]:

	model	best_score	best_params
0	linear_regression	0.763902	{'normalize': True}
1	lasso	0.736687	{'alpha': 1, 'selection': 'cyclic'}
2	decision_tree	0.684342	{'criterion': 'mse', 'splitter': 'best'}

3.Lasso

Ridge regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of coefficients.The lasso coefficients minimize a penalized residual sum of squares. A linear model that estimates sparse coefficients.

Mathematically, it consists of a linear model trained with ℓ1 prior as regularizer. The objective function to minimize is:

min∑(yi-y_pred)^2+λ|m|

In [84]:
```python
from sklearn.linear_model import Lasso
lr_lasso=Lasso(random_state=2,alpha=2.0,selection='cyclic')
lr_lasso.fit(x_train,y_train)
```

Out[84]:
```
Lasso(alpha=2.0, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=
2,
      selection='cyclic', tol=0.0001, warm_start=False)
```

In [85]:
```python
lr_lasso.intercept_
```

Out[85]:
```
7.736090221484883
```

In [86]:
```python
pred=lr_lasso.predict(x_test)
pred
```

Out[86]:
```
array([19.8920971 , 40.33334922, 33.09928201, 25.10017864, 25.70056007,
       15.96040624, 24.22983905, 24.6200777 , 38.05219679])
```

In [87]:
```python
test_pred = lr_lasso.predict(x_test)
train_pred = lr_lasso.predict(x_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)
print('====================================')
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)
```

```
Test set evaluation:
_____________________________________
MAE: 2.9620984837717574
MSE: 12.856576831523316
RMSE: 3.5856068986328267
R2 Square 0.8261464568692173
====================================
Train set evaluation:
_____________________________________
MAE: 2.7260527683671927
```

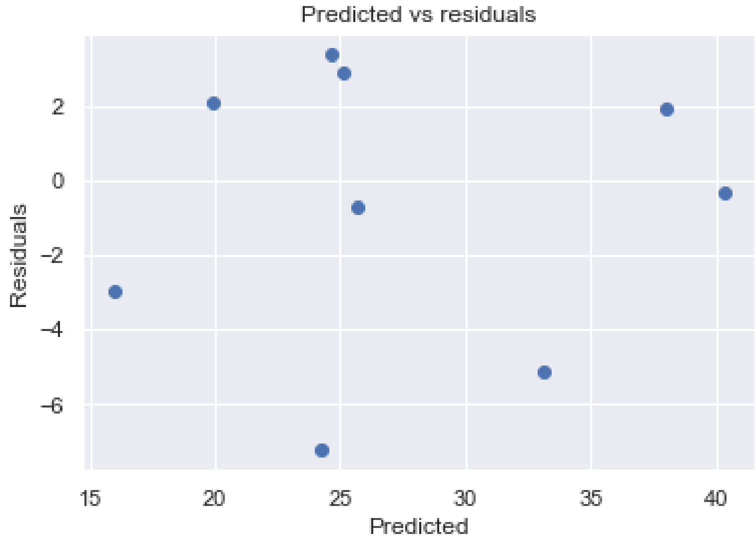
```
MSE: 13.339179370939574
RMSE: 3.652284130642025
R2 Square 0.8222763822662787
```

In [88]:

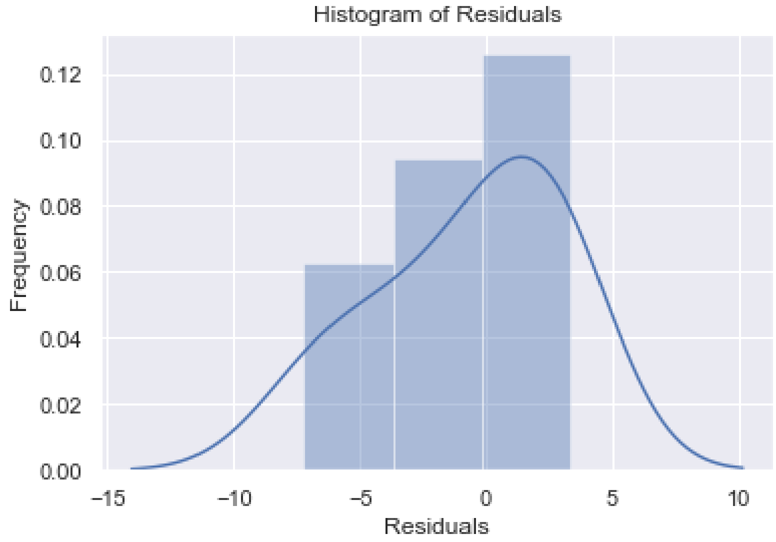
```
# Visualizing the differences between actual prices and predicted value
s
plt.scatter(y_test, pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```

In [89]:

```
# Checking residuals
plt.scatter(pred,y_test-pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```



In [90]:
```
# Checking Normality of errors
sns.distplot(y_test-pred)
plt.title("Histogram of Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()
```



In [91]:
```
def predict_price(OverallCond,area,bhk,ageofbuilding,location):
    loc_index = np.where(x.columns==location)[0][0]

    X = np.zeros(len(x.columns))
    X[0] = OverallCond
    X[1] = area
    X[2] = bhk
    X[3] = ageofbuilding
    if loc_index >= 0:
        X[loc_index] = 1

    return lr_lasso.predict([X])[0]
```

In [92]:
```
predict_price(9,684,2,0,'arvind marg')
```

Out[92]: 23.13428679804992

In [93]:
```
results_df_2 = pd.DataFrame(data=[["Lasso", *evaluate(y_test, test_pred
) ]],
                            columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2
 Square',])
```

```
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[93]:

	Model	MAE	MSE	RMSE	R2 Square
0	Linear Regression	2.782366	11.419699	3.379304	0.845577
1	Support Vector Machine	2.871226	12.949228	3.598504	0.824894
2	Lasso	2.962098	12.856577	3.585607	0.826146

4.Ridge Regression

Ridge regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of coefficients. The ridge coefficients minimize a penalized residual sum of squares,

min∑(yi-y_pred)^2+λ[mi^2] where i=1,2,....,n and mi is the slope.

In [94]:
```
from sklearn.linear_model import Ridge
model = Ridge(alpha=100, solver='cholesky', tol=0.0001, random_state=42
)
model.fit(x_train, y_train)

pred = model.predict(x_test)

test_pred = model.predict(x_test)
train_pred = model.predict(x_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)
print('====================================')
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)
```

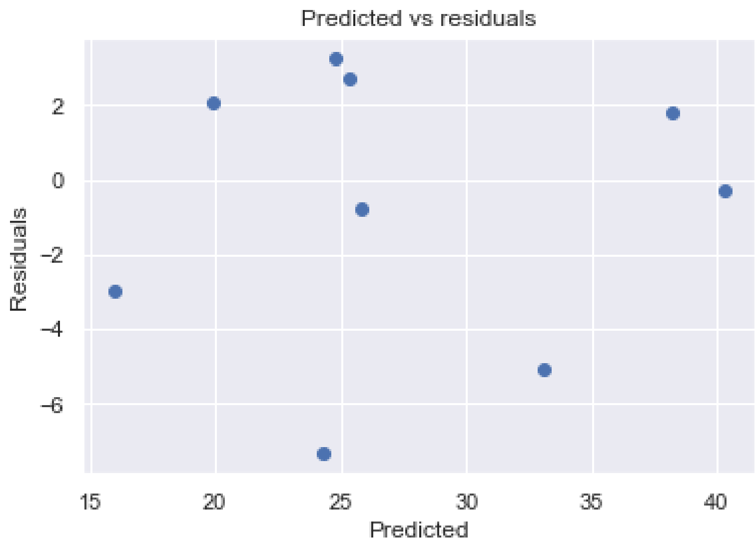
```
Test set evaluation:
_____________________________________
MAE: 2.9142324745969574
```

```
MSE: 12.698143679144913
RMSE: 3.5634454786266776
R2 Square 0.8282888751234161
=====================================
Train set evaluation:
_____________________________________
MAE: 2.6631028257770346
MSE: 12.934488096302742
RMSE: 3.5964549345574652
R2 Square 0.8276682579876907
```

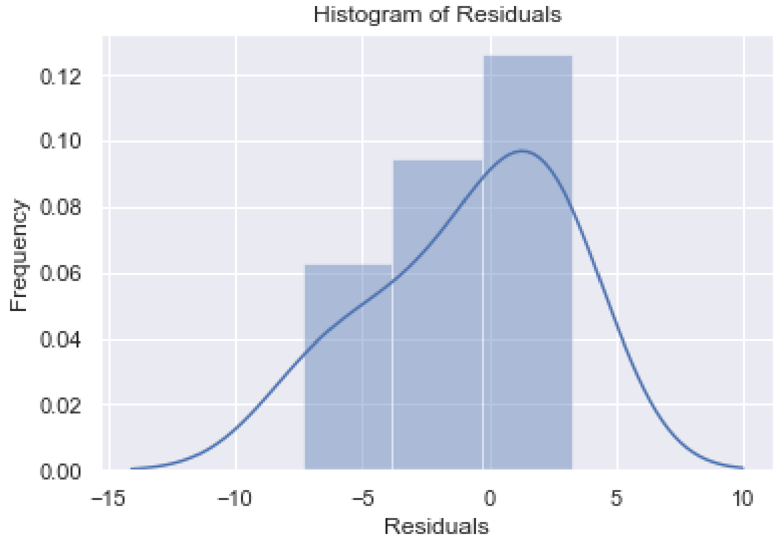
In [95]:
```python
# Visualizing the differences between actual prices and predicted value
s
plt.scatter(y_test, pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```

In [96]:
```python
# Checking residuals
```

```python
plt.scatter(pred,y_test-pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```



In [97]:
```python
# Checking Normality of errors
sns.distplot(y_test-pred)
plt.title("Histogram of Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()
```



In [98]:
```
results_df_2 = pd.DataFrame(data=[["Ridge Regression", *evaluate(y_test, test_pred) ]],
                            columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[98]:

	Model	MAE	MSE	RMSE	R2 Square
0	Linear Regression	2.782366	11.419699	3.379304	0.845577
1	Support Vector Machine	2.871226	12.949228	3.598504	0.824894
2	Lasso	2.962098	12.856577	3.585607	0.826146
3	Ridge Regression	2.914232	12.698144	3.563445	0.828289

Ensemble Learning

5. Bagging

In [99]:
```python
from sklearn.ensemble import BaggingRegressor
from sklearn import tree
```

In [100]:
```python
bg=BaggingRegressor(tree.DecisionTreeRegressor(random_state=20))
```

In [101]:
```python
bg.fit(x_train,y_train)
```

Out[101]:
```
BaggingRegressor(base_estimator=DecisionTreeRegressor(ccp_alpha=0.0,
                                                      criterion='mse',
                                                      max_depth=None,
                                                      max_features=Non
e,
                                                      max_leaf_nodes=No
ne,
                                                      min_impurity_decr
ease=0.0,
                                                      min_impurity_spli
t=None,
                                                      min_samples_leaf=
1,
                                                      min_samples_split
=2,
                                                      min_weight_fracti
on_leaf=0.0,
                                                      presort='deprecat
ed',
                                                      random_state=20,
                                                      splitter='best'),
                 bootstrap=True, bootstrap_features=False, max_features
=1.0,
                 max_samples=1.0, n_estimators=10, n_jobs=None, oob_sco
re=False,
                 random_state=None, verbose=0, warm_start=False)
```

In [102]:
```python
pred=bg.predict(x_test)
pred
```

Out[102]:
```
array([21.178, 37.2  , 30.5  , 29.35 , 28.352, 13.85 , 24.456, 24.405,
       42.   ])
```

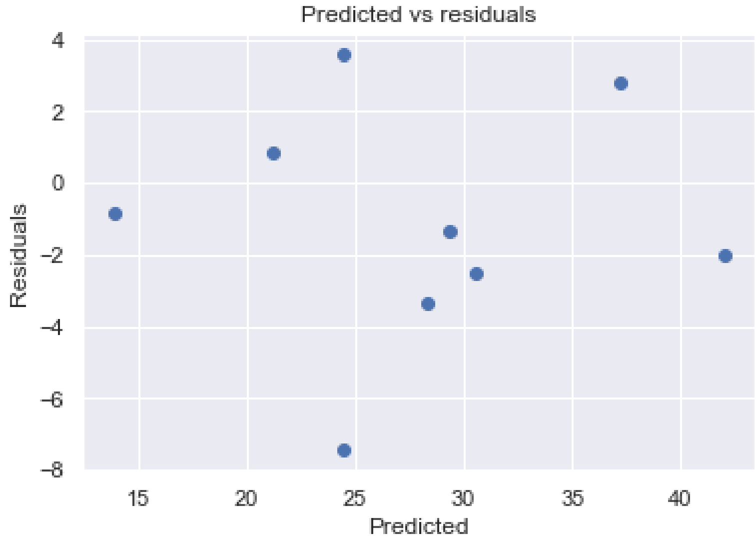
In [103]:
```
test_pred = bg.predict(x_test)
train_pred = bg.predict(x_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)
print('====================================')
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)
```

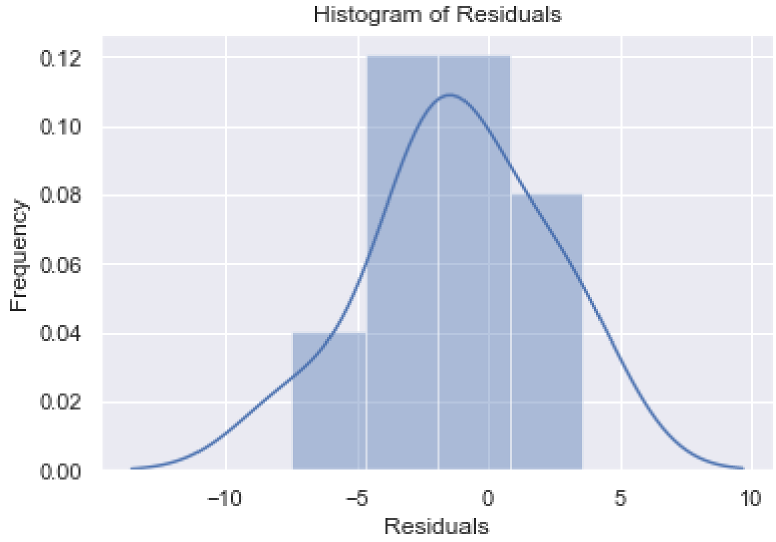
```
Test set evaluation:
_____________________________________
MAE: 2.747222222222222
MSE: 11.229172111111106
RMSE: 3.3509956895094786
R2 Square 0.848153098330551
====================================
Train set evaluation:
_____________________________________
MAE: 1.4286
MSE: 3.5975016857142865
RMSE: 1.8967081182180578
R2 Square 0.9520689394295724
```

In [104]:
```
# Visualizing the differences between actual prices and predicted value
s
plt.scatter(y_test, pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```

In [105]:
```
# Checking residuals
plt.scatter(pred,y_test-pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```



In [106]:
```
# Checking Normality of errors
sns.distplot(y_test-pred)
plt.title("Histogram of Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()
```



In [107]:
```
def predict_price(OverallCond,area,bhk,ageofbuilding,location):
    loc_index = np.where(x.columns==location)[0][0]

    X = np.zeros(len(x.columns))
    X[0] = OverallCond
    X[1] = area
    X[2] = bhk
    X[3] = ageofbuilding
    if loc_index >= 0:
        X[loc_index] = 1

    return bg.predict([X])[0]
```

In [108]:
```
predict_price(9,684,2,0,'arvind marg')
```

Out[108]: 23.55

In [109]:
```
#  lets try =model is not overfit
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
```

```
cv = ShuffleSplit(n_splits=3, test_size=0.2, random_state=0)

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

Out[109]:
```
array([0.7649708 , 0.81388056, 0.76718494])
```

In [110]:
```
results_df_2 = pd.DataFrame(data=[["Bagging", *evaluate(y_test, test_pr
ed) ]],
                            columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2
 Square'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[110]:

	Model	MAE	MSE	RMSE	R2 Square
0	Linear Regression	2.782366	11.419699	3.379304	0.845577
1	Support Vector Machine	2.871226	12.949228	3.598504	0.824894
2	Lasso	2.962098	12.856577	3.585607	0.826146
3	Ridge Regression	2.914232	12.698144	3.563445	0.828289
4	Bagging	2.747222	11.229172	3.350996	0.848153

6.Random forest

In [111]:
```
# 6.Random forest
from sklearn.ensemble import RandomForestRegressor
rfr=RandomForestRegressor(n_estimators=100,min_samples_split=2,min_samp
les_leaf=2,max_depth=780,max_features='auto',random_state=1)
```

In [112]:
```
rfr.fit(x_train,y_train)
```

Out[112]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=780, max_features='auto', max_leaf_node
s=None,
                      max_samples=None, min_impurity_decrease=0.0,
```

```
                                   min_impurity_split=None, min_samples_leaf=2,
                                   min_samples_split=2, min_weight_fraction_leaf=0.
0,
                                   n_estimators=100, n_jobs=None, oob_score=False,
                                   random_state=1, verbose=0, warm_start=False)
```

In [113]:
```
print('Random Forest:',rfr.score(x_test,y_test))
```

```
Random Forest: 0.8514510702238111
```

In [114]:
```
pred=rfr.predict(x_test)
```

In [115]:
```
test_pred = rfr.predict(x_test)
train_pred = rfr.predict(x_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)
print('====================================')
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)
```

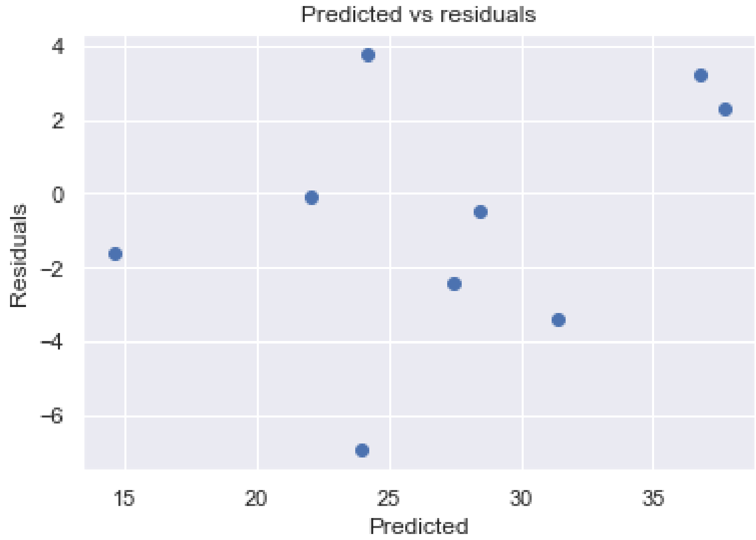
```
Test set evaluation:
_____________________________________
MAE: 2.7005042283950633
MSE: 10.985285053819402
RMSE: 3.3144056863666225
R2 Square 0.8514510702238112
====================================
Train set evaluation:
_____________________________________
MAE: 1.9300817907647894
MSE: 7.388568718974037
RMSE: 2.7181921784476604
R2 Square 0.9015589245714011
```

In [116]:
```
# Visualizing the differences between actual prices and predicted value
s
plt.scatter(y_test, pred)
```

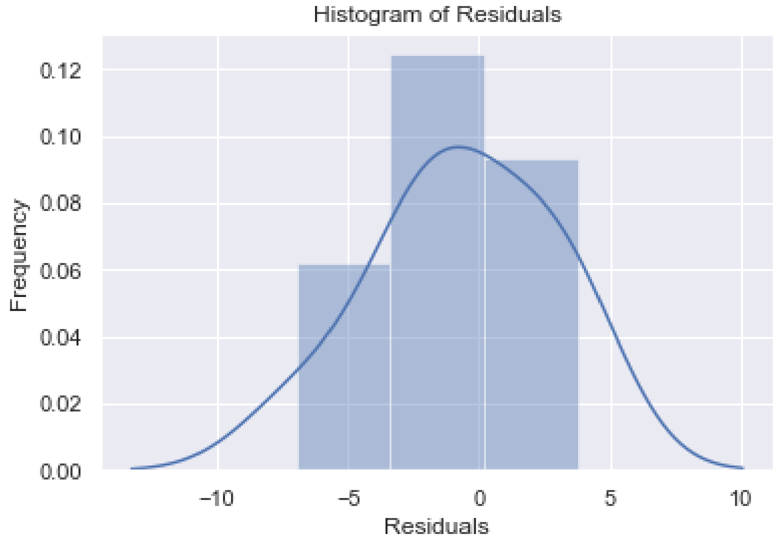
```
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```

In [117]:

```
# Checking residuals
plt.scatter(pred,y_test-pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```



In [118]:
```
# Checking Normality of errors
sns.distplot(y_test-pred)
plt.title("Histogram of Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()
```



Hyperparameter tunning

In [119]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [120]:
```
rfr=RandomForestRegressor()
```

In [121]:
```
import warnings
warnings.filterwarnings('ignore')# we want to ignore warning
```

In [122]:
```
from sklearn.ensemble import RandomForestRegressor
import numpy as np
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1000, n
um = 10)]
# Number of features to consider at every split
max_features = ['auto']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 1000,10)]
```

```
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10,14]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4,6,8]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}
print(random_grid)
```

```
{'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000],
'max_features': ['auto'], 'max_depth': [10, 120, 230, 340, 450, 560, 67
0, 780, 890, 1000], 'min_samples_split': [2, 5, 10, 14], 'min_samples_l
eaf': [1, 2, 4, 6, 8]}
```

In [123]:
```
rf=RandomForestRegressor()
rf_randomcv=RandomizedSearchCV(estimator=rf,param_distributions=random_
grid,n_iter=10,cv=3,verbose=2,
                               random_state=100,n_jobs=-1)
```

In [124]:
```
### fit the randomized model
rf_randomcv.fit(x_train,y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:   21.6s finished
```

Out[124]:
```
RandomizedSearchCV(cv=3, error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   max_samples=None,
```

```
                                                min_impurity_decreas
e=0.0,
                                                min_impurity_split=N
one,
                                                min_samples_leaf=1,
                                                min_samples_split=2,
                                                min_weight_fraction_
leaf=0.0,
                                                n_estimators=100,
                                                n_jobs=None, oob_sco
re=Fals...
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'max_depth': [10, 120, 230, 34
0, 450,
                                                   560, 670, 780, 89
0,
                                                   1000],
                                        'max_features': ['auto'],
                                        'min_samples_leaf': [1, 2, 4,
6, 8],
                                        'min_samples_split': [2, 5, 10,
14],
                                        'n_estimators': [100, 200, 300,
400,
                                                      500, 600, 700,
800,
                                                      900, 1000]},
                   pre_dispatch='2*n_jobs', random_state=100, refit=Tru
e,
                   return_train_score=False, scoring=None, verbose=2)
```

In [125]:
```
rf_randomcv.best_params_
```

Out[125]:
```
{'n_estimators': 1000,
 'min_samples_split': 5,
 'min_samples_leaf': 2,
 'max_features': 'auto',
 'max_depth': 1000}
```

In [126]:
```
best_random_grid=rf_randomcv.best_estimator_
best_random_grid
```

Out[126]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=1000, max_features='auto', max_leaf_nod
es=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=2,
                      min_samples_split=5, min_weight_fraction_leaf=0.
0,
                      n_estimators=1000, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

In [127]:
```
test_pred = best_random_grid.predict(x_test)
train_pred = best_random_grid.predict(x_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)
print('====================================')
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)
```

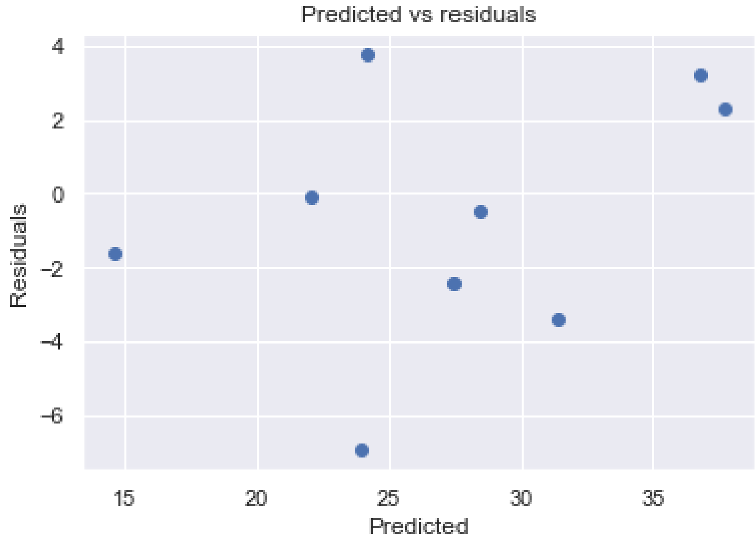
```
Test set evaluation:
_____________________________________
MAE: 2.882783834616019
MSE: 12.039249893332519
RMSE: 3.4697622243220816
R2 Square 0.8371987910918307
====================================
Train set evaluation:
_____________________________________
MAE: 2.0650270307153216
MSE: 8.389809125020802
RMSE: 2.896516722724176
R2 Square 0.8882189684739912
```

In [128]:
```
# Visualizing the differences between actual prices and predicted value
s
plt.scatter(y_test, pred)
```

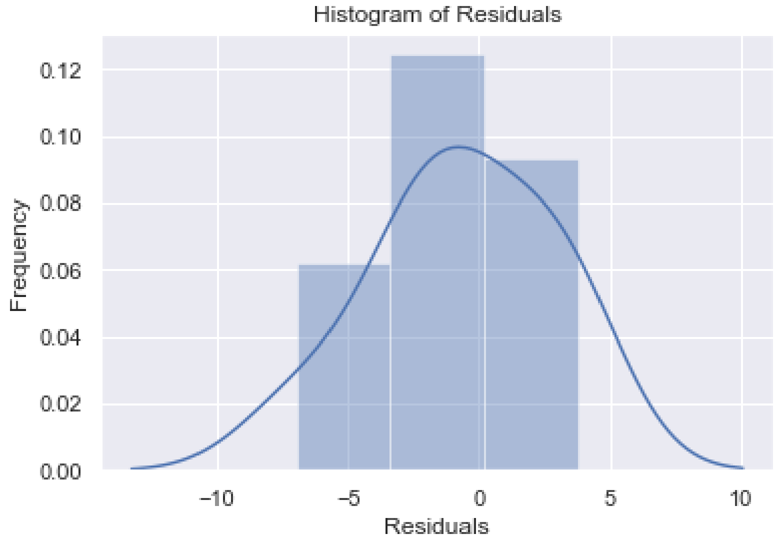
```
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```

In [129]:

```
# Checking residuals
plt.scatter(pred,y_test-pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```



In [130]:
```
# Checking Normality of errors
sns.distplot(y_test-pred)
plt.title("Histogram of Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()
```



In [131]:
```
def predict_price(OverallCond,area,bhk,ageofbuilding,location):
    loc_index = np.where(x.columns==location)[0][0]

    X = np.zeros(len(x.columns))
    X[0] = OverallCond
    X[1] = area
    X[2] = bhk
    X[3] = ageofbuilding
    if loc_index >= 0:
        X[loc_index] = 1

    return best_random_grid.predict([X])[0]
```

In [132]:
```
predict_price(9,684,2,0,'arvind marg')
```

Out[132]: 21.534304768758986

In [133]:
```
results_df_2 = pd.DataFrame(data=[["Random Forest", *evaluate(y_test, t
est_pred) ]],
                            columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2
 Square'])
```

```
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[133]:

	Model	MAE	MSE	RMSE	R2 Square
0	Linear Regression	2.782366	11.419699	3.379304	0.845577
1	Support Vector Machine	2.871226	12.949228	3.598504	0.824894
2	Lasso	2.962098	12.856577	3.585607	0.826146
3	Ridge Regression	2.914232	12.698144	3.563445	0.828289
4	Bagging	2.747222	11.229172	3.350996	0.848153
5	Random Forest	2.882784	12.039250	3.469762	0.837199

Boosting

7. AdaBoost

In [134]:

```
from sklearn.ensemble import AdaBoostRegressor
adab=AdaBoostRegressor()
```

In [135]:

```
adab.fit(x_train,y_train)
```

Out[135]:

```
AdaBoostRegressor(base_estimator=None, learning_rate=1.0, loss='linea
r',
                  n_estimators=50, random_state=None)
```

In [136]:

```
pred=adab.predict(x_test)
pred
```

Out[136]:

```
array([23.25      , 35.6       , 30.68181818, 27.25      , 27.14055556,
       14.671875  , 25.255     , 25.255     , 45.        ])
```

In [137]:

```
test_pred = adab.predict(x_test)
```

```
train_pred = adab.predict(x_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)
print('====================================')
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)
```

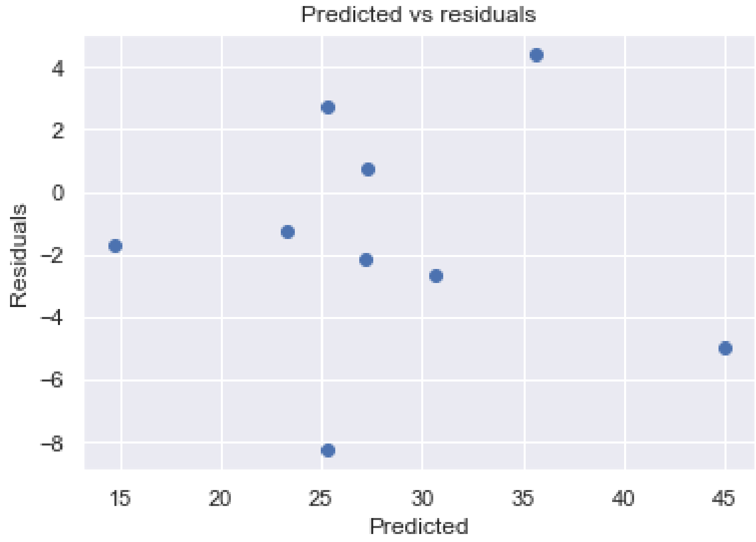
```
Test set evaluation:
_____________________________________
MAE: 3.210472081930415
MSE: 15.192704762486152
RMSE: 3.897782031166719
R2 Square 0.7945560791717231
====================================
Train set evaluation:
_____________________________________
MAE: 1.3670514172335597
MSE: 2.597539117523418
RMSE: 1.611688281747875
R2 Square 0.9653918703442247
```

In [138]:

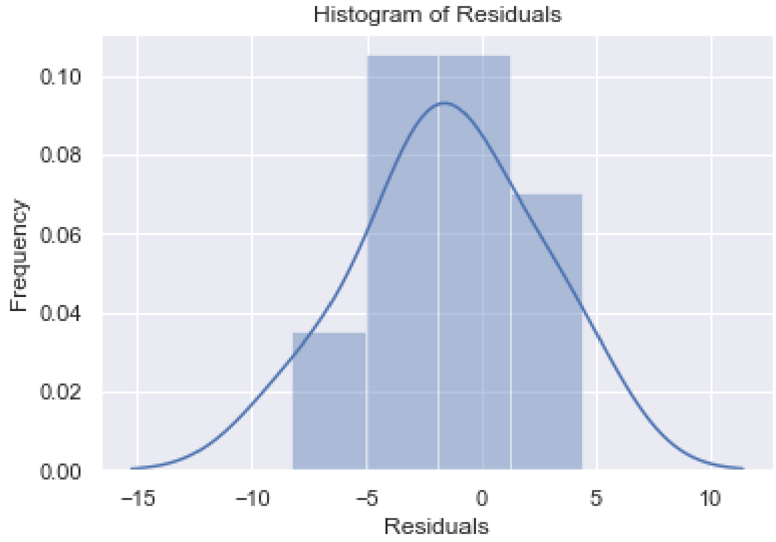
```
# Visualizing the differences between actual prices and predicted value
s
plt.scatter(y_test, pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```

In [139]:

```
# Checking residuals
plt.scatter(pred,y_test-pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```



In [140]:
```
# Checking Normality of errors
sns.distplot(y_test-pred)
plt.title("Histogram of Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()
```



In [141]:
```
def predict_price(OverallCond,area,bhk,ageofbuilding,location):
    loc_index = np.where(x.columns==location)[0][0]

    X = np.zeros(len(x.columns))
    X[0] = OverallCond
    X[1] = area
    X[2] = bhk
    X[3] = ageofbuilding
    if loc_index >= 0:
        X[loc_index] = 1

    return adab.predict([X])[0]
```

In [142]:
```
predict_price(9,684,2,0,'arvind marg')
```

Out[142]: 23.982142857142858

In [143]:
```
results_df_2 = pd.DataFrame(data=[["Adaboost", *evaluate(y_test, test_p
red) ]],
                            columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2
 Square'])
```

```
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[143]:

	Model	MAE	MSE	RMSE	R2 Square
0	Linear Regression	2.782366	11.419699	3.379304	0.845577
1	Support Vector Machine	2.871226	12.949228	3.598504	0.824894
2	Lasso	2.962098	12.856577	3.585607	0.826146
3	Ridge Regression	2.914232	12.698144	3.563445	0.828289
4	Bagging	2.747222	11.229172	3.350996	0.848153
5	Random Forest	2.882784	12.039250	3.469762	0.837199
6	Adaboost	3.210472	15.192705	3.897782	0.794556

8.GradientBoosting

In [144]:

```
from sklearn.ensemble import GradientBoostingRegressor
gbr=GradientBoostingRegressor()
gbr.fit(x_train,y_train)
```

Out[144]:

```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman
_mse',
                          init=None, learning_rate=0.1, loss='ls', max_
depth=3,
                          max_features=None, 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=10
0,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_star
t=False)
```

In [145]:
```
pred=gbr.predict(x_test)
pred
```

Out[145]:
```
array([18.1513634 , 37.35688315, 30.76372438, 28.38539182, 29.54777362,
       14.2098971 , 24.07263301, 24.9220838 , 43.35434339])
```

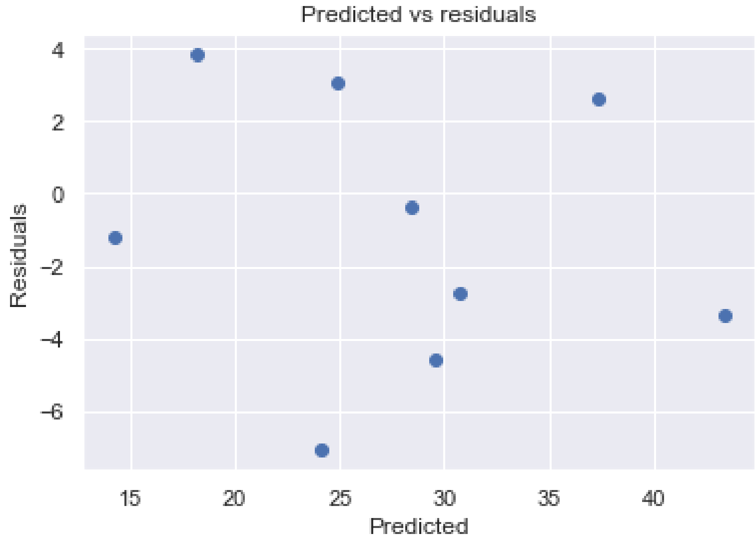
In [146]:
```
est_pred = gbr.predict(x_test)
train_pred = gbr.predict(x_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)
print('====================================')
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)
```

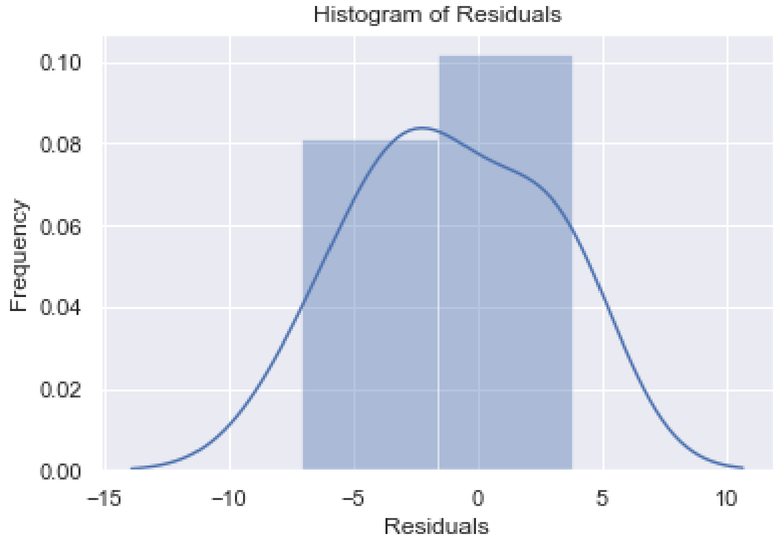
```
Test set evaluation:
_____________________________________
MAE: 3.210472081930415
MSE: 15.192704762486152
RMSE: 3.897782031166719
R2 Square 0.7945560791717231
====================================
Train set evaluation:
_____________________________________
MAE: 0.22039402856122678
MSE: 0.08054660193475044
RMSE: 0.28380733241893247
R2 Square 0.9989268430168059
```

In [147]:
```
# Visualizing the differences between actual prices and predicted value
s
plt.scatter(y_test, pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```

In [148]:
```
# Checking residuals
plt.scatter(pred,y_test-pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```



In [149]:
```
# Checking Normality of errors
sns.distplot(y_test-pred)
plt.title("Histogram of Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()
```



In [150]:
```
def predict_price(OverallCond,area,bhk,ageofbuilding,location):
    loc_index = np.where(x.columns==location)[0][0]

    X = np.zeros(len(x.columns))
    X[0] = OverallCond
    X[1] = area
    X[2] = bhk
    X[3] = ageofbuilding
    if loc_index >= 0:
        X[loc_index] = 1

    return gbr.predict([X])[0]
```

In [151]:
```
predict_price(9,684,2,0,'arvind marg')
```

Out[151]:
```
25.23584490465953
```

In [152]:
```
results_df_2 = pd.DataFrame(data=[["GradientBoosting", *evaluate(y_test
, test_pred) ]],
                            columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2
 Square'])
```

```
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[152]:

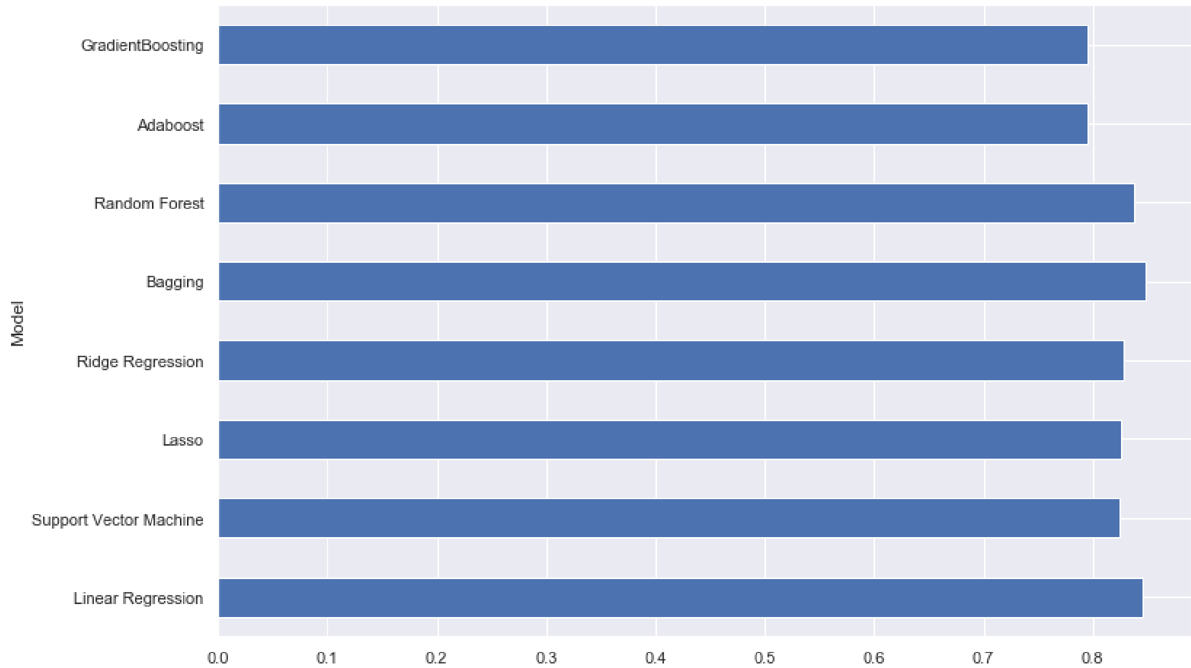
	Model	MAE	MSE	RMSE	R2 Square
0	Linear Regression	2.782366	11.419699	3.379304	0.845577
1	Support Vector Machine	2.871226	12.949228	3.598504	0.824894
2	Lasso	2.962098	12.856577	3.585607	0.826146
3	Ridge Regression	2.914232	12.698144	3.563445	0.828289
4	Bagging	2.747222	11.229172	3.350996	0.848153
5	Random Forest	2.882784	12.039250	3.469762	0.837199
6	Adaboost	3.210472	15.192705	3.897782	0.794556
7	GradientBoosting	3.210472	15.192705	3.897782	0.794556

Models Comparison

In [153]:

```
results_df.set_index('Model', inplace=True)
results_df['R2 Square'].plot(kind='barh', figsize=(12, 8))
```

Out[153]: <matplotlib.axes._subplots.AxesSubplot at 0x10265cf0>



Conclusion

when we look at the model accurcy table Bagging,Linear model,Random forest is working very well.In which Bagging is working very well if we compare with all other models.Bagging have 0.871260 accuracy,it means almost 87% values are predicted well. We have built our model on very small dataset,if we increase this dataset accuracy will be increase.

References:

https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

In [ ]: