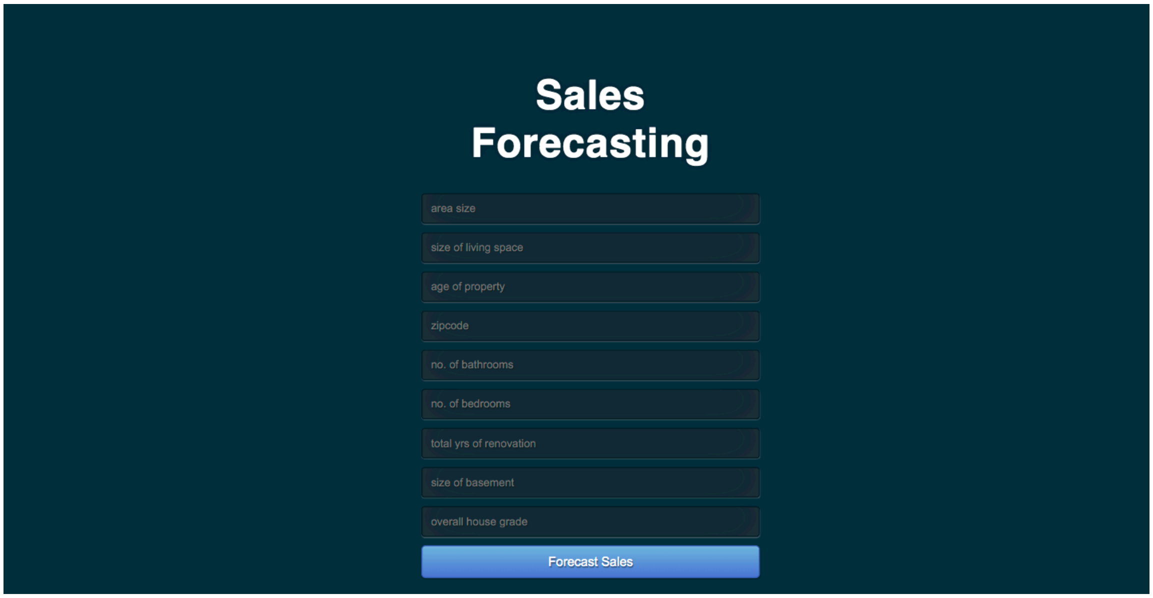


Sales Forcast

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, SGDRegressor, Ridge, Lass
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, Baggi
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from xgboost import XGBRegressor
from sklearn.decomposition import PCA, KernelPCA
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.pipeline import Pipeline
```

In [ ]:
```
data = pd.read_csv('/content/kc_house_data.csv')
data
```

Out[ ]:

	id	date	price	bedrooms	bathrooms	sqft_liv
0	7129300520	20141013T000000	221900.0	3	1.00	1
1	6414100192	20141209T000000	538000.0	3	2.25	2
2	5631500400	20150225T000000	180000.0	2	1.00	
3	2487200875	20141209T000000	604000.0	4	3.00	1
4	1954400510	20150218T000000	510000.0	3	2.00	1
...	...	...	...	...	...	
21608	263000018	20140521T000000	360000.0	3	2.50	1
21609	6600060120	20150223T000000	400000.0	4	2.50	2
21610	1523300141	20140623T000000	402101.0	2	0.75	1
21611	291310100	20150116T000000	400000.0	3	2.50	1
21612	1523300157	20141015T000000	325000.0	2	0.75	1

21613 rows × 21 columns

Field name	Field description
date	Date of sale
price	Final transaction amount
bedrooms	Number of bedrooms
bathrooms	Number of bathrooms
sqft_living	Living space size (in square feet)
sqft_lot	Lot size (in square feet)
floors	Number of floors
waterfront	Is house on waterfront (1: yes, 0: not)
view	Categorical variable for view of the house
condition	Categorical variable for condition of the house
grade	Overall house grade based on King County grading system
sqft_above	Size of the house excluding basement (in square feet)
sqft_basement	Size of the basement (in square feet)
yr_built	Year house was built
yr_renovated	Year house was renovated (if renovated)
zipcode	ZIP Code of the house
lat	Latitude of house
long	Longitude of house
sqft_living15	Size of living space in 2015 (in square feet)
sqrt_lot15	Size of lot in 2015 (in square feet)

Some of the fields contain codes for specific values. These are explained below.

Condition	Condition	Description
1	Poor	Many repairs needed. House is showing serious deterioration.
2	Fair	Some repairs needed immediately. Much deferred maintenance is needed.
3	Average	Depending upon age of improvement, normal amount of upkeep for the age of the home.
4	Good	Condition above the norm for the age of the home. This indicates extra attention and care has been taken to maintain it.
5	Very Good	Excellent maintenance and updating on home; not a total renovation.

Grade	Description
1-3	Falls short of minimum building standards; normally cabin or inferior structure.
4	Generally older low quality construction. The house does not meet code.
5	Lower construction costs and workmanship. The house has small, simple design.
6	Lowest grade currently meeting building codes. Low-quality materials and simple designs were used.
7	Average grade of construction and design. This is commonly seen in plats and older subdivisions.
8	Just above average in construction and design. Houses of this quality usually have better materials in both the exterior and interior finishes.
9	Better architectural design, with extra exterior and interior design and quality.
10	Homes of this quality generally have high-quality features. Finish work is better, and more design quality is seen in the floor plans and larger square footage.
11	Custom design and higher quality finish work, with added amenities of solid woods, bathroom fixtures, and more luxurious options.
12	Custom design and excellent builders. All materials are of the highest quality and all conveniences are present.
13	Generally custom designed and built, approaching the mansion level. These houses have a large amount of highest quality cabinet work, wood trim, and marble with large entries.

View	Description
0	Unknown
1	Fair
2	Average
3	Good
4	Excellent

In [ ]:
```
#see the datatype of each column
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21613 non-null  int64  
 1   date           21613 non-null  object 
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64  
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64  
 6   sqft_lot       21613 non-null  int64  
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64  
 9   view           21613 non-null  int64  
 10  condition      21613 non-null  int64  
 11  grade          21613 non-null  int64  
 12  sqft_above     21611 non-null  float64
 13  sqft_basement  21613 non-null  int64  
 14  yr_built       21613 non-null  int64  
 15  yr_renovated   21613 non-null  int64  
 16  zipcode        21613 non-null  int64  
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64  
 20  sqft_lot15     21613 non-null  int64  
dtypes: float64(6), int64(14), object(1)
memory usage: 3.5+ MB
```

In [ ]:

```
data.isnull().any().sum() #you can also add .sum()
```

Out[ ]: 1

In [ ]:

```
#fill all the values with 0
data.fillna(0, inplace=True)
```

Feature Creation

In [ ]:

```
#format the date
d =[]
for i in data['date'].values:
    d.append(i[:4])

data['date'] = d

# convert everything to same datatype
for i in data.columns:
    data[i]=data[i].astype(float)

#make a new column age of the house
data['age'] = data['date'] - data['yr_built']

#calculate the total years of renovation
data['renov_age'] = np.abs(data['yr_renovated'] - data['yr_built'])
```

```python
data['renov_age'] = data.renov_age.apply(lambda x: x if len(str(int(x)))==2

#remove unwanted columns like yr_built, date, id
data.drop(['id','date', 'yr_built', 'yr_renovated'], axis=1, inplace=True)
data.head()
```

Out[ ]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	vi
0	221900.0	3.0	1.00	1180.0	5650.0	1.0	0.0	
1	538000.0	3.0	2.25	2570.0	7242.0	2.0	0.0	
2	180000.0	2.0	1.00	770.0	10000.0	1.0	0.0	
3	604000.0	4.0	3.00	1960.0	5000.0	1.0	0.0	
4	510000.0	3.0	2.00	1680.0	8080.0	1.0	0.0	

Dealing With Highly Correlated Features

In [ ]:

```python
data.corr()
```

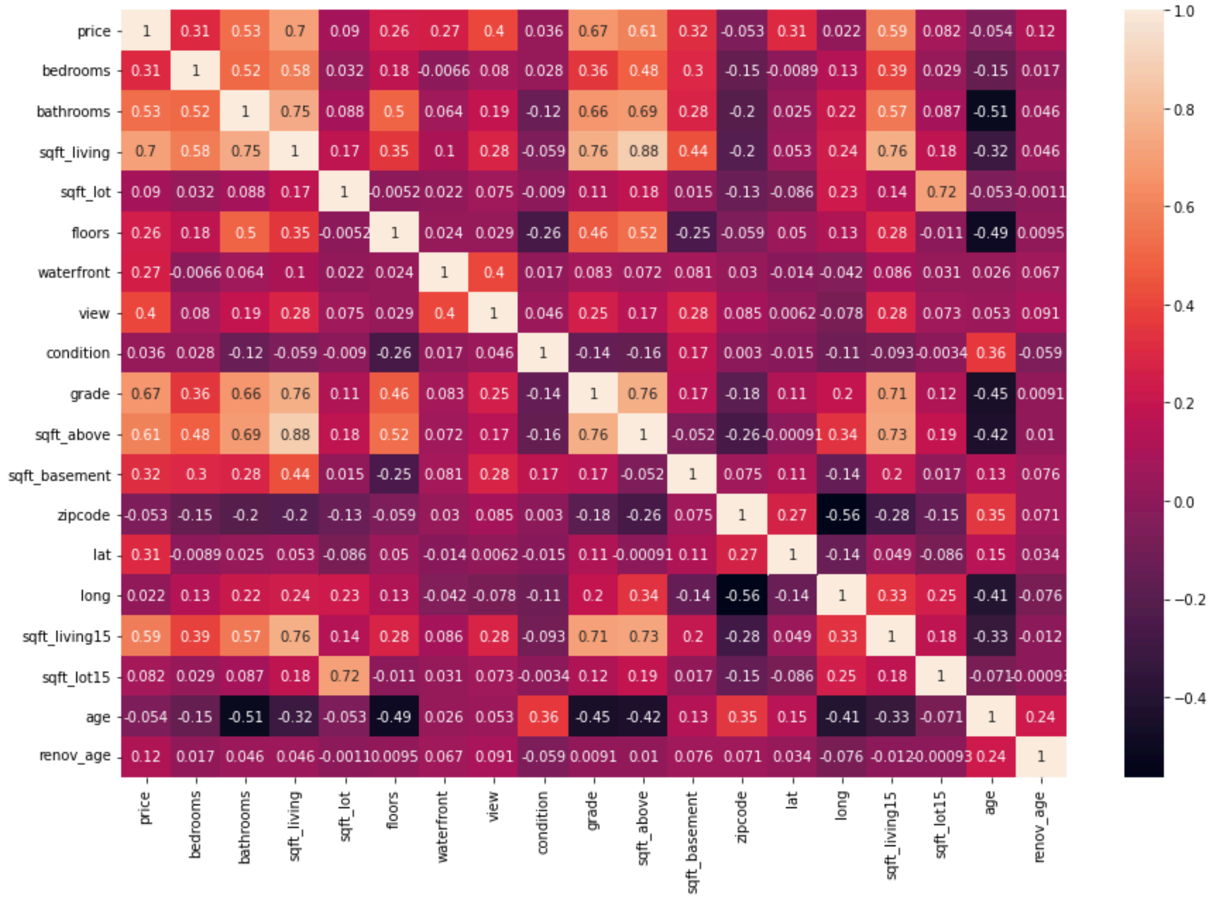
Out[ ]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	flo
price	1.000000	0.308350	0.525138	0.702035	0.089661	0.256
bedrooms	0.308350	1.000000	0.515884	0.576671	0.031703	0.175
bathrooms	0.525138	0.515884	1.000000	0.754665	0.087740	0.500
sqft_living	0.702035	0.576671	0.754665	1.000000	0.172826	0.353
sqft_lot	0.089661	0.031703	0.087740	0.172826	1.000000	-0.005
floors	0.256794	0.175429	0.500653	0.353949	-0.005201	1.000
waterfront	0.266369	-0.006582	0.063744	0.103818	0.021604	0.023
view	0.397293	0.079532	0.187737	0.284611	0.074710	0.029
condition	0.036362	0.028472	-0.124982	-0.058753	-0.008958	-0.263
grade	0.667434	0.356967	0.664983	0.762704	0.113621	0.458
sqft_above	0.605416	0.477479	0.685273	0.876288	0.183510	0.523
sqft_basement	0.323816	0.303093	0.283770	0.435043	0.015286	-0.245
zipcode	-0.053203	-0.152668	-0.203866	-0.199430	-0.129574	-0.059
lat	0.307003	-0.008931	0.024573	0.052529	-0.085683	0.049
long	0.021626	0.129473	0.223042	0.240223	0.229521	0.125
sqft_living15	0.585379	0.391638	0.568634	0.756420	0.144608	0.279
sqft_lot15	0.082447	0.029244	0.087175	0.183286	0.718557	-0.011
age	-0.053951	-0.154324	-0.506407	-0.318488	-0.052990	-0.489
renov_age	0.117200	0.016968	0.045950	0.045653	-0.001141	0.009

In [ ]:

```
# plotting correlation heatmap
plt.figure(figsize=(15,10))
dataplot = sns.heatmap(data.corr(), annot=True)

# displaying heatmap
plt.show()
```



In [ ]:
```
for i , r in data.corr().iterrows():
  print(i)
```

```
price
bedrooms
bathrooms
sqft_living
sqft_lot
floors
waterfront
view
condition
grade
sqft_above
sqft_basement
zipcode
lat
long
sqft_living15
sqft_lot15
age
renov_age
```

In [ ]:
```
for i , r in data.corr().iterrows():
  print(r)
```

```
price            1.000000
bedrooms         0.308350
bathrooms        0.525138
sqft_living      0.702035
sqft_lot         0.089661
floors           0.256794
waterfront       0.266369
view             0.397293
condition        0.036362
grade            0.667434
sqft_above       0.605416
sqft_basement    0.323816
zipcode         -0.053203
lat              0.307003
long             0.021626
sqft_living15    0.585379
sqft_lot15       0.082447
age             -0.053951
renov_age        0.117200
Name: price, dtype: float64
price            0.308350
bedrooms         1.000000
bathrooms        0.515884
sqft_living      0.576671
sqft_lot         0.031703
floors           0.175429
waterfront      -0.006582
view             0.079532
condition        0.028472
grade            0.356967
sqft_above       0.477479
sqft_basement    0.303093
zipcode         -0.152668
lat             -0.008931
long             0.129473
sqft_living15    0.391638
sqft_lot15       0.029244
age             -0.154324
renov_age        0.016968
Name: bedrooms, dtype: float64
price            0.525138
bedrooms         0.515884
bathrooms        1.000000
sqft_living      0.754665
sqft_lot         0.087740
floors           0.500653
waterfront       0.063744
view             0.187737
condition       -0.124982
grade            0.664983
sqft_above       0.685273
sqft_basement    0.283770
zipcode         -0.203866
lat              0.024573
long             0.223042
sqft_living15    0.568634
```

```
sqft_lot15       0.087175
age             -0.506407
renov_age        0.045950
Name: bathrooms, dtype: float64
price            0.702035
bedrooms         0.576671
bathrooms        0.754665
sqft_living      1.000000
sqft_lot         0.172826
floors           0.353949
waterfront       0.103818
view             0.284611
condition       -0.058753
grade            0.762704
sqft_above       0.876288
sqft_basement    0.435043
zipcode         -0.199430
lat              0.052529
long             0.240223
sqft_living15    0.756420
sqft_lot15       0.183286
age             -0.318488
renov_age        0.045653
Name: sqft_living, dtype: float64
price            0.089661
bedrooms         0.031703
bathrooms        0.087740
sqft_living      0.172826
sqft_lot         1.000000
floors          -0.005201
waterfront       0.021604
view             0.074710
condition       -0.008958
grade            0.113621
sqft_above       0.183510
sqft_basement    0.015286
zipcode         -0.129574
lat             -0.085683
long             0.229521
sqft_living15    0.144608
sqft_lot15       0.718557
age             -0.052990
renov_age       -0.001141
Name: sqft_lot, dtype: float64
price            0.256794
bedrooms         0.175429
bathrooms        0.500653
sqft_living      0.353949
sqft_lot        -0.005201
floors           1.000000
waterfront       0.023698
view             0.029444
condition       -0.263768
grade            0.458183
sqft_above       0.523867
sqft_basement   -0.245705
```

```
zipcode         -0.059121
lat              0.049614
long             0.125419
sqft_living15    0.279885
sqft_lot15      -0.011269
age             -0.489640
renov_age        0.009475
Name: floors, dtype: float64
price            0.266369
bedrooms        -0.006582
bathrooms        0.063744
sqft_living      0.103818
sqft_lot         0.021604
floors           0.023698
waterfront       1.000000
view             0.401857
condition        0.016653
grade            0.082775
sqft_above       0.072076
sqft_basement    0.080588
zipcode          0.030285
lat             -0.014274
long            -0.041910
sqft_living15    0.086463
sqft_lot15       0.030703
age              0.026093
renov_age        0.066727
Name: waterfront, dtype: float64
price            0.397293
bedrooms         0.079532
bathrooms        0.187737
sqft_living      0.284611
sqft_lot         0.074710
floors           0.029444
waterfront       0.401857
view             1.000000
condition        0.045990
grade            0.251321
sqft_above       0.167673
sqft_basement    0.276947
zipcode          0.084827
lat              0.006157
long            -0.078400
sqft_living15    0.280439
sqft_lot15       0.072575
age              0.053458
renov_age        0.091207
Name: view, dtype: float64
price            0.036362
bedrooms         0.028472
bathrooms       -0.124982
sqft_living     -0.058753
sqft_lot        -0.008958
floors          -0.263768
waterfront       0.016653
view             0.045990
```

```
condition        1.000000
grade           -0.144674
sqft_above      -0.158195
sqft_basement    0.174105
zipcode          0.003026
lat             -0.014941
long            -0.106500
sqft_living15   -0.092824
sqft_lot15      -0.003406
age              0.360665
renov_age       -0.058816
Name: condition, dtype: float64
price            0.667434
bedrooms         0.356967
bathrooms        0.664983
sqft_living      0.762704
sqft_lot         0.113621
floors           0.458183
waterfront       0.082775
view             0.251321
condition       -0.144674
grade            1.000000
sqft_above       0.755781
sqft_basement    0.168392
zipcode         -0.184862
lat              0.114084
long             0.198372
sqft_living15    0.713202
sqft_lot15       0.119248
age             -0.447415
renov_age        0.009081
Name: grade, dtype: float64
price            0.605416
bedrooms         0.477479
bathrooms        0.685273
sqft_living      0.876288
sqft_lot         0.183510
floors           0.523867
waterfront       0.072076
view             0.167673
condition       -0.158195
grade            0.755781
sqft_above       1.000000
sqft_basement   -0.052204
zipcode         -0.261038
lat             -0.000914
long             0.343760
sqft_living15    0.731729
sqft_lot15       0.194051
age             -0.424347
renov_age        0.010148
Name: sqft_above, dtype: float64
price            0.323816
bedrooms         0.303093
bathrooms        0.283770
sqft_living      0.435043
```

```
sqft_lot         0.015286
floors          -0.245705
waterfront       0.080588
view             0.276947
condition        0.174105
grade            0.168392
sqft_above      -0.052204
sqft_basement    1.000000
zipcode          0.074845
lat              0.110538
long            -0.144765
sqft_living15    0.200355
sqft_lot15       0.017276
age              0.132865
renov_age        0.075818
Name: sqft_basement, dtype: float64
price           -0.053203
bedrooms        -0.152668
bathrooms       -0.203866
sqft_living     -0.199430
sqft_lot        -0.129574
floors          -0.059121
waterfront       0.030285
view             0.084827
condition        0.003026
grade           -0.184862
sqft_above      -0.261038
sqft_basement    0.074845
zipcode          1.000000
lat              0.267048
long            -0.564072
sqft_living15   -0.279033
sqft_lot15      -0.147221
age              0.346864
renov_age        0.071398
Name: zipcode, dtype: float64
price            0.307003
bedrooms        -0.008931
bathrooms        0.024573
sqft_living      0.052529
sqft_lot        -0.085683
floors           0.049614
waterfront      -0.014274
view             0.006157
condition       -0.014941
grade            0.114084
sqft_above      -0.000914
sqft_basement    0.110538
zipcode          0.267048
lat              1.000000
long            -0.135512
sqft_living15    0.048858
sqft_lot15      -0.086419
age              0.147647
renov_age        0.033734
Name: lat, dtype: float64
```

```
price            0.021626
bedrooms         0.129473
bathrooms        0.223042
sqft_living      0.240223
sqft_lot         0.229521
floors           0.125419
waterfront      -0.041910
view            -0.078400
condition       -0.106500
grade            0.198372
sqft_above       0.343760
sqft_basement   -0.144765
zipcode         -0.564072
lat             -0.135512
long             1.000000
sqft_living15    0.334605
sqft_lot15       0.254451
age             -0.409323
renov_age       -0.076260
Name: long, dtype: float64
price            0.585379
bedrooms         0.391638
bathrooms        0.568634
sqft_living      0.756420
sqft_lot         0.144608
floors           0.279885
waterfront       0.086463
view             0.280439
condition       -0.092824
grade            0.713202
sqft_above       0.731729
sqft_basement    0.200355
zipcode         -0.279033
lat              0.048858
long             0.334605
sqft_living15    1.000000
sqft_lot15       0.183192
age             -0.326552
renov_age       -0.012424
Name: sqft_living15, dtype: float64
price            0.082447
bedrooms         0.029244
bathrooms        0.087175
sqft_living      0.183286
sqft_lot         0.718557
floors          -0.011269
waterfront       0.030703
view             0.072575
condition       -0.003406
grade            0.119248
sqft_above       0.194051
sqft_basement    0.017276
zipcode         -0.147221
lat             -0.086419
long             0.254451
sqft_living15    0.183192
```

```
sqft_lot15       1.000000
age             -0.070954
renov_age       -0.000929
Name: sqft_lot15, dtype: float64
price           -0.053951
bedrooms        -0.154324
bathrooms       -0.506407
sqft_living     -0.318488
sqft_lot        -0.052990
floors          -0.489640
waterfront       0.026093
view             0.053458
condition        0.360665
grade           -0.447415
sqft_above      -0.424347
sqft_basement    0.132865
zipcode          0.346864
lat              0.147647
long            -0.409323
sqft_living15   -0.326552
sqft_lot15      -0.070954
age              1.000000
renov_age        0.236057
Name: age, dtype: float64
price            0.117200
bedrooms         0.016968
bathrooms        0.045950
sqft_living      0.045653
sqft_lot        -0.001141
floors           0.009475
waterfront       0.066727
view             0.091207
condition       -0.058816
grade            0.009081
sqft_above       0.010148
sqft_basement    0.075818
zipcode          0.071398
lat              0.033734
long            -0.076260
sqft_living15   -0.012424
sqft_lot15      -0.000929
age              0.236057
renov_age        1.000000
Name: renov_age, dtype: float64
```

In [ ]:

```
#print highly correlated variables
corr_features =[]

for i , r in data.corr().iterrows():
    k=0 #counter
    for j in range(len(r)):
        if i!= r.index[k]:
            if r.values[k] >=0.5:
                corr_features.append([i, r.index[k], r.values[k]])
        k += 1
corr_features
```

Out[ ]:
```
[['price', 'bathrooms', 0.5251375054139628],
 ['price', 'sqft_living', 0.7020350546118005],
 ['price', 'grade', 0.6674342560202353],
 ['price', 'sqft_above', 0.6054162591641183],
 ['price', 'sqft_living15', 0.5853789035795692],
 ['bedrooms', 'bathrooms', 0.5158836376158312],
 ['bedrooms', 'sqft_living', 0.5766706925022448],
 ['bathrooms', 'price', 0.5251375054139628],
 ['bathrooms', 'bedrooms', 0.5158836376158312],
 ['bathrooms', 'sqft_living', 0.7546652789673752],
 ['bathrooms', 'floors', 0.5006531725878688],
 ['bathrooms', 'grade', 0.6649825338780723],
 ['bathrooms', 'sqft_above', 0.6852729704767271],
 ['bathrooms', 'sqft_living15', 0.568634289578226],
 ['sqft_living', 'price', 0.7020350546118005],
 ['sqft_living', 'bedrooms', 0.5766706925022448],
 ['sqft_living', 'bathrooms', 0.7546652789673752],
 ['sqft_living', 'grade', 0.7627044764584776],
 ['sqft_living', 'sqft_above', 0.8762879508115581],
 ['sqft_living', 'sqft_living15', 0.7564202590172237],
 ['sqft_lot', 'sqft_lot15', 0.7185567524330374],
 ['floors', 'bathrooms', 0.5006531725878688],
 ['floors', 'sqft_above', 0.5238665894982677],
 ['grade', 'price', 0.6674342560202353],
 ['grade', 'bathrooms', 0.6649825338780723],
 ['grade', 'sqft_living', 0.7627044764584776],
 ['grade', 'sqft_above', 0.7557805208597812],
 ['grade', 'sqft_living15', 0.7132020930151698],
 ['sqft_above', 'price', 0.6054162591641183],
 ['sqft_above', 'bathrooms', 0.6852729704767271],
 ['sqft_above', 'sqft_living', 0.8762879508115581],
 ['sqft_above', 'floors', 0.5238665894982677],
 ['sqft_above', 'grade', 0.7557805208597812],
 ['sqft_above', 'sqft_living15', 0.7317286450832593],
 ['sqft_living15', 'price', 0.5853789035795692],
 ['sqft_living15', 'bathrooms', 0.568634289578226],
 ['sqft_living15', 'sqft_living', 0.7564202590172237],
 ['sqft_living15', 'grade', 0.7132020930151698],
 ['sqft_living15', 'sqft_above', 0.7317286450832593],
 ['sqft_lot15', 'sqft_lot', 0.7185567524330374]]
```

In [ ]:
```python
#let us remove highly correlated features that is above 0.8
feat =[]
for i in corr_features:
    if i[2] >= 0.8: # 2 is the correlation value as it is shown in the corr_
        feat.append(i[0])
        feat.append(i[1])

data.drop(list(set(feat)), axis=1, inplace=True)
```

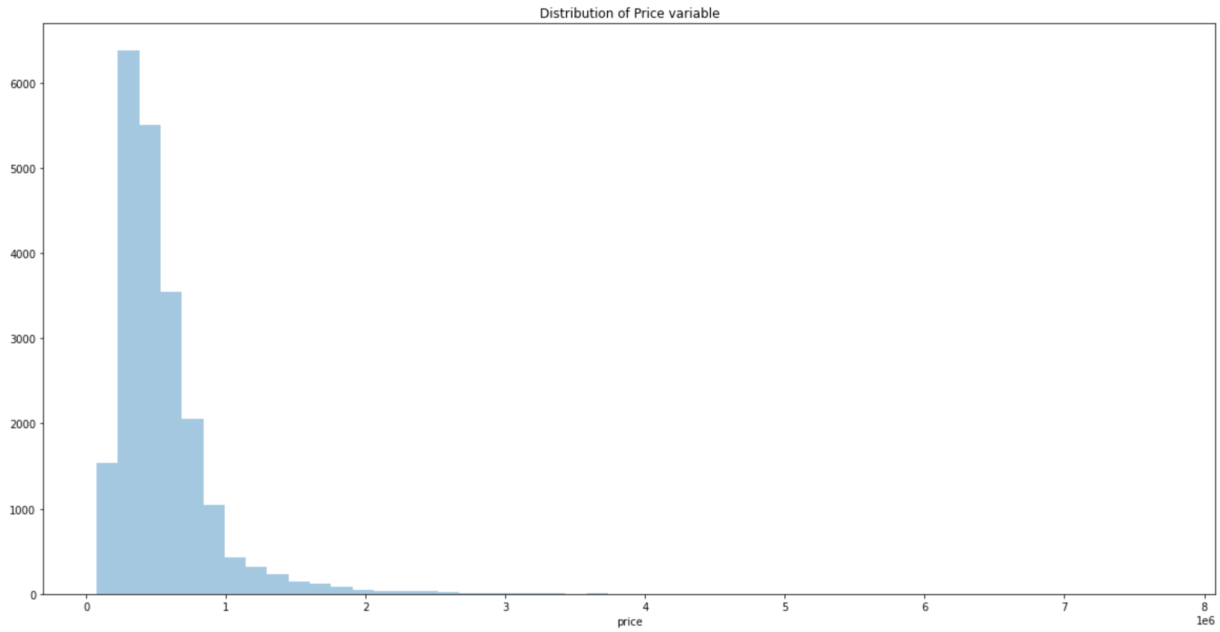
In [ ]:
```python

```

Outlier Detection

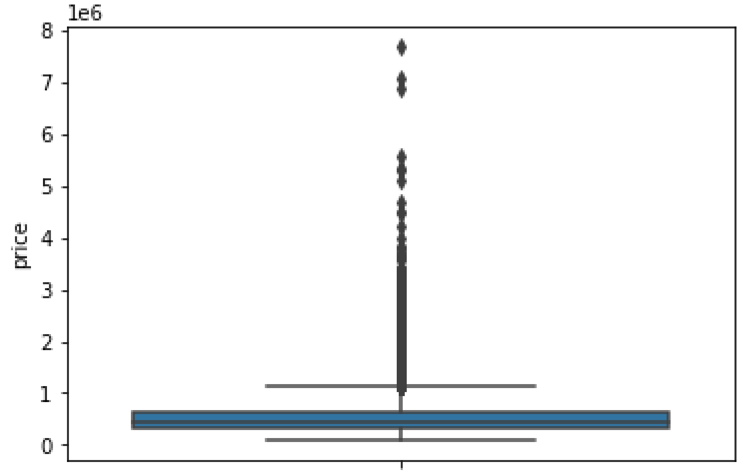
Price Feature Before Dealing With Outliers

In [ ]:
```
#plot the (dependent) variable to see its distribution
# plt.title()
plt.figure(figsize=(20,10))
sns.distplot(data.price, kde=False).set_title('Distribution of Price variabl
plt.show()
```



In [ ]:
```
# creating boxplots to see the outliers in the price variable

plt.figure(figsize=(6,4))
sns.boxplot(y=data['price'])
plt.show()
```



Using Interquartile Range

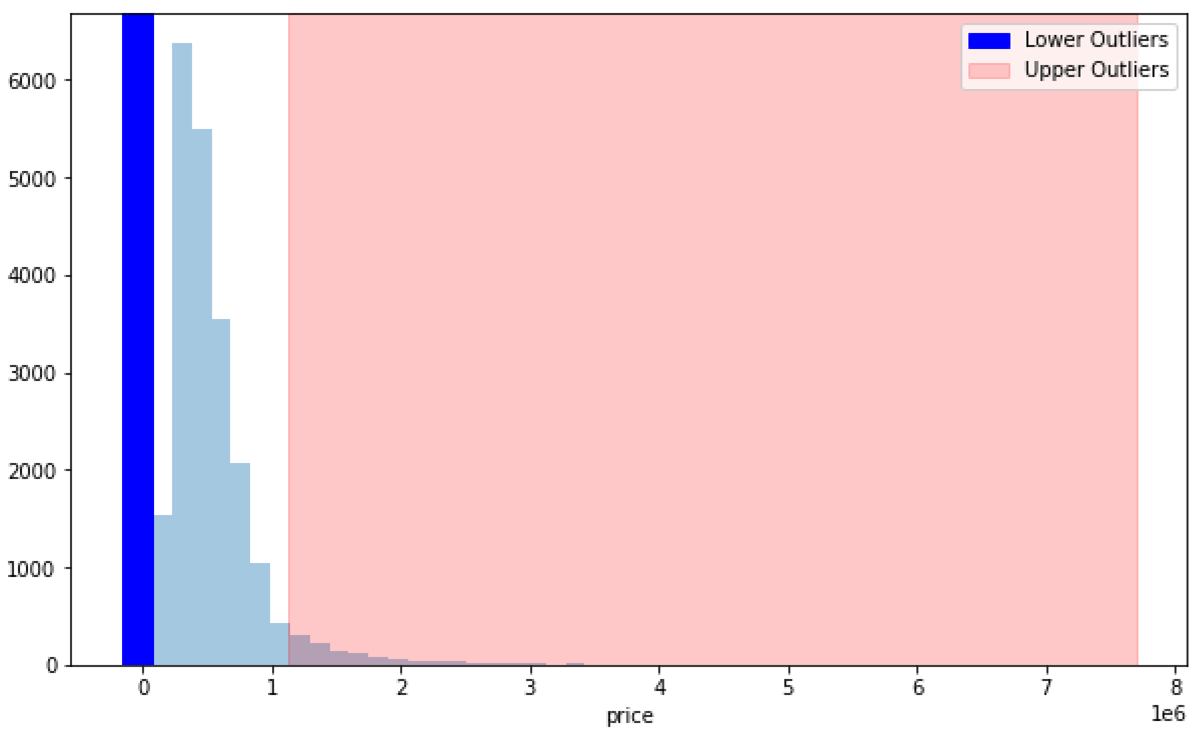
In [ ]:
```
#let us numerically draw conclusions
#creating function that can calculate interquartile range of the data
def calc_interquartile(data, column):
    global lower, upper
```

```
    #calculating the first and third quartile
    first_quartile, third_quartile = np.percentile(data[column], 25), np.per
    #calculate the interquartilerange
    iqr = third_quartile - first_quartile
    # outlier cutoff (1.5 is a generally taken as a threshold)
    cutoff = iqr*1.5
    #calculate the lower and upper limits
    lower, upper = first_quartile - cutoff , third_quartile + cutoff
    #remove the outliers from the columns
    upper_outliers = data[data[column] > upper]
    lower_outliers = data[data[column] < lower]
    print('Lower outliers', lower_outliers.shape[0])
    print('Upper outliers', upper_outliers.shape[0])
    return print('total outliers', upper_outliers.shape[0] + lower_outliers.
```

In [ ]:

```
#plotting outliers graph for 'price' feature
calc_interquartile(data, 'price')
plt.figure(figsize = (10,6))
sns.distplot(data['price'], kde=False)
print(upper, lower)
plt.axvspan(xmin = lower,xmax= data['price'].min(),alpha=1, color='blue', la
plt.axvspan(xmin = upper,xmax= data['price'].max(),alpha=0.2, color='red', l
plt.legend()
plt.show()
```

```
Lower outliers 0
Upper outliers 1146
total outliers 1146
1129575.0 -162625.0
```



Using ZScore

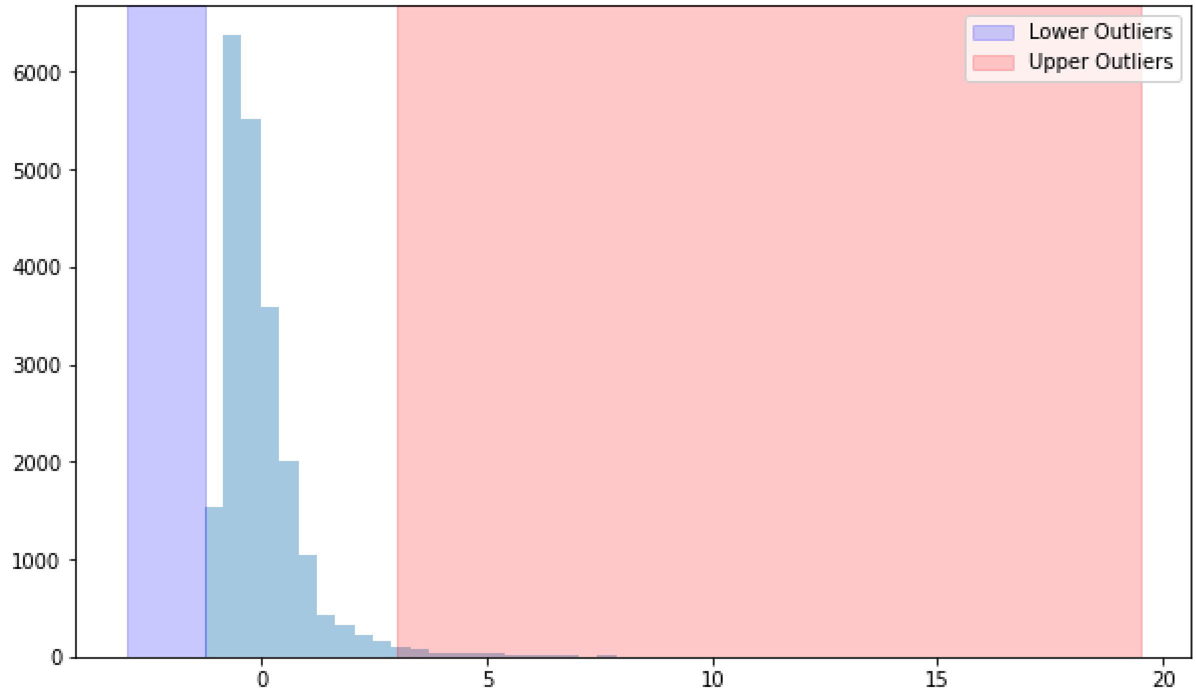
In [ ]:
```
""" creating function for calculating zscore which is subtracting the mean f
is less than -3 or greater than 3, then that data point is an outlier"""

# from scipy.stats import zscore

def z_score(data, column):
    #creating global variables for plotting the graph for better demonstrati
    global zscore, outlier
    #creating lists to store zscore and outliers
    zscore = []
    outlier =[]
    # for zscore generally taken thresholds are 2.5, 3 or 3.5 hence i took 3
    threshold = 3
    # calculating the mean of the passed column
    mean = np.mean(data[column])
    # calculating the standard deviation of the passed column
    std = np.std(data[column])
    for i in data[column]:
        z = (i-mean)/std
        zscore.append(z)
        #if the zscore is greater than threshold = 3 that means it is an out
        if np.abs(z) > threshold:
            outlier.append(i)
    return print('total outliers', len(outlier))
```

In [ ]:
```
#plotting outliers graph for 'price' feature
z_score(data, 'price')
plt.figure(figsize = (10,6))
sns.distplot(zscore, kde=False)
print(upper, lower)
plt.axvspan(xmin = -3 ,xmax= min(zscore),alpha=0.2, color='blue', label='Low
plt.axvspan(xmin = 3 ,xmax= max(zscore),alpha=0.2, color='red', label='Upper
plt.legend()
plt.show()
```

```
total outliers 406
1129575.0 -162625.0
```



In [ ]:

```
#remove the outliers from price using zscore
dj=[]
for i in data.price:
    if i in set(outlier):
        dj.append(0.0)
    else:
        dj.append(i)

data['P'] = dj

x = data.drop(data[data['P'] == 0.0].index)
x.shape
```

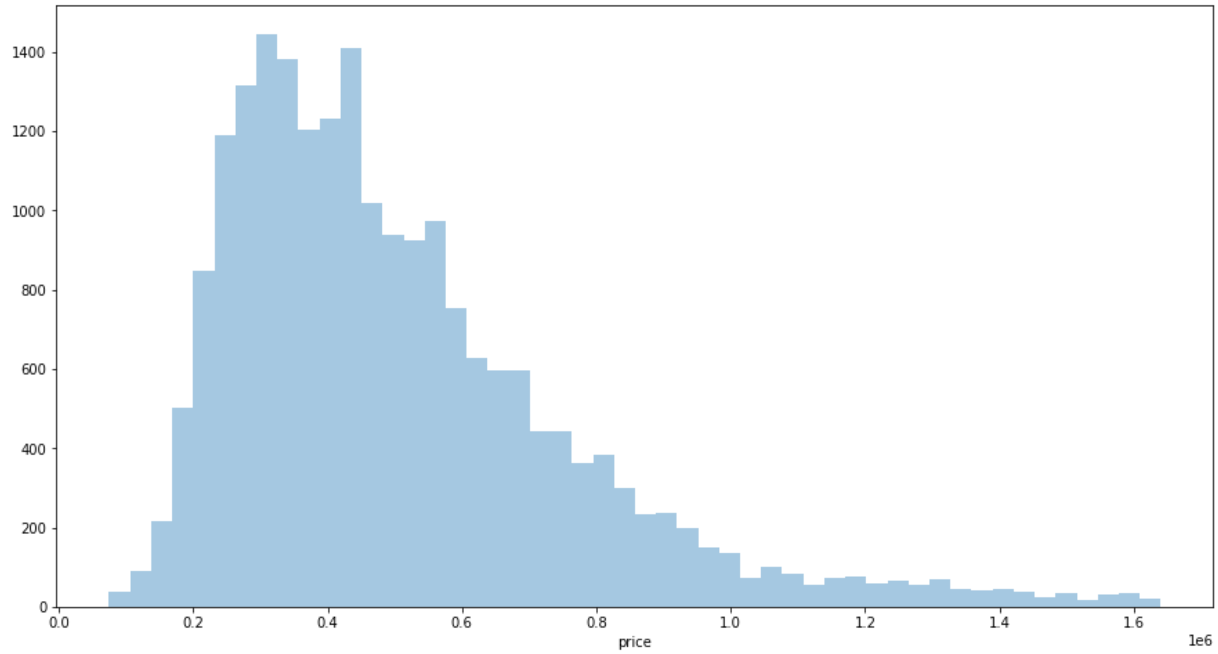
Out[ ]:

```
(21207, 18)
```

In [ ]:

```
plt.figure(figsize = (15,8))
sns.distplot(x['price'], kde=False)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: Future
Warning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level func
tion with similar flexibility) or `histplot` (an axes-level function for his
tograms).
  warnings.warn(msg, FutureWarning)
```



Comparing the data

In [ ]:
```
#isolation forest
import warnings
warnings.filterwarnings(action='ignore')
from sklearn.ensemble import IsolationForest
iso = IsolationForest()
outlier = iso.fit_predict(data)
```

In [ ]:
```
outlier
```

Out[ ]:
```
array([1, 1, 1, ..., 1, 1, 1])
```

In [ ]:
```
print(set(outlier))
```

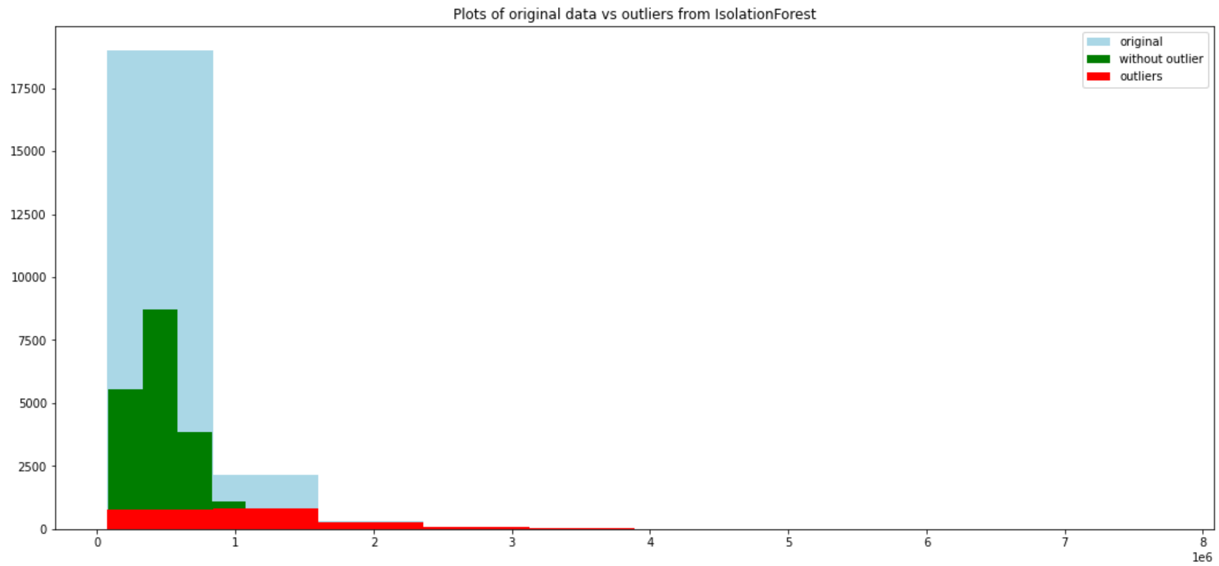
```
{1, -1}
```

-1 for outliers, 1 for non-outliers

In [ ]:
```
#mask variable contains all the outliers
mask = outlier == -1
#task variable contains all the non-outliers data
task = outlier == 1
#creating dataframe containing outliers
df_1 = data[mask]
#creating dataframe containing non-outliers
df_2 = data[task]
```

In [ ]:
```
#plotting graph to show the original data, outliers and non-outliers
plt.figure(figsize=(18, 8))
plt.title('Plots of original data vs outliers from IsolationForest')
plt.hist(data['price'], label= 'original', color='lightblue')
plt.hist(df_2['price'], label='without outlier', color='green')
plt.hist(df_1['price'], label='outliers', color='red')
```

```
plt.legend()
plt.show()
```



Model Building

In [ ]:
```
#defining the independent and dependent variable
X = x.drop(['price','P'], axis=1) #independent varaibles
Y = x['price'] #dependent variable
```

In [ ]:
```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, r
lr = LinearRegression()
lr.fit(x_train, y_train)
pred = lr.predict(x_test)
r2_score(y_test, pred)
```

Out[ ]: 0.6917485447465683

Creating Pipeline for all models

In [ ]:
```
sc = ('Scaler', StandardScaler())
est =[]
est.append(('LinearRegression', Pipeline([sc, ('LinearRegression', LinearReg
est.append(('Ridge', Pipeline([sc, ('Ridge', Ridge())])))
est.append(('Lasso', Pipeline([sc, ('Lasso', Lasso())])))
est.append(('BayesianRidge', Pipeline([sc, ('BayesianRidge', BayesianRidge()
est.append(('ElasticNet', Pipeline([sc,('Elastic', ElasticNet())])))
est.append(('SGD', Pipeline([sc,('SGD', SGDRegressor())])))
est.append(('Huber', Pipeline([sc,('Huber', HuberRegressor())])))
est.append(('RANSAC', Pipeline([sc,('RANSAC', RANSACRegressor())])))
est.append(('GradientBoosting', Pipeline([sc,('GradientBoosting',GradientBoo
est.append(('AdaBoost', Pipeline([sc, ('AdaBoost', AdaBoostRegressor())])))
est.append(('ExtraTree', Pipeline([sc,('ExtraTrees', ExtraTreesRegressor())]
est.append(('RandomForest', Pipeline([sc,('RandomForest', RandomForestRegres
est.append(('Bagging', Pipeline([sc,('Bagging', BaggingRegressor())])))
est.append(('KNeighbors', Pipeline([sc,('KNeighbors', KNeighborsRegressor())
```

```
est.append(('DecisionTree', Pipeline([sc,('DecisionTree', DecisionTreeRegres
est.append(('XGB', Pipeline([sc,('XGB', XGBRegressor())])))
```

In [ ]:

```
# using KFold cross validation
import warnings
warnings.filterwarnings(action='ignore')
seed = 4
splits = 7
score = 'r2'
models_score =[]
for i in est:
    kfold = KFold(n_splits=splits, random_state=seed, shuffle=True)
    results = cross_val_score(i[1], x_train, y_train, cv=kfold, scoring=scor
    models_score.append({i[0] : '{}'.format(results.mean())})
```

```
[04:49:47] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.
[04:49:48] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.
[04:49:48] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.
[04:49:49] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.
[04:49:50] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.
[04:49:50] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.
[04:49:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.
```

In [ ]:

```
models_score
```

Out[ ]:

```
[{'LinearRegression': '0.6051185072742342'},
 {'Ridge': '0.6051186922715839'},
 {'Lasso': '0.6051185258409818'},
 {'BayesianRidge': '0.6051194092434848'},
 {'ElasticNet': '0.5501122694481356'},
 {'SGD': '0.6044891894319965'},
 {'Huber': '0.6003343240956742'},
 {'RANSAC': '0.4873795938528364'},
 {'GradientBoosting': '0.7599320183544173'},
 {'AdaBoost': '0.3947811064237354'},
 {'ExtraTree': '0.7489626633577496'},
 {'RandomForest': '0.7755587832151665'},
 {'Bagging': '0.7535903440906441'},
 {'KNeighbors': '0.6424188946681761'},
 {'DecisionTree': '0.5601529410166883'},
 {'XGB': '0.759882787923888'}]
```

results might vary when run it at different times.

Hyperparameter Tuning

In [ ]:

```
#Tuning only XGB as it has the higher accuracy
est =[]
```

```
est.append(('XGB', Pipeline([sc,('XGB', XGBRegressor())])))

best = []

parameters = {

              'XGB': {'XGB__learning_rate': [0.1,0.2,0.3,0.4],

                         'XGB__max_depth': [4,6,8],
                      'XGB__n_estimators': [100,500,1000,1500]}
             }

for i in est:
    kfold = KFold(n_splits=5, random_state=seed, shuffle=True)
    grid = GridSearchCV(estimator=i[1], param_grid = parameters[i[0]], cv =
    grid.fit(x_train, y_train)
    best.append((i[0], grid.best_score_,  grid.best_params_))
```

In [ ]:

```
#implementing it with best parameters
xgb = XGBRegressor(learning_rate=0.1, max_depth=4, n_estimators=1000)
xgb.fit(x_train, y_train)
pred = xgb.predict(x_test)
xgb.score(x_test,y_test)
```

```
[02:19:33] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.
```

Out[ ]: 0.8733764163860631

In [ ]:

```
x_train
```

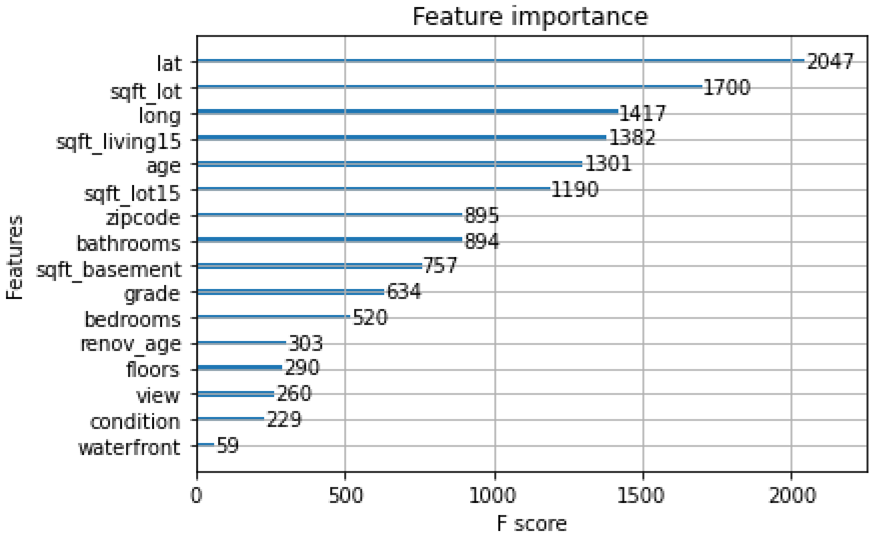
Out[ ]:

	bedrooms	bathrooms	sqft_lot	floors	waterfront	view	condition	gı
2610	3.0	2.50	30886.0	2.0	0.0	0.0	3.0	
17502	3.0	2.50	71002.0	1.0	0.0	0.0	4.0	
15475	2.0	1.50	7200.0	1.0	0.0	0.0	3.0	
10195	5.0	3.50	4800.0	2.0	0.0	2.0	3.0	
2514	4.0	2.50	4736.0	2.0	0.0	0.0	3.0	
...	...	...	...	...	...	...	...	
11471	5.0	2.25	16553.0	2.0	0.0	2.0	5.0	
12161	3.0	1.75	39639.0	1.0	0.0	0.0	4.0	
5482	4.0	2.50	6605.0	2.0	0.0	0.0	3.0	
873	4.0	2.50	10514.0	2.0	0.0	0.0	3.0	
16077	3.0	2.50	35171.0	2.0	0.0	0.0	3.0	

14844 rows × 16 columns

Feature Selection

In [ ]:
```
from xgboost import plot_importance
plot_importance(xgb)
plt.show()
```



I will use the top 9 features as my final fetaures to build my model and deploy it. I can also consider all these important features shown above. NB: **Lat** and **Long** are very important features in this regard. They are telling us that the location of the building is an important factor in determining the price or value of the property. The zip code is also telling us the same thing. In this example, I will omit the **Lat** and **Long** because if i do the deployment and I want people to check the value of their property, it is mostly difficult for them to provide the exact latitude and logitute of their property. In reality, I would have gone extra mile to collect extra data on the location of the properties or houses and add that data to my dataset which i can use to build my model instead of using latitude and longitude.

I will rather include the **zipcode** which also tells us the location of the houses.

In [ ]:
```
x
```

Out[ ]:

	price	bedrooms	bathrooms	sqft_lot	floors	waterfront	view	co
0	221900.0	3.0	1.00	5650.0	1.0	0.0	0.0	
1	538000.0	3.0	2.25	7242.0	2.0	0.0	0.0	
2	180000.0	2.0	1.00	10000.0	1.0	0.0	0.0	
3	604000.0	4.0	3.00	5000.0	1.0	0.0	0.0	
4	510000.0	3.0	2.00	8080.0	1.0	0.0	0.0	
...	...	...	...	...	...	...	...	
21608	360000.0	3.0	2.50	1131.0	3.0	0.0	0.0	
21609	400000.0	4.0	2.50	5813.0	2.0	0.0	0.0	
21610	402101.0	2.0	0.75	1350.0	2.0	0.0	0.0	
21611	400000.0	3.0	2.50	2388.0	2.0	0.0	0.0	
21612	325000.0	2.0	0.75	1076.0	2.0	0.0	0.0	

21207 rows × 18 columns

In [ ]:

```
Y #Y contains only the price
```

Out[ ]:

```
0        221900.0
1        538000.0
2        180000.0
3        604000.0
4        510000.0
           ...
21608    360000.0
21609    400000.0
21610    402101.0
21611    400000.0
21612    325000.0
Name: price, Length: 21207, dtype: float64
```

In [ ]:

```
SelectedFeatures = x[['sqft_lot','sqft_living15','age','zipcode','bathrooms'
SelectedFeatures
```

Out[ ]:

	sqft_lot	sqft_living15	age	zipcode	bathrooms	bedrooms	renov_age
0	5650.0	1340.0	59.0	98178.0	1.00	3.0	0.0
1	7242.0	1690.0	63.0	98125.0	2.25	3.0	40.0
2	10000.0	2720.0	82.0	98028.0	1.00	2.0	0.0
3	5000.0	1360.0	49.0	98136.0	3.00	4.0	0.0
4	8080.0	1800.0	28.0	98074.0	2.00	3.0	0.0
...	...	...	...	...	...	...	...
21608	1131.0	1530.0	5.0	98103.0	2.50	3.0	0.0
21609	5813.0	1830.0	1.0	98146.0	2.50	4.0	0.0
21610	1350.0	1020.0	5.0	98144.0	0.75	2.0	0.0
21611	2388.0	1410.0	11.0	98027.0	2.50	3.0	0.0
21612	1076.0	1020.0	6.0	98144.0	0.75	2.0	0.0

21207 rows × 9 columns

In [ ]:

```
x_train, x_test, y_train, y_test = train_test_split(SelectedFeatures, Y, tes

#implementing it with best parameters
xgb = XGBRegressor(learning_rate=0.1, max_depth=4, n_estimators=1000)
xgb.fit(x_train, y_train)
pred = xgb.predict(x_test)
xgb.score(x_test,y_test)
```

```
[02:19:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:line
ar is now deprecated in favor of reg:squarederror.
```

Out[ ]: 0.8346848971893831

In [ ]:

```
lr=LinearRegression()
lr.fit(x_train, y_train)
pred = lr.predict(x_test)
lr.score(x_test,y_test)
```

Out[ ]: 0.5935645159913372

In [ ]:

```
import pickle

# Creating a pickle file for the classifier
pickle.dump(xgb, open('model.pkl', 'wb'))
```

In [ ]:

```

```