XGBoost Regression & SVR

September 13, 2025

[1]:
```python
import numpy as np
import pandas as pd
```

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

[2]:
```
               id        date     price  bedrooms  bathrooms  sqft_living  \
0      7129300520  10/13/2014  221900.0         3       1.00         1180
1      6414100192   12/9/2014  538000.0         3       2.25         2570
2      5631500400   2/25/2015  180000.0         2       1.00          770
3      2487200875   12/9/2014  604000.0         4       3.00         1960
4      1954400510   2/18/2015  510000.0         3       2.00         1680
...           ...         ...       ...       ...        ...          ...
21592   263000018   5/21/2014  360000.0         3       2.50         1530
21593  6600060120   2/23/2015  400000.0         4       2.50         2310
21594  1523300141   6/23/2014  402101.0         2       0.75         1020
21595   291310100   1/16/2015  400000.0         3       2.50         1600
21596  1523300157  10/15/2014  325000.0         2       0.75         1020

       sqft_lot  floors  waterfront  view  ...  grade  sqft_above  \
0          5650     1.0           0     0  ...      7        1180
1          7242     2.0           0     0  ...      7        2170
2         10000     1.0           0     0  ...      6         770
3          5000     1.0           0     0  ...      7        1050
4          8080     1.0           0     0  ...      8        1680
...         ...     ...         ...   ...  ...    ...         ...
21592      1131     3.0           0     0  ...      8        1530
21593      5813     2.0           0     0  ...      8        2310
21594      1350     2.0           0     0  ...      7        1020
21595      2388     2.0           0     0  ...      8        1600
21596      1076     2.0           0     0  ...      7        1020

       sqft_basement  yr_built  yr_renovated  zipcode      lat     long  \
0                  0      1955             0    98178  47.5112 -122.257
1                400      1951          1991    98125  47.7210 -122.319
2                  0      1933             0    98028  47.7379 -122.233
3                910      1965             0    98136  47.5208 -122.393
4                  0      1987             0    98074  47.6168 -122.045
```

```
...              ...      ...            ...      ...      ...      ...
21592              0     2009              0    98103  47.6993 -122.346
21593              0     2014              0    98146  47.5107 -122.362
21594              0     2009              0    98144  47.5944 -122.299
21595              0     2004              0    98027  47.5345 -122.069
21596              0     2008              0    98144  47.5941 -122.299

       sqft_living15  sqft_lot15
0               1340        5650
1               1690        7639
2               2720        8062
3               1360        5000
4               1800        7503
...              ...         ...
21592           1530        1509
21593           1830        7200
21594           1020        2007
21595           1410        1287
21596           1020        1357

[21597 rows x 21 columns]
```

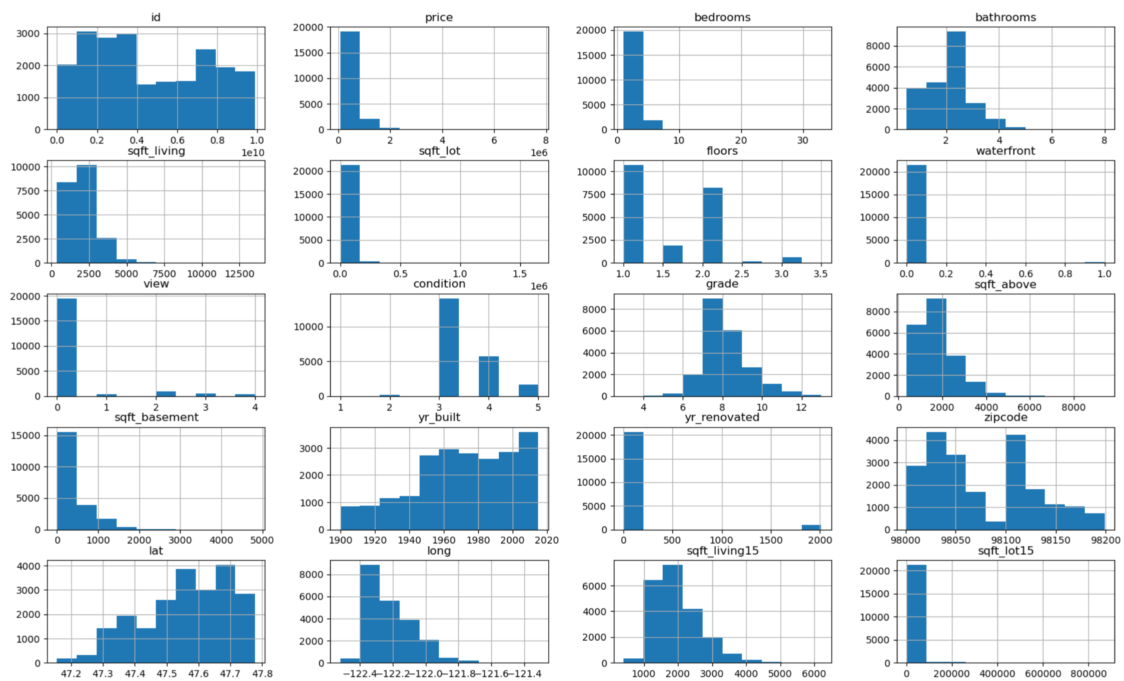
[4]:
```python
df.describe()
```

[4]:
```
                 id         price      bedrooms     bathrooms   sqft_living  \
count  2.159700e+04  2.159700e+04  21597.000000  21597.000000  21597.000000
mean   4.580474e+09  5.402966e+05      3.373200      2.115826   2080.321850
std    2.876736e+09  3.673681e+05      0.926299      0.768984    918.106125
min    1.000102e+06  7.800000e+04      1.000000      0.500000    370.000000
25%    2.123049e+09  3.220000e+05      3.000000      1.750000   1430.000000
50%    3.904930e+09  4.500000e+05      3.000000      2.250000   1910.000000
75%    7.308900e+09  6.450000e+05      4.000000      2.500000   2550.000000
max    9.900000e+09  7.700000e+06     33.000000      8.000000  13540.000000

           sqft_lot        floors    waterfront          view     condition  \
count  2.159700e+04  21597.000000  21597.000000  21597.000000  21597.000000
mean   1.509941e+04      1.494096      0.007547      0.234292      3.409825
std    4.141264e+04      0.539683      0.086549      0.766390      0.650546
min    5.200000e+02      1.000000      0.000000      0.000000      1.000000
25%    5.040000e+03      1.000000      0.000000      0.000000      3.000000
50%    7.618000e+03      1.500000      0.000000      0.000000      3.000000
75%    1.068500e+04      2.000000      0.000000      0.000000      4.000000
max    1.651359e+06      3.500000      1.000000      4.000000      5.000000

              grade    sqft_above  sqft_basement      yr_built  yr_renovated  \
count  21597.000000  21597.000000   21597.000000  21597.000000  21597.000000
mean       7.657915   1788.596842     291.725008   1970.999676     84.464787
```

std	1.173200	827.759761	442.667800	29.375234	401.821438
min	3.000000	370.000000	0.000000	1900.000000	0.000000
25%	7.000000	1190.000000	0.000000	1951.000000	0.000000
50%	7.000000	1560.000000	0.000000	1975.000000	0.000000
75%	8.000000	2210.000000	560.000000	1997.000000	0.000000
max	13.000000	9410.000000	4820.000000	2015.000000	2015.000000

	zipcode	lat	long	sqft_living15	sqft_lot15
count	21597.000000	21597.000000	21597.000000	21597.000000	21597.000000
mean	98077.951845	47.560093	-122.213982	1986.620318	12758.283512
std	53.513072	0.138552	0.140724	685.230472	27274.441950
min	98001.000000	47.155900	-122.519000	399.000000	651.000000
25%	98033.000000	47.471100	-122.328000	1490.000000	5100.000000
50%	98065.000000	47.571800	-122.231000	1840.000000	7620.000000
75%	98118.000000	47.678000	-122.125000	2360.000000	10083.000000
max	98199.000000	47.777600	-121.315000	6210.000000	871200.000000

[5]:
```
df.hist(figsize= (20,12));
```



First of all we usually should do preprocessing operation like null values or misunderstanding values. For e.g, we see the df['yr_renovated] that shows the year of renewing the building. here 0 means that this building had no renovation (It does not mean the year). There are some solutions: 1. if the df['yr_renovated] = 0, so we can put df['yr_built] instead. 2. or we can make them 0 or 1 to show that the house is renovated or not.

But we skip here.

[6]:
```
import xgboost as xgb
```

[7]:
```
x = df.iloc[:, 3 :].values
y = df['price'].values
```

[9]:
```
np.shape(x)
```

[9]:
```
(21597, 18)
```

[10]:
```
np.shape(y)
```

[10]:
```
(21597,)
```

[11]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state = 123)
```

[12]:
```
xgb_reg = xgb.XGBRegressor(colsample_bytree = 0.8, learning_rate = 0.1,␣
↪max_depth = 6, n_estimators = 1000, verbosity = 3)
```

[14]:
```
xgb_reg.fit(x_train, y_train)
p = xgb_reg.predict(x_test)
```

```
[18:35:50] ======== Monitor (0): HostSketchContainer ========
[18:35:50] AllReduce: 0.001281s, 1 calls @ 1281us

[18:35:50] MakeCuts: 0.001342s, 1 calls @ 1342us

[18:35:50] INFO: C:\actions-
runner\_work\xgboost\xgboost\src\data\iterative_dmatrix.cc:53: Finished
constructing the `IterativeDMatrix`: (16197, 18, 291546).
[18:35:50] DEBUG: C:\actions-runner\_work\xgboost\xgboost\src\gbm\gbtree.cc:131:
Using tree method: 0
[18:35:51] ======== Monitor (0): GBTree ========
[18:35:51] BoostNewTrees: 0.884339s, 1000 calls @ 884339us

[18:35:51] CommitModel: 0.000409s, 1000 calls @ 409us

[18:35:51] ======== Monitor (0): HistUpdater ========
[18:35:51] BuildHistogram: 0.266365s, 5000 calls @ 266365us

[18:35:51] EvaluateSplits: 0.248679s, 6000 calls @ 248679us

[18:35:51] InitData: 0.0159s, 1000 calls @ 15900us

[18:35:51] InitRoot: 0.11689s, 1000 calls @ 116890us

[18:35:51] LeafPartition: 0.000123s, 1000 calls @ 123us
```

```
[18:35:51] UpdatePosition: 0.160456s, 6000 calls @ 160456us

[18:35:51] UpdatePredictionCache: 0.022868s, 1000 calls @ 22868us

[18:35:51] UpdateTree: 0.855876s, 1000 calls @ 855876us

[18:35:51] DEBUG: C:\actions-runner\_work\xgboost\xgboost\src\gbm\gbtree.cc:131:
Using tree method: 0
```

[15]:
```
from sklearn.metrics import mean_absolute_error
```

What is mean_absolute_error? It is a regression metric used to evaluate how well a model is predicting continuous numerical values. It measures the average absolute difference between actual and predicted values.

Interpretation:

Lower MAE means better predictions.

It is in the same unit as the target variable (e.g., dollars, meters).

Unlike MSE, MAE doesn't square the errors, so it's more robust to outliers.

[16]:
```
from sklearn import metrics
print(metrics.mean_absolute_error(y_test, p))
```

```
63091.339743923614
```

As above, we can not use classification_report or confusion_matrix. because they are related to classification.

the above unit equals to our target unit. Target unit is dollar, so the error is 63091$.

[17]:
```
df['price'].mean()
```

[17]:
```
540296.5735055795
```

[18]:
```
df['price'].std()
```

[18]:
```
367368.1401013936
```

[19]:
```
MSE = metrics.mean_squared_error(y_test, p)
MSE
```

[19]:
```
14744813931.751701
```

[20]:
```
RMSE = np.sqrt(metrics.mean_squared_error(y_test, p))
RMSE
```

[20]:
```
121428.22543277037
```

[21]:
```
metrics.r2_score(y_test, p)
```

[21]: 0.8889036036305592

This calculates the R^2 score (coefficient of determination) between the actual values y_test and the predicted values p.

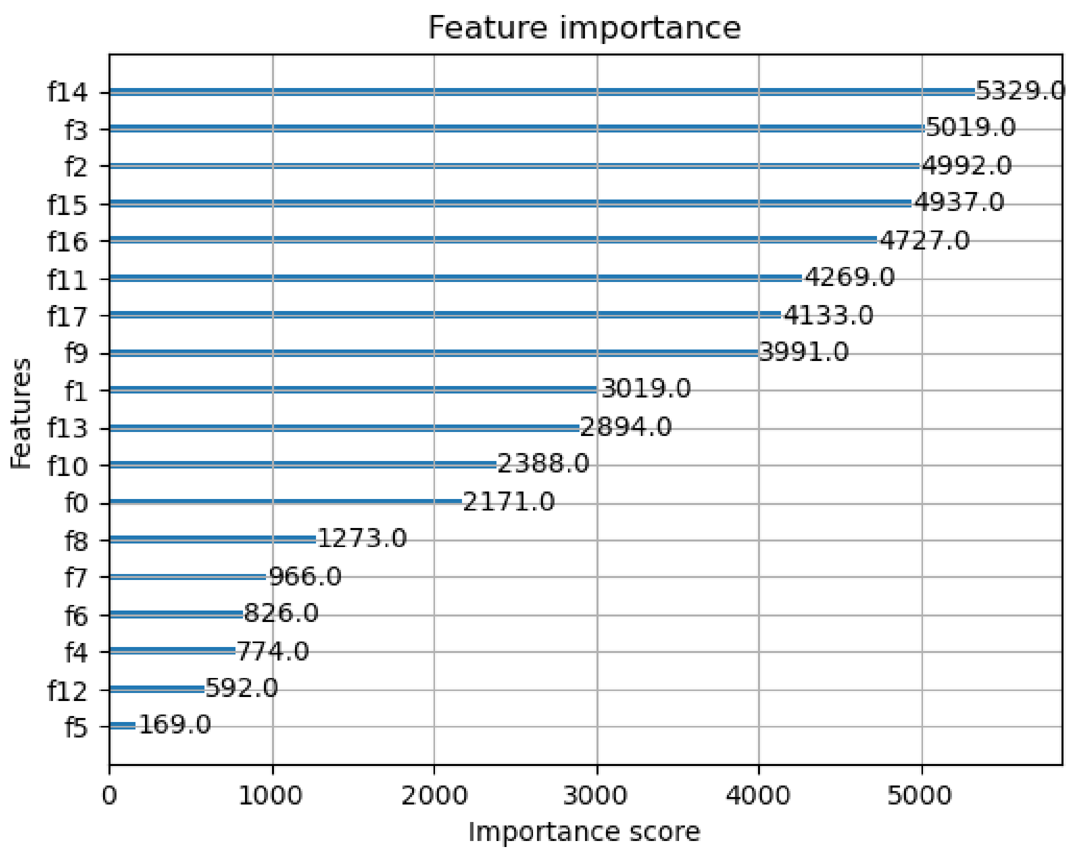
Interpretation:

$R^2 = 1 \rightarrow$ Perfect prediction

$R^2 = 0 \rightarrow$ Model does no better than predicting the mean

$R^2 < 0 \rightarrow$ Model performs worse than the mean

[22]:
```
xgb.plot_importance(xgb_reg);
```



Each f are features of tables of x. f14 means the 14th col of x. Lets see the columns.

[27]:
```
x_table = df.iloc[:, 3 :]
x_table.columns
```

[27]:
```
Index(['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'long',
       'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

[28]:
```
print(x_table.columns[14])
print(x_table.columns[3])
print(x_table.columns[2])
print(x_table.columns[15])
print(x_table.columns[16])
```

```
lat
sqft_lot
sqft_living
long
sqft_living15
```

0.0.1 Lets predict with SVR:

[29]:
```
from sklearn.svm import SVR
svr_reg = SVR()
svr_reg.fit(x_train, y_train)
p = svr_reg.predict(x_test)
```

[30]:
```
metrics.mean_absolute_error(y_test, p)
```

[30]:
```
219941.48670598533
```

SVR is Support Vector Regression.

As result XGBR predicts better than SVR. Thats better to use hyperparameters to improve the prediction.