House Price Predictor

July 16, 2024

1 House Price Predictor using Linear Regression Model

1.1 Import All the Necessary Libraries

```
[1]: import pandas as pd import numpy as np import seaborn as
sns import matplotlib.pyplot as plt from
sklearn.linear_model import LinearRegression from
sklearn.tree import DecisionTreeRegressor from
sklearn.ensemble import RandomForestRegressor from
sklearn.preprocessing import
StandardScaler,QuantileTransformer from
sklearn.model_selection import train_test_split from
sklearn.metrics import mean_squared_error, r2_score

%matplotlib inline
```

1.2 Working on Train dataframe

```
[2]: traindf = pd.read_csv('train.csv')
```

```
[3]: traindf.columns
```

```
[3]: Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea',
'Street',
           'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
           'LandSlope', 'Neighborhood', 'Condition1', 'Condition2',
           'BldgType',
           'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt',
           'YearRemodAdd',
           'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd',
           'MasVnrType',
           'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation',
           'BsmtQual',
           'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
           'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF',
           'Heating',
           'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF',
           '2ndFlrSF',
           'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath',
           'FullBath',
           'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
        'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu',
        'GarageType',
        'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea',
        'GarageQual',
        'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
        'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea',
        'PoolQC',
'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
        'SaleCondition', 'SalePrice'],
       dtype='object')
```

```
[4]: numeric_df = traindf.select_dtypes(include='number')
     correlation_matrix = numeric_df.corr()
     correlation_matrix['SalePrice'].sort_values(ascending =
     False)
```

```
[4]: SalePrice      1.000000
    OverallQual     0.790982
     GrLivArea      0.708624
    GarageCars      0.640409
    GarageArea      0.623431
    TotalBsmtSF     0.613581
     1stFlrSF       0.605852
     FullBath       0.560664
    TotRmsAbvGrd    0.533723
     YearBuilt      0.522897
    YearRemodAdd    0.507101
    GarageYrBlt     0.486362
    MasVnrArea      0.477493
    Fireplaces      0.466929
    BsmtFinSF1      0.386420
    LotFrontage     0.351799
    WoodDeckSF      0.324413
     2ndFlrSF       0.319334
    OpenPorchSF     0.315856
     HalfBath       0.284108
     LotArea        0.263843
    BsmtFullBath    0.227122
     BsmtUnfSF      0.214479
    BedroomAbvGr    0.168213
    ScreenPorch     0.111447
     PoolArea       0.092404
     MoSold         0.046432
     3SsnPorch      0.044584
    BsmtFinSF2     -
                   0.011378
    BsmtHalfBath   -
                   0.016844
```

```
 MiscVal      -
              0.021190
 Id           -
              0.021917
LowQualFinSF  -
              0.025606
 YrSold       -
              0.028923
OverallCond   -
              0.077856
MSSubClass    -
              0.084284
 EnclosedPorch-
              0.128578
KitchenAbvGr  -
              0.135907
 Name: SalePrice, dtype: float64
```

[5]:
```
req_tr =␣
↪["GarageArea","OverallQual","TotalBsmtSF","1stFlrSF","2ndFlrSF","LowQualFinSF","GrLiv
                                                                                  Area",
```

[6]:
```
selected_tr = traindf[req_tr]
```

[7]:
```
selected_tr.loc[:, 'TotalBath'] =
                                (selected_tr['BsmtFullBath'].fillna(0)
                                +
                                selected_tr['BsmtHalfBath'].fillna(0)
                                + selected_tr['FullBath'].fillna(0) +
                                selected_tr['HalfBath'].fillna(0))

selected_tr.loc[:, 'TotalSF'] = (selected_tr['TotalBsmtSF'].fillna(0)
+
                               selected_tr['1stFlrSF'].fillna(0) +
                               selected_tr['2ndFlrSF'].fillna(0) +
                               selected_tr['LowQualFinSF'].fillna(0) +
                               selected_tr['GrLivArea'].fillna(0))
```

```
C:\Users\jagan\AppData\Local\Temp\ipykernel_4956\2208519363.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a
DataFrame. Try using .loc[row_indexer,col_indexer] = value
instead

See the caveats in the documentation:
https://pandas.pydata.org/pandasdocs/stable/user_guide/indexing.html#r
eturning-a-view-versus-a-copy selected_tr.loc[:, 'TotalBath'] =
(selected_tr['BsmtFullBath'].fillna(0) +
C:\Users\jagan\AppData\Local\Temp\ipykernel_4956\2208519363.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a
DataFrame. Try using .loc[row_indexer,col_indexer] = value
instead

See the caveats in the documentation:
https://pandas.pydata.org/pandasdocs/stable/user_guide/indexing.html#r
eturning-a-view-versus-a-copy selected_tr.loc[:, 'TotalSF'] =
(selected_tr['TotalBsmtSF'].fillna(0) +
```

```
[8]: selected_tr
```

```
[8]:     GarageArea  OverallQual  TotalBsmtSF  1stFlrSF  2ndFlrSF  LowQualFinSF  \
     0          548            7          856       856       854             0

     1          460            6         1262      1262         0             0

     2          608            7          920       920       866             0

     3          642            7          756       961       756             0

     4          836            8         1145      1145      1053             0

     …           …            …            …         …         …             …

     1455       460            6          953       953       694             0

     1456       500            6         1542      2073         0             0

     1457       252            7         1152      1188      1152             0

     1458       240            5         1078      1078         0             0

     1459       276            5         1256      1256         0             0

       GrLivArea  BsmtFullBath  BsmtHalfBath  FullBath  HalfBath  TotRmsAbvGrd  \
     0      1710             1             0         2         1             8
     1      1262             0             1         2         0             6
     2      1786             1             0         2         1             6
     3      1717             1             0         1         0             7
     4      2198             1             0         2         1             9
     …         …             …             …         …         …             …
     1455   1647             0             0         2         1             7
     1456   2073             1             0         2         0             7
     1457   2340             0             0         2         0             9
     1458   1078             1             0         1         0             5
     1459   1256             1             0         1         1             6

           SalePrice  TotalBath  TotalSF
```

```
0       208500         4    4276
1       181500         3    3786
2       223500         4    4492
3       140000         2    4190
4       250000         4    5541
…          …         …       …
1455    175000         3    4247
1456    210000         3    5688
1457    266500         2    5832
1458    142125         2    3234
1459    147500         3    3768

[1460 rows x 15 columns]
```

1.3 Keeping only the necessary columns

```
[9]: train_df =␣
      ↪selected_tr[['TotRmsAbvGrd','TotalBath','GarageArea','TotalSF','OverallQual','SalePrice']]
```

```
[10]: train_df
```

```
[10]:    TotRmsAbvGrd TotalBath GarageArea TotalSF OverallQual SalePrice
      0               8         4          548     4276           7     208500
      1               6         3          460     3786           6     181500
      2               6         4          608     4492           7     223500
      3               7         2          642     4190           7     140000
      4               9         4          836     5541           8     250000
      …            …        …         …      …          …        …
      1455            7         3          460     4247           6     175000
      1456            7         3          500     5688           6     210000
      1457            9         2          252     5832           7     266500
      1458            5         2          240     3234           5     142125
      1459            6         3          276     3768           5     147500

      [1460 rows x 6 columns]
```

1.4 Splitting the dataset and Creating Pipeline

[11]:
```
 from sklearn.model_selection import train_test_split
train_set,test_set =train_test_split(train_df,test_size =
0.2,random_state = 42) print(f"Rows in train set:
{len(train_set)}\nRows in test set: ↪{len(test_set)}\n")
```

```
Rows in train set: 1168
Rows in test set:292
```

[12]:
```
housing = train_set.drop("SalePrice",axis=1)
housing_labels = train_set["SalePrice"].copy()
```

[13]:
```
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
my_pipeline = Pipeline([
    ('imputer',SimpleImputer(strategy="median")),
    ('std_scaler',StandardScaler())
])
```

[14]:
```
X_train = my_pipeline.fit_transform(housing)
```

[15]:
```
X_train
```

```
[15]: array([[-0.96456591, -0.48377079, -0.86383727, -0.13352109, -
 0.82044456], [ 0.27075534, 0.61127627, -0.45626397, -0.13428593, -
                                                      0.08893368],
         [-1.58222654, -1.57881784, -2.25716927, -1.32207838, -
         0.82044456],
         …,
         [-0.96456591, -0.48377079, 0.45366713, -1.16605156, -
         0.82044456],
         [ 0.27075534, -0.48377079, -1.23349678, -0.26966215,
         0.64257719],
         [ 0.27075534, -0.48377079, 0.87071888, 0.28025593,
         0.64257719]])
```

[16]:
```
Y_train = housing_labels
```

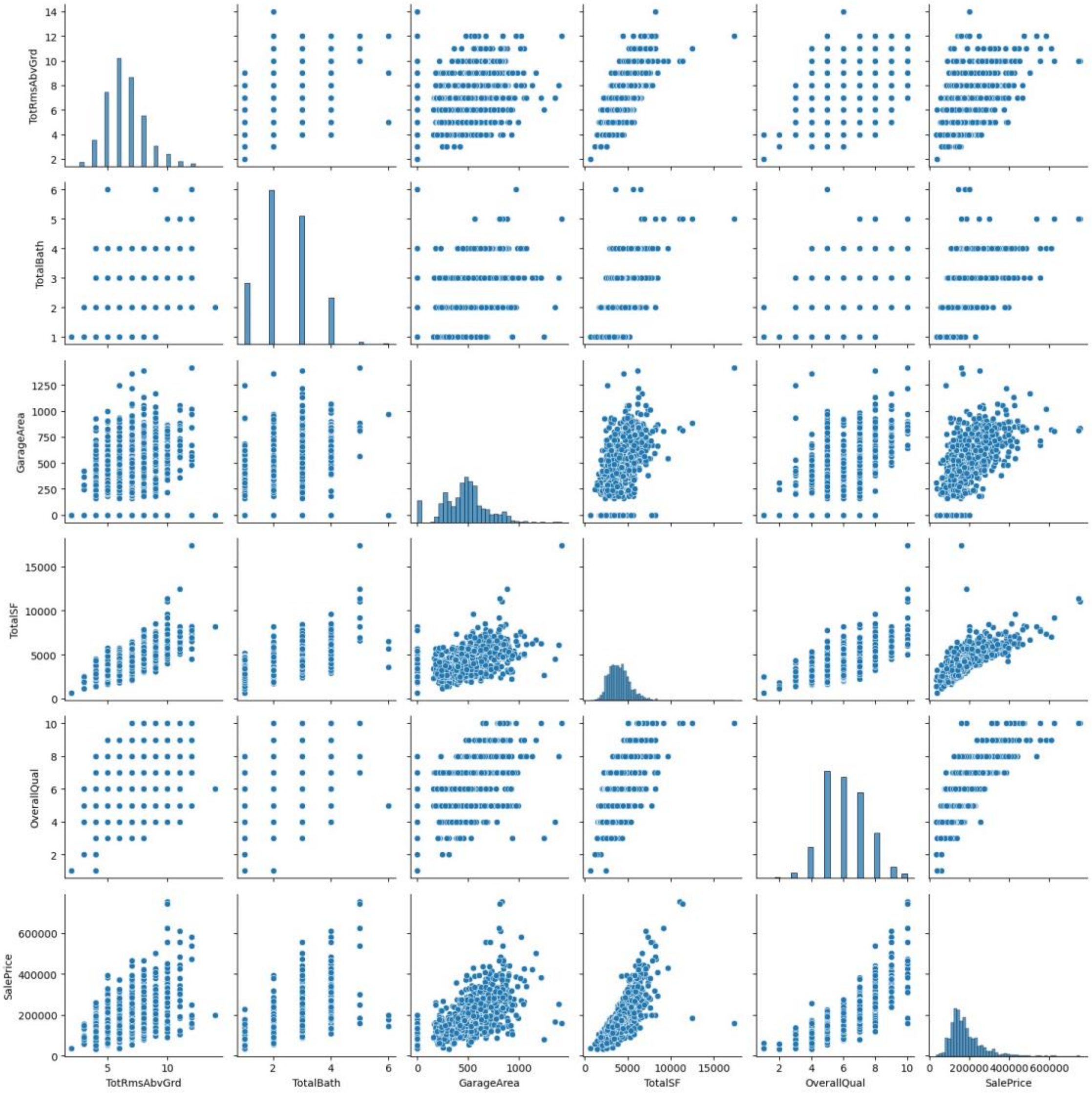
[17]:
```
Y_train.shape
```

```
[17]: (1168,)
```

1.5 Correlations

[18]:

```
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
%matplotlib inline
sns.pairplot(train_df)
plt.tight_layout()
plt.show()
```



[19]:

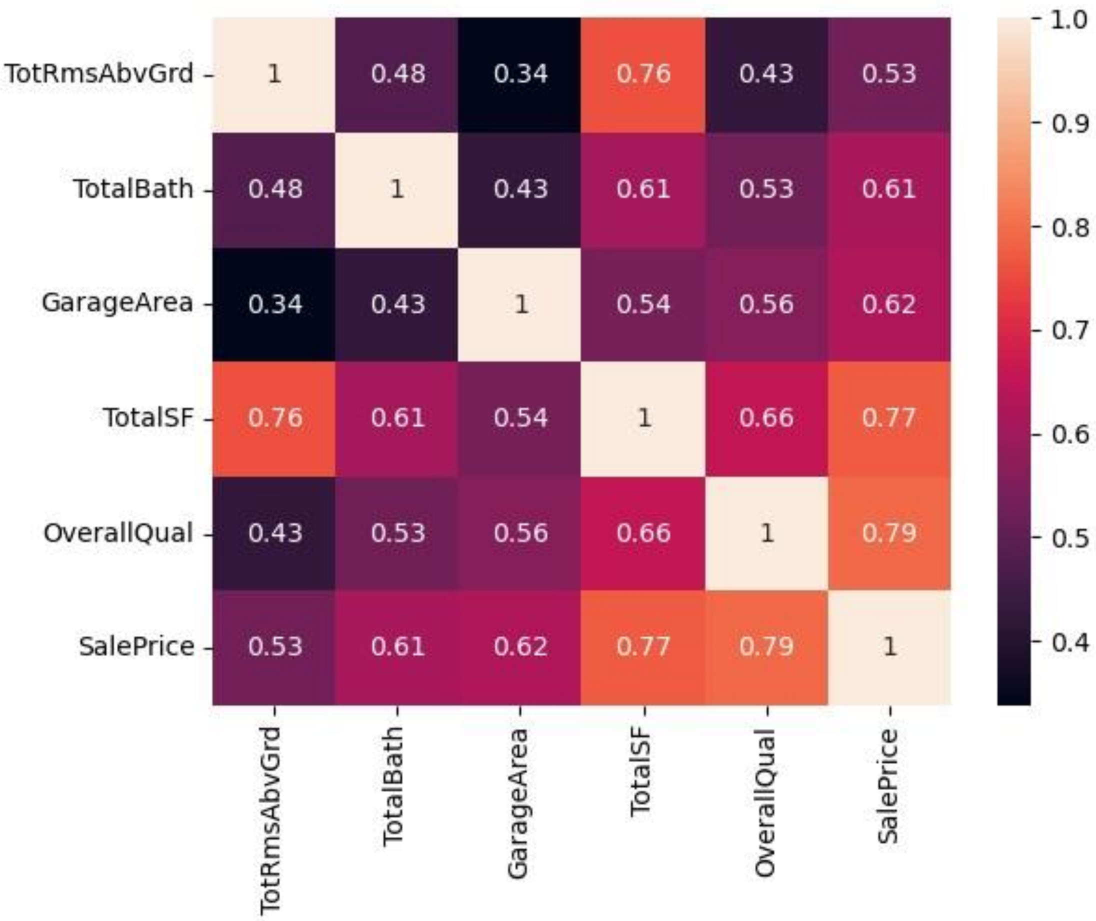
```
 corr_matrix = train_df.corr()
corr_matrix['SalePrice'].sort_values(ascending = False)
```

[19]:

```
 SalePrice     1.000000
OverallQual    0.790982
 TotalSF       0.773909
 GarageArea     0.623431
 TotalBath      0.613005
  TotRmsAbvGrd 0.533723
Name: SalePrice, dtype: float64
```

```
[20]: sns.heatmap(train_df.corr(),annot = True)
```

```
[20]: <Axes: >
```



1.6 Working with Test Dataframe

```
[21]: testdf = pd.read_csv("test.csv")
```

```
[22]: testdf.head()
```

```
[22]:   Id MSSubClass MSZoning LotFrontage LotArea Street Alley LotShape \
     0  1461      20     RH     80.0  11622 Pave  NaN   Reg
     1  1462      20     RL     81.0  14267 Pave  NaN   IR1
     2  1463      60     RL     74.0  13830 Pave  NaN   IR1
     3  1464      60     RL     78.0  9978  Pave  NaN   IR1
```

```
4  1465     120   RL    43.0  5005  Pave  NaN   IR1

  LandContour Utilities … ScreenPorch PoolArea PoolQC Fence
  MiscFeature \
0     Lvl    AllPub …    120   0     NaN MnPrv   NaN 1 Lvl   AllPub …
0     0      NaN   NaN   Gar2
2     Lvl    AllPub …    0     0     NaN MnPrv   NaN 3 Lvl   AllPub …
0     0      NaN   NaN   NaN
4         HLS    AllPub …        144        0    NaN    NaN         NaN

  MiscVal MoSold YrSold SaleType SaleCondition
0       0    6     2010  WD    Normal
1       12500      6     2010  WD    Normal
2       0    3     2010  WD    Normal
3       0    6     2010  WD    Normal
4       0    1     2010  WD    Normal

[5 rows x 80 columns]
```

```
[23]: req_tst =␣
↪["GarageArea","OverallQual","TotalBsmtSF","1stFlrSF","2ndFlrSF","LowQualFinSF","GrLiv
                                                                                 Area",
```

```
[24]: selected_tst = testdf[req_tst]
```

```
[25]: selected_tst.loc[:, 'TotalBath'] =
(selected_tst['BsmtFullBath'].fillna(0) +
                                        selected_tst['BsmtHalfBath'].fillna(0)
                                        + selected_tst['FullBath'].fillna(0) +
                                        selected_tst['HalfBath'].fillna(0))

      selected_tst.loc[:, 'TotalSF'] =
      (selected_tst['TotalBsmtSF'].fillna(0) +
                                      selected_tst['1stFlrSF'].fillna(0) +
                                      selected_tst['2ndFlrSF'].fillna(0) +
                                      selected_tst['LowQualFinSF'].fillna(0)
                                      + selected_tst['GrLivArea'].fillna(0))
```

```
C:\Users\jagan\AppData\Local\Temp\ipykernel_4956\771691818.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a
DataFrame. Try using .loc[row_indexer,col_indexer] = value
instead

See the caveats in the documentation:
https://pandas.pydata.org/pandasdocs/stable/user_guide/indexing.html#r
eturning-a-view-versus-a-copy selected_tst.loc[:, 'TotalBath'] =
    (selected_tst['BsmtFullBath'].fillna(0) +
    C:\Users\jagan\AppData\Local\Temp\ipykernel_4956\771691818.py:6:
    SettingWithCopyWarning:
    A value is trying to be set on a copy of a slice from a
    DataFrame. Try using .loc[row_indexer,col_indexer] = value
    instead

    See the caveats in the documentation:
    https://pandas.pydata.org/pandasdocs/stable/user_guide/indexing.html#r
    eturning-a-view-versus-a-copy
  selected_tst.loc[:, 'TotalSF'] = (selected_tst['TotalBsmtSF'].fillna(0) +
```

```
[26]: selected_tst
```

```
[26]:    GarageArea OverallQual TotalBsmtSF 1stFlrSF 2ndFlrSF LowQualFinSF \
0          730.0  5     882.0 896   0     0

1          312.0  6     1329.0      1329  0     0

2          482.0  5     928.0 928   701   0

3          470.0  6     926.0 926   678   0

4          506.0  8     1280.0      1280  0     0

     …          …           …           …        …        …            …

1454          0.0   4     546.0 546   546   0

1455          286.04      546.0 546   546   0

1456          576.05      1224.0      1224  0     0

1457          0.0   5     912.0 970   0     0

1458          650.07      996.0 996   1004  0

        GrLivArea BsmtFullBath BsmtHalfBath FullBath HalfBath TotRmsAbvGrd \
     0          896    0.0    0.0    1     0     5
     1          1329   0.0    0.0    1     1     6
     2          1629   0.0    0.0    2     1     6
     3          1604   0.0    0.0    2     1     7
     4          1280   0.0    0.0    2     0     5
     …          …           …           …        …        …            …
     1454       1092   0.0    0.0    1     1     5
     1455       1092   0.0    0.0    1     1     6
     1456       1224   1.0    0.0    1     0     7
     1457       970    0.0    1.0    1     0     6
     1458       2000   0.0    0.0    2     1     9

            TotalBath TotalSF
```

```
0     1.0   2674.0 1
2.0   3987.0 2   3.0
4186.0 3   3.0
4134.0
4          2.0   3840.0
…          …      …
1454        2.0   2730.0
1455        2.0   2730.0
1456        2.0   3672.0
1457        2.0   2852.0
1458        3.0   4996.0
[1459 rows x 14 columns ]
```

```
[27]: test_df_unproc =␣
       ↪selected_tst[['TotRmsAbvGrd','TotalBath','GarageArea','TotalSF','OverallQual']]
```

```
[28]: test_df_unproc
```

```
[28]:    TotRmsAbvGrd TotalBath GarageArea TotalSF OverallQual
0                5 1.0   730.0 2674.0     5

1                6 2.0   312.0 3987.0     6

2                6 3.0   482.0 4186.0     5

3                7 3.0   470.0 4134.0     6

4                5 2.0   506.0 3840.0     8

      …             …       …          …      …          …

1454             5 2.0   0.0   2730.0     4

1455             6 2.0   286.0 2730.0     4

1456             7 2.0   576.0 3672.0     5

1457             6 2.0   0.0   2852.0     5

1458             9 3.0   650.0 4996.0     7

      [1459 rows x 5 columns ]
```

```
[29]: test_df = test_df_unproc.fillna(test_df_unproc.mean())
```

```
[30]: x_test = my_pipeline.
       ↪transform(test_df[['TotRmsAbvGrd','TotalBath','GarageArea','TotalSF','OverallQual']].
       ↪values)
```

```
[31]: x_test
```

```
 [31]: array([[-0.96456591, -1.57881784, 1.2024646 , -1.10333489, -
0.82044456], [-0.34690528, -0.48377079, -0.77853123, -0.09910341, -
                                                       0.08893368],
         [-0.34690528, 0.61127627, 0.02713693, 0.05309923, -0.82044456],
         …,
         [ 0.27075534, -0.48377079, 0.47262403, -0.34002719, -
         0.82044456],
         [-0.34690528, -0.48377079, -2.25716927, -0.96719384, -
         0.82044456],
         [ 1.50607659, 0.61127627, 0.82332664, 0.67261751, 0.64257719]])
```

1.7 Model Selection

```
[32]: #model = LinearRegression()
      #model = DecisionTreeRegressor()
      model = RandomForestRegressor()
      model.fit(X_train,Y_train)
```

```
[32]: RandomForestRegressor()
```

```
[33]: y_train_pred = model.predict(X_train)
```

```
[34]: y_train_pred[:5]
```

```
[34]: array([145283. , 172053.95, 89698. , 168814. , 141625. ])
```

```
[35]: some_data = housing.iloc[:5]
      some_labels = housing_labels.iloc[:5]
```

```
[36]: proc_data = my_pipeline.transform(some_data)
```

```
[37]: model.predict(proc_data)
```

```
[37]: array([145283. , 172053.95, 89698. , 168814. , 141625. ])
```

```
[38]: list(some_labels)
```

```
[38]: [145000, 178000, 85000, 175000, 127000]
```

```
[39]: train_mse = mean_squared_error(Y_train,y_train_pred)
```

```
[40]: train_rmse = np.sqrt(train_mse)
```

```
[41]: print(f"Training MSE: {train_mse:.2f}, Training RMSE: {train_rmse:.2f}")
```

```
Training MSE: 182754306.10, Training RMSE: 13518.67
```

1.8 Cross - Validation

```
[42]: from sklearn.model_selection import cross_val_score
      scores =␣
        ↪cross_val_score(model,X_train,Y_train,scoring="neg_mean_squared_error",cv =␣
        ↪200)
      rmse_scores = np.sqrt(-scores)
```

```
[43]: rmse_scores
```

```
   [43]: array([ 23277.76446447, 13031.67234441, 24491.16208088,
  12916.01052409, 46110.0745834 , 12685.13165022, 18886.3441575 ,
                                                   11071.47547444,
         11616.79173023, 52066.94494877, 31510.80666534,
         29639.23399926,
        12151.07107851,9319.99965266, 17079.07143005, 20884.17467223,
         18824.49033057, 33193.65937336, 36854.67178624,
         23093.95361294,
         23292.9684995 , 19432.72111562, 18547.99982299,
         27613.27744408,
         20402.00284752, 17822.98485906, 42015.72882012,
         36937.17830823,
 175165.06203852, 49690.4027053 , 23330.91946822, 31513.45824178,
         22887.73799821, 31010.13490391, 47989.62053758,
         13329.38499857,
         21928.82341947, 28502.91443722, 17245.77254429,
         31486.07563826,
         22230.82920158, 32465.60889805, 27843.04913432,
         35600.14766459, 33868.94917467, 31457.61022018, 27956.1485217
         , 40078.26812993,
         20590.99533685, 21056.55475528, 19898.88278442,
         50438.13615389,
         39470.50551592, 35012.49792537, 22559.83629881,
         25338.56615794,
         10921.06969935, 27397.02810506, 25766.11480393,
         31086.04232992,
 201242.98515478, 12028.23570696, 26659.38149476, 39101.03975416,
         22930.10835334, 25187.16704816, 41651.52119048,
         29282.34773083,
          8224.18203629, 14246.79672003, 56886.03402084, 27417.3801067
         , 55466.02380919, 18062.31089481, 46371.11498073,
         45814.2412921 ,
         37923.06090531, 29540.32205912, 40406.01041065,
         33253.30878663,
         28357.63588478, 31339.75470533, 106127.87363791,
         10572.52219235,
    44907.81340181, 31278.70874266, 17627.15366185,
    23986.51913983,
    34952.10095183, 82162.55603195, 20900.97615594, 23172.9641621
    ,
    26814.03423907, 24597.32265251, 20439.71940229,
    22240.29752072,
    23770.5040223 , 28314.47719879, 69329.96980133,
    13772.33040116,
    32481.10212308, 29130.99328727, 44167.46596499,
    53298.49040065,
    18883.32266379, 17803.70762922, 35231.98484315,
    20413.09211208,
    18867.41154384, 14814.45213528, 19582.28398618,
    11716.30128734,
    10636.0999438 , 22564.76237833, 31336.81397651,
    31644.56494109,
    75496.23776138, 30016.20028031, 20454.74205292,
    30066.09581106,
    30854.08791016, 25820.47922475, 13727.55889958,
    19292.76478605,
    30038.27850124, 20108.84141508, 27973.08744932,
    45101.61936736,
  38876.15567145, 16128.61505828, 33935.44165938,8855.15987218,
    25776.08507509, 25150.93116638, 33212.31806431,
    14352.15252709,
    44717.67500818, 32649.13731135, 28537.97918331,
    12465.11203373,
    26243.67447064, 25454.07721303, 27022.5697848 ,
    10317.17889282,
    21891.22473637, 29261.49148125, 33452.4737976 , 13743.8454676
    ,
    11277.83822064, 16237.91736661, 24039.63841877, 26656.8558185
    ,
    15413.36324152, 42944.44827572, 31109.2988122 ,
    15811.60360654,
    25686.08533944, 25779.34969772, 20482.6668366 ,
    25274.52807206,
    53481.51708005, 27726.34633021, 27098.11067491,
    18400.37491698,
    51419.98502293, 20869.90451782, 12858.92838191,
    61556.68668169,
    21786.96406963, 31011.97109104, 33354.55797674,
    22402.05803861,
    14163.05024854, 14270.74072466, 19769.06816004, 17101.5888594
    ,
 24282.57518006, 17461.77872288, 38024.22855112,
 20424.51415501,
36820.8916111 ,9820.05229768, 15945.06899661,6109.83019202,
 19796.08373925, 29891.36122746, 26424.62050824,
 11914.26957176,
 27561.72469719, 42619.58945645, 27378.43479366,
 21589.13644804,
 17407.48298693, 13255.09382259, 16332.29125757,
 13944.69038795,
 62926.75609394, 19634.57473268, 21195.25876501,
 32191.88871351])
```

[44]:

```
def print_scores(scores):
    print("Scores:",scores)
    print("Mean:",scores.mean())
    print("Standard Deviation",scores.std())
```

[45]:

```
print_scores(rmse_scores)
```

```
Scores: [ 23277.76446447 13031.67234441 24491.16208088 12916.01052409
46110.0745834 12685.13165022 18886.3441575  11071.47547444
  11616.79173023 52066.94494877 31510.80666534 29639.23399926
  12151.07107851 9319.99965266 17079.07143005 20884.17467223
  18824.49033057 33193.65937336 36854.67178624 23093.95361294
  23292.9684995 19432.72111562 18547.99982299 27613.27744408
  20402.00284752 17822.98485906 42015.72882012 36937.17830823
 175165.06203852 49690.4027053 23330.91946822 31513.45824178
  22887.73799821 31010.13490391 47989.62053758 13329.38499857
  21928.82341947 28502.91443722 17245.77254429 31486.07563826
  22230.82920158 32465.60889805 27843.04913432 35600.14766459
33868.94917467 31457.61022018 27956.148521740078.26812993
  20590.99533685 21056.55475528 19898.88278442 50438.13615389
  39470.50551592 35012.49792537 22559.83629881 25338.56615794
  10921.06969935 27397.02810506 25766.11480393 31086.04232992
 201242.98515478 12028.23570696 26659.38149476 39101.03975416
  22930.10835334      25187.16704816      41651.52119048
  29282.34773083       8224.18203629      14246.79672003
  56886.03402084       27417.3801067      55466.02380919
  18062.31089481 46371.11498073 45814.2412921
  37923.06090531 29540.32205912 40406.01041065 33253.30878663
  28357.63588478 31339.75470533 106127.87363791 10572.52219235
  44907.81340181 31278.70874266 17627.15366185 23986.51913983
  34952.10095183 82162.55603195 20900.97615594 23172.9641621
  26814.03423907 24597.32265251 20439.71940229 22240.29752072
  23770.5040223 28314.47719879 69329.96980133 13772.33040116
  32481.10212308 29130.99328727 44167.46596499 53298.49040065
  18883.32266379 17803.70762922 35231.98484315 20413.09211208
  18867.41154384 14814.45213528 19582.28398618 11716.30128734
  10636.0999438 22564.76237833 31336.81397651 31644.56494109
  75496.23776138 30016.20028031 20454.74205292 30066.09581106
  30854.08791016 25820.47922475 13727.55889958 19292.76478605
  30038.27850124 20108.84141508 27973.08744932 45101.61936736
38876.15567145 16128.61505828 33935.441659388855.15987218
  25776.08507509 25150.93116638 33212.31806431 14352.15252709
  44717.67500818 32649.13731135 28537.97918331 12465.11203373
26243.67447064 25454.07721303 27022.569784810317.17889282
21891.22473637 29261.49148125 33452.473797613743.8454676
  11277.83822064 16237.91736661 24039.63841877 26656.8558185
15413.36324152 42944.44827572 31109.298812215811.60360654
25686.08533944 25779.34969772 20482.666836625274.52807206
  53481.51708005 27726.34633021 27098.11067491 18400.37491698
  51419.98502293 20869.90451782 12858.92838191 61556.68668169
  21786.96406963 31011.97109104 33354.55797674 22402.05803861
  14163.05024854 14270.74072466 19769.06816004 17101.5888594
  24282.57518006 17461.77872288 38024.22855112 20424.51415501
 36820.8916111  9820.05229768 15945.06899661 6109.83019202
  19796.08373925 29891.36122746 26424.62050824 11914.26957176
  27561.72469719 42619.58945645 27378.43479366 21589.13644804
  17407.48298693 13255.09382259 16332.29125757 13944.69038795
  62926.75609394 19634.57473268 21195.25876501 32191.88871351]
Mean: 29103.74552461538
Standard Deviation 21139.34340094215
```

[46]:

```
y_pred=model.predict(x_test)
```

[47]:

```
y_pred
```

[47]:

```
array([129596.33, 155331.14, 148147.5 , …, 138024.85, 107234.5 ,
       231268. ])
```

[48]:

```
 pred=pd.DataFrame(y_pred)
sub_df=pd.read_csv('sample_submission.csv')
datasets=pd.concat([sub_df['Id'],pred],axis=1)
datasets.columns=['Id','SalePrice']
datasets.to_csv('sample_submission.csv',index=False)
```