```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
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
```

```
traindf = pd.read_csv('train.csv')
```

```
traindf.columns
```

```
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', 'TotalBath', 'TotalSF'],
      dtype='object')
```

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

```
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
```

```
req_tr = ["GarageArea","OverallQual","TotalBsmtSF","1stFlrSF","2ndFlrSF","LowQualFinSF","GrLivArea","BsmtFullBath","BsmtHalfBath","FullBath"
```

```
selected_tr = traindf[req_tr]
```

```
selected_tr
```

	GarageArea	OverallQual	TotalBsmtSF	1stFlrSF	2ndFlrSF	LowQualFinSF	GrLivArea
0	548	7	856	856	854	0	1710
1	460	6	1262	1262	0	0	1262
2	608	7	920	920	866	0	1786
3	642	7	756	961	756	0	1717
4	836	8	1145	1145	1053	0	2198
...	...	...	...	...	...	...	...
1455	460	6	953	953	694	0	1647
1456	500	6	1542	2073	0	0	2073
1457	252	7	1152	1188	1152	0	2340
1458	240	5	1078	1078	0	0	1078
1459	276	5	1256	1256	0	0	1256

1460 rows × 15 columns

```
train_df = selected_tr[['TotRmsAbvGrd','TotalBath','GarageArea','TotalSF','OverallQual','SalePrice']]
```

```
train_df
```

	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 × 6 columns

```
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
```

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

```
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())
])
```

```
X_train = my_pipeline.fit_transform(housing)
X_train
```

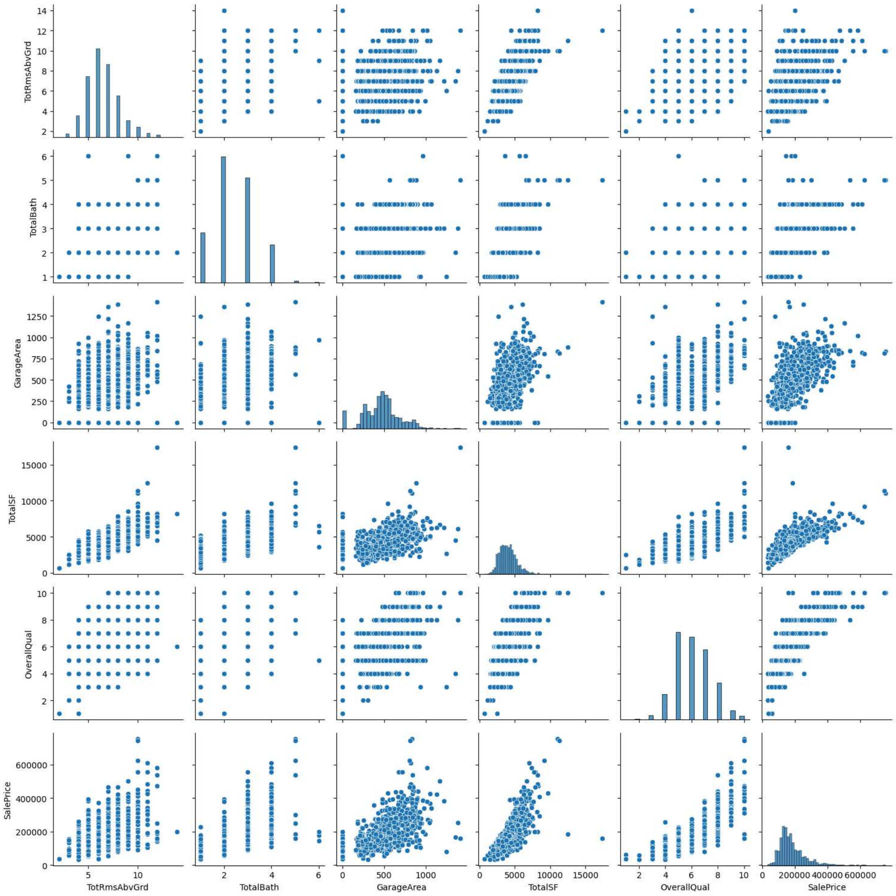
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-5243e64da559> in <cell line: 1>()
----> 1 X_train = my_pipeline.fit_transform(housing)
      2 X_train

NameError: name 'my_pipeline' is not defined
```

```
Y_train = housing_labels
Y_train.shape
```

```
(1168,)
```

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

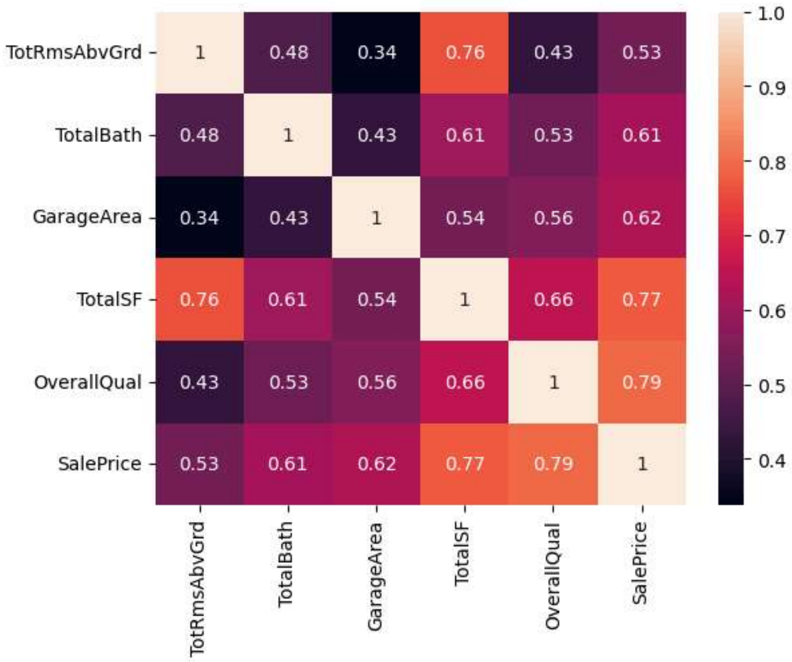

TotRmsAbvGrd
TotalBath
GarageArea
TotalSF
OverallQual
SalePrice

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

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

```
sns.heatmap(train_df.corr(),annot = True)
```

```
<Axes: >
```




```
testdf = pd.read_csv("test.csv")
testdf.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContou
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	Lv
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	Lv
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	Lv
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	Lv
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS

5 rows × 80 columns

```
req_tst = ["GarageArea","OverallQual","TotalBsmtSF","1stFlrSF","2ndFlrSF","LowQualFinSF","GrLivArea","BsmtFullBath","BsmtHalfBath","FullBath
```

```
selected_tst = testdf[req_tst]
```

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

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

```
<ipython-input-29-3654aa847672>: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/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
  selected_tst.loc[:, 'TotalBath'] = (selected_tst['BsmtFullBath'].fillna(0) + selected_tst['BsmtHalfBath'].fillna(0) + selected_tst['Fu
<ipython-input-29-3654aa847672>:3: 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/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
    selected_tst.loc[:, 'TotalSF'] = (selected_tst['TotalBsmtSF'].fillna(0) + selected_tst['1stFlrSF'].fillna(0) + selected_tst['2ndFlrSF'
```

```
selected_tst
```

	GarageArea	OverallQual	TotalBsmtSF	1stFlrSF	2ndFlrSF	LowQualFinSF	GrLivArea
0	730.0	5	882.0	896	0	0	896
1	312.0	6	1329.0	1329	0	0	1329
2	482.0	5	928.0	928	701	0	1629
3	470.0	6	926.0	926	678	0	1604
4	506.0	8	1280.0	1280	0	0	1280
...	...	...	...	...	...	...	...
1454	0.0	4	546.0	546	546	0	1092
1455	286.0	4	546.0	546	546	0	1092
1456	576.0	5	1224.0	1224	0	0	1224
1457	0.0	5	912.0	970	0	0	970
1458	650.0	7	996.0	996	1004	0	2000

1459 rows × 14 columns

```
test_df_unproc = selected_tst[['TotRmsAbvGrd','TotalBath','GarageArea','TotalSF','OverallQual']]
test_df_unproc
```

	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 × 5 columns

```
test_df = test_df_unproc.fillna(test_df_unproc.mean())
```

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

```
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]])
```

```
#model = LinearRegression()
#model = DecisionTreeRegressor()
model = RandomForestRegressor()
model.fit(X_train,Y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
y_train_pred = model.predict(X_train)
```

```
y_train_pred[:5]
```

```
array([148521.18, 172056.9 ,  90154.  , 166354.87, 136974.  ])
```

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

```
proc_data = my_pipeline.transform(some_data)
```

```
model.predict(proc_data)
```

```
array([148521.18, 172056.9 ,  90154.  , 166354.87, 136974.  ])
```

```
list(some_labels)
```

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

```
train_mse = mean_squared_error(Y_train,y_train_pred)
```

```
train_rmse = np.sqrt(train_mse)
print(f"Training MSE: {train_mse:.2f}, Training RMSE: {train_rmse:.2f}")
```

```
Training MSE: 163787325.95, Training RMSE: 12797.94
```

```
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)
```

```
rmse_scores
```

```
array([ 20134.98781949,  14686.87903424,  25562.67014137,  11905.34310038,
        45553.74485603,  12497.36432866,  20265.04862479,  12889.92955452,
        11261.55705183,  52747.54088485,  34609.31832455,  29043.5516443 ,
        14251.0653049 ,   9526.57305502,  20334.1112828 ,  21620.68600092,
        19603.56284842,  32427.16875131,  36371.87422057,  22903.17476099,
        29016.6143909 ,  17732.6279336 ,  17531.1583017 ,  27387.69596271,
        19728.8207265 ,  19055.45171987,  42351.70427429,  38289.73939472,
       169442.94448404,  51948.6841921 ,  20039.87728897,  31810.0452124 ,
        19298.84508228,  31237.14614987,  48310.76375112,  12092.20975801,
        24145.4982442 ,  29326.85866748,  19506.94453812,  31756.81340151,
        25479.30428831,  32190.70892996,  29135.34518748,  33369.33615978,
        33584.70848518,  31432.59335172,  26504.67187058,  43618.47998814,
        20860.76813753,  20657.5541872 ,  21071.80558272,  53215.5982509 ,
        38228.71510965,  36641.28538325,  22073.26195159,  29934.96832478,
         9650.5560506 ,  25607.4089106 ,  22936.90169392,  30883.30469553,
       196725.30642846,  13491.25136136,  24715.70765939,  35421.61088696,
        23144.38847077,  24795.38663266,  41835.06611542,  30494.15015245,
         6555.91515732,  15579.91845922,  56876.132711  ,  25937.21951802,
        57422.75017867,  19575.1187025 ,  46137.92518233,  45489.93083409,
        37558.36328828,  28587.41603817,  38103.47081189,  32716.94616841,
        26202.3327639 ,  32227.48042114, 107726.2272588 ,  10285.87273477,
        48266.92082725,  31804.01462906,  18613.94256777,  23131.08842726,
        36113.34216564,  78378.17033651,  18775.59369568,  21453.69985058,
        25668.55756481,  24129.84654464,  21071.21765313,  21733.66626709,
        24587.97829035,  26258.57635744,  68989.42232601,  12352.74900534,
        32906.76054716,  27480.11429687,  45547.59333722,  54560.41029764,
        18115.99299691,  19096.94691454,  36792.92575559,  18896.67704669,
        19276.91598619,  14519.23555926,  19970.06860648,  12423.77754214,
        12094.27428157,  18969.35804675,  29244.15652398,  33901.53161987,
        75631.53392333,  28047.73904368,  18379.3750864 ,  26041.29291982,
        28324.9575722 ,  29756.48018735,  14313.75217015,  18665.93479935,
        31275.61234144,  21641.25100101,  28743.87731493,  46061.1868786 ,
        37400.46496854,  15846.09278188,  36268.70180244,   7769.55188641,
        25458.80855174,  26958.30089985,  28917.79069789,  11310.055774  ,
        47552.44402414,  33432.36993426,  30679.33718408,  12573.17246057,
```

```
            24470.12191997,  24497.22777048,  26025.19328146,  14213.07304743,
            20817.14421494,  29400.88794654,  32918.50467223,  12146.96259758,
            12853.61488526,  14973.26035789,  23699.98583852,  25934.69156306,
            13975.19548295,  42630.04201458,  29441.60406063,  13362.67776993,
            26307.66408174,  25364.09111996,  20196.22774016,  25136.92955551,
            57581.48168316,  27483.57960713,  26304.55335392,  16805.65646023,
            48771.61740836,  18986.33355552,  12223.46789884,  61086.90902796,
            21223.61539812,  30941.98171267,  31486.9433332 ,  22021.52735504,
            15835.77256873,  12102.93082367,  20000.9617597 ,  15518.48566011,
            22174.64973984,  15999.94009796,  35208.88688937,  21299.11173005,
            36057.75718532,  10168.83999179,  16645.47265271,   7039.58608775,
            19899.05658098,  28872.26749384,  26077.70350845,  12630.91552385,
            26073.00929526,  41985.17469325,  29530.10815622,  19655.47974547,
            17120.754699  ,  12897.45423417,  17405.99170086,  17088.11487907,
            59854.26733955,  20239.14386941,  25271.46448594,  29172.88345019])
```

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

```
print_scores(rmse_scores)
```

```
Scores: [ 20134.98781949  14686.87903424  25562.67014137  11905.34310038
   45553.74485603  12497.36432866  20265.04862479  12889.92955452
   11261.55705183  52747.54088485  34609.31832455  29043.5516443
   14251.0653049    9526.57305502  20334.1112828   21620.68600092
   19603.56284842  32427.16875131  36371.87422057  22903.17476099
   29016.6143909   17732.6279336   17531.1583017   27387.69596271
   19728.8207265   19055.45171987  42351.70427429  38289.73939472
  169442.94448404  51948.6841921   20039.87728897  31810.0452124
   19298.84508228  31237.14614987  48310.76375112  12092.20975801
   24145.4982442   29326.85866748  19506.94453812  31756.81340151
   25479.30428831  32190.70892996  29135.34518748  33369.33615978
   33584.70848518  31432.59335172  26504.67187058  43618.47998814
   20860.76813753  20657.5541872   21071.80558272  53215.5982509
   38228.71510965  36641.28538325  22073.26195159  29934.96832478
    9650.5560506   25607.4089106   22936.90169392  30883.30469553
  196725.30642846  13491.25136136  24715.70765939  35421.61088696
   23144.38847077  24795.38663266  41835.06611542  30494.15015245
    6555.91515732  15579.91845922  56876.132711    25937.21951802
   57422.75017867  19575.1187025   46137.92518233  45489.93083409
   37558.36328828  28587.41603817  38103.47081189  32716.94616841
   26202.3327639   32227.48042114 107726.2272588   10285.87273477
   48266.92082725  31804.01462906  18613.94256777  23131.08842726
   36113.34216564  78378.17033651  18775.59369568  21453.69985058
   25668.55756481  24129.84654464  21071.21765313  21733.66626709
   24587.97829035  26258.57635744  68989.42232601  12352.74900534
   32906.76054716  27480.11429687  45547.59333722  54560.41029764
   18115.99299691  19096.94691454  36792.92575559  18896.67704669
   19276.91598619  14519.23555926  19970.06860648  12423.77754214
   12094.27428157  18969.35804675  29244.15652398  33901.53161987
   75631.53392333  28047.73904368  18379.3750864   26041.29291982
   28324.9575722   29756.48018735  14313.75217015  18665.93479935
   31275.61234144  21641.25100101  28743.87731493  46061.1868786
   37400.46496854  15846.09278188  36268.70180244   7769.55188641
   25458.80855174  26958.30089985  28917.79069789  11310.055774
   47552.44402414  33432.36993426  30679.33718408  12573.17246057
   24470.12191997  24497.22777048  26025.19328146  14213.07304743
   20817.14421494  29400.88794654  32918.50467223  12146.96259758
   12853.61488526  14973.26035789  23699.98583852  25934.69156306
   13975.19548295  42630.04201458  29441.60406063  13362.67776993
   26307.66408174  25364.09111996  20196.22774016  25136.92955551
   57581.48168316  27483.57960713  26304.55335392  16805.65646023
   48771.61740836  18986.33355552  12223.46789884  61086.90902796
   21223.61539812  30941.98171267  31486.9433332   22021.52735504
   15835.77256873  12102.93082367  20000.9617597   15518.48566011
   22174.64973984  15999.94009796  35208.88688937  21299.11173005
   36057.75718532  10168.83999179  16645.47265271   7039.58608775
   19899.05658098  28872.26749384  26077.70350845  12630.91552385
   26073.00929526  41985.17469325  29530.10815622  19655.47974547
   17120.754699    12897.45423417  17405.99170086  17088.11487907
   59854.26733955  20239.14386941  25271.46448594  29172.88345019]
Mean: 28970.871605843375
Standard Deviation 20827.428551337565
```

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

```
y_pred
```

```
array([130460.83, 155886.5 , 145079.  , ..., 138131.5 , 110141.5 ,
       235697.2 ])
```

```
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)
```

```
traindf['TotalBath'] = traindf['FullBath'] + traindf['HalfBath'] + traindf['BsmtFullBath'] + traindf['BsmtHalfBath']
traindf['TotalSF'] = traindf['1stFlrSF'] + traindf['2ndFlrSF'] + traindf['TotalBsmtSF']
```

```
req_tr = ["GarageArea","OverallQual","TotalBath","TotalSF","TotRmsAbvGrd","SalePrice"]
selected_tr = traindf[req_tr].fillna(0)
```

```
train_set, test_set = train_test_split(selected_tr, test_size=0.2, random_state=42)
```

```
X_train = train_set.drop("SalePrice", axis=1)
Y_train = train_set["SalePrice"].copy()
```

```
X_test = test_set.drop("SalePrice", axis=1)
Y_test = test_set["SalePrice"].copy()
```

```
pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler())
])
```

```
X_train_prepared = pipeline.fit_transform(X_train)
X_test_prepared = pipeline.transform(X_test)
```

```
model = LinearRegression()
model.fit(X_train_prepared, Y_train)
```

```
▾ LinearRegression
LinearRegression()
```

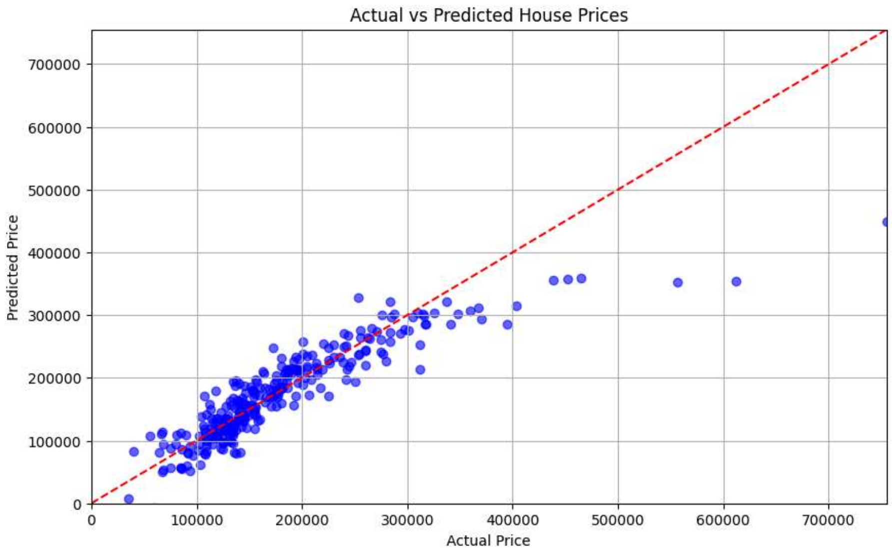
```
Y_train_pred = model.predict(X_train_prepared)
train_mse = mean_squared_error(Y_train, Y_train_pred)
train_rmse = np.sqrt(train_mse)
train_r2 = r2_score(Y_train, Y_train_pred)
print(f"Training MSE: {train_mse:.2f}, Training RMSE: {train_rmse:.2f}, Training R^2: {train_r2:.2f}")
```

```
Training MSE: 1451670460.71, Training RMSE: 38100.79, Training R^2: 0.76
```

```
Y_test_pred = model.predict(X_test_prepared)
test_mse = mean_squared_error(Y_test, Y_test_pred)
test_rmse = np.sqrt(test_mse)
test_r2 = r2_score(Y_test, Y_test_pred)
print(f"Test MSE: {test_mse:.2f}, Test RMSE: {test_rmse:.2f}, Test R^2: {test_r2:.2f}")
```

```
Test MSE: 1568951446.76, Test RMSE: 39609.99, Test R^2: 0.80
```

```
plt.figure(figsize=(10, 6))
plt.scatter(Y_test, Y_test_pred, color='blue', alpha=0.6)
plt.title('Actual vs Predicted House Prices')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.xlim(0, max(Y_test.max(), Y_test_pred.max()))
plt.ylim(0, max(Y_test.max(), Y_test_pred.max()))
plt.plot([0, max(Y_test.max(), Y_test_pred.max())], [0, max(Y_test.max(), Y_test_pred.max())], color='red', linestyle='--')
plt.grid(True)
plt.show()
```




```
errors = Y_test_pred - Y_test
plt.figure(figsize=(10, 6))
sns.histplot(errors, bins=30, kde=True, color='blue')
plt.title('Distribution of Prediction Errors')
plt.xlabel('Prediction Error')
plt.ylabel('Count')
plt.xlim(-100000, 100000)
```