```
Task 1
```

In [58]:
```
#Import All the Necessary Libraries
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
```

```
Working on Train dataframe
```

In [59]:
```
traindf = pd.read_csv('train.csv')
```

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

Out[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')
```

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

Out[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
```

In [5]:
```
req_tr = ["GarageArea","OverallQual","TotalBsmtSF","1stFlrSF","2ndFlrSF","LowQualFinSF","GrLivArea","BsmtFullBath","BsmtHalfBath'
```

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

In [12]:
```
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))
```

In [8]:
```
selected_tr
```

Out[8]:

	GarageArea	OverallQual	TotalBsmtSF	1stFlrSF	2ndFlrSF	LowQualFinSF	GrLivArea	BsmtFullBath	BsmtHalfBath	FullBath	HalfBath	TotRmsAbvGrd	S
0	548	7	856	856	854	0	1710	1	0	2	1	8	
1	460	6	1262	1262	0	0	1262	0	1	2	0	6	
2	608	7	920	920	866	0	1786	1	0	2	1	6	
3	642	7	756	961	756	0	1717	1	0	1	0	7	
4	836	8	1145	1145	1053	0	2198	1	0	2	1	9	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1455	460	6	953	953	694	0	1647	0	0	2	1	7	
1456	500	6	1542	2073	0	0	2073	1	0	2	0	7	
1457	252	7	1152	1188	1152	0	2340	0	0	2	0	9	
1458	240	5	1078	1078	0	0	1078	1	0	1	0	5	
1459	276	5	1256	1256	0	0	1256	1	0	1	1	6	

1460 rows × 15 columns

```
Keeping only the necessary columns
```

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

In [10]:
```
train_df
```

Out[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 × 6 columns

```
Splitting the dataset and Creating Pipeline
```

In [14]:
```
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
```

In [15]:
```
housing = train_set.drop("SalePrice",axis=1)
housing_labels = train_set["SalePrice"].copy()
```

In [16]:
```
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())
])
```

In [17]:
```
X_train = my_pipeline.fit_transform(housing)
```

In [18]:
```
X_train
```

Out[18]:
```
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]])
```

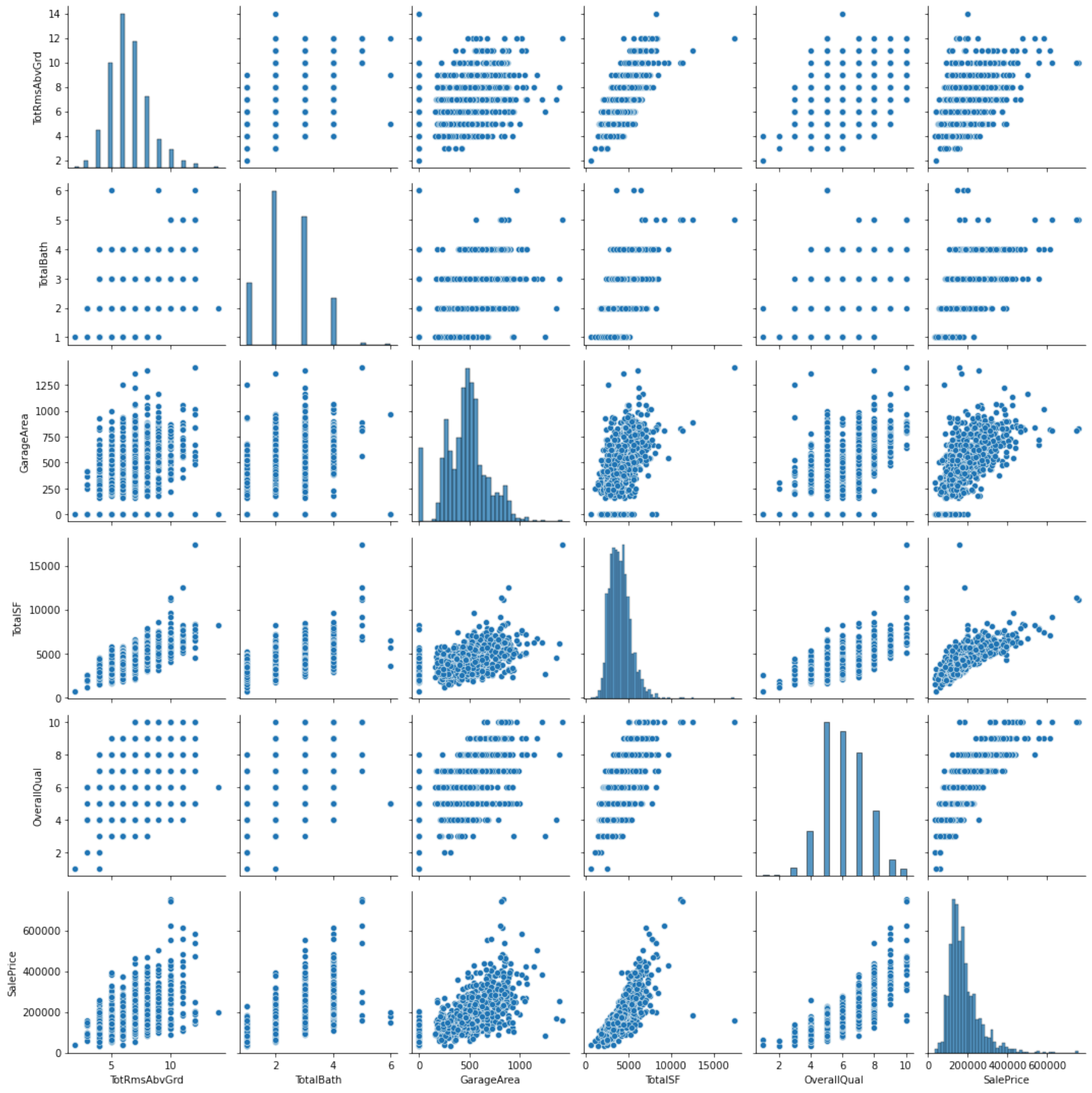
In [19]:
```
Y_train = housing_labels
```

In [20]:
```
Y_train.shape
```

Out[20]:
```
(1168,)
```

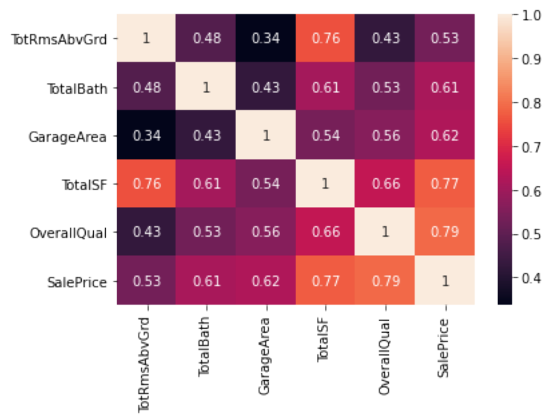
Correlations

In [22]:
```
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
%matplotlib inline
sns.pairplot(train_df)
plt.tight_layout()
plt.show()
```



```
In [23]: sns.heatmap(train_df.corr(),annot = True)

Out[23]: <AxesSubplot:>
```



	TotRmsAbvGrd	TotalBath	GarageArea	TotalSF	OverallQual	SalePrice
TotRmsAbvGrd	1	0.48	0.34	0.76	0.43	0.53
TotalBath	0.48	1	0.43	0.61	0.53	0.61
GarageArea	0.34	0.43	1	0.54	0.56	0.62
TotalSF	0.76	0.61	0.54	1	0.66	0.77
OverallQual	0.43	0.53	0.56	0.66	1	0.79
SalePrice	0.53	0.61	0.62	0.77	0.79	1

Working with Test Dataframe

```
In [25]: testdf = pd.read_csv("test.csv")
         testdf.head()

Out[25]:
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	ScreenPorch	PoolArea	PoolQC	Fence	MiscFeatu
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	...	120	0	NaN	MnPrv	Na
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	NaN	Ga
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	MnPrv	Na
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	NaN	Na
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPub	...	144	0	NaN	NaN	Na

5 rows × 80 columns

```
In [26]: req_tst = ["GarageArea","OverallQual","TotalBsmtSF","1stFlrSF","2ndFlrSF","LowQualFinSF","GrLivArea","BsmtFullBath","BsmtHalfBath
         selected_tst = testdf[req_tst]

In [27]: selected_tst = testdf[req_tst]

In [32]: 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))
```

In [30]:
```
selected_tst
```

Out[30]:

	GarageArea	OverallQual	TotalBsmtSF	1stFlrSF	2ndFlrSF	LowQualFinSF	GrLivArea	BsmtFullBath	BsmtHalfBath	FullBath	HalfBath	TotRmsAbvGrd
0	730.0	5	882.0	896	0	0	896	0.0	0.0	1	0	5
1	312.0	6	1329.0	1329	0	0	1329	0.0	0.0	1	1	6
2	482.0	5	928.0	928	701	0	1629	0.0	0.0	2	1	6
3	470.0	6	926.0	926	678	0	1604	0.0	0.0	2	1	7
4	506.0	8	1280.0	1280	0	0	1280	0.0	0.0	2	0	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1454	0.0	4	546.0	546	546	0	1092	0.0	0.0	1	1	5
1455	286.0	4	546.0	546	546	0	1092	0.0	0.0	1	1	6
1456	576.0	5	1224.0	1224	0	0	1224	1.0	0.0	1	0	7
1457	0.0	5	912.0	970	0	0	970	0.0	1.0	1	0	6
1458	650.0	7	996.0	996	1004	0	2000	0.0	0.0	2	1	9

1459 rows × 14 columns

In [33]:
```
test_df_unproc = selected_tst[['TotRmsAbvGrd','TotalBath','GarageArea','TotalSF','OverallQual']]
```

In [34]:
```
test_df_unproc
```

Out[34]:

	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

In [35]:
```
test_df = test_df_unproc.fillna(test_df_unproc.mean())
```

In [36]:
```
x_test = my_pipeline.transform(test_df[['TotRmsAbvGrd','TotalBath','GarageArea','TotalSF','OverallQual']].values)
```

In [37]:
```
x_test
```

Out[37]:
```
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 Selection
```

In [39]:
```
#model = LinearRegression()
#model = DecisionTreeRegressor()
model = RandomForestRegressor()
model.fit(X_train,Y_train)
```

Out[39]:
```
RandomForestRegressor()
```

In [40]:
```
y_train_pred = model.predict(X_train)
```

```
In [41]: y_train_pred[:5]

Out[41]: array([146834.84, 172931.32,  91300.  , 166744.24, 140191.  ])

In [42]: some_data = housing.iloc[:5]
         some_labels = housing_labels.iloc[:5]

In [43]: proc_data = my_pipeline.transform(some_data)

In [44]: model.predict(proc_data)

Out[44]: array([146834.84, 172931.32,  91300.  , 166744.24, 140191.  ])

In [45]: list(some_labels)

Out[45]: [145000, 178000, 85000, 175000, 127000]

In [46]: train_mse = mean_squared_error(Y_train,y_train_pred)

In [47]: train_rmse = np.sqrt(train_mse)

In [48]: print(f"Training MSE: {train_mse:.2f}, Training RMSE: {train_rmse:.2f}")

         Training MSE: 172026550.98, Training RMSE: 13115.89
```

Cross - Validation

```
In [50]: 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)
```

In [52]:
```
rmse_scores
```

Out[52]:
```
array([ 21797.00669202,  14957.23898344,  23275.47683597,  11572.31366899,
        48469.42139054,   7978.34457094,  20108.00276321,  12182.28705366,
         9675.6005755 ,  50051.95097995,  35336.34359263,  28413.51341703,
        13376.37160055,   9231.85991138,  19188.14282691,  24133.7195857 ,
        22196.33020079,  34130.85574345,  37975.93581947,  22577.63227323,
        29179.25399623,  18375.66206345,  17160.70071119,  27878.48541779,
        19603.82459691,  16825.41801781,  43740.14145525,  39724.65220952,
       176909.66719632,  51788.88255228,  23334.31657806,  33784.70364157,
        22282.38778279,  32334.5193015 ,  50705.24900549,  13047.87527871,
        22625.86361465,  29734.56486361,  19631.81205684,  30324.90502457,
        21632.85519688,  31509.02307562,  28060.15770012,  32925.9132895 ,
        33147.81554893,  29222.27202686,  27771.70311292,  42393.55519823,
        21369.22916065,  22467.10057453,  20924.18662807,  48161.4827102 ,
        41136.68890422,  36070.41333717,  23224.24710799,  26647.34160335,
        10923.56368389,  27395.41929729,  25133.96265023,  30794.43284436,
       200176.50686324,  12964.3965824 ,  23573.53115501,  35304.32045918,
        22433.83451002,  25678.94236018,  37458.0628356 ,  29555.53944044,
         7661.74250814,  14727.787973  ,  57329.12632426,  27221.77836773,
        57520.85344648,  18551.85315539,  46763.06902435,  43296.50993689,
        40860.09435628,  29802.20027902,  41610.13598278,  32778.20515812,
        27980.9459166 ,  32012.30602063, 102727.14956704,   8512.05566991,
        48702.8238754 ,  33746.41055608,  18892.61485873,  23173.03608053,
        34232.51683334,  76160.87323065,  19534.11024824,  22763.47101524,
        26778.91449607,  24839.86969716,  22023.51121385,  22899.09792426,
        25177.27969627,  27054.33547047,  70353.74989691,  13075.34584013,
        30295.02368228,  27852.836771  ,  47272.47658452,  52837.22871236,
        18265.70827883,  18609.07531439,  37755.77969763,  18381.18261301,
        19391.26991251,  15537.5580907 ,  19831.66264374,  12679.66154048,
        12374.18002601,  19689.58169192,  29660.72321625,  34117.9541451 ,
        76540.12052656,  29696.42354019,  18924.90446707,  25584.9575775 ,
        30115.01267773,  28243.26709808,  16246.69567596,  20069.33052675,
        33494.84124277,  20665.78796561,  30613.53973346,  46517.14561505,
        39252.64005476,  16479.40451005,  33323.31507917,   7912.69298667,
        24606.93935367,  27606.29374646,  26867.35518677,  12344.08722316,
        47467.0233914 ,  34801.16337931,  28816.48668026,  12774.76565494,
        25667.81631088,  25781.74953679,  26568.81983178,  11793.86336533,
        27023.99016464,  28976.83715756,  32867.98942372,  13969.1685425 ,
        12431.51201239,  15682.35086843,  25441.42785125,  26944.52144272,
        14107.63519776,  41800.21756193,  30936.17488118,  11967.64690939,
        23653.52173679,  25313.56984546,  21167.41561914,  23787.55509279,
        52678.95257977,  27239.65401155,  26241.70676363,  19016.82308447,
        49723.35586252,  21594.08841625,  12986.52271444,  61732.56257263,
        20709.83670122,  29636.14876585,  29617.10636478,  23944.94913123,
        14187.16088132,  13210.82900748,  14164.14058122,  18636.91790398,
        22962.13911521,  16815.15778132,  39186.89137075,  21304.89640533,
        33535.667977  ,  11000.74891384,  15984.82892073,   6314.40760104,
        17886.30632806,  31777.16028452,  26004.08548421,  13937.61202777,
        26535.04161049,  38994.06371719,  29611.16999628,  19890.36672566,
        17924.8571899 ,  13186.03558123,  15470.62428205,  17103.37379449,
        58208.00583386,  20068.74696506,  17829.19613253,  29203.44959104])
```

In [53]:
```
def print_scores(scores):
    print("Scores:",scores)
    print("Mean:",scores.mean())
    print("Standard Deviation",scores.std())
```

In [54]:
```
print_scores(rmse_scores)
```

```
Scores: [ 21797.00669202  14957.23898344  23275.47683597  11572.31366899
  48469.42139054   7978.34457094  20108.00276321  12182.28705366
   9675.6005755   50051.95097995  35336.34359263  28413.51341703
  13376.37160055   9231.85991138  19188.14282691  24133.7195857
  22196.33020079  34130.85574345  37975.93581947  22577.63227323
  29179.25399623  18375.66206345  17160.70071119  27878.48541779
  19603.82459691  16825.41801781  43740.14145525  39724.65220952
 176909.66719632  51788.88255228  23334.31657806  33784.70364157
  22282.38778279  32334.5193015   50705.24900549  13047.87527871
  22625.86361465  29734.56486361  19631.81205684  30324.90502457
  21632.85519688  31509.02307562  28060.15770012  32925.9132895
  33147.81554893  29222.27202686  27771.70311292  42393.55519823
  21369.22916065  22467.10057453  20924.18662807  48161.4827102
  41136.68890422  36070.41333717  23224.24710799  26647.34160335
  10923.56368389  27395.41929729  25133.96265023  30794.43284436
 200176.50686324  12964.3965824   23573.53115501  35304.32045918
  22433.83451002  25678.94236018  37458.0628356   29555.53944044
   7661.74250814  14727.787973    57329.12632426  27221.77836773
  57520.85344648  18551.85315539  46763.06902435  43296.50993689
  40860.09435628  29802.20027902  41610.13598278  32778.20515812
  27980.9459166   32012.30602063 102727.14956704   8512.05566991
  48702.8238754   33746.41055608  18892.61485873  23173.03608053
  34232.51683334  76160.87323065  19534.11024824  22763.47101524
  26778.91449607  24839.86969716  22023.51121385  22899.09792426
  25177.27969627  27054.33547047  70353.74989691  13075.34584013
  30295.02368228  27852.836771    47272.47658452  52837.22871236
  18265.70827883  18609.07531439  37755.77969763  18381.18261301
  19391.26991251  15537.5580907   19831.66264374  12679.66154048
  12374.18002601  19689.58169192  29660.72321625  34117.9541451
  76540.12052656  29696.42354019  18924.90446707  25584.9575775
  30115.01267773  28243.26709808  16246.69567596  20069.33052675
  33494.84124277  20665.78796561  30613.53973346  46517.14561505
  39252.64005476  16479.40451005  33323.31507917   7912.69298667
  24606.93935367  27606.29374646  26867.35518677  12344.08722316
  47467.0233914   34801.16337931  28816.48668026  12774.76565494
  25667.81631088  25781.74953679  26568.81983178  11793.86336533
  27023.99016464  28976.83715756  32867.98942372  13969.1685425
  12431.51201239  15682.35086843  25441.42785125  26944.52144272
  14107.63519776  41800.21756193  30936.17488118  11967.64690939
  23653.52173679  25313.56984546  21167.41561914  23787.55509279
  52678.95257977  27239.65401155  26241.70676363  19016.82308447
  49723.35586252  21594.08841625  12986.52271444  61732.56257263
  20709.83670122  29636.14876585  29617.10636478  23944.94913123
  14187.16088132  13210.82900748  14164.14058122  18636.91790398
  22962.13911521  16815.15778132  39186.89137075  21304.89640533
  33535.667977    11000.74891384  15984.82892073   6314.40760104
  17886.30632806  31777.16028452  26004.08548421  13937.61202777
  26535.04161049  38994.06371719  29611.16999628  19890.36672566
  17924.8571899   13186.03558123  15470.62428205  17103.37379449
  58208.00583386  20068.74696506  17829.19613253  29203.44959104]
Mean: 29089.48487732674
Standard Deviation 21087.336412413624
```

In [55]:
```
y_pred=model.predict(x_test)
```

In [56]:
```
y_pred
```

Out[56]:
```
array([133177.5 , 154364.32, 144508.37, ..., 140685.  , 108002.  ,
       235164.1 ])
```

In [57]:
```
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)
```

In [ ]: