House Prices : Advanced Regression Techniques

Aim: Predict the sale price of a house

Features (80) :

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, Bedroom, Kitchen, 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

Kaggle dataset: [https://www.kaggle.com/c/house-prices-advanced-regression-techniques (https://www.kaggle.com/c/house-prices-advanced-regression-techniques)](https://www.kaggle.com/c/house-prices-advanced-regression-techniques)

In [1]:
```
# import necessary libraries

import pandas as pd
import sys
import numpy as np
import seaborn as sns
from math import sqrt
from pylab import rcParams

from sklearn import metrics
from sklearn.metrics import mean_squared_error
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

import sklearn
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import ElasticNet, Lasso
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.kernel_ridge import KernelRidge

from sklearn.ensemble import StackingRegressor

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
```

--- 1. LOADING & LOOKING AT THE DATA ---

- The housing dataset is available on Kaggle under “House Prices: Advanced Regression Techniques”. The “train.csv” file contains the training data and “test.csv” contains the testing data. The training data contains data for 1460 rows which corresponds to 1460 house’s data and 80 columns which correspond to the feature of those houses. Similarly, the testing data contains data of 1461 houses and their 79 attributes.

In [2]:
```
# load dataset
csv_path = "train.csv"
df_train = pd.read_csv(csv_path, sep = ',')

csv_path = "test.csv"
df_test = pd.read_csv(csv_path, sep = ',')
```

In [3]:
```
# check shape
print(df_train.shape)
print(df_test.shape)
```

```
(1460, 81)
(1459, 80)
```

In [4]:
```python
# look a first 10 rows of training data
df_train.head(10)
```

Out[4]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN
5	6	50	RL	85.0	14115	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	MnPrv	Shed
6	7	20	RL	75.0	10084	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN
7	8	60	RL	NaN	10382	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	Shed
8	9	50	RM	51.0	6120	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN
9	10	190	RL	50.0	7420	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN

10 rows × 81 columns

In [5]:

```python
# look a first 10 rows of testing data
df_test.head(10)
```

Out[5]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	ScreenPorch	PoolArea	PoolQC	Fenc
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	...	120	0	NaN	MnPr
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	Na
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	MnPr
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	Na
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPub	...	144	0	NaN	Na
5	1466	60	RL	75.0	10000	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	Na
6	1467	20	RL	NaN	7980	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	GdPr
7	1468	60	RL	63.0	8402	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	Na
8	1469	20	RL	85.0	10176	Pave	NaN	Reg	Lvl	AllPub	...	0	0	NaN	Na
9	1470	20	RL	70.0	8400	Pave	NaN	Reg	Lvl	AllPub	...	0	0	NaN	MnPr

10 rows × 80 columns

In [6]:
```python
# see all the column names
df_train.columns
```

Out[6]:
```
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 [7]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
 20  YearRemodAdd   1460 non-null   int64
 21  RoofStyle      1460 non-null   object
 22  RoofMatl       1460 non-null   object
 23  Exterior1st    1460 non-null   object
 24  Exterior2nd    1460 non-null   object
 25  MasVnrType     1452 non-null   object
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object
 28  ExterCond      1460 non-null   object
 29  Foundation     1460 non-null   object
 30  BsmtQual       1423 non-null   object
 31  BsmtCond       1423 non-null   object
 32  BsmtExposure   1422 non-null   object
 33  BsmtFinType1   1423 non-null   object
 34  BsmtFinSF1     1460 non-null   int64
 35  BsmtFinType2   1422 non-null   object
```

```
36  BsmtFinSF2     1460 non-null   int64
37  BsmtUnfSF      1460 non-null   int64
38  TotalBsmtSF    1460 non-null   int64
39  Heating        1460 non-null   object
40  HeatingQC      1460 non-null   object
41  CentralAir     1460 non-null   object
42  Electrical     1459 non-null   object
43  1stFlrSF       1460 non-null   int64
44  2ndFlrSF       1460 non-null   int64
45  LowQualFinSF   1460 non-null   int64
46  GrLivArea      1460 non-null   int64
47  BsmtFullBath   1460 non-null   int64
48  BsmtHalfBath   1460 non-null   int64
49  FullBath       1460 non-null   int64
50  HalfBath       1460 non-null   int64
51  BedroomAbvGr   1460 non-null   int64
52  KitchenAbvGr   1460 non-null   int64
53  KitchenQual    1460 non-null   object
54  TotRmsAbvGrd   1460 non-null   int64
55  Functional     1460 non-null   object
56  Fireplaces     1460 non-null   int64
57  FireplaceQu    770 non-null    object
58  GarageType     1379 non-null   object
59  GarageYrBlt    1379 non-null   float64
60  GarageFinish   1379 non-null   object
61  GarageCars     1460 non-null   int64
62  GarageArea     1460 non-null   int64
63  GarageQual     1379 non-null   object
64  GarageCond     1379 non-null   object
65  PavedDrive     1460 non-null   object
66  WoodDeckSF     1460 non-null   int64
67  OpenPorchSF    1460 non-null   int64
68  EnclosedPorch  1460 non-null   int64
69  3SsnPorch      1460 non-null   int64
70  ScreenPorch    1460 non-null   int64
71  PoolArea       1460 non-null   int64
72  PoolQC         7 non-null      object
73  Fence          281 non-null    object
74  MiscFeature    54 non-null     object
75  MiscVal        1460 non-null   int64
76  MoSold         1460 non-null   int64
77  YrSold         1460 non-null   int64
```

```
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

- There are 1460 rows and 81 columns
- There are columns with large number of null entries like PoolQC, MiscFeature
- The columns have Three types of datatypes: float64(3), int64(35), object(43)

In [8]:
```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1459 entries, 0 to 1458
Data columns (total 80 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1459 non-null   int64  
 1   MSSubClass     1459 non-null   int64  
 2   MSZoning       1455 non-null   object 
 3   LotFrontage    1232 non-null   float64
 4   LotArea        1459 non-null   int64  
 5   Street         1459 non-null   object 
 6   Alley          107 non-null    object 
 7   LotShape       1459 non-null   object 
 8   LandContour    1459 non-null   object 
 9   Utilities      1457 non-null   object 
 10  LotConfig      1459 non-null   object 
 11  LandSlope      1459 non-null   object 
 12  Neighborhood   1459 non-null   object 
 13  Condition1     1459 non-null   object 
 14  Condition2     1459 non-null   object 
 15  BldgType       1459 non-null   object 
 16  HouseStyle     1459 non-null   object 
 17  OverallQual    1459 non-null   int64  
 18  OverallCond    1459 non-null   int64  
 19  YearBuilt      1459 non-null   int64  
 20  YearRemodAdd   1459 non-null   int64  
 21  RoofStyle      1459 non-null   object 
 22  RoofMatl       1459 non-null   object 
 23  Exterior1st    1458 non-null   object 
 24  Exterior2nd    1458 non-null   object 
 25  MasVnrType     1443 non-null   object 
 26  MasVnrArea     1444 non-null   float64
 27  ExterQual      1459 non-null   object 
 28  ExterCond      1459 non-null   object 
 29  Foundation     1459 non-null   object 
 30  BsmtQual       1415 non-null   object 
 31  BsmtCond       1414 non-null   object 
 32  BsmtExposure   1415 non-null   object 
 33  BsmtFinType1   1417 non-null   object 
 34  BsmtFinSF1     1458 non-null   float64
 35  BsmtFinType2   1417 non-null   object 
```

```
36  BsmtFinSF2     1458 non-null   float64
37  BsmtUnfSF      1458 non-null   float64
38  TotalBsmtSF    1458 non-null   float64
39  Heating        1459 non-null   object
40  HeatingQC      1459 non-null   object
41  CentralAir     1459 non-null   object
42  Electrical     1459 non-null   object
43  1stFlrSF       1459 non-null   int64
44  2ndFlrSF       1459 non-null   int64
45  LowQualFinSF   1459 non-null   int64
46  GrLivArea      1459 non-null   int64
47  BsmtFullBath   1457 non-null   float64
48  BsmtHalfBath   1457 non-null   float64
49  FullBath       1459 non-null   int64
50  HalfBath       1459 non-null   int64
51  BedroomAbvGr   1459 non-null   int64
52  KitchenAbvGr   1459 non-null   int64
53  KitchenQual    1458 non-null   object
54  TotRmsAbvGrd   1459 non-null   int64
55  Functional     1457 non-null   object
56  Fireplaces     1459 non-null   int64
57  FireplaceQu    729 non-null    object
58  GarageType     1383 non-null   object
59  GarageYrBlt    1381 non-null   float64
60  GarageFinish   1381 non-null   object
61  GarageCars     1458 non-null   float64
62  GarageArea     1458 non-null   float64
63  GarageQual     1381 non-null   object
64  GarageCond     1381 non-null   object
65  PavedDrive     1459 non-null   object
66  WoodDeckSF     1459 non-null   int64
67  OpenPorchSF    1459 non-null   int64
68  EnclosedPorch  1459 non-null   int64
69  3SsnPorch      1459 non-null   int64
70  ScreenPorch    1459 non-null   int64
71  PoolArea       1459 non-null   int64
72  PoolQC         3 non-null      object
73  Fence          290 non-null    object
74  MiscFeature    51 non-null     object
75  MiscVal        1459 non-null   int64
76  MoSold         1459 non-null   int64
77  YrSold         1459 non-null   int64
```

```
 78  SaleType       1458 non-null   object
 79  SaleCondition  1459 non-null   object
dtypes: float64(11), int64(26), object(43)
memory usage: 912.0+ KB
```

- There are 1459 rows and 80 columns
- There are columns with large number of null entries like PoolQC, MiscFeature etc
- The columns have Three types of datatypes: float64(11), int64(26), object(43)

Looking at the label to predict

In [9]:
```
df_train['SalePrice'].describe()
```

Out[9]:
```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

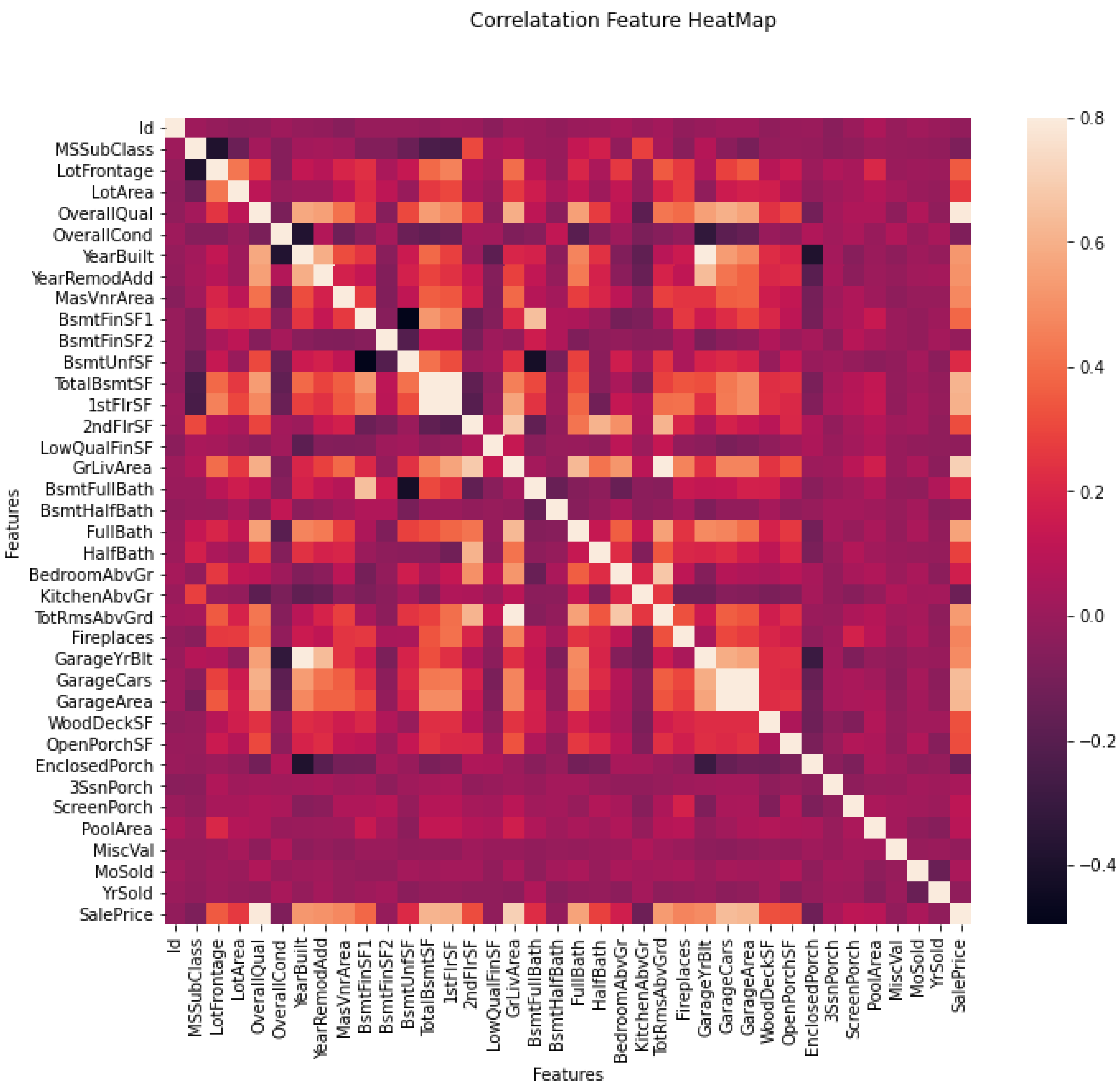
- The average SalePrice of a house is 180,921
- The Maximum SalePrice of a house is 755,000 and Minimum 34,900

In [10]:

```
#correlation matrix
corr_mat = df_train.corr()
f, ax = plt.subplots(figsize=(12, 9))

sns.heatmap(corr_mat, vmax=.8,square=True)

plt.suptitle("Correlatation Feature HeatMap")
plt.xlabel("Features")
plt.ylabel("Features")
```

Out[10]:

```
Text(133.44000000000005, 0.5, 'Features')
```

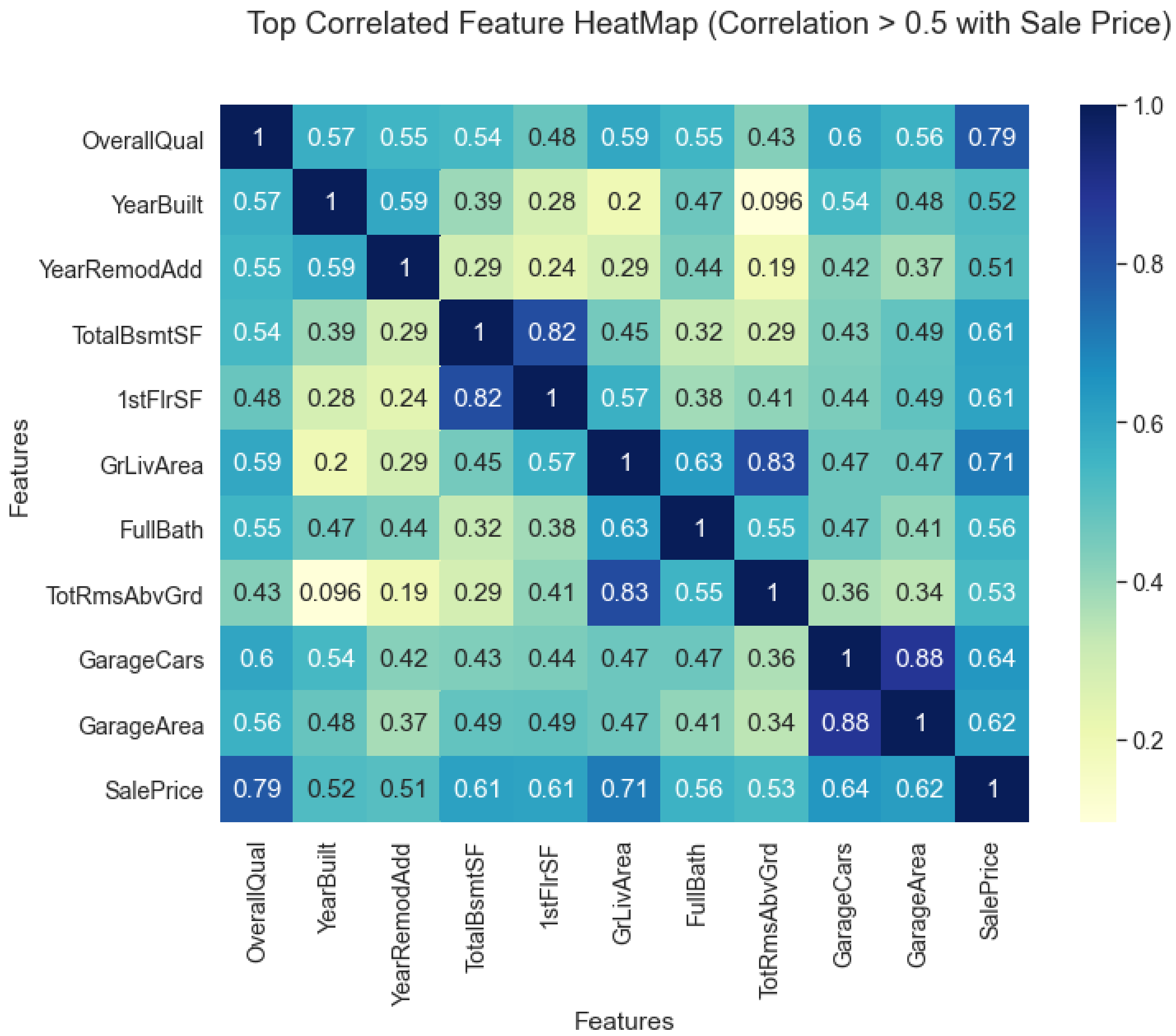

Correlatation Feature HeatMap
Features
Features
Id
MSSubClass
LotFrontage
LotArea
OverallQual
OverallCond
YearBuilt
YearRemodAdd
MasVnrArea
BsmtFinSF1
BsmtFinSF2
BsmtUnfSF
TotalBsmtSF
1stFlrSF
2ndFlrSF
LowQualFinSF
GrLivArea
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageYrBlt
GarageCars
GarageArea
WoodDeckSF
OpenPorchSF
EnclosedPorch
3SsnPorch
ScreenPorch
PoolArea
MiscVal
MoSold
YrSold
SalePrice
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

In [11]:

```
# most correlated features
corr_mat = df_train.corr()

sns.set(font_scale = 1.3)
plt.figure(figsize = (11,8))

top_corr = corr_mat.index[abs(corr_mat["SalePrice"])>0.5]
g = sns.heatmap(df_train[top_corr].corr(),annot=True,cmap="YlGnBu")
plt.suptitle("Top Correlated Feature HeatMap (Correlation > 0.5 with Sale Price)")
plt.xlabel("Features")
plt.ylabel("Features")
```

Out[11]:

```
Text(71.5, 0.5, 'Features')
```

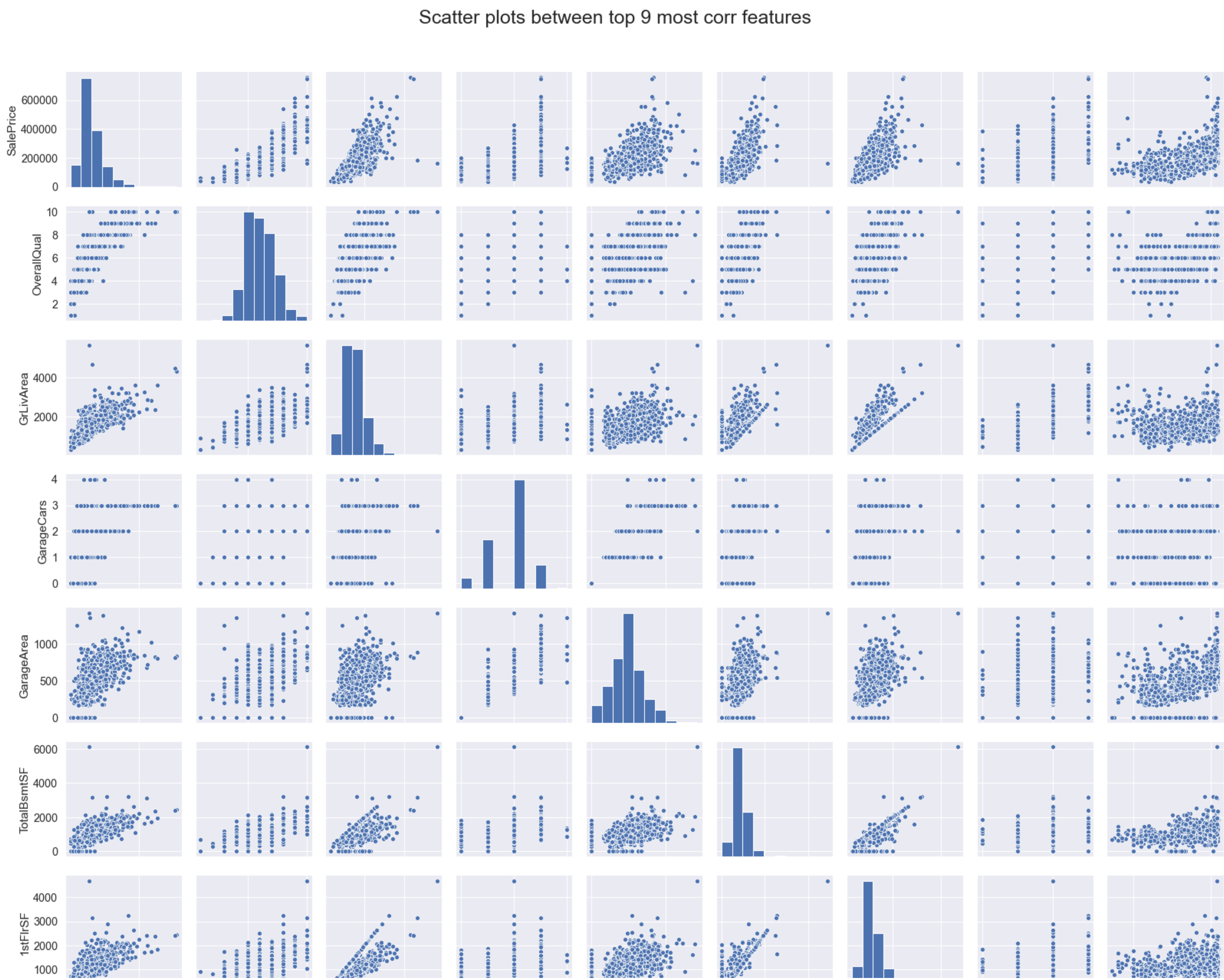


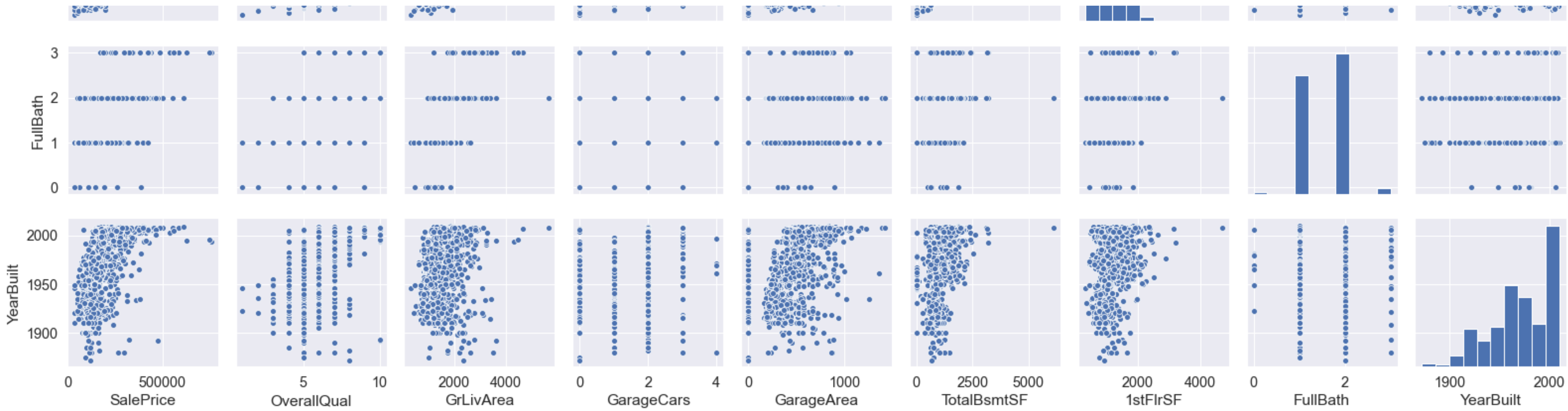
- OverallQual and GrLivArea seem to be the most correlated to SalePrice

In [12]:
```
print("Correlation Values")

corr = df_train.corr().drop('SalePrice')
corr.sort_values(["SalePrice"], ascending = False, inplace = True)
print(corr.SalePrice)
```

```
Correlation Values
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 [13]:
```
rcParams['figure.figsize'] = 5,5
cols = ['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars','GarageArea', 'TotalBsmtSF','1stFlrSF','FullBath','YearBu
sns_plot = sns.pairplot(df_train[cols])

plt.suptitle('Scatter plots between top 9 most corr features', y=1.04, size=25)
plt.tight_layout()
plt.show()
```


Scatter plots between top 9 most corr features
SalePrice
OverallQual
GrLivArea
GarageCars
GarageArea
TotalBsmtSF
1stFlrSF
0
200000
400000
600000
2
4
6
8
10
2000
4000
0
1
2
3
4
0
500
1000
0
2000
4000
6000
1000
2000
3000
4000


FullBath
YearBuilt
SalePrice
OverallQual
GrLivArea
GarageCars
GarageArea
TotalBsmtSF
1stFlrSF
FullBath
YearBuilt
0
500000
5
10
2000
4000
0
2
4
0
1000
0
2500
5000
2000
4000
0
2
1900
2000
3
2
1
0
2000
1950
1900

In [14]:

```
rcParams['figure.figsize'] = 5,5
cols = ['SalePrice','EnclosedPorch', 'KitchenAbvGr', 'MSSubClass', 'LowQualFinSF','YrSold', 'OverallCond']
sns_plot = sns.pairplot(df_train[cols])

plt.suptitle('Scatter plots between least 6  corr features', y=1.04, size=20)
plt.tight_layout()
plt.show()
```

Scatter plots between least 6 corr features
SalePrice
EnclosedPorch
KitchenAbvGr
MSSubClass
LowQualFinSF

--- 2. HANDLING DATA ---

Drop Id Column

In [15]:
```
#drop id as it is not required for training or prediction
train_ID = df_train['Id']
test_ID = df_test['Id']

df_train.drop(['Id'], axis=1, inplace=True)
df_test.drop(['Id'], axis=1, inplace=True)

df_train.shape, df_test.shape
```

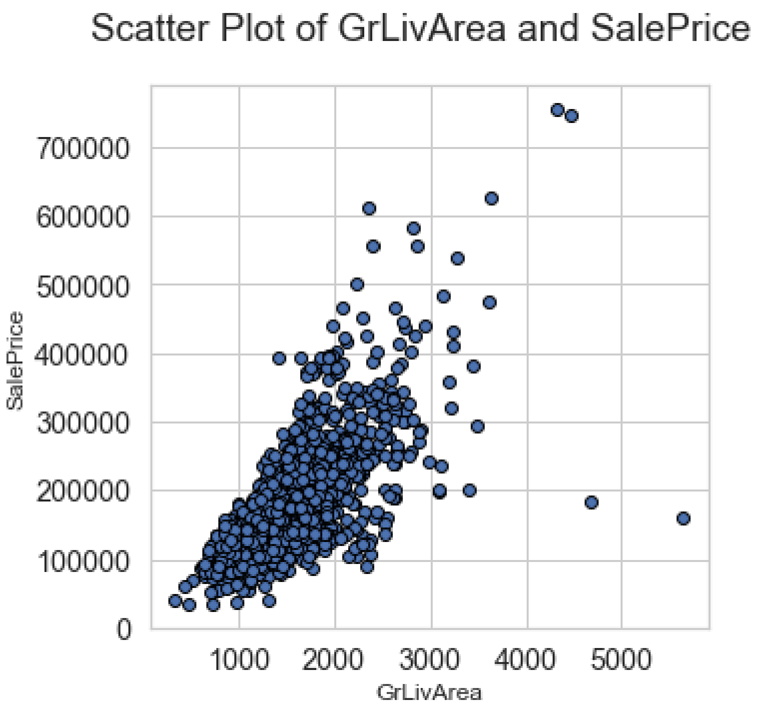
Out[15]:
```
((1460, 80), (1459, 79))
```

Checking for Outliers

In [16]:
```
sns.set_style('whitegrid')
edgecolor = 'black'

fig = plt.figure(figsize=(12,12))

#function to plot scatter plot between a feature and the Sale Price
def scatter_plot(a):
    fig, ax = plt.subplots()
    ax.scatter(x = df_train[a], y = df_train['SalePrice'], edgecolor=edgecolor)
    plt.ylabel('SalePrice', fontsize=12)
    plt.xlabel(a, fontsize=12)
    plt.suptitle("Scatter Plot of "+ a + " and SalePrice")
    plt.show()
```

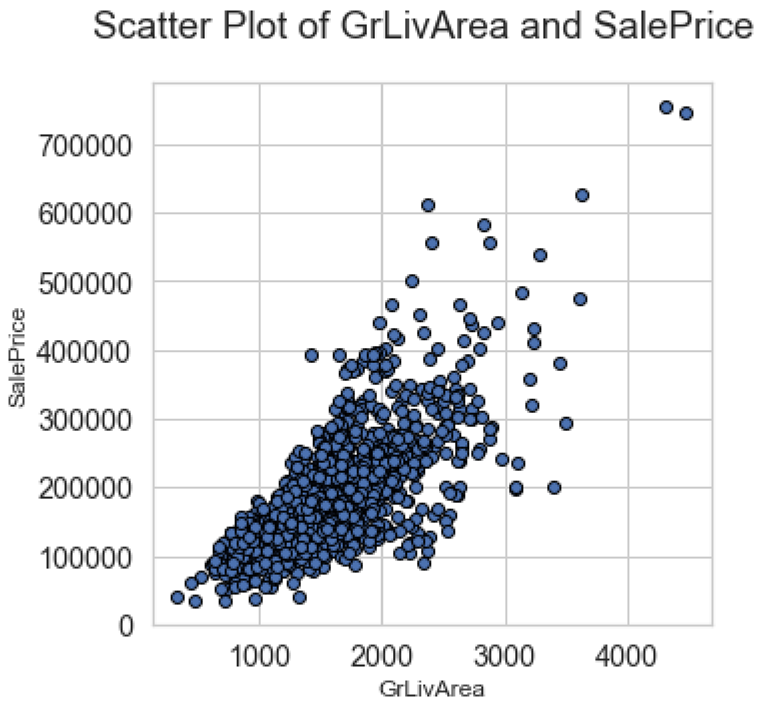
```
<Figure size 864x864 with 0 Axes>
```

In [17]:
```
scatter_plot('GrLivArea')
```

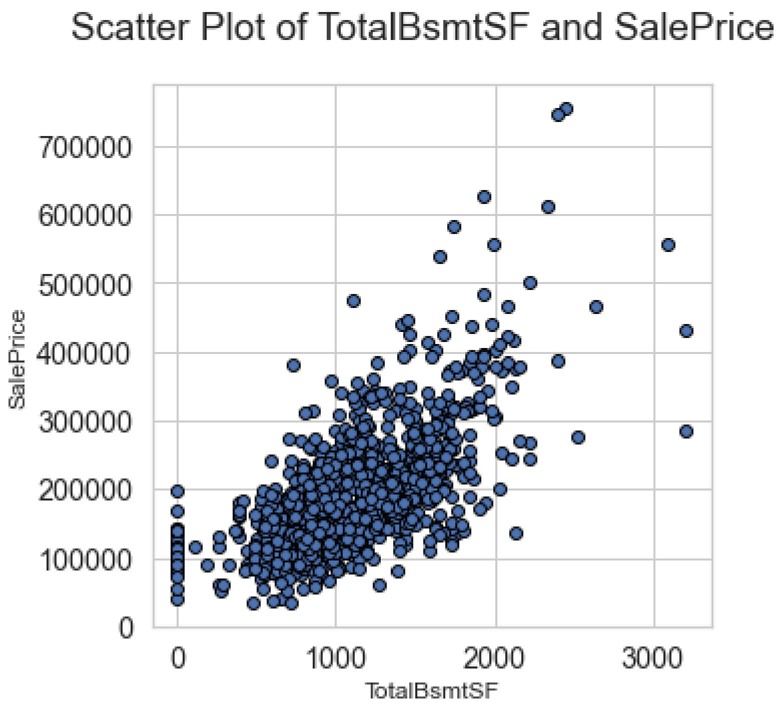


- It can be observed that there are large outliers which can negatively affect the prediction of sale price highly
- So the outliers need to be deleted

In [18]:
```
#Deleting outliers
df_train =  df_train.drop( df_train[( df_train['GrLivArea'] > 4000) & ( df_train['SalePrice']<300000)].index)

#Check the graphic again
scatter_plot('GrLivArea')
```

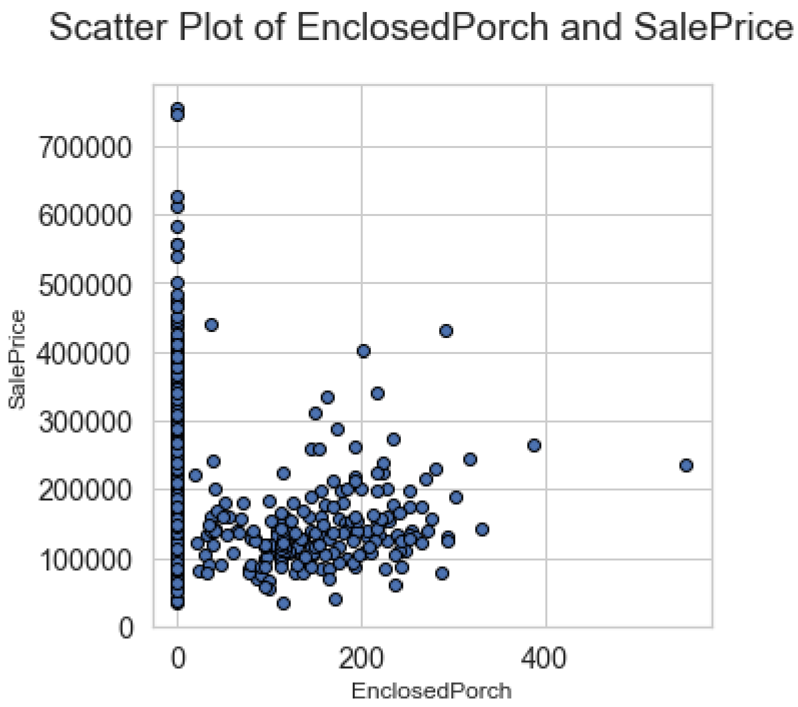


In [19]:
```
scatter_plot('TotalBsmtSF')
```



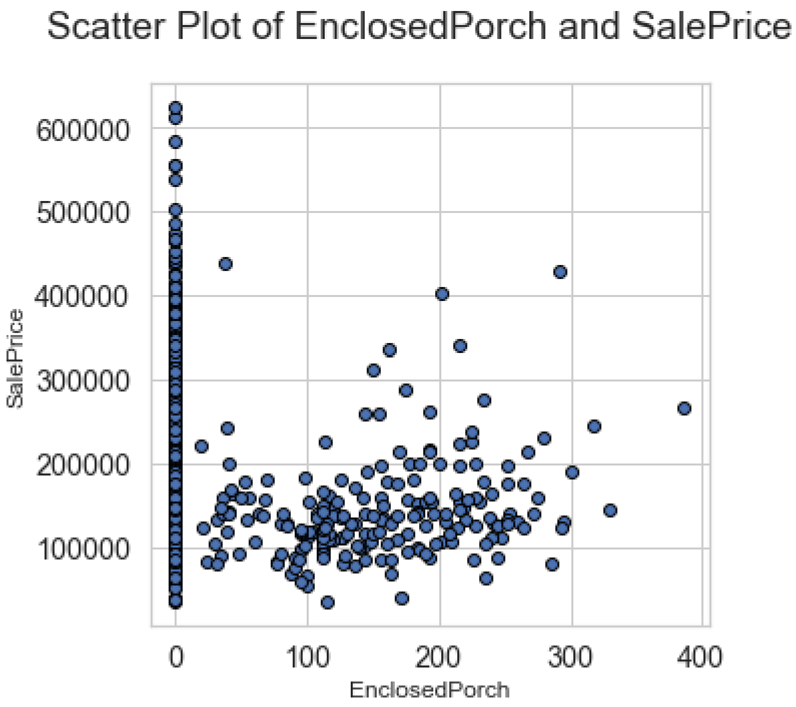
- There arent too large outliers, we do not need to delete any points

In [20]:
```
scatter_plot('EnclosedPorch')
```



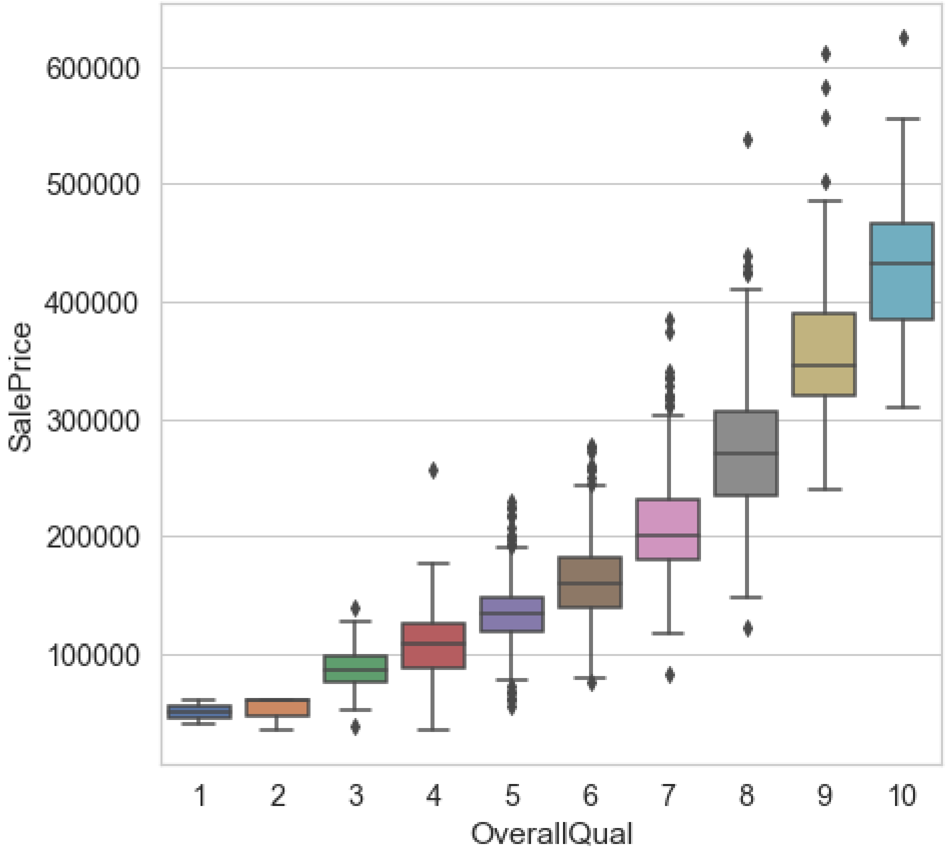
- There is are some outliers that should be deleted so that it doesnt affect our predictions much

In [21]:
```
#Deleting outliers
df_train =  df_train.drop( df_train[( df_train['EnclosedPorch']>400)].index)

#Deleting outliers
df_train =  df_train.drop( df_train[( df_train['SalePrice']>700000)].index)

#check plot again
scatter_plot('EnclosedPorch')
```



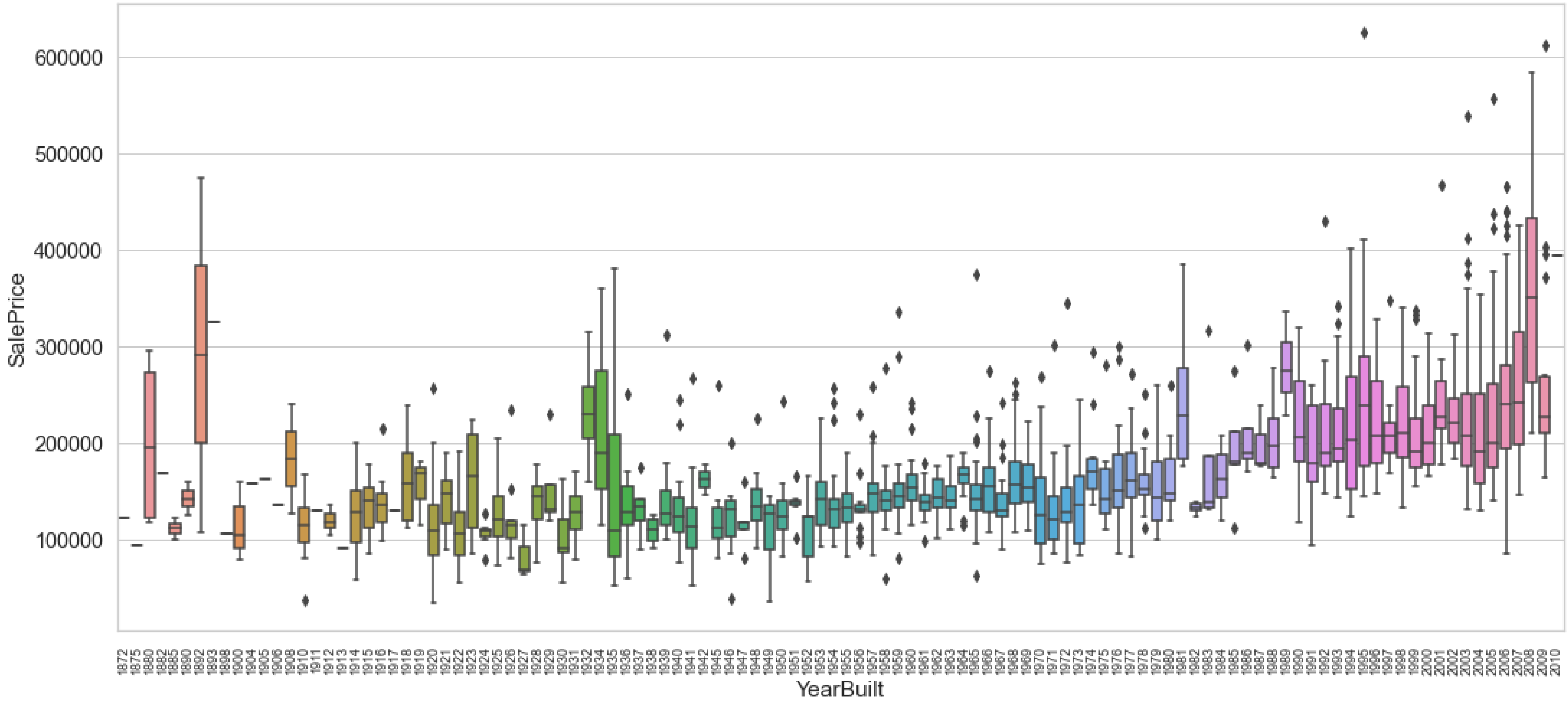
In [22]:
```
# plot a box plot for categorical feature : Overall Quality

fig = plt.figure(figsize=(7,7))
data = pd.concat([df_train['SalePrice'], df_train['OverallQual']], axis=1)
sns.boxplot(x = df_train['OverallQual'], y="SalePrice", data = data)
```

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x1ef8b00fb80>



In [23]:
```python
# plot a box plot for categorical feature : Year Built
fig = plt.figure(figsize=(18,8))

data = pd.concat([df_train['SalePrice'], df_train['YearBuilt']], axis=1)
sns.boxplot(x= df_train['YearBuilt'], y="SalePrice", data=data)
plt.xticks(rotation=90,fontsize= 9)
```

Out[23]:
```
(array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
         13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
         26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
         39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
         52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
         65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,
         78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
         91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103,
        104, 105, 106, 107, 108, 109, 110, 111]),
 <a list of 112 Text major ticklabel objects>)
```

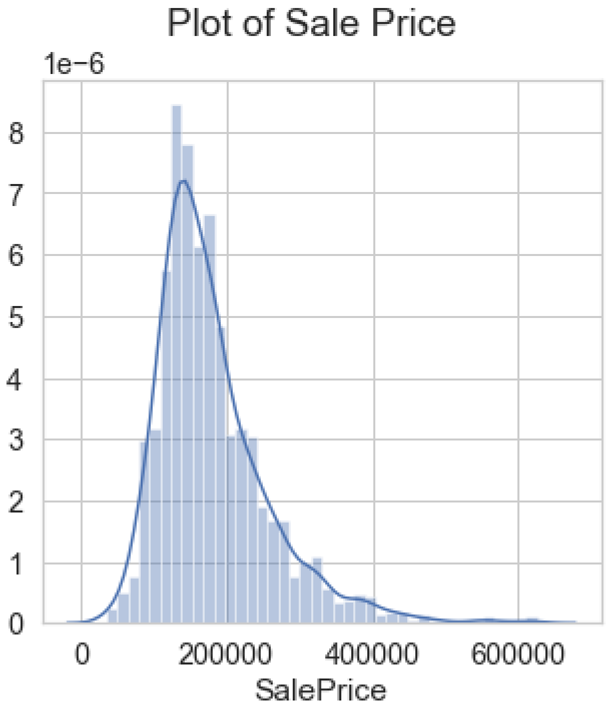


SalePrice
YearBuilt

In [24]:

```
sns.distplot(df_train['SalePrice'])

plt.suptitle( "Plot of Sale Price")

print("Skewness: %f" % df_train['SalePrice'].skew())
print("Kurtosis: %f" % df_train['SalePrice'].kurt())
```

```
Skewness: 1.567473
Kurtosis: 3.888317
```



In [25]:
```
# applying log transformation to correct the positive skewness in the data
# taking logs means that errors in predicting expensive and cheap houses will affect the result equally

df_train['SalePrice'] = np.log(df_train['SalePrice'])
plt.suptitle("Plot of Sale Price after log transformation")
sns.distplot(df_train['SalePrice'])
plt.show()
```

In [26]:
```
df_train['SalePrice'].describe()
```

Out[26]:
```
count    1455.000000
mean       12.021706
std         0.396112
min        10.460242
25%        11.774520
50%        12.001505
75%        12.272562
max        13.345507
Name: SalePrice, dtype: float64
```

In [27]:
```
df_train['SalePrice']
```

Out[27]:
```
0       12.247694
1       12.109011
2       12.317167
3       11.849398
4       12.429216
          ...
1455    12.072541
1456    12.254863
1457    12.493130
1458    11.864462
1459    11.901583
Name: SalePrice, Length: 1455, dtype: float64
```

In [28]:
```
df_train.shape
```

Out[28]:
```
(1455, 80)
```

Handling missing data

In [29]:
```
#function to see the missing data in a dataframe
def missing_data(df,n):
    total = df.isnull().sum().sort_values(ascending=False)          # Total No of missing values
    percentage = (df.isnull().sum() / df.isnull().count()).sort_values(ascending=False)*100  # % of Missing values
    No_unique_val = df.nunique()                                     # No of unique values
    missing_data = pd.concat([total, percentage, No_unique_val], axis=1,
                             keys=['Total No of missing val', '% of Missing val','No of unique val'], sort = False)

    print(missing_data.head(n))
```

In [30]:
```
#training data
missing_data(df_train,20)
```

```
              Total No of missing val  % of Missing val  No of unique val
PoolQC                           1451         99.725086                 2
MiscFeature                      1401         96.288660                 4
Alley                            1364         93.745704                 2
Fence                            1176         80.824742                 4
FireplaceQu                       690         47.422680                 5
LotFrontage                       259         17.800687               109
GarageType                         81          5.567010                 6
GarageCond                         81          5.567010                 5
GarageFinish                       81          5.567010                 3
GarageQual                         81          5.567010                 5
GarageYrBlt                        81          5.567010                97
BsmtFinType2                       38          2.611684                 6
BsmtExposure                       38          2.611684                 4
BsmtQual                           37          2.542955                 4
BsmtCond                           37          2.542955                 4
BsmtFinType1                       37          2.542955                 6
MasVnrArea                          8          0.549828               324
MasVnrType                          8          0.549828                 4
Electrical                          1          0.068729                 5
RoofMatl                            0          0.000000                 7
```

In [31]:
```
df_train['PoolQC'].unique()
```

Out[31]:
```
array([nan, 'Fa', 'Gd'], dtype=object)
```

- PoolQC,Alley have only two unique values

- PoolQC has 99.7% of missing data, which means most of the values are NA: No Pool ie most of the houses do not have a pool
- PoolQC,Alley,MiscFeature will be dropped due to large number of missing values

In [32]:
```
#test data
missing_data(df_test,34)
```

	Total No of missing val	% of Missing val	No of unique val
PoolQC	1456	99.794380	2
MiscFeature	1408	96.504455	3
Alley	1352	92.666210	2
Fence	1169	80.123372	4
FireplaceQu	730	50.034270	5
LotFrontage	227	15.558602	115
GarageCond	78	5.346127	5
GarageFinish	78	5.346127	3
GarageYrBlt	78	5.346127	97
GarageQual	78	5.346127	4
GarageType	76	5.209047	6
BsmtCond	45	3.084304	4
BsmtExposure	44	3.015764	4
BsmtQual	44	3.015764	4
BsmtFinType1	42	2.878684	6
BsmtFinType2	42	2.878684	6
MasVnrType	16	1.096642	4
MasVnrArea	15	1.028101	303
MSZoning	4	0.274160	5
BsmtHalfBath	2	0.137080	3
Utilities	2	0.137080	1
Functional	2	0.137080	7
BsmtFullBath	2	0.137080	4
BsmtFinSF2	1	0.068540	161
BsmtFinSF1	1	0.068540	669
BsmtUnfSF	1	0.068540	793
TotalBsmtSF	1	0.068540	736
Exterior2nd	1	0.068540	15
SaleType	1	0.068540	9
Exterior1st	1	0.068540	13
KitchenQual	1	0.068540	4
GarageArea	1	0.068540	459
GarageCars	1	0.068540	6
OverallQual	0	0.000000	10

In [33]:
```
df_test['Utilities'].unique()
```

Out[33]:
```
array(['AllPub', nan], dtype=object)
```

- all records mostly "AllPub" for Utilities

- PoolQC,Alley,MiscFeature will be dropped due to large number of missing values
- Utilities has only 1 unique value
- Utility will also be dropped

In [34]:
```
# calculate total number of null values in training data
null_train = df_train.isnull().sum().sum()
print(null_train)

# calculate total number of null values in test data
null_test = df_test.isnull().sum().sum()
print(null_test)
```

```
6950
7000
```

In [35]:
```
# save the 'SalePrice'column as train_label
train_label = df_train['SalePrice'].reset_index(drop=True)

# # drop 'SalePrice' column from df_train
df_train = df_train.drop(['SalePrice'], axis=1)
# # now df_train contains all training features
```

In [36]:
```python
# function to HANDLE the missing data in a dataframe
def missing (df):

    # drop theses columns due to large null values or many same values
    df = df.drop(['Utilities','PoolQC','MiscFeature','Alley'], axis=1)

    # Null value likely means No Fence so fill as "None"
    df["Fence"] = df["Fence"].fillna("None")

    # Null value likely means No Fireplace so fill as "None"
    df["FireplaceQu"] = df["FireplaceQu"].fillna("None")

    # Lot frontage is the feet of street connected to property, which is likely similar to the neighbourhood houses, sc
    df["LotFrontage"] = df["LotFrontage"].fillna(df["LotFrontage"].median())

    # Null value likely means  typical(Typ)
    df["Functional"] = df["Functional"].fillna("Typ")

    # Only one null value so fill as the most frequent value(mode)
    df['KitchenQual'] = df['KitchenQual'].fillna(df['KitchenQual'].mode()[0])

    # Only one null value so fill as the most frequent value(mode)
    df['Electrical'] = df['Electrical'].fillna(df['Electrical'].mode()[0])

    # Very few null value so fill with the most frequent value(mode)
    df['SaleType'] = df['SaleType'].fillna(df['SaleType'].mode()[0])

    # Null value likely means no masonry veneer
    df["MasVnrType"] = df["MasVnrType"].fillna("None") #so fill as "None" (since categorical feature)
    df["MasVnrArea"] = df["MasVnrArea"].fillna(0)      #so fill as o

    # Only one null value so fill as the most frequent value(mode)
    df['Exterior1st'] = df['Exterior1st'].fillna(df['Exterior1st'].mode()[0])
    df['Exterior2nd'] = df['Exterior2nd'].fillna(df['Exterior2nd'].mode()[0])

    #MSZoning is general zoning classification,Very few null value so fill with the most frequent value(mode)
    df['MSZoning'] = df['MSZoning'].fillna(df['MSZoning'].mode()[0])

    #Null value likely means no Identified type of dwelling so fill as "None"
    df['MSSubClass'] = df['MSSubClass'].fillna("None")
```

```python
    # Null value likely means No Garage, so fill as "None" (since these are categorical features)
    for col in ('GarageType', 'GarageFinish', 'GarageQual', 'GarageCond'):
        df[col] = df[col].fillna('None')

    # Null value likely means No Garage and no cars in garage, so fill as 0
    for col in ('GarageYrBlt', 'GarageArea', 'GarageCars'):
        df[col] = df[col].fillna(0)

    # Null value likely means No Basement, so fill as 0
    for col in ('BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF','TotalBsmtSF', 'BsmtFullBath', 'BsmtHalfBath'):
        df[col] = df[col].fillna(0)

    # Null value likely means No Basement, so fill as "None" (since these are categorical features)
    for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'):
        df[col] = df[col].fillna('None')

    return df
```

In [37]:

```python
df_train = missing(df_train)
df_test = missing(df_test)
```

In [38]:

```python
# calculate total number of null values in training data
null_train = df_train.isnull().sum().sum()
print(null_train)

# calculate total number of null values in test data
null_test = df_test.isnull().sum().sum()
print(null_test)
```

```
0
0
```

In [39]:

```python
df_train.shape,df_test.shape
```

Out[39]:

```
((1455, 75), (1459, 75))
```

In [40]:
```
def add_new_cols(df):

    df['Total_SF'] = df['TotalBsmtSF'] + df['1stFlrSF'] + df['2ndFlrSF']

    df['Total_Bathrooms'] = (df['FullBath'] + (0.5 * df['HalfBath']) + df['BsmtFullBath']
                             + (0.5 * df['BsmtHalfBath']))

    df['Total_Porch_SF'] = (df['OpenPorchSF'] + df['3SsnPorch'] + df['EnclosedPorch'] +
                            df['ScreenPorch'] + df['WoodDeckSF'])

    df['Total_Square_Feet'] = (df['BsmtFinSF1'] + df['BsmtFinSF2'] + df['1stFlrSF'] + df['2ndFlrSF'])

    df['Total_Quality'] = df['OverallQual'] + df['OverallCond']

    return df
```

In [41]:
```
# add the new columns
df_train = add_new_cols(df_train)
df_test = add_new_cols(df_test)
```

In [42]:
```
df_train.shape,df_test.shape
```

Out[42]:
```
((1455, 80), (1459, 80))
```

Check data types

In [43]:
```
#training data
g1 = df_train.columns.to_series().groupby(df_train.dtypes).groups
```

In [44]:
```
{k.name: v for k, v in g1.items()}
```

Out[44]:
```
{'int64': Index(['MSSubClass', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt',
        'YearRemodAdd', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF',
        '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
        'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
        'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'GarageArea', 'WoodDeckSF',
        'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea',
        'MiscVal', 'MoSold', 'YrSold', 'Total_SF', 'Total_Porch_SF',
        'Total_Square_Feet', 'Total_Quality'],
       dtype='object'),
 'float64': Index(['LotFrontage', 'MasVnrArea', 'GarageYrBlt', 'Total_Bathrooms'], dtype='object'),
 'object': Index(['MSZoning', 'Street', 'LotShape', 'LandContour', 'LotConfig',
        'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
        'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd',
        'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
        'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Heating',
        'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual', 'Functional',
        'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond',
        'PavedDrive', 'Fence', 'SaleType', 'SaleCondition'],
       dtype='object')}
```

In [45]:
```
#testing data
g2 = df_test.columns.to_series().groupby(df_test.dtypes).groups
```

In [46]:
```
{k.name: v for k, v in g2.items()}
```

Out[46]:
```
{'int64': Index(['MSSubClass', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt',
        'YearRemodAdd', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea',
        'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd',
        'Fireplaces', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
        'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold',
        'Total_Porch_SF', 'Total_Quality'],
       dtype='object'),
 'float64': Index(['LotFrontage', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
        'TotalBsmtSF', 'BsmtFullBath', 'BsmtHalfBath', 'GarageYrBlt',
        'GarageCars', 'GarageArea', 'Total_SF', 'Total_Bathrooms',
        'Total_Square_Feet'],
       dtype='object'),
 'object': Index(['MSZoning', 'Street', 'LotShape', 'LandContour', 'LotConfig',
        'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
        'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd',
        'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
        'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Heating',
        'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual', 'Functional',
        'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond',
        'PavedDrive', 'Fence', 'SaleType', 'SaleCondition'],
       dtype='object')}
```

In [47]:
```
#get dummy values for categorical data
df_train = pd.get_dummies(df_train)
df_test = pd.get_dummies(df_test)

print(df_train.shape)
print(df_test.shape)
```

```
(1455, 292)
(1459, 278)
```

In [48]:
```
#align the training and testing data
df_train, df_test = df_train.align(df_test, join = 'inner', axis=1)
```

In [49]:
```
print(df_train.shape)
print(df_test.shape)
```

```
(1455, 278)
(1459, 278)
```

In [50]:
```
# calculate total number of null values in training data
null_train = df_train.isnull().sum().sum()
print(null_train)

# calculate total number of null values in test data
null_test = df_test.isnull().sum().sum()
print(null_test)
```

```
0
0
```

In [51]:
```
df_train.head(5)
```

Out[51]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...	SaleType_C
0	60	65.0	8450	7	5	2003	2003	196.0	706	0	...	
1	20	80.0	9600	6	8	1976	1976	0.0	978	0	...	
2	60	68.0	11250	7	5	2001	2002	162.0	486	0	...	
3	70	60.0	9550	7	5	1915	1970	0.0	216	0	...	
4	60	84.0	14260	8	5	2000	2000	350.0	655	0	...	

5 rows × 278 columns

In [52]:
```
df_test.head(5)
```

Out[52]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...	SaleType_C
0	20	80.0	11622	5	6	1961	1961	0.0	468.0	144.0	...	
1	20	81.0	14267	6	6	1958	1958	108.0	923.0	0.0	...	
2	60	74.0	13830	5	5	1997	1998	0.0	791.0	0.0	...	
3	60	78.0	9978	6	6	1998	1998	20.0	602.0	0.0	...	
4	120	43.0	5005	8	5	1992	1992	0.0	263.0	0.0	...	

5 rows × 278 columns

In [53]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1455 entries, 0 to 1459
Columns: 278 entries, MSSubClass to SaleCondition_Partial
dtypes: float64(4), int64(37), uint8(237)
memory usage: 854.2 KB
```

In [54]:
```
X_test = df_test            # testing features
```

In [56]:
```
df_train["SalePrice"] = train_label
```

In [58]:
```
df_train.head()
```

Out[58]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...	SaleType_N
0	60	65.0	8450	7	5	2003	2003	196.0	706	0	...	
1	20	80.0	9600	6	8	1976	1976	0.0	978	0	...	
2	60	68.0	11250	7	5	2001	2002	162.0	486	0	...	
3	70	60.0	9550	7	5	1915	1970	0.0	216	0	...	
4	60	84.0	14260	8	5	2000	2000	350.0	655	0	...	

5 rows × 279 columns

In [59]:
```
train_set, valid_set = train_test_split(df_train,train_size= 0.7, shuffle=False)

X_train = train_set.drop(["SalePrice"], axis=1)  # training features
y_train = train_set["SalePrice"].copy()             # training label

X_valid = valid_set.drop(["SalePrice"], axis=1)  # testing features
y_valid = valid_set["SalePrice"].copy()               # testing label
```

In [60]:
```
print("X_train shape: {}".format(X_train.shape))
print("y_train shape: {}".format(y_train.shape))
print()
print("X_valid shape: {}".format(X_valid.shape))
print("y_valid shape: {}".format(y_valid.shape))
print()
print("X_test shape: {}".format(X_test.shape))
```

```
X_train shape: (1018, 278)
y_train shape: (1018,)

X_valid shape: (437, 278)
y_valid shape: (437,)

X_test shape: (1459, 278)
```

Check data type and null values

In [61]:
```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1018 entries, 0 to 1020
Columns: 278 entries, MSSubClass to SaleCondition_Partial
dtypes: float64(4), int64(37), uint8(237)
memory usage: 569.6 KB
```

In [62]:
```
X_valid.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 437 entries, 1021 to 1459
Columns: 278 entries, MSSubClass to SaleCondition_Partial
dtypes: float64(4), int64(37), uint8(237)
memory usage: 244.5 KB
```

In [63]:
```
y_train
```

Out[63]:
```
0       12.247694
1       12.109011
2       12.317167
3       11.849398
4       12.429216
          ...
1016    12.271345
1017    12.078239
1018    12.175613
1019    11.373663
1020    12.160029
Name: SalePrice, Length: 1018, dtype: float64
```

In [64]:
```
y_valid
```

Out[64]:
```
1021    12.567237
1022    11.630709
1023    12.028739
1024    12.588191
1025    11.561716
          ...
1455          NaN
1456          NaN
1457          NaN
1458          NaN
1459          NaN
Name: SalePrice, Length: 437, dtype: float64
```

In [65]:
```
null_t_x = X_train.isnull().sum().sum()
print(null_t_x)

null_t_y = y_train.isnull().sum().sum()
print(null_t_y)
```

```
0
0
```

In [66]:
```
null_v_x = X_valid.isnull().sum().sum()
print(null_v_x)

null_v_y = y_valid.isnull().sum().sum()
print(null_v_y)
```

```
0
5
```

- No null values in X_valid

- There are 5 null values in y_valid

In [67]:
```
np.where(np.isnan(y_valid))
```

Out[67]:
```
(array([432, 433, 434, 435, 436], dtype=int64),)
```

In [68]:
```
# replace null values by mean value of y_valid column
mean = np.nanmean(y_valid)
y_valid = np.nan_to_num(y_valid,nan = mean)
```

In [69]:
```
#check again
np.where(np.isnan(y_valid))
```

Out[69]:
```
(array([], dtype=int64),)
```

In [70]:
```
y_valid.dtype
```

Out[70]:
```
dtype('float64')
```

In [71]:
```
print("Valid data shape:")
print(X_valid.shape, y_valid.shape)
print()
```

```
Valid data shape:
(437, 278) (437,)
```

--- 3. SET CROSS VALIDATION AND RMSE --

Cross Validation

- done to avoid underfitting/overfitting of data and to get a better understanging of how good our models are performing
- split data into k subsets, and train on k-1 of those subset,leaving one for testing
- performing 10-fold cross validation for each model#

In [72]:

```
# calculating cross validation score with scoring set to negative mean absolute error
def cross_validation(model):

    scores = np.sqrt(-cross_val_score(model, X_train, y_train, cv = 12, scoring = "neg_mean_squared_error"))
    mean = np.mean(scores)
    print("Mean CV score: ",mean)
```

RMSE

In [73]:

```
# function to calculate Root mean square error (RMSE)
def rmse(y_pred, y_train):

    rmse_ = np.sqrt(metrics.mean_squared_error(y_pred,y_train))
    print("rmse: ", rmse_)
```

Plot Label

In [74]:
```
# function to plot actual vs predicited label
def actual_vs_pred_plot(y_train,y_pred):

    fig = plt.figure(figsize=(12,12))
    fig, ax = plt.subplots()

    ax.scatter(y_train, y_pred,color = "teal",edgecolor = 'lightblue')
    ax.plot([y_train.min(),y_train.max()], [y_train.min(), y_train.max()], 'k--',lw=0.2)
    ax.set_xlabel('Actual')
    ax.set_ylabel('Predicted')
    plt.suptitle("Actual vs Predicted Scatter Plot",size=14)
    plt.show()
```

-- 4. DATA MODELLING --

MODELS

1. LINEAR REGRESSION MODEL

- Linear Regression is the first model used. In this model, the target value is expected to be a linear combination of the features. The coefficients are set to minimize the residual sum of squares between the target predicted and the observed features

In [139]:
```
reg = linear_model.LinearRegression()
```

In [140]:
```
cross_validation(reg)
```

```
Mean CV score:  0.4752199075347933
```

In [141]:

```
#fit on training
model_reg = reg.fit(X_train, y_train)

#predict value of sale price on the training set
y1_pred = reg.predict(X_train)

#caculate root mean square error
rmse(y1_pred,y_train)
```

```
rmse:  0.3442710335666392
```

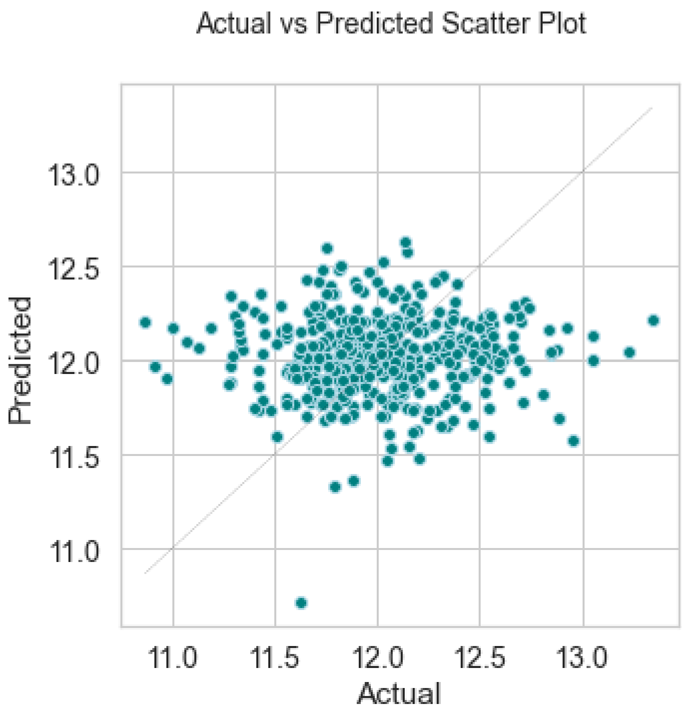
In [142]:

```
#predict value of sale price on the validation set
y1_pred_v = reg.predict(X_valid)

#caculate root mean square error
rmse(y1_pred_v, y_valid)
```

```
rmse:  0.42793480397157035
```

```
In [143]: #plot
actual_vs_pred_plot(y_valid,y1_pred_v)
```

```
<Figure size 864x864 with 0 Axes>
```



2. RIDGE MODEL

- The second model used is Ridge Regression. Ridge Regression is a regularized version of linear regression. The parameter alpha is used to regularize the model. For alpha equal to zero, ridge regression is just a linear regression. RidgeCV model is used to implement ridge regression as it has a built-in cross validation of the alpha parameter. Sixteen different values of alpha between 7e-4 and 20 were used with a 10-fold cross validation. A pipeline using min-max scaler was built to apply to training, validation and testing data.

In [144]:
```
# to find the best value of alphas from this list, i will use RidgeCV
alphas_ = [ 7e-4, 5e-4, 3e-4, 1e-4, 1e-3, 5e-2, 1e-2, 0.1, 0.3, 1, 3, 5, 10, 15, 18, 20]

# use robust scaler as unlike other scalers, the centering and scaling of ro bust scaler
#is based on percentiles and are therefore is not influenced by a few number of very large marginal outliers.

ridge = make_pipeline(MinMaxScaler(), linear_model.RidgeCV(alphas = alphas_, cv = 10))
```

In [145]:
```
cross_validation(ridge)
```

```
Mean CV score:  0.41672707496259215
```

In [146]:
```
#fit
model_ridge = ridge.fit(X_train, y_train)

#predict value of sale price on the training set
y2_pred = ridge.predict(X_train)

#caculate root mean square error
rmse(y2_pred,y_train)
```

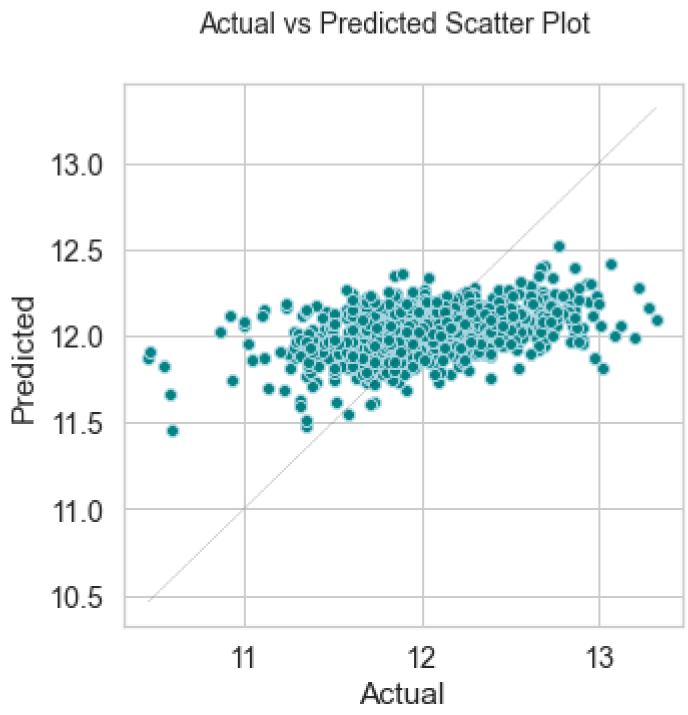
```
rmse:  0.36727237018186476
```

In [147]:
```
#predict value of sale price on the valid set
y2_pred_v = ridge.predict(X_valid)

#caculate root mean square error
rmse(y2_pred_v, y_valid)
```

```
rmse:  0.3957886167433282
```

In [148]:
```
#plot
actual_vs_pred_plot(y_train,y2_pred)
```

<Figure size 864x864 with 0 Axes>



3. LASSO MODEL

- Lasso regression is also a regularized version of linear regression. Lasso regression automatically performs feature selection and can estimates sparse coefficients. LassoCV model was used to implement lasso regression as it has a built-in cross validation of the alpha parameter. Different values of alpha were set with a 10-fold cross validation. Robust scaler was used in a pipeline to scale the training, validation and testing data.

In [149]:
```
# to find the best value of alphas from this list, i will use LassoCV
alpha2 = [0.0001, 0.0002, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008]

#use robust scaler so that predictions are not influenced by a few number of very large marginal outliers

lasso = make_pipeline(RobustScaler(), linear_model.LassoCV(alphas = alpha2, random_state=42,cv=12,max_iter=2000))
```

In [150]:
```
cross_validation(lasso)
```

```
Mean CV score:  0.4297643272515321
```

In [151]:
```
#fit
model_lasso = lasso.fit(X_train, y_train)

#predict value of quality on the training set
y3_pred = lasso.predict(X_train)

#caculate root mean square error
rmse(y3_pred,y_train)
```

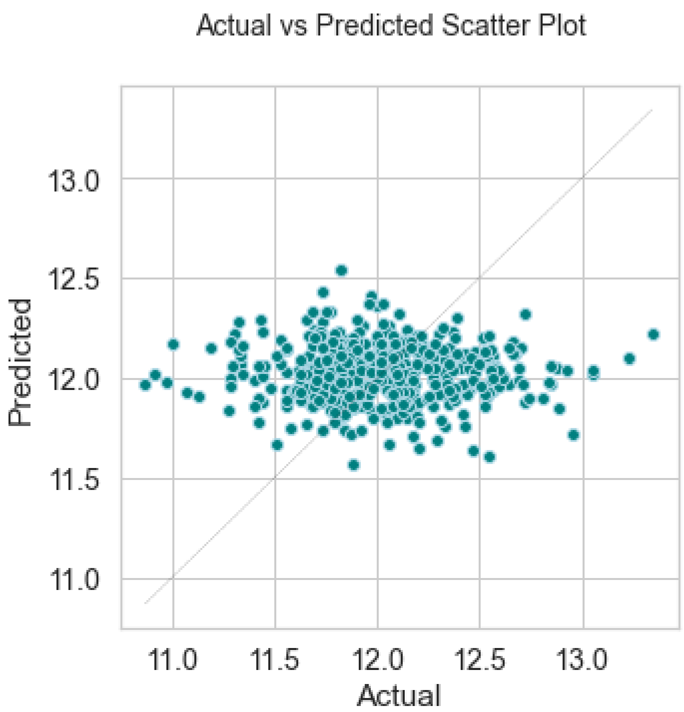
```
rmse:  0.36267996691815335
```

In [152]:
```
#predict value of sale price on the validation set
y3_pred_v = lasso.predict(X_valid)

#caculate root mean square error
rmse(y3_pred_v, y_valid)
```

```
rmse:  0.4059493256188701
```

In [153]:
```
actual_vs_pred_plot(y_valid,y3_pred_v)
```

<Figure size 864x864 with 0 Axes>



4. K-NEAREST NEIGHBOUR REGRESSION MODEL

- K -nearest neighbour regressor is another popular model for regression tasks. It is a simple supervised machine learning model. The numbers of neighbours were set to three different values and the performance of this model was noted. Weights were set to uniform to assign equal weights to all points in each neighbourhood. The algorithm used was set to auto so that the best performing algorithm on the values was used. The leaf size was set to 25.

In [154]:
```
from sklearn.neighbors import KNeighborsRegressor

# N = 5 #
neigh = KNeighborsRegressor(n_neighbors = 5,
                            weights = 'uniform',
                            algorithm = 'auto',
                            leaf_size=25)
neigh.fit(X_train,y_train)

#predict value of sale price on the training set
y4_pred = neigh.predict(X_train)

#caculate root mean square error
rmse(y4_pred,y_train)
```

```
rmse:  0.34885424380933583
```

In [155]:
```
# N = 7 #
neigh1 = KNeighborsRegressor(n_neighbors = 7,
                             weights = 'uniform',
                             leaf_size=25)
neigh1.fit(X_train,y_train)

#predict value of quality on the training set
y_pred = neigh1.predict(X_train)

#caculate root mean square error
rmse(y_pred,y_train)
```

```
rmse:  0.3665712393534244
```

In [156]:
```
# N = 9 #
neigh2 = KNeighborsRegressor(n_neighbors = 9,
                             weights = 'uniform',
                             leaf_size=25)
neigh2.fit(X_train,y_train)

#predict value of quality on the training set
y_pred = neigh2.predict(X_train)

#caculate root mean square error
rmse(y_pred,y_train)
```

```
rmse:  0.37262338937265044
```

In [157]:
```
# N=5 performs best
```

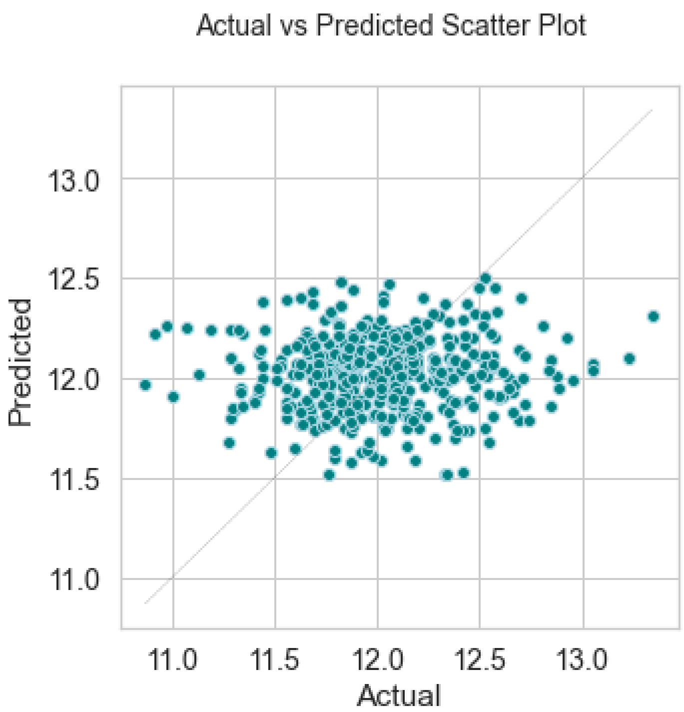
In [158]:
```
#predict value of sale price on the validation set
y4_pred_v = neigh.predict(X_valid)

#caculate root mean square error
rmse(y4_pred_v, y_valid)
```

```
rmse:  0.41351487769327555
```

Note: rmse increases when values of k(no. of neighbours) increase

In [159]:
```
actual_vs_pred_plot(y_valid,y4_pred_v)
```

```
<Figure size 864x864 with 0 Axes>
```



5. DECISION TREE MODEL

- Decision tree model is also used to fit this data as it does not require much data cleaning and is not influenced by outliers. Decision trees can, unlike linear models, fit linearly inseparable datasets. The values of minimum leaves were set between 1 to 9 because a very small number of minimum leaves can cause overfitting whereas a large number of minimum leaves will prevent the tree from learning. Maximum depth of 7 and 9 were used to fit the data for predictions.

In [113]:
```
from sklearn import tree
```

In [114]:
```
# set max depth to 5
tree_regr1 = tree.DecisionTreeRegressor(max_depth = 7, min_samples_leaf=5,random_state=42)

# set max depth to 9
tree_regr2 = tree.DecisionTreeRegressor(max_depth = 9,min_samples_leaf=9,random_state=42)

#fit the traning data to a decision tree model
tree_regr11 = tree_regr1.fit(X_train,y_train)
tree_regr12 = tree_regr2.fit(X_train,y_train)

#predict value of sale price on the training set
y1 = tree_regr1.predict(X_train)
y2 = tree_regr2.predict(X_train)
```

In [115]:
```
cross_validation(tree_regr1)
cross_validation(tree_regr2)
```

```
Mean CV score:  0.4440722344760503
Mean CV score:  0.45825272349446555
```

In [116]:
```
#caculate root mean square error
rmse(y1,y_train)
```

```
rmse:  0.3238501847516405
```

In [117]:
```
rmse(y2,y_train)
```

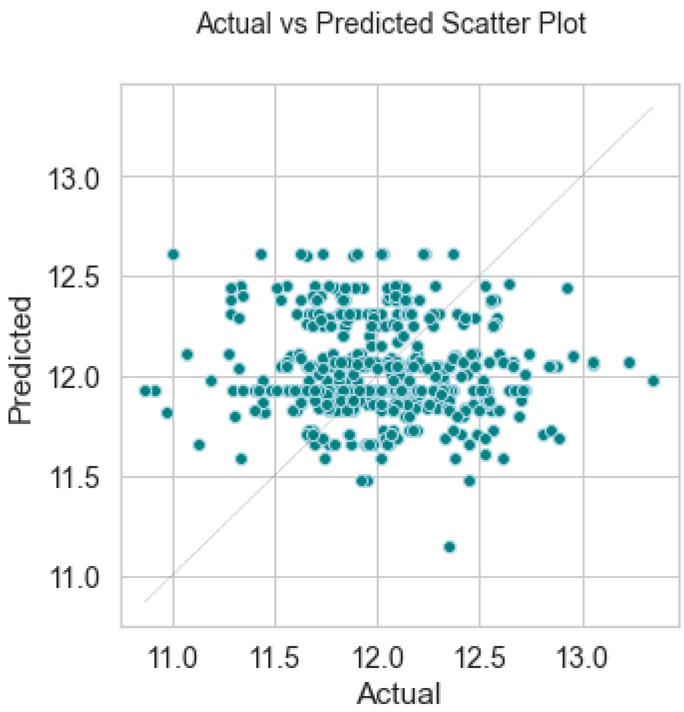
```
rmse:  0.319434991726199
```

In [118]:
```
#predict value of sale price on the validation set
y5_pred_v = tree_regr2.predict(X_valid)

#caculate root mean square error
rmse(y5_pred_v, y_valid)
```

```
rmse:  0.4583579345988703
```

In [119]:
```
#plot
actual_vs_pred_plot(y_valid,y5_pred_v)
```

<Figure size 864x864 with 0 Axes>



6. Random Forest MODEL

- Random forest model is an ensemble method based on randomized decision trees. Grid search was used to select the best parameters with a 5-fold cross validation. The number of trees in the forest was set to 200 with a maximum depth of 5 and 3 minimum leaves.

In [134]:
```
rforest = RandomForestRegressor(n_estimators=200,max_depth=13,random_state=42)
```

In [146]:
```
# grid search to find best value of C, gamma and epsilon
param_grid  = {'n_estimators': [100,150,200,250,300,350,400],
               'max_depth': [5,7,9,11,13,15,17],
               'min_samples_leaf': [3,5,7,9,11,13,15]}

# set cross validation to 5
clf = GridSearchCV(rforest, param_grid, cv = 5, n_jobs = -2)
clf.fit(X_train,y_train)
```

Out[146]:
```
GridSearchCV(cv=5,
             estimator=RandomForestRegressor(max_depth=7, min_samples_leaf=5,
                                             n_estimators=250,
                                             random_state=42),
             n_jobs=-2,
             param_grid={'max_depth': [5, 7, 9, 11, 13, 15, 17],
                         'min_samples_leaf': [3, 5, 7, 9, 11, 13, 15],
                         'n_estimators': [100, 150, 200, 250, 300, 350, 400]})
```

In [147]:
```
clf.best_params_
```

Out[147]:
```
{'max_depth': 5, 'min_samples_leaf': 3, 'n_estimators': 200}
```

In [135]:
```
rforest = RandomForestRegressor(n_estimators=, max_depth=5, min_samples_leaf=3, random_state=42)
```

In [155]:
```
cross_validation(rforest)
```

```
Mean CV score:  0.403804172243945
```

In [156]:
```
#fit
model_rforest = rforest.fit(X_train, y_train)

#predict value of sale price on the training set
y6_pred = rforest.predict(X_train)

#caculate root mean square error
rmse(y6_pred,y_train)
```

```
rmse:  0.3494772667130121
```

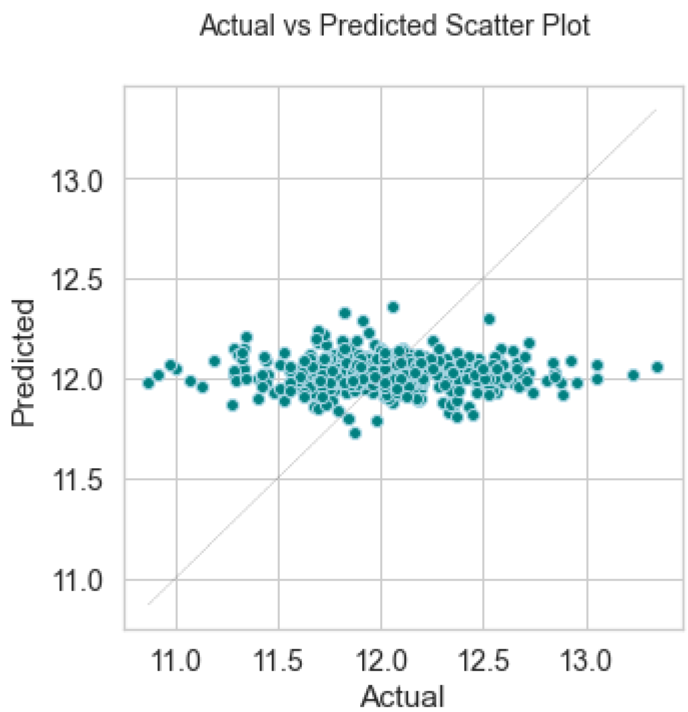
In [157]:
```
#predict value of sale price on the validation set
y6_pred_v = rforest.predict(X_valid)

#caculate root mean square error
rmse(y6_pred_v, y_valid)
```

```
rmse:  0.38616747296757176
```

In [158]:
```
#0: 0.38852359192540425
#1: 0.38616747296757176
```

In [159]:
```
#plot
actual_vs_pred_plot(y_valid, y6_pred_v)
```

```
<Figure size 864x864 with 0 Axes>
```



7. Support Vector Regressor MODEL

- Support vector regressor is another powerful model. It is memory efficient and offers different kernels to choose from. Grid search was used to find the best value of the hyperparameters C, gamma and epsilon. The sigmoid kernel was used along with the default value of epsilon.

In [75]:
```
svr_basic = SVR(C = 10, gamma = 0.001)
```

In [114]:
```
# grid search to find best value of C, gamma and epsilon and default kernel 'rbf'
param_grid  = {'C': [5,7,10,15,20,30],'gamma': [0.001, 0.0001, 0.0011, 0.00011], 'epsilon': [0.1, 0.01, 0.001, 0.005, 0

# set cross validation to 5
clf = GridSearchCV(svr_basic, param_grid, cv = 10, n_jobs = -2)
clf.fit(X_train,y_train)
```

Out[114]:
```
GridSearchCV(cv=10, estimator=SVR(C=10, gamma=0.001), n_jobs=-2,
             param_grid={'C': [5, 7, 10, 15, 20, 30],
                         'epsilon': [0.1, 0.01, 0.001, 0.005, 0.007, 0.008,
                                     0.009],
                         'gamma': [0.001, 0.0001, 0.0011, 0.00011]})
```

In [115]:
```
clf.best_params_
```

Out[115]:
```
{'C': 5, 'epsilon': 0.1, 'gamma': 0.0011}
```

In [116]:
```
#make final SVR model with best parameters found from grid search
svr = make_pipeline(MinMaxScaler(), SVR(C= 5, epsilon= 0.1, gamma=0.0011, kernel = "sigmoid"))
```

In [117]:
```
cross_validation(svr)
```

```
Mean CV score:  0.40963206887105647
```

In [118]:

```
#fit
model_svr = svr.fit(X_train, y_train)

#predict value of sale price on the training set
y7_pred = svr.predict(X_train)

#caculate root mean square error
rmse(y7_pred,y_train)
```

```
rmse:  0.38245878515315423
```

In [119]:

```
#predict value of sale price on the validation set
y7_pred_v = svr.predict(X_valid)

#caculate root mean square error
rmse(y7_pred_v, y_valid)
```

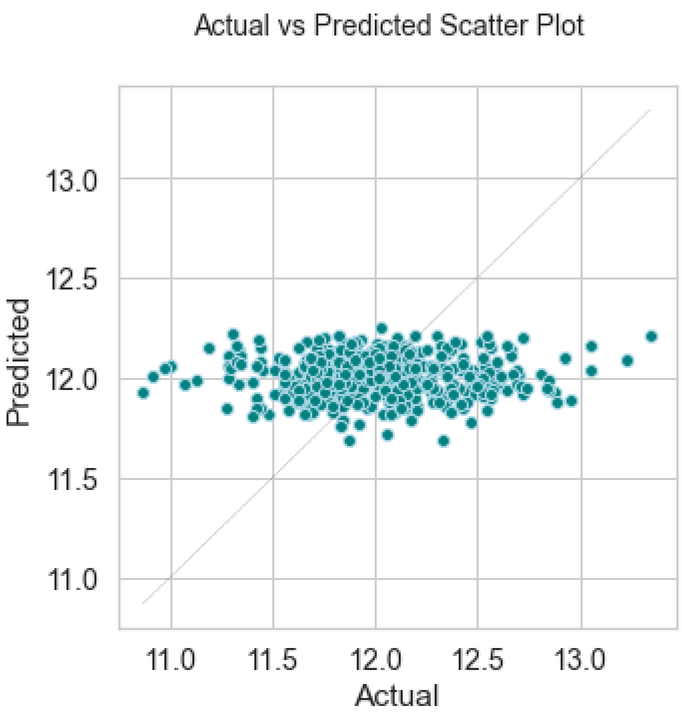
```
rmse:  0.3900469727418305
```

In [113]:

```
# Linear - 0.4338387095039476
# Sigmoid - 0.3900469727418305
# With sigmoid as default kernel - 0.396705456249O4924
# rbf - 0.39420253052849114
```

In [120]:
```
actual_vs_pred_plot(y_valid, y7_pred_v)
```

```
<Figure size 864x864 with 0 Axes>
```



8. Gradient Boosting Regressor MODEL

- Gradient boosting regression is an ensemble of weak prediction models. Two gradient boosting models with different depths were evaluated. The loss was set to ‘huber’ which is a combination of least square regression and a highly robust loss function.

In [121]:
```
# set max depth to 4, min_samples_leaf to 15
gbr1 = GradientBoostingRegressor(n_estimators=200, learning_rate=0.05, max_depth = 7,
                                 min_samples_leaf=7, loss='huber', random_state =42)
```

In [122]:
```
# set max depth to 7, min_samples_leaf to 10
gbr2 = GradientBoostingRegressor(n_estimators=200, learning_rate=0.05, max_depth = 9,
                                min_samples_leaf=10, loss='huber', random_state =42)
```

In [123]:
```
cross_validation(gbr1)
cross_validation(gbr2)
```

```
Mean CV score:  0.4287974195276836
Mean CV score:  0.4292489907640939
```

In [124]:
```
#fit
model_gbr1 = gbr1.fit(X_train, y_train)
model_gbr2 = gbr2.fit(X_train, y_train)

#predict value of sale price on the training set
y_g1_pred = gbr1.predict(X_train)
y_g2_pred = gbr2.predict(X_train)

#caculate root mean square error
rmse(y_g1_pred,y_train)
rmse(y_g2_pred,y_train)
```

```
rmse:  0.15045439854847656
rmse:  0.13917493901563793
```

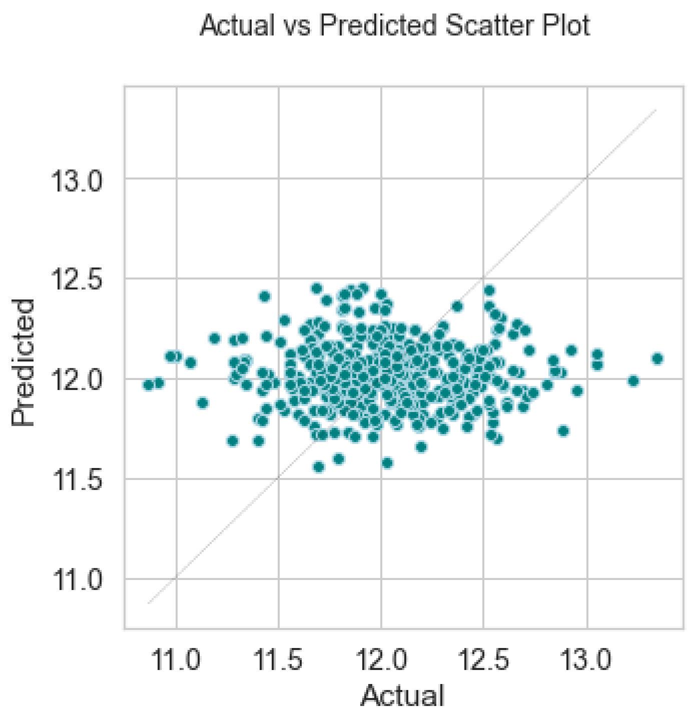
- model gbr2 performs best

In [125]:
```
#predict value of sale price on the validation set
y8_pred_v = gbr2.predict(X_valid)

#caculate root mean square error
rmse(y8_pred_v, y_valid)
```

```
rmse:  0.4118219430457788
```

In [126]:
```
# plot for gbr2
actual_vs_pred_plot(y_valid, y8_pred_v)
```

<Figure size 864x864 with 0 Axes>



9. STACKED REGRESSOR MODEL

- The final model used is the stacked regressor model. Stacking allows the power of each individual estimator to be used by using their output as a final estimator input. Random forest, Support vector regressor, K -nearest neighbour regressor and ridge regressor were stacked with random forest as the final estimator.

In [160]:
```
# using Random Forest,Support Vector Regressor and Gradient Boosting to build a stack model because they have lower RMS
estimators = [('Random Forest', rforest),
              ("Support Vector Regressor",svr),
              ("K",neigh),
              ("Ridge",ridge)
              ]
```

In [161]:
```
stacked = StackingRegressor(estimators = estimators, final_estimator = rforest, cv=5)
```

In [162]:
```
cross_validation(stacked)
```

```
Mean CV score:  0.4093903027876036
```

In [163]:
```
#fit
model_stack = stacked.fit(X_train, y_train)

#predict value of sale price on the training set
y9_pred = stacked.predict(X_train)

#caculate root mean square error
rmse(y9_pred,y_train)
```

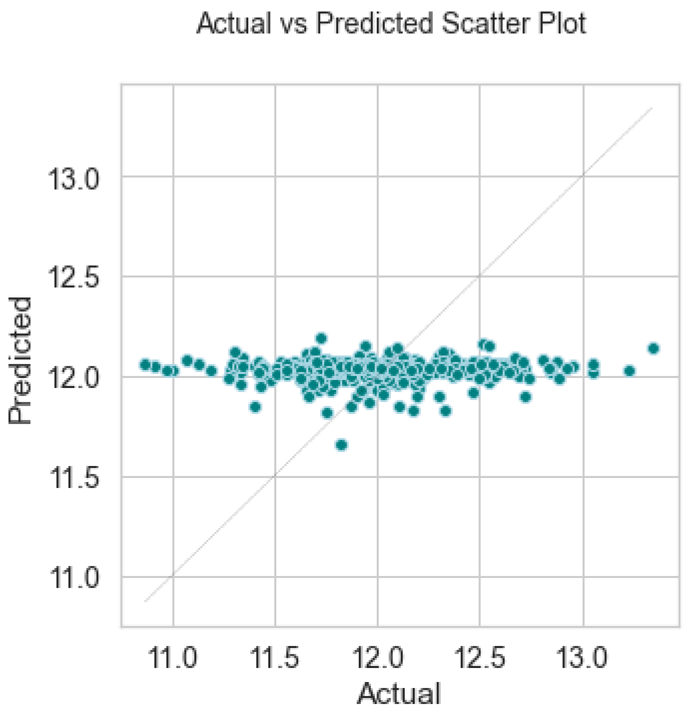
```
rmse:  0.40380119116088625
```

In [164]:
```
#predict value of sale price on the validation set
y9_pred_v = stacked.predict(X_valid)

#caculate root mean square error
rmse(y9_pred_v, y_valid)
```

```
rmse:  0.3769718491202983
```

In [166]:
```
# plot
actual_vs_pred_plot(y_valid,y9_pred_v)
```

```
<Figure size 864x864 with 0 Axes>
```



Observations

RMSE:

- linear reg : 0.42793480397157035
- ridge : 0.3957886167433282
- lasso : 0.4059493256188701
- k-nearest neighbour(k=5) : 0.41351487769327555
- decision tree(maxdepth=9) : 0.4583579345988703
- random forest : 0.38616747296757176

- Support Vector Regressor : 0.3900469727418305
- Gradient Boosting Regressor : 0.4118219430457788
- Stacked Regressor model : 0.3769718491202983

How errors compare:

- The lowest error is of : Stacked Regressor model
- The largest error is of : decision tree(maxdepth=9)
- Therefore Stacked Regressor model will be applied to the test data as it is the best performing model

-- 5. TEST DATA PREDICTION ---

In [ ]:
```
csv_path = "sample_submission.csv"
df_sub = pd.read_csv(csv_path, sep = ',')
```

In [168]:
```
df_sub.shape
```

In [169]:
```
df_sub.head()
```

In [170]:
```
X_test.shape
```

In [171]:
```
#predict value of sale price on the training set
y_final_pred = stacked.predict(X_test)

y_final_pred
```

Out[171]:
```
array([11.95412697, 12.24090799, 12.02787313, ..., 12.11104746,
       11.80525731, 12.00073532])
```

In [172]:
```python
#undo the log tranformation to get predictions in terms of original label
predictions = np.expm1(y_final_pred)
print(predictions)
```

```
[155456.39222118 207088.84134974 167354.09213307 ... 181869.0069438
 133953.74714746 162873.51239215]
```

In [173]:
```python
submit = pd.DataFrame()
submit['Id'] = test_ID
submit['SalePrice'] = predictions
submit.to_csv('submission.csv',index=False)
```

In [174]:
```python
submit
```

Out[174]:

	Id	SalePrice
0	1461	155456.392221
1	1462	207088.841350
2	1463	167354.092133
3	1464	181226.365422
4	1465	157800.657188
...	...	...
1454	2915	170084.706175
1455	2916	166670.224702
1456	2917	181869.006944
1457	2918	133953.747147
1458	2919	162873.512392

1459 rows × 2 columns