Importing Libraries

In [1]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
te = pd.read_csv("test.csv")
```

Basic Information of the "Test" Dataset

In [3]:
```
te.head()
```

Out[3]:

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

5 rows × 80 columns

In [4]:
```
te.tail()
```

Out[4]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	ScreenP
1454	2915	160	RM	21.0	1936	Pave	NaN	Reg	Lvl	AllPub	...	
1455	2916	160	RM	21.0	1894	Pave	NaN	Reg	Lvl	AllPub	...	
1456	2917	20	RL	160.0	20000	Pave	NaN	Reg	Lvl	AllPub	...	
1457	2918	85	RL	62.0	10441	Pave	NaN	Reg	Lvl	AllPub	...	
1458	2919	60	RL	74.0	9627	Pave	NaN	Reg	Lvl	AllPub	...	

5 rows × 80 columns

In [5]:
```
te.describe()
```

Out[5]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	M
count	1459.000000	1459.000000	1232.000000	1459.000000	1459.000000	1459.000000	1459.000000	1459.000000	1
mean	2190.000000	57.378341	68.580357	9819.161069	6.078821	5.553804	1971.357779	1983.662783	
std	421.321334	42.746880	22.376841	4955.517327	1.436812	1.113740	30.390071	21.130467	
min	1461.000000	20.000000	21.000000	1470.000000	1.000000	1.000000	1879.000000	1950.000000	
25%	1825.500000	20.000000	58.000000	7391.000000	5.000000	5.000000	1953.000000	1963.000000	
50%	2190.000000	50.000000	67.000000	9399.000000	6.000000	5.000000	1973.000000	1992.000000	
75%	2554.500000	70.000000	80.000000	11517.500000	7.000000	6.000000	2001.000000	2004.000000	
max	2919.000000	190.000000	200.000000	56600.000000	10.000000	9.000000	2010.000000	2010.000000	1

8 rows × 37 columns

In [6]:
```
te.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
```

In [7]:
```
te.columns
```

Out[7]:
```
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'],
      dtype='object')
```

In [8]:
```
te.shape
```

Out[8]:
```
(1459, 80)
```

Null Value Analysis

In [9]:
```
te.isnull().sum()
```

Out[9]:
```
Id                0
MSSubClass        0
MSZoning          4
LotFrontage     227
LotArea           0
                 ...
MiscVal           0
MoSold            0
YrSold            0
SaleType          1
SaleCondition     0
Length: 80, dtype: int64
```

In [10]:
```
te = te.drop(["MSZoning", "Street", "Alley", "LotShape", "LandContour", "Utilities", "PoolQC", "F
              "SaleType", "SaleCondition", "LotConfig", "LandSlope", "Neighborhood", "Condition1"
              "BldgType", "HouseStyle", "RoofStyle", "PavedDrive", "RoofMatl", "Exterior1st", "Ga
              "MasVnrType", "ExterQual", "ExterCond", "Foundation", "BsmtQual", "BsmtCond", "Bsmt
              "BsmtFinType1", "BsmtFinType2", "Heating", "HeatingQC", "CentralAir", "Electrical",
              "Functional", "FireplaceQu", "GarageType", "GarageFinish", "GarageQual", "Id", "Yea
              "YrSold"], axis=1)
```

In [11]:
```
te.head()
```

Out[11]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	To
0	20	80.0	11622	5	6	0.0	468.0	144.0	270.0	
1	20	81.0	14267	6	6	108.0	923.0	0.0	406.0	
2	60	74.0	13830	5	5	0.0	791.0	0.0	137.0	
3	60	78.0	9978	6	6	20.0	602.0	0.0	324.0	
4	120	43.0	5005	8	5	0.0	263.0	0.0	1017.0	

5 rows × 33 columns

Treatment for Null Values

In [12]:
```
te1= te.copy()
```

In [13]:
```
te1.head()
```

Out[13]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	To
0	20	80.0	11622	5	6	0.0	468.0	144.0	270.0	
1	20	81.0	14267	6	6	108.0	923.0	0.0	406.0	
2	60	74.0	13830	5	5	0.0	791.0	0.0	137.0	
3	60	78.0	9978	6	6	20.0	602.0	0.0	324.0	
4	120	43.0	5005	8	5	0.0	263.0	0.0	1017.0	

5 rows × 33 columns

In [14]:
```
te1.isnull().sum()
```

Out[14]:
```
MSSubClass         0
LotFrontage      227
LotArea            0
OverallQual        0
OverallCond        0
MasVnrArea        15
BsmtFinSF1         1
BsmtFinSF2         1
BsmtUnfSF          1
TotalBsmtSF        1
1stFlrSF           0
2ndFlrSF           0
LowQualFinSF       0
GrLivArea          0
BsmtFullBath       2
BsmtHalfBath       2
FullBath           0
HalfBath           0
BedroomAbvGr       0
KitchenAbvGr       0
TotRmsAbvGrd       0
Fireplaces         0
GarageYrBlt       78
GarageCars         1
GarageArea         1
WoodDeckSF         0
OpenPorchSF        0
EnclosedPorch      0
3SsnPorch          0
ScreenPorch        0
PoolArea           0
MiscVal            0
MoSold             0
dtype: int64
```

In [15]:
```
te1["LotFrontage"].mean()
```

Out[15]: 68.58035714285714

In [16]:
```
te1["LotFrontage"]=te["LotFrontage"].fillna(te["LotFrontage"].mean())
```

In [17]:
```
te1["MasVnrArea"].mean()
```

Out[17]: 100.70914127423822

In [18]:
```
te1["MasVnrArea"]=te["MasVnrArea"].fillna(te["MasVnrArea"].mean())
```

In [19]:
```
te1["BsmtFinSF1"].mean()
```

Out[19]: 439.2037037037037

In [20]:
```
te1["BsmtFinSF1"]=te["BsmtFinSF1"].fillna(te["BsmtFinSF1"].mean())
```

In [21]:
```
te1["BsmtFinSF2"].mean()
```

Out[21]: 52.61934156378601

In [22]:
```
te1["BsmtFinSF2"]=te["BsmtFinSF2"].fillna(te["BsmtFinSF2"].mean())
```

In [23]:
```
te1["BsmtUnfSF"].mean()
```

Out[23]: 554.2949245541838

In [24]:
```
te1["BsmtUnfSF"]=te["BsmtUnfSF"].fillna(te["BsmtUnfSF"].mean())
```

In [25]:
```
te1["TotalBsmtSF"].mean()
```

Out[25]: 1046.1179698216736

In [26]:
```
te1["TotalBsmtSF"]=te["TotalBsmtSF"].fillna(te["TotalBsmtSF"].mean())
```

In [27]:
```
te1["BsmtFullBath"].mean()
```

Out[27]: 0.4344543582704187

In [28]:
```
te1["BsmtFullBath"]=te["BsmtFullBath"].fillna(te["BsmtFullBath"].mean())
```

In [29]:
```
te1["BsmtFullBath"].mean()
```

Out[29]: 0.4344543582704186

In [30]:
```
te1["BsmtFullBath"]=te["BsmtFullBath"].fillna(te["BsmtFullBath"].mean())
```

In [31]:
```
te1["BsmtHalfBath"].mean()
```

Out[31]: 0.06520247083047358

In [32]:
```
te1["BsmtHalfBath"]=te["BsmtHalfBath"].fillna(te["BsmtHalfBath"].mean())
```

In [33]:
```
te1["GarageYrBlt"].mean()
```

Out[33]: 1977.7212165097756

In [34]:
```
te1["GarageYrBlt"]=te["GarageYrBlt"].fillna(te["GarageYrBlt"].mean())
```

In [35]:
```
te1["GarageCars"].mean()
```

Out[35]: 1.7661179698216736

In [36]:
```
te1["GarageCars"]=te["GarageCars"].fillna(te["GarageCars"].mean())
```

In [37]:
```
te1["GarageArea"].mean()
```

Out[37]:
```
472.76886145404666
```

In [38]:
```
te1["GarageArea"]=te["GarageArea"].fillna(te["GarageArea"].mean())
```

In [39]:
```
te1.isnull().sum()
```

Out[39]:
```
MSSubClass       0
LotFrontage      0
LotArea          0
OverallQual      0
OverallCond      0
MasVnrArea       0
BsmtFinSF1       0
BsmtFinSF2       0
BsmtUnfSF        0
TotalBsmtSF      0
1stFlrSF         0
2ndFlrSF         0
LowQualFinSF     0
GrLivArea        0
BsmtFullBath     0
BsmtHalfBath     0
FullBath         0
HalfBath         0
BedroomAbvGr     0
KitchenAbvGr     0
TotRmsAbvGrd     0
Fireplaces       0
GarageYrBlt      0
GarageCars       0
GarageArea       0
WoodDeckSF       0
OpenPorchSF      0
EnclosedPorch    0
3SsnPorch        0
ScreenPorch      0
PoolArea         0
MiscVal          0
MoSold           0
dtype: int64
```

Exploratory Analysis

In [40]:
```
# Analysis based on Numbers of Houses sold based on Quality
```

In [41]:
```
te1.head()
```

Out[41]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	Tc
0	20	80.0	11622	5	6	0.0	468.0	144.0	270.0	
1	20	81.0	14267	6	6	108.0	923.0	0.0	406.0	
2	60	74.0	13830	5	5	0.0	791.0	0.0	137.0	
3	60	78.0	9978	6	6	20.0	602.0	0.0	324.0	
4	120	43.0	5005	8	5	0.0	263.0	0.0	1017.0	

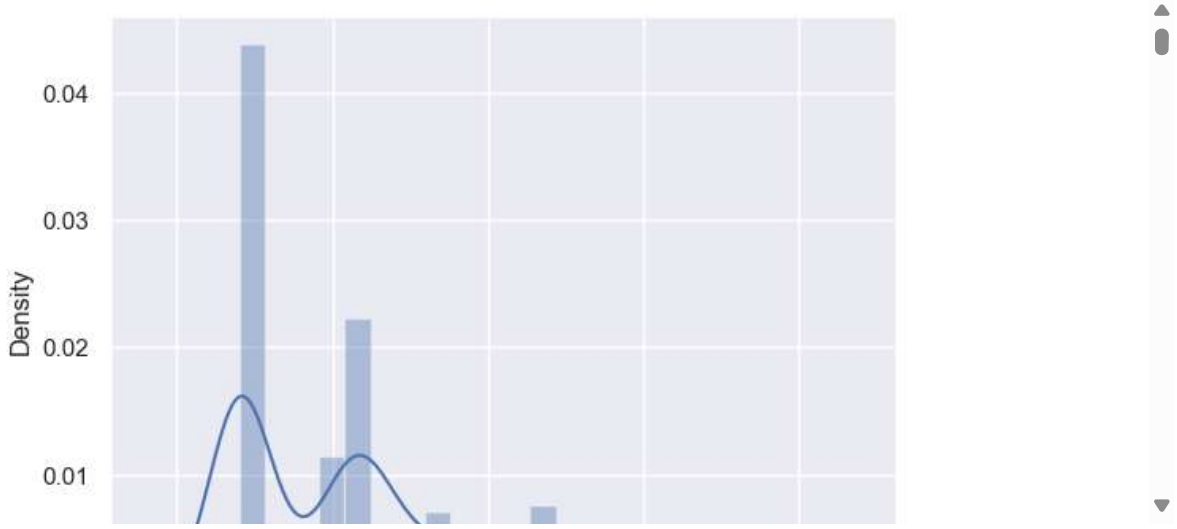
5 rows × 33 columns

In [42]:
```
te1.duplicated().sum()
```

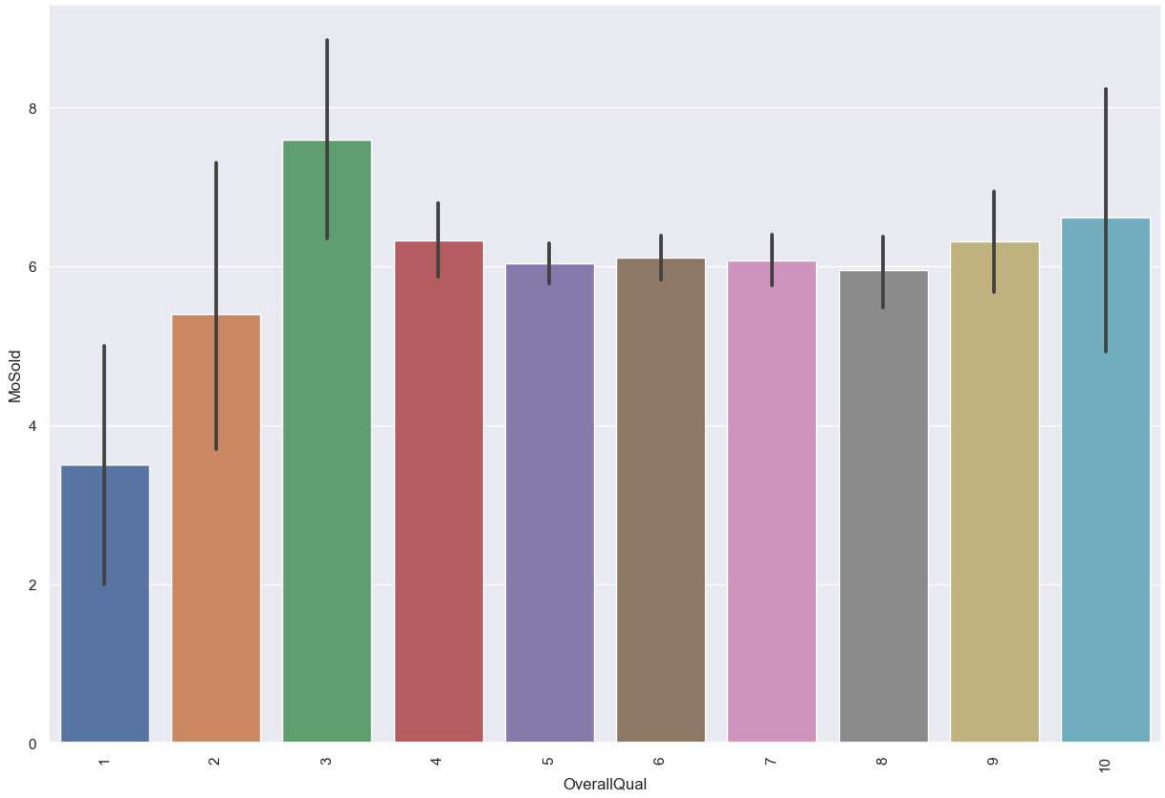
Out[42]:
```
0
```

In [43]:
```
def distplots(col):
    sns.distplot(te1[col])
    plt.show()

for i in list(te1)[0:]:
    distplots(i)
```



In [44]:
```
plt.figure(figsize=(15,10), dpi=100)
sns.barplot (x='OverallQual', y='MoSold', data=te1)
plt.xticks(rotation=90)
plt.show()
```

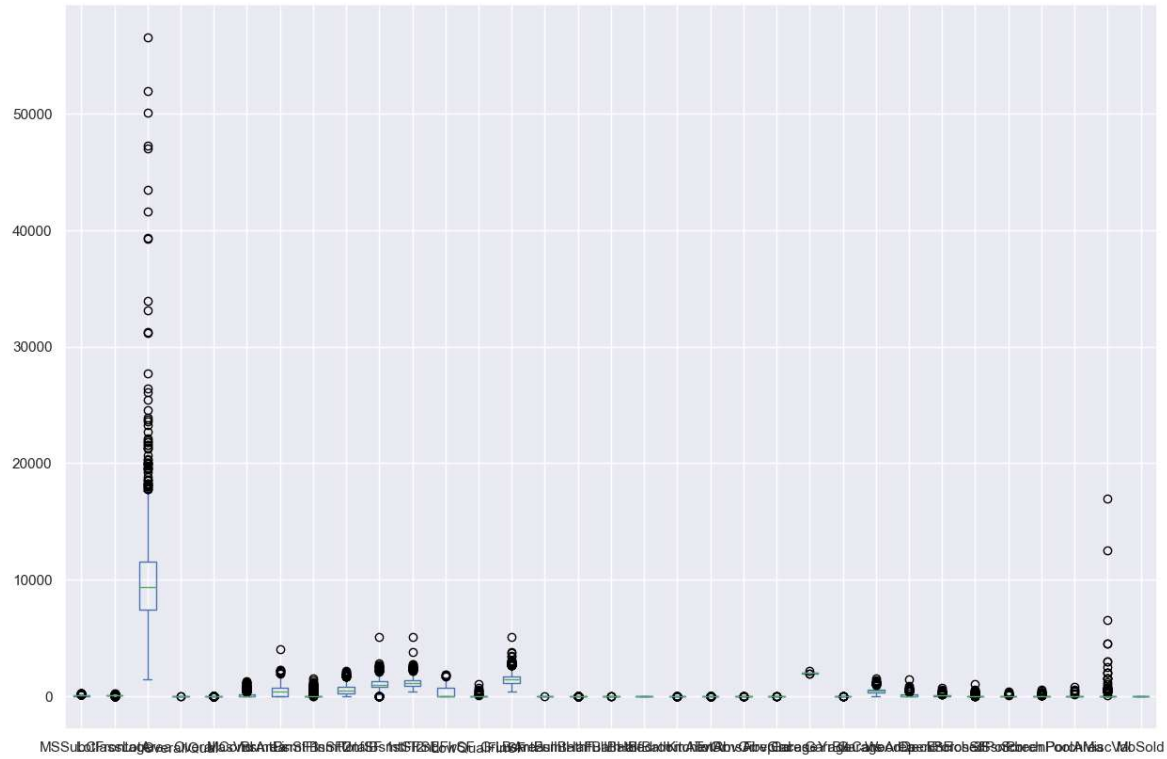


Outlier Analysis

In [45]:
```
#The Graph Before Outler Treatment
```

In [46]:
```
te1.plot(kind='box', figsize=(15,10))
```

Out[46]: <Axes: >



Treatment of Outliers using Capping Method

In [47]:
```
te2=te1.copy()
```

In [48]:
```
te2.columns
```

Out[48]:
```
Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
       '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'],
      dtype='object')
```

In [49]:
```
num_list=[]
for i in te2:
    if te2[i].dtype!=object:
        num_list.append(i)
```

In [50]:
```
num_list
```

Out[50]:
```
['MSSubClass',
 'LotFrontage',
 'LotArea',
 'OverallQual',
 'OverallCond',
 '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']
```

In [51]:
```
cat_list=[]
for i in te2:
    if te2[i].dtype==object:
        cat_list.append(i)
```

In [52]:
```
cat_list
```

Out[52]:
```
[]
```

In [53]:
```
te2 = te1.drop(['BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', '
                'BsmtFullBath', 'BsmtHalfBath', 'FullBath','HalfBath', 'BedroomAbvGr', 'KitchenAb
                'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF','OpenPorchS
                '3SsnPorch', 'ScreenPorch', 'PoolArea','MiscVal', 'MoSold'], axis=1)
```

In [54]:
```
te2.head()
```

Out[54]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea
0	20	80.0	11622	5	6	0.0
1	20	81.0	14267	6	6	108.0
2	60	74.0	13830	5	5	0.0
3	60	78.0	9978	6	6	20.0
4	120	43.0	5005	8	5	0.0

In [55]:
```
MSSubClass_q1 = te2['MSSubClass'].quantile(0.25)
MSSubClass_q3 = te2['MSSubClass'].quantile(0.75)
MSSubClass_iqr = MSSubClass_q3 -  MSSubClass_q1
MSSubClass_upper = MSSubClass_q3 + 1.5 * MSSubClass_iqr
MSSubClass_lower = MSSubClass_q1 - 1.5 * MSSubClass_iqr
```

In [56]:
```
te2['MSSubClass'] = np.where(te2['MSSubClass'] > MSSubClass_upper, MSSubClass_upper,
                             np.where(te2['MSSubClass'] < MSSubClass_lower, MSSubClass_lower,
                                      te2['MSSubClass']) )
```

In [57]:
```
LotFrontage_q1 = te2['LotFrontage'].quantile(0.25)
LotFrontage_q3 = te2['LotFrontage'].quantile(0.75)
LotFrontage_iqr = LotFrontage_q3 -  LotFrontage_q1
LotFrontage_upper = LotFrontage_q3 + 1.5 * LotFrontage_iqr
LotFrontage_lower = LotFrontage_q1 - 1.5 * LotFrontage_iqr
```

In [58]:
```
te2['LotFrontage'] = np.where(te2['LotFrontage'] > LotFrontage_upper, LotFrontage_upper,
                              np.where(te2['LotFrontage'] < LotFrontage_lower, LotFrontage_lower,
                                       te2['LotFrontage']) )
```

In [59]:
```
LotArea_q1 = te2['LotArea'].quantile(0.25)
LotArea_q3 = te2['LotArea'].quantile(0.75)
LotArea_iqr = LotArea_q3 -  LotArea_q1
LotArea_upper = LotArea_q3 + 1.5 * LotArea_iqr
LotArea_lower = LotArea_q1 - 1.5 * LotArea_iqr
```

In [60]:
```
te2['LotArea'] = np.where(te2['LotArea'] > LotArea_upper, LotArea_upper,
                      np.where(te2['LotArea'] < LotArea_lower, LotArea_lower,
                               te2['LotArea']) )
```

In [61]:
```
OverallQual_q1 = te2['OverallQual'].quantile(0.25)
OverallQual_q3 = te2['OverallQual'].quantile(0.75)
OverallQual_iqr = OverallQual_q3 -  OverallQual_q1
OverallQual_upper = OverallQual_q3 + 1.5 * OverallQual_iqr
OverallQual_lower = OverallQual_q1 - 1.5 * OverallQual_iqr
```

In [62]:
```
te2['OverallQual'] = np.where(te2['OverallQual'] > OverallQual_upper, OverallQual_upper,
                      np.where(te2['OverallQual'] < OverallQual_lower, OverallQual_lower,
                               te2['OverallQual']) )
```

In [63]:
```
OverallCond_q1 = te2['OverallCond'].quantile(0.25)
OverallCond_q3 = te2['OverallCond'].quantile(0.75)
OverallCond_iqr = OverallCond_q3 -  OverallCond_q1
OverallCond_upper = OverallCond_q3 + 1.5 * OverallCond_iqr
OverallCond_lower = OverallCond_q1 - 1.5 * OverallCond_iqr
```

In [64]:
```
te2['OverallCond'] = np.where(te2['OverallCond'] > OverallCond_upper, OverallCond_upper,
                      np.where(te2['OverallCond'] < OverallCond_lower, OverallCond_lower,
                               te2['OverallCond']) )
```

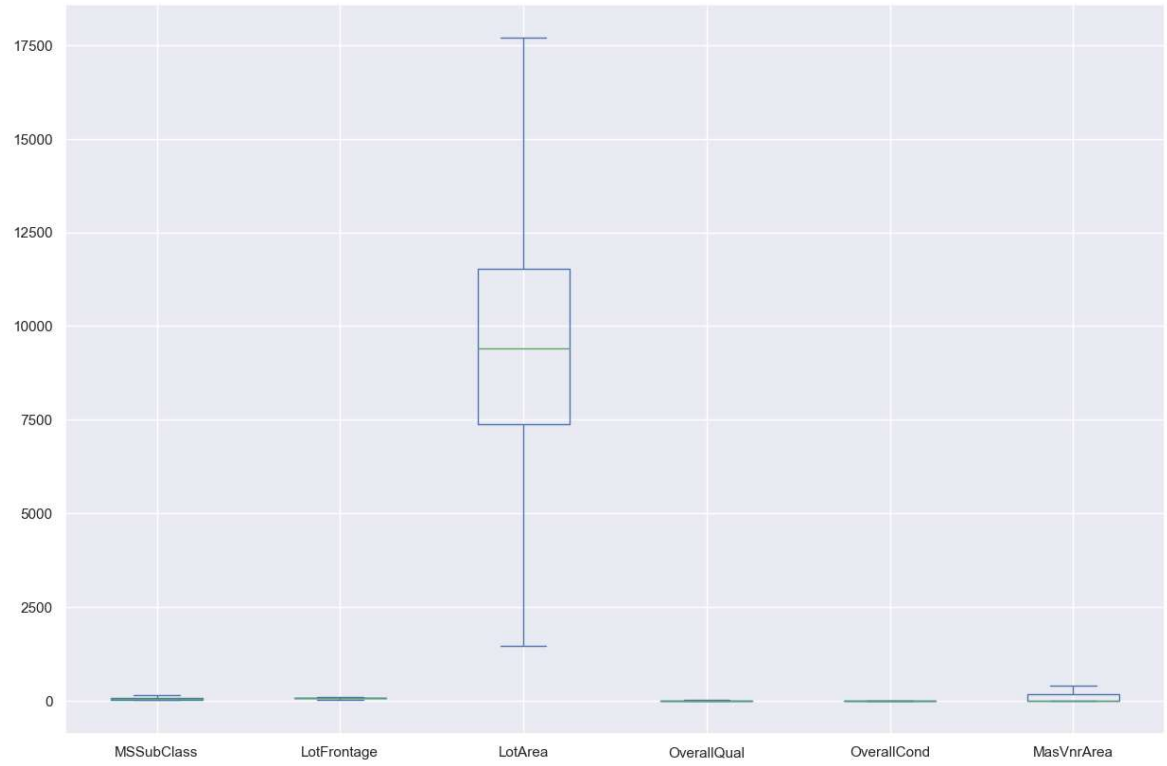
In [65]:
```
OverallCond_q1 = te2['MasVnrArea'].quantile(0.25)
OverallCond_q3 = te2['MasVnrArea'].quantile(0.75)
OverallCond_iqr = OverallCond_q3 -  OverallCond_q1
OverallCond_upper = OverallCond_q3 + 1.5 * OverallCond_iqr
OverallCond_lower = OverallCond_q1 - 1.5 * OverallCond_iqr
```

In [66]:
```
te2['MasVnrArea'] = np.where(te2['MasVnrArea'] > OverallCond_upper, OverallCond_upper,
                      np.where(te2['MasVnrArea'] < OverallCond_lower, OverallCond_lower,
                               te2['MasVnrArea']) )
```

In [67]:
```
# The Graph after the Outlier Treatment
```

In [68]:
```
te2.plot(kind='box', figsize=(15,10))
```

Out[68]: <Axes: >



Inter Quartile Range

In [69]:
```
Q1 = te2.quantile(0.25)
Q3 = te2.quantile(0.75)
IQR = Q3 - Q1
```

In [70]:
```
pos_outlier = Q3 + 1.5 * IQR
neg_outlier = Q3 - 1.5 * IQR
```

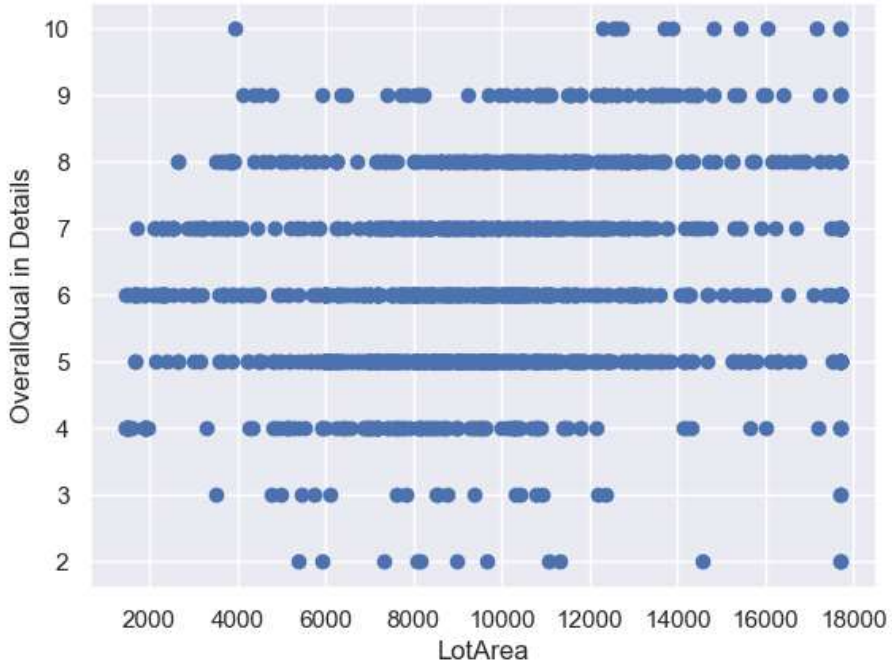
In [71]:

```
print(Q1)
print("Q1************"*5)
print(Q3)
print("Q3************"*5)
print(IQR)
print("IQR************"*5)
print(pos_outlier)
print("pos************"*5)
print(neg_outlier)
print("neg************"*5)
```

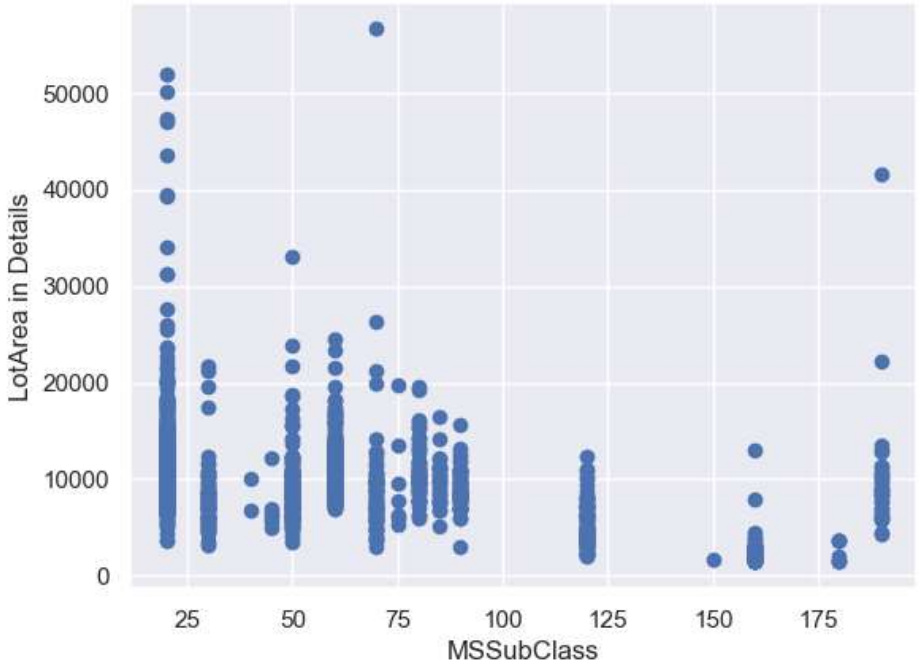
```
MSSubClass       20.0
LotFrontage      60.0
LotArea        7391.0
OverallQual       5.0
OverallCond       5.0
MasVnrArea        0.0
Name: 0.25, dtype: float64
Q1************Q1************Q1************Q1************Q1************
MSSubClass       70.0
LotFrontage      78.0
LotArea        11517.5
OverallQual        7.0
OverallCond        6.0
MasVnrArea       162.0
Name: 0.75, dtype: float64
Q3************Q3************Q3************Q3************Q3************
MSSubClass       50.0
LotFrontage      18.0
LotArea        4126.5
OverallQual       2.0
OverallCond       1.0
MasVnrArea      162.0
dtype: float64
IQR************IQR************IQR************IQR************IQR************
MSSubClass       145.00
LotFrontage      105.00
LotArea        17707.25
OverallQual       10.00
OverallCond        7.50
MasVnrArea       405.00
dtype: float64
pos************pos************pos************pos************pos************
MSSubClass       -5.00
LotFrontage      51.00
LotArea        5327.75
OverallQual       4.00
OverallCond       4.50
MasVnrArea      -81.00
dtype: float64
neg************neg************neg************neg************neg************
```

Plots to Understand

In [72]:
```
plt.scatter(te2['LotArea'], te2['OverallQual'])
plt.xlabel("LotArea")
plt.ylabel("OverallQual in Details")
plt.show()
```

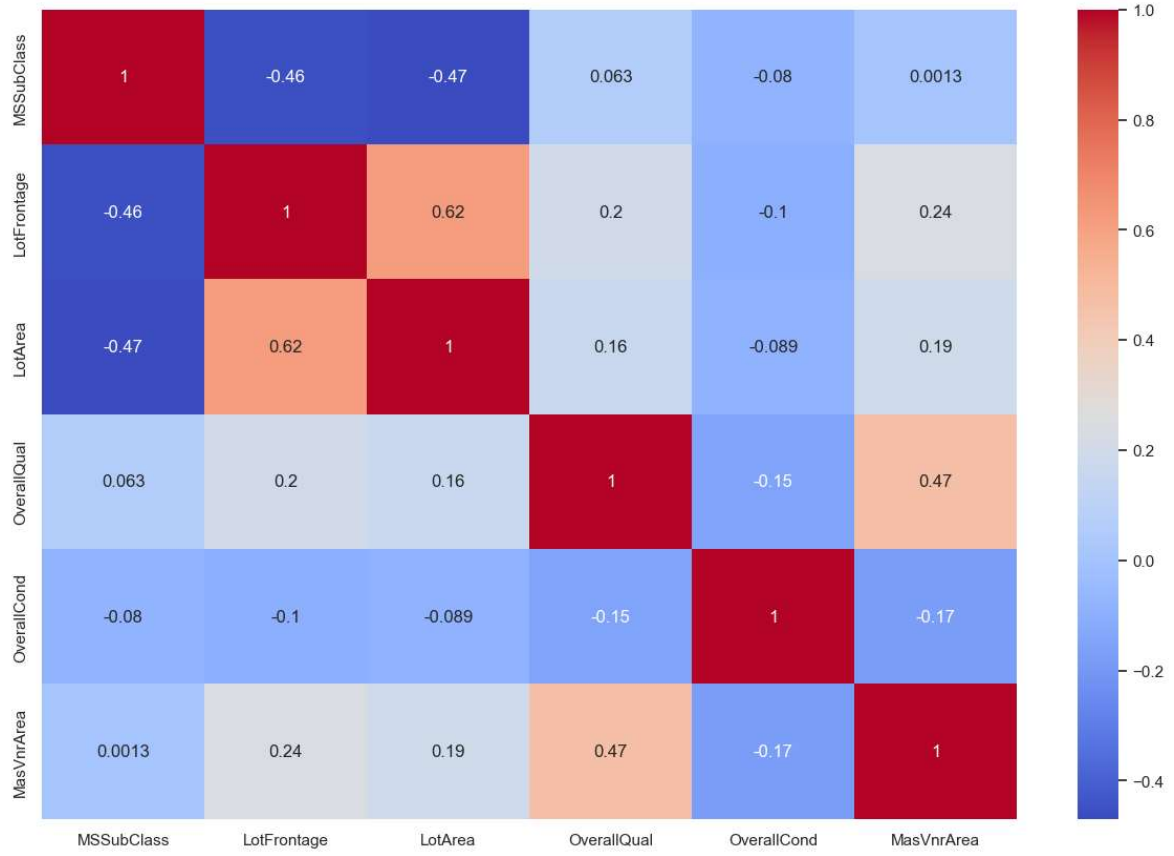


In [73]:
```
plt.scatter(te['MSSubClass'], te['LotArea'])
plt.xlabel("MSSubClass")
plt.ylabel("LotArea in Details")
plt.show()
```



In [74]:

```
plt.figure(figsize=(15,10))
corr = te2.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.show()
```



In [75]:
```
sns.pairplot(te2)
```

Out[75]:
```
<seaborn.axisgrid.PairGrid at 0x17a0ed49810>
```

In [ ]:
```

```

In [76]:
```
te3 = te.copy()
```

In [77]:
```
te3.head()
```

Out[77]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	Tc
0	20	80.0	11622	5	6	0.0	468.0	144.0	270.0	
1	20	81.0	14267	6	6	108.0	923.0	0.0	406.0	
2	60	74.0	13830	5	5	0.0	791.0	0.0	137.0	
3	60	78.0	9978	6	6	20.0	602.0	0.0	324.0	
4	120	43.0	5005	8	5	0.0	263.0	0.0	1017.0	

5 rows × 33 columns

In [78]:
```
te3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1459 entries, 0 to 1458
Data columns (total 33 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   MSSubClass     1459 non-null   int64  
 1   LotFrontage    1232 non-null   float64
 2   LotArea        1459 non-null   int64  
 3   OverallQual    1459 non-null   int64  
 4   OverallCond    1459 non-null   int64  
 5   MasVnrArea     1444 non-null   float64
 6   BsmtFinSF1     1458 non-null   float64
 7   BsmtFinSF2     1458 non-null   float64
 8   BsmtUnfSF      1458 non-null   float64
 9   TotalBsmtSF    1458 non-null   float64
 10  1stFlrSF       1459 non-null   int64  
 11  2ndFlrSF       1459 non-null   int64  
 12  LowQualFinSF   1459 non-null   int64  
 13  GrLivArea      1459 non-null   int64  
 14  BsmtFullBath   1457 non-null   float64
 15  BsmtHalfBath   1457 non-null   float64
 16  FullBath       1459 non-null   int64  
 17  HalfBath       1459 non-null   int64  
 18  BedroomAbvGr   1459 non-null   int64  
 19  KitchenAbvGr   1459 non-null   int64  
 20  TotRmsAbvGrd   1459 non-null   int64  
 21  Fireplaces     1459 non-null   int64  
 22  GarageYrBlt    1381 non-null   float64
 23  GarageCars     1458 non-null   float64
 24  GarageArea     1458 non-null   float64
 25  WoodDeckSF     1459 non-null   int64  
 26  OpenPorchSF    1459 non-null   int64  
 27  EnclosedPorch  1459 non-null   int64  
 28  3SsnPorch      1459 non-null   int64  
 29  ScreenPorch    1459 non-null   int64  
 30  PoolArea       1459 non-null   int64  
 31  MiscVal        1459 non-null   int64  
 32  MoSold         1459 non-null   int64  
dtypes: float64(11), int64(22)
memory usage: 376.3 KB
```

In [79]:
```
# Drop Columns
```

In [80]:
```
te3 = te3.drop(["MasVnrArea", "BsmtFinSF1", "BsmtFinSF2", "BsmtUnfSF", "TotalBsmtSF",
                "1stFlrSF", "2ndFlrSF", "LowQualFinSF", "GrLivArea", "BsmtFullBath", "BsmtHalfBath"
                "BedroomAbvGr", "KitchenAbvGr", "TotRmsAbvGrd","Fireplaces", "GarageYrBlt", "Garage
                "WoodDeckSF","OpenPorchSF", "EnclosedPorch", "3SsnPorch", "ScreenPorch", "PoolArea"
                  "MoSold"], axis=1)
```

In [81]:
```
te3.head()
```

Out[81]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond
0	20	80.0	11622	5	6
1	20	81.0	14267	6	6
2	60	74.0	13830	5	5
3	60	78.0	9978	6	6
4	120	43.0	5005	8	5

In [82]:
```
# Null Values Analysis
```

In [83]:
```
te3.isnull().sum()
```

Out[83]:
```
MSSubClass       0
LotFrontage    227
LotArea          0
OverallQual      0
OverallCond      0
dtype: int64
```

In [84]:
```
te3["LotFrontage"].mean()
```

Out[84]:
```
68.58035714285714
```

In [85]:
```
te3["LotFrontage"]=te3["LotFrontage"].fillna(te3["LotFrontage"].mean())
```

In [86]:
```
te3.isnull().sum()
```

Out[86]:
```
MSSubClass     0
LotFrontage    0
LotArea        0
OverallQual    0
OverallCond    0
dtype: int64
```

One Hot Encoding

In [87]:
```
te3
```

Out[87]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond
0	20	80.0	11622	5	6
1	20	81.0	14267	6	6
2	60	74.0	13830	5	5
3	60	78.0	9978	6	6
4	120	43.0	5005	8	5
...	...	...	...	...	...
1454	160	21.0	1936	4	7
1455	160	21.0	1894	4	5
1456	20	160.0	20000	5	7
1457	85	62.0	10441	5	5
1458	60	74.0	9627	7	5

1459 rows × 5 columns

In [88]:
```
# Treating Null Values
```

In [89]:
```
MSSubClass_q1 = te3['MSSubClass'].quantile(0.25)
MSSubClass_q3 = te3['MSSubClass'].quantile(0.75)
MSSubClass_iqr = MSSubClass_q3 -  MSSubClass_q1
MSSubClass_upper = MSSubClass_q3 + 1.5 * MSSubClass_iqr
MSSubClass_lower = MSSubClass_q1 - 1.5 * MSSubClass_iqr
```

In [90]:
```
te3['MSSubClass'] = np.where(te3['MSSubClass'] > MSSubClass_upper, MSSubClass_upper,
                             np.where(te3['MSSubClass'] < MSSubClass_lower, MSSubClass_lower,
                                      te3['MSSubClass']) )
```

In [91]:
```
LotFrontage_q1 = te3['LotFrontage'].quantile(0.25)
LotFrontage_q3 = te3['LotFrontage'].quantile(0.75)
LotFrontage_iqr = LotFrontage_q3 -  LotFrontage_q1
LotFrontage_upper = LotFrontage_q3 + 1.5 * LotFrontage_iqr
LotFrontage_lower = LotFrontage_q1 - 1.5 * LotFrontage_iqr
```

In [92]:
```
te3['LotFrontage'] = np.where(te3['LotFrontage'] > LotFrontage_upper, LotFrontage_upper,
                              np.where(te3['LotFrontage'] < LotFrontage_lower, LotFrontage_lower,
                                       te3['LotFrontage']) )
```

In [93]:
```
LotArea_q1 = te3['LotArea'].quantile(0.25)
LotArea_q3 = te3['LotArea'].quantile(0.75)
LotArea_iqr = LotArea_q3 -  LotArea_q1
LotArea_upper = LotArea_q3 + 1.5 * LotArea_iqr
LotArea_lower = LotArea_q1 - 1.5 * LotArea_iqr
```

In [94]:
```
te3['LotArea'] = np.where(te3['LotArea'] > LotArea_upper, LotArea_upper,
                    np.where(te3['LotArea'] < LotArea_lower, LotArea_lower,
                             te3['LotArea']) )
```

In [95]:
```
OverallQual_q1 = te3['OverallQual'].quantile(0.25)
OverallQual_q3 = te3['OverallQual'].quantile(0.75)
OverallQual_iqr = OverallQual_q3 -  OverallQual_q1
OverallQual_upper = OverallQual_q3 + 1.5 * OverallQual_iqr
OverallQual_lower = OverallQual_q1 - 1.5 * OverallQual_iqr
```

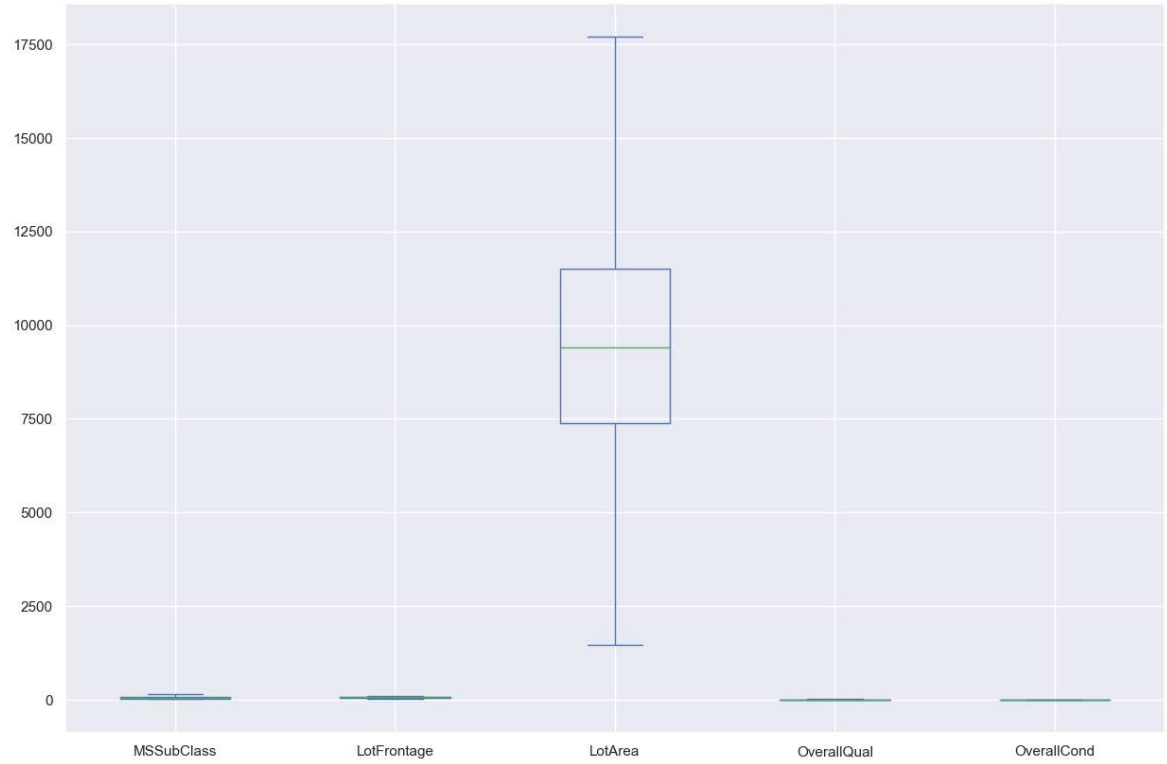
In [96]:
```
te3['OverallQual'] = np.where(te3['OverallQual'] > OverallQual_upper, OverallQual_upper,
                    np.where(te3['OverallQual'] < OverallQual_lower, OverallQual_lower,
                             te3['OverallQual']) )
```

In [97]:
```
OverallCond_q1 = te3['OverallCond'].quantile(0.25)
OverallCond_q3 = te3['OverallCond'].quantile(0.75)
OverallCond_iqr = OverallCond_q3 -  OverallCond_q1
OverallCond_upper = OverallCond_q3 + 1.5 * OverallCond_iqr
OverallCond_lower = OverallCond_q1 - 1.5 * OverallCond_iqr
```

In [98]:
```
te3['OverallCond'] = np.where(te3['OverallCond'] > OverallCond_upper, OverallCond_upper,
                    np.where(te3['OverallCond'] < OverallCond_lower, OverallCond_lower,
                             te3['OverallCond']) )
```

In [99]:
```
te3.plot(kind='box', figsize=(15,10))
```

Out[99]: <Axes: >



In [100]:
```
te3
```

Out[100]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond
0	20.0	80.0	11622.00	5.0	6.0
1	20.0	81.0	14267.00	6.0	6.0
2	60.0	74.0	13830.00	5.0	5.0
3	60.0	78.0	9978.00	6.0	6.0
4	120.0	43.0	5005.00	8.0	5.0
...	...	...	...	...	...
1454	145.0	33.0	1936.00	4.0	7.0
1455	145.0	33.0	1894.00	4.0	5.0
1456	20.0	105.0	17707.25	5.0	7.0
1457	85.0	62.0	10441.00	5.0	5.0
1458	60.0	74.0	9627.00	7.0	5.0

1459 rows × 5 columns

Feature Scaling

In [101]:
```
x = te3.iloc[:,0:-1]
y = te3['OverallCond']
```

In [102]:
```
x.head()
```

Out[102]:

	MSSubClass	LotFrontage	LotArea	OverallQual
0	20.0	80.0	11622.0	5.0
1	20.0	81.0	14267.0	6.0
2	60.0	74.0	13830.0	5.0
3	60.0	78.0	9978.0	6.0
4	120.0	43.0	5005.0	8.0

In [103]:
```
y.head()
```

Out[103]:
```
0    6.0
1    6.0
2    5.0
3    6.0
4    5.0
Name: OverallCond, dtype: float64
```

In [104]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc_x = sc.fit_transform(x)
pd.DataFrame(sc_x)
```

Out[104]:

	0	1	2	3
0	-0.934709	0.688567	0.584370	-0.754355
1	-0.934709	0.746893	1.314169	-0.056002
2	0.116040	0.338612	1.193593	-0.754355
3	0.116040	0.571916	0.130764	-0.056002
4	1.692162	-1.469489	-1.241368	1.340703
...	...	...	...	...
1454	2.348880	-2.052748	-2.088154	-1.452707
1455	2.348880	-2.052748	-2.099743	-1.452707
1456	-0.934709	2.146714	2.263389	-0.754355
1457	0.772757	-0.361298	0.258513	-0.754355
1458	0.116040	0.338612	0.033917	0.642350

1459 rows × 4 columns

In [105]:
```
sc_x
```

Out[105]:
```
array([[-0.93470869,  0.68856728,  0.58437016, -0.75435458],
       [-0.93470869,  0.74689314,  1.3141686 , -0.05600221],
       [ 0.11603963,  0.33861214,  1.19359321, -0.75435458],
       ...,
       [-0.93470869,  2.1467137 ,  2.26338943, -0.75435458],
       [ 0.77275733, -0.36129814,  0.25851309, -0.75435458],
       [ 0.11603963,  0.33861214,  0.03391727,  0.64235016]])
```

In [106]:
```
tex = pd.DataFrame(sc_x, columns = x.columns[:])
tex.head(2)
```

Out[106]:

	MSSubClass	LotFrontage	LotArea	OverallQual
0	-0.934709	0.688567	0.584370	-0.754355
1	-0.934709	0.746893	1.314169	-0.056002

In [107]:
```
variable = sc_x
variable.shape
```

Out[107]:
```
(1459, 4)
```

In [108]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
variable = sc_x

vif = pd.DataFrame()

vif['Variance Inflation Factor'] = [variance_inflation_factor(variable, i ) for i in range(variab

vif['Features'] = x.columns
```

In [109]:
```
vif
```

Out[109]:

	Variance Inflation Factor	Features
0	1.417216	MSSubClass
1	1.760776	LotFrontage
2	1.756153	LotArea
3	1.087642	OverallQual

Split the data into train and test

In [110]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=101)
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(1094, 4) (365, 4) (1094,) (365,)
```

Building Model

1. Linear Regression

In [111]:
```
# A. SKLearn Approach
```

In [112]:
```
from sklearn.linear_model import LinearRegression
lm=LinearRegression()
lr=lm.fit(x_train, y_train)
```

In [113]:
```
print(lm.intercept_)
print()
print(lm.coef_)
```

```
6.782723183259975

[-4.04271305e-03 -5.22269065e-03 -2.15695136e-05 -7.55728787e-02]
```

In [114]:
```
y_pred_test_price = lm.predict(x_test) # Test Dataset
y_pred_train_price = lm.predict(x_train) # Training Dataset
```

In [115]:
```
y_test
```

Out[115]:
```
666     7.5
104     5.0
528     5.0
18      5.0
1151    6.0
       ...
1113    5.0
803     5.0
778     5.0
954     5.0
1258    6.0
Name: OverallCond, Length: 365, dtype: float64
```

In [116]:
```
y_train
```

Out[116]:
```
801     5.0
1426    6.0
1240    5.0
1138    5.0
1381    5.0
       ...
1417    7.5
75      6.0
599     7.0
1361    7.0
863     5.0
Name: OverallCond, Length: 1094, dtype: float64
```

In [117]:
```
from sklearn.metrics import r2_score
print ('Training Accuracy of Model Using Linear Regression Method : ', r2_score(y_train, y_pred_t
print ('Test Accuracy of Model Using Linear Regression Method : ', r2_score(y_test, y_pred_test_p
```

```
Training Accuracy of Model Using Linear Regression Method :  0.04746358806499229
Test Accuracy of Model Using Linear Regression Method :  0.04365536402959924
```

In [118]:
```
d = (0.04746358806499229 - 0.04365536402959924)
d
```

Out[118]:
```
0.0038082240353930485
```

In [119]:
```
# The Difference between Training Accuracy and Test Accuracy is almost 0.004. Hence there is a Ov
```

In [120]:
```
# B. Using OLS Approach method
```

In [121]:
```
from statsmodels.regression.linear_model import OLS
import statsmodels.regression.linear_model as smf
```

In [122]:
```
te_model = smf.OLS(endog = y_train, exog=x_train).fit()
```

In [123]:
```
te_model.summary()
```

Out[123]:

OLS Regression Results

Dep. Variable:	OverallCond	**R-squared (uncentered):**	0.938
Model:	OLS	**Adj. R-squared (uncentered):**	0.938
Method:	Least Squares	**F-statistic:**	4109.
Date:	Fri, 10 Nov 2023	**Prob (F-statistic):**	0.00
Time:	13:49:13	**Log-Likelihood:**	-1920.0
No. Observations:	1094	**AIC:**	3848.
Df Residuals:	1090	**BIC:**	3868.
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
MSSubClass	0.0137	0.001	12.092	0.000	0.011	0.016
LotFrontage	0.0380	0.003	13.491	0.000	0.032	0.044
LotArea	4.124e-05	1.53e-05	2.704	0.007	1.13e-05	7.12e-05
OverallQual	0.2636	0.027	9.611	0.000	0.210	0.317

Omnibus:	24.604	**Durbin-Watson:**	2.019
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	20.285
Skew:	0.258	**Prob(JB):**	3.94e-05
Kurtosis:	2.577	**Cond. No.**	6.66e+03

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 6.66e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

In [124]:
```
#C. Gradient Descent Approach Method
```

In [125]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(sc_x, y, test_size=0.25, random_state=101)
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(1094, 4) (365, 4) (1094,) (365,)
```

In [126]:
```
from sklearn.linear_model import SGDRegressor
```

In [127]:
```
tegd_model = SGDRegressor()
tegd_model.fit(x_train, y_train)
```

Out[127]:

▾ SGDRegressor

SGDRegressor()

In [128]:
```
y_pred_train_tegd = tegd_model.predict(x_train)
y_pred_test_tegd = tegd_model.predict(x_test)
```

In [129]:
```
print("Training Accuracy of Model Using Gradient Descent Approach Method :", r2_score(y_train, y_
print()
print("Test Accuracy of Model Using Gradient Descent Approach Method :", r2_score(y_test, y_pred_
```

```
Training Accuracy of Model Using Gradient Descent Approach Method : 0.047160946738845855

Test Accuracy of Model Using Gradient Descent Approach Method : 0.044714700701612764
```

2 : Ridge Regression (L2- Regularization)

In [130]:
```
# closure to zero but not exact zero
# penalty - 0.3
```

In [131]:
```
from sklearn.linear_model import Ridge
te.ridge = Ridge(alpha=0.3)
te.rr = te.ridge.fit(x_train, y_train)
print("TE.Ridge Model :", (te.ridge.coef_))
```

```
TE.Ridge Model : [-0.15382401 -0.08950643 -0.07814249 -0.10820197]
```

In [132]:
```
y_pred_train_ridge = te.ridge.predict(x_train)
y_pred_test_ridge = te.ridge.predict(x_test)
```

In [133]:
```
print("Training Accuracy of Model Using Ridge Regression Method :", r2_score(y_train, y_pred_trai
print()
print("Test Accuracy of Model Using Ridge Regression Method :", r2_score(y_test, y_pred_test_ridg
```

```
Training Accuracy of Model Using Ridge Regression Method : 0.04746358222817082

Test Accuracy of Model Using Ridge Regression Method : 0.04365729630774673
```

In [ ]:
```
```

3. Decision Tree

In [134]:
```
from sklearn.tree import DecisionTreeRegressor
dtree = DecisionTreeRegressor()
dtree.fit(x_train, y_train)
```

Out[134]:
```
DecisionTreeRegressor

DecisionTreeRegressor()
```

In [135]:
```
y_pred_train_tedt = dtree.predict(x_train)
y_pred_test_tedt = dtree.predict(x_test)
```

In [136]:
```
print("Training Accuracy of Model Using Decision Tree Method :", r2_score(y_train, y_pred_train_t
print()
print("Test Accuracy of Model Using Decision Tree Method :", r2_score(y_test, y_pred_test_tedt))
```

```
Training Accuracy of Model Using Decision Tree Method : 0.9429429963374778

Test Accuracy of Model Using Decision Tree Method : -0.4123776946105402
```

4. Bagging

In [137]:
```
from sklearn.ensemble import BaggingRegressor
bagging = BaggingRegressor()
bagging.fit(x_train, y_train)
```

Out[137]:
```
BaggingRegressor

BaggingRegressor()
```

In [138]:
```
y_pred_train_tebg = dtree.predict(x_train)
y_pred_test_tebg = dtree.predict(x_test)
```

In [139]:
```
print("Training Accuracy of Model Using Bagging Method :", r2_score(y_train, y_pred_train_tebg))
print()
print("Test Accuracy of Model Using Bagging Method :", r2_score(y_test, y_pred_test_tebg))
```

```
Training Accuracy of Model Using Bagging Method : 0.9429429963374778

Test Accuracy of Model Using Bagging Method : -0.4123776946105402
```

5. Random Forest Regression

In [140]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()
rf.fit(x_train, y_train)
```

Out[140]:
```
RandomForestRegressor
RandomForestRegressor()
```

In [141]:
```
y_pred_train_terf = rf.predict(x_train)
y_pred_test_terf = rf.predict(x_test)
```

In [142]:
```
print("Training Accuracy of Model Using Random Forest Method :", r2_score(y_train, y_pred_train_t
print()
print("Test Accuracy of Model Using Random Forest Method :", r2_score(y_test, y_pred_test_terf))
```

```
Training Accuracy of Model Using Random Forest Method : 0.8297475216796728

Test Accuracy of Model Using Random Forest Method : 0.15262953554479586
```

6. Lasso regularization

In [143]:
```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=0.1)
lasso.fit(x_train, y_train)
print("Lasso Model :", (lasso.coef_))
```

```
Lasso Model : [-0.         -0.         -0.         -0.04554661]
```

In [144]:
```
y_pred_train_tela = lasso.predict(x_train)
y_pred_test_tela = lasso.predict(x_test)
```

In [145]:
```
print("Training Accuracy of Model Using Lasso Regularization Method :", r2_score(y_train, y_pred_
print()
print("Test Accuracy of Model Using Lasso Regularization Method :", r2_score(y_test, y_pred_test_
```

```
Training Accuracy of Model Using Lasso Regularization Method : 0.012722528218197193

Test Accuracy of Model Using Lasso Regularization Method : 0.010590527496321278
```

7. Support Vector Machine

In [146]:
```
from sklearn.svm import SVR

svm_linear = SVR(kernel='linear')
sv = svm_linear.fit(x_train, y_train)
y_pred_train_teli=svm_linear.predict(x_train)
y_pred_test_teli=svm_linear.predict(x_test)
```

In [147]:
```
print("Training Accuracy of Model Using Support Vector Machine Method :", r2_score(y_train, y_pre
print()
print("Test Accuracy of Model Using Support Vector Machine Method :", r2_score(y_test, y_pred_tes
```

```
Training Accuracy of Model Using Support Vector Machine Method : -0.10647107911975895

Test Accuracy of Model Using Support Vector Machine Method : -0.09995232155055134
```

8. AdaBoost Regressor

In [148]:
```
from sklearn.ensemble import AdaBoostRegressor
ada = AdaBoostRegressor()
ada.fit(x_train, y_train)
```

Out[148]:
```
AdaBoostRegressor

AdaBoostRegressor()
```

In [149]:
```
y_pred_train_tead = ada.predict(x_train)
y_pred_test_tead = ada.predict(x_test)
```

In [150]:
```
print("Training Accuracy of Model Using Support Vector Machine Method :", r2_score(y_train, y_pre
print()
print("Test Accuracy of Model Using Support Vector Machine Method :", r2_score(y_test, y_pred_tes
```

```
Training Accuracy of Model Using Support Vector Machine Method : 0.13520109979860584

Test Accuracy of Model Using Support Vector Machine Method : 0.09485753831555976
```

9. Gradient Boost Regressor

In [151]:
```
from sklearn.ensemble import GradientBoostingRegressor
gdb = GradientBoostingRegressor()
gdb.fit(x_train, y_train)
```

Out[151]:
```
GradientBoostingRegressor

GradientBoostingRegressor()
```

In [152]:
```
y_pred_train_tegb = gdb.predict(x_train)
y_pred_test_tegb = gdb.predict(x_test)
```

In [153]:
```
print("Training Accuracy of Model Using Support Vector Machine Method :", r2_score(y_train, y_pre
print()
print("Test Accuracy of Model Using Support Vector Machine Method :", r2_score(y_test, y_pred_tes
```

```
Training Accuracy of Model Using Support Vector Machine Method : 0.38191049194262505

Test Accuracy of Model Using Support Vector Machine Method : 0.19603294638402757
```

10. XGBoost Regressor

In [154]:
```
xgb = GradientBoostingRegressor()
xg = xgb.fit(x_train, y_train)
```

In [155]:
```
y_pred_train_texg = xgb.predict(x_train)
y_pred_test_texg = xgb.predict(x_test)
```

In [156]:
```
print("Training Accuracy of Model Using Support Vector Machine Method :", r2_score(y_train, y_pre
print()
print("Test Accuracy of Model Using Support Vector Machine Method :", r2_score(y_test, y_pred_tes
```

```
Training Accuracy of Model Using Support Vector Machine Method : 0.38191049194262505

Test Accuracy of Model Using Support Vector Machine Method : 0.19626697439915752
```

11. KNN Regressor

In [157]:
```
from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor()
knn.fit(x_train, y_train)
```

Out[157]:
```
KNeighborsRegressor

KNeighborsRegressor()
```

In [158]:
```
y_pred_train_teknn = gdb.predict(x_train)
y_pred_test_teknn = gdb.predict(x_test)
```

In [159]:
```
print("Training Accuracy of Model Using Support Vector Machine Method :", r2_score(y_train, y_pre
print()
print("Test Accuracy of Model Using Support Vector Machine Method :", r2_score(y_test, y_pred_tes
```

```
Training Accuracy of Model Using Support Vector Machine Method : 0.38191049194262505

Test Accuracy of Model Using Support Vector Machine Method : 0.19603294638402757
```

12. ElasticNet

In [160]:
```
from sklearn.linear_model import ElasticNet
elastic = ElasticNet(alpha=0.3, l1_ratio=0.1)
elastic.fit(x_train, y_train)
```

Out[160]:
```
ElasticNet

ElasticNet(alpha=0.3, l1_ratio=0.1)
```

In [161]:
```
y_pred_train_teel = elastic.predict(x_train)
y_pred_test_teel = elastic.predict(x_test)
```

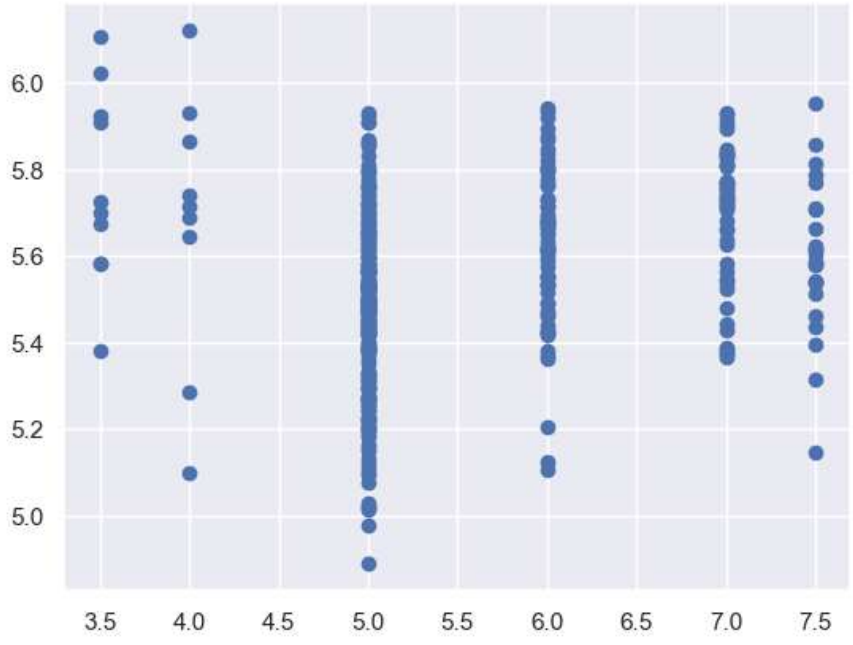
In [162]:
```
print("Training Accuracy of Model Using ElasticNet Method :", r2_score(y_train, y_pred_train_teel
print()
print("Test Accuracy of Model Using ElasticNet Method :", r2_score(y_test, y_pred_test_teel))
```

```
Training Accuracy of Model Using ElasticNet Method : 0.03654550929351874

Test Accuracy of Model Using ElasticNet Method : 0.03438186396193654
```

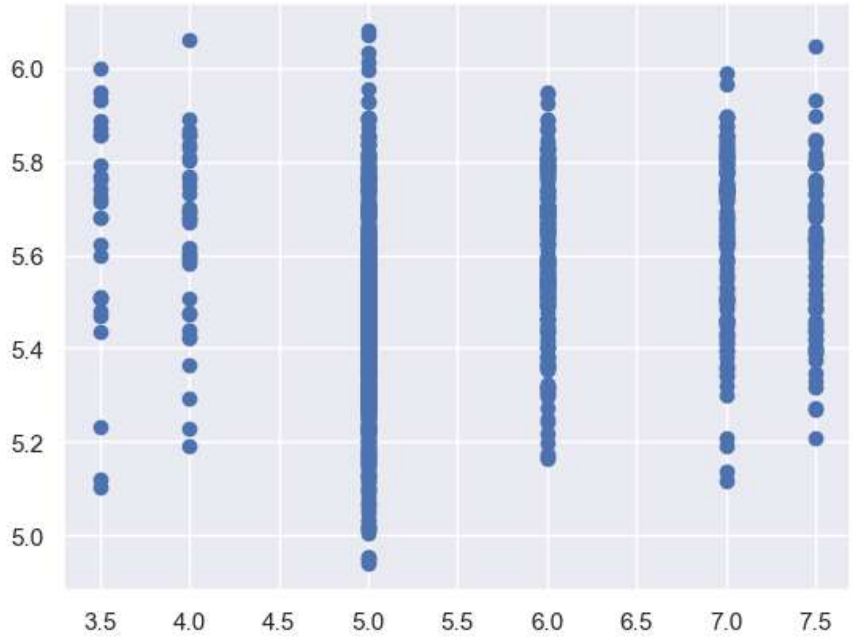
In [163]:
```
plt.scatter(y_test, y_pred_test_price)
```

Out[163]:
```
<matplotlib.collections.PathCollection at 0x17a17b1dc60>
```

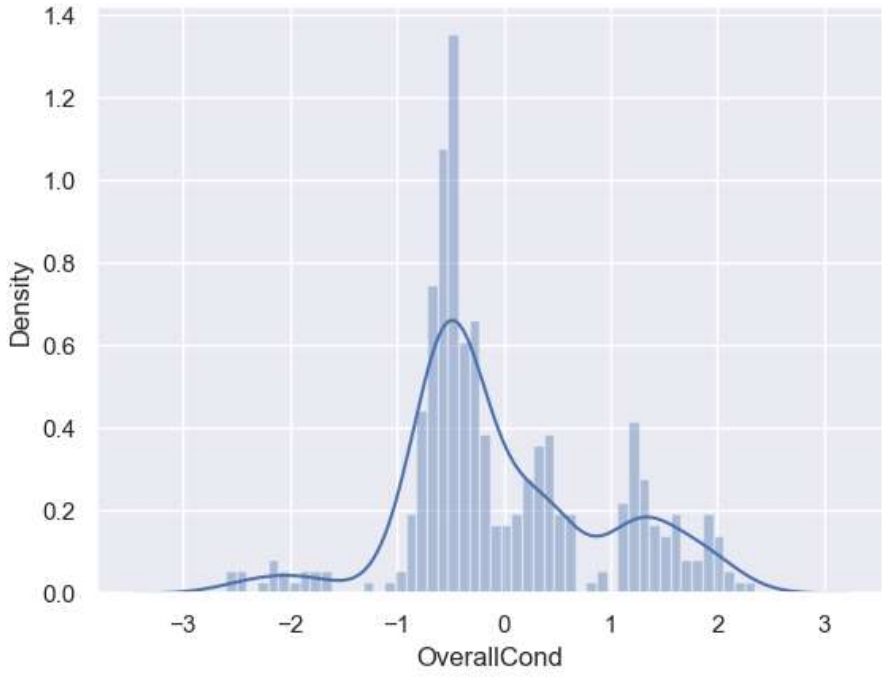


In [164]:
```
plt.scatter(y_train, y_pred_train_price)
```

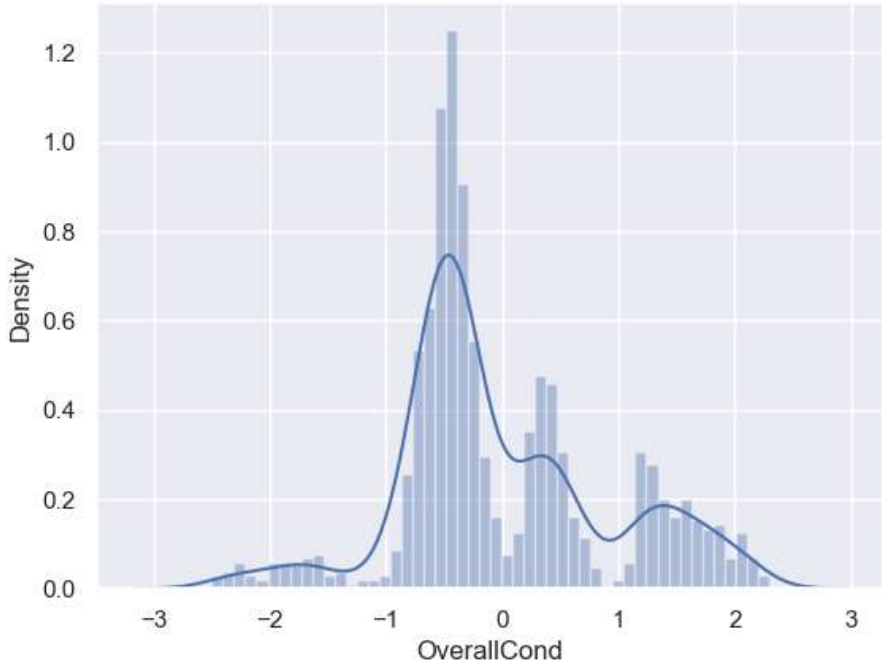
Out[164]: <matplotlib.collections.PathCollection at 0x17a17ba8b80>



In [165]:
```
sns.distplot((y_test - y_pred_test_price), bins=50)
plt.show()
```



In [166]:
```
sns.distplot((y_train - y_pred_train_price), bins=50)
plt.show()
```



Performance matrix

In [167]:
```
#Mean Absolute Error (MAE)
```

In [168]:
```
from sklearn import metrics
```

In [169]:
```
MAE = (metrics.mean_absolute_error(y_test, y_pred_test_price))
print("MAE :", MAE)
```

```
MAE : 0.748119218869478
```

In [170]:
```
#Mean Absolute Percent Error (MAPE)
```

In [171]:
```
MAPE = (metrics.mean_absolute_error(y_test, y_pred_test_price)/100)
print("MAPE :", MAPE)
```

```
MAPE : 0.00748119218869478
```

In [172]:
```
#Mean Squared Error (MSE)
```

In [173]:
```
MSE = (metrics.mean_squared_error(y_test,y_pred_test_price))
print("MSE :", MSE)
```

```
MSE : 0.8757776701470656
```

In [174]:
```
#Root Mean Squared Error (RMSE)
```

In [175]:
```
RMSE = np.sqrt(metrics.mean_squared_error(y_test,y_pred_test_price))
print("RMSE :", RMSE)
```

```
RMSE : 0.9358299365520776
```

Final

In [176]:
```
print(sc_x)
```

```
[[-0.93470869  0.68856728  0.58437016 -0.75435458]
 [-0.93470869  0.74689314  1.3141686  -0.05600221]
 [ 0.11603963  0.33861214  1.19359321 -0.75435458]
 ...
 [-0.93470869  2.1467137   2.26338943 -0.75435458]
 [ 0.77275733 -0.36129814  0.25851309 -0.75435458]
 [ 0.11603963  0.33861214  0.03391727  0.64235016]]
```

In [177]:
```
print(tex.head(2))
```

```
   MSSubClass  LotFrontage   LotArea  OverallQual
0   -0.934709     0.688567  0.584370    -0.754355
1   -0.934709     0.746893  1.314169    -0.056002
```

In [178]:
```
print(variable.shape)
```

```
(1459, 4)
```

In [179]:
```
print(vif)
```

```
   Variance Inflation Factor     Features
0                   1.417216   MSSubClass
1                   1.760776  LotFrontage
2                   1.756153      LotArea
3                   1.087642  OverallQual
```

In [180]:
```
print(lm.intercept_)
```

```
6.782723183259975
```

In [181]:
```
print(lm.coef_)
```

```
[-4.04271305e-03 -5.22269065e-03 -2.15695136e-05 -7.55728787e-02]
```

In [182]:
```
Model = ["Using Linear Regression Method: SKlearn Approach", "Using Linear Regression Method: SKl
        "Random Forest Regression", "Lasso regularization", "Support Vector Machine", "AdaBoost R

Train = [r2_score(y_train, y_pred_train_price), r2_score(y_train, y_pred_train_tegd), r2_score(y_
        r2_score(y_train, y_pred_train_teli), r2_score(y_train, y_pred_train_tead), r2_score(y_tr

Test = [r2_score(y_test, y_pred_test_price), r2_score(y_test, y_pred_test_tegd), r2_score(y_test,
        r2_score(y_test, y_pred_test_teli), r2_score(y_test, y_pred_test_tead), r2_score(y_test,
```

In [183]:

```
Final_Result = pd.DataFrame ({'Model' : Model, 'Train': Train, 'Test': Test})
Final_Result['Difference'] = Final_Result['Train'] - Final_Result['Test']
print(Final_Result)
```

```
                                                Model     Train      Test  \
0    Using Linear Regression Method: SKlearn Approach  0.047464  0.043655
1   Using Linear Regression Method: SKlearn Approa...  0.047161  0.044715
2               Ridge Regression (L2- Regularization)  0.047464  0.043657
3      Building Model - DecisionTree Regression Model  0.942943 -0.412378
4                                    Bagging Method  0.942943 -0.412378
5                           Random Forest Regression  0.829748  0.152630
6                               Lasso regularization  0.012723  0.010591
7                         Support Vector Machine -0.106471 -0.099952
8                             AdaBoost Regressor  0.135201  0.094858
9                           Gradient Boost Regressor  0.381910  0.196033
10                                XGBoost Regressor  0.381910  0.196267
11                                    KNN Regressor  0.381910  0.196033
12                                       ElasticNet  0.036546  0.034382

    Difference
0     0.003808
1     0.002446
2     0.003806
3     1.355321
4     1.355321
5     0.677118
6     0.002132
7    -0.006519
8     0.040344
9     0.185878
10    0.185644
11    0.185878
12    0.002164
```

END