```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import os
from scipy import stats
from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, ElasticNet,
ElasticNetCV
from sklearn.linear_model import LogisticRegression
import sklearn.metrics as metrics
from sklearn import preprocessing
from numpy import array
from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import Lasso, LassoCV
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeRegressor

%matplotlib inline
sns.set()

df_train = pd.read_csv("train.csv")
df_test = pd.read_csv("test.csv")

print(df_train.shape)
print("*"*50)
print(df_test.shape)
```

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

```
df_train.head()
```

```
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape
\
0   1          60       RL         65.0     8450   Pave   NaN      Reg

1   2          20       RL         80.0     9600   Pave   NaN      Reg

2   3          60       RL         68.0    11250   Pave   NaN      IR1

3   4          70       RL         60.0     9550   Pave   NaN      IR1
```

```
4   5          60       RL         84.0    14260   Pave   NaN      IR1


  LandContour Utilities  ... PoolArea PoolQC Fence MiscFeature MiscVal
MoSold  \
0         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
2
1         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
5
2         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
9
3         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
2
4         Lvl    AllPub  ...        0    NaN   NaN         NaN       0
12

  YrSold  SaleType  SaleCondition  SalePrice
0   2008        WD         Normal     208500
1   2007        WD         Normal     181500
2   2008        WD         Normal     223500
3   2006        WD        Abnorml     140000
4   2008        WD         Normal     250000

[5 rows x 81 columns]
```

```
df_test.head()
```

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

  LandContour Utilities  ... ScreenPorch PoolArea PoolQC  Fence
MiscFeature  \
0         Lvl    AllPub  ...         120        0    NaN  MnPrv
NaN
1         Lvl    AllPub  ...           0        0    NaN    NaN
Gar2
2         Lvl    AllPub  ...           0        0    NaN  MnPrv
NaN
3         Lvl    AllPub  ...           0        0    NaN    NaN
NaN
```

```
4         HLS    AllPub  ...         144        0    NaN    NaN
NaN

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

[5 rows x 80 columns]
```

EDA

```
df_train.describe()
```

```
                Id   MSSubClass  LotFrontage        LotArea
OverallQual  \
count  1460.000000  1460.000000  1201.000000    1460.000000
1460.000000
mean    730.500000    56.897260    70.049958   10516.828082
6.099315
std     421.610009    42.300571    24.284752    9981.264932
1.382997
min       1.000000    20.000000    21.000000    1300.000000
1.000000
25%     365.750000    20.000000    59.000000    7553.500000
5.000000
50%     730.500000    50.000000    69.000000    9478.500000
6.000000
75%    1095.250000    70.000000    80.000000   11601.500000
7.000000
max    1460.000000   190.000000   313.000000  215245.000000
10.000000

       OverallCond    YearBuilt  YearRemodAdd   MasVnrArea
BsmtFinSF1  ...  \
count  1460.000000  1460.000000   1460.000000  1452.000000
1460.000000  ...
mean      5.575342  1971.267808   1984.865753   103.685262
443.639726  ...
std       1.112799    30.202904     20.645407   181.066207
456.098091  ...
min       1.000000  1872.000000   1950.000000     0.000000
0.000000  ...
25%       5.000000  1954.000000   1967.000000     0.000000
0.000000  ...
50%       5.000000  1973.000000   1994.000000     0.000000
383.500000  ...
75%       6.000000  2000.000000   2004.000000   166.000000
```

```
712.250000  ...
max       9.000000  2010.000000   2010.000000  1600.000000
5644.000000  ...

        WoodDeckSF  OpenPorchSF  EnclosedPorch    3SsnPorch
ScreenPorch  \
count  1460.000000  1460.000000    1460.000000  1460.000000
1460.000000
mean     94.244521    46.660274      21.954110     3.409589
15.060959
std     125.338794    66.256028      61.119149    29.317331
55.757415
min       0.000000     0.000000       0.000000     0.000000
0.000000
25%       0.000000     0.000000       0.000000     0.000000
0.000000
50%       0.000000    25.000000       0.000000     0.000000
0.000000
75%     168.000000    68.000000       0.000000     0.000000
0.000000
max     857.000000   547.000000     552.000000   508.000000
480.000000

          PoolArea       MiscVal       MoSold       YrSold
SalePrice
count  1460.000000   1460.000000  1460.000000  1460.000000
1460.000000
mean      2.758904     43.489041     6.321918  2007.815753
180921.195890
std      40.177307    496.123024     2.703626     1.328095
79442.502883
min       0.000000      0.000000     1.000000  2006.000000
34900.000000
25%       0.000000      0.000000     5.000000  2007.000000
129975.000000
50%       0.000000      0.000000     6.000000  2008.000000
163000.000000
75%       0.000000      0.000000     8.000000  2009.000000
214000.000000
max     738.000000  15500.000000    12.000000  2010.000000
755000.000000

[8 rows x 38 columns]
```

```
df_test.describe()
```

```
                Id   MSSubClass  LotFrontage       LotArea
OverallQual  \
count  1459.000000  1459.000000  1232.000000   1459.000000
1459.000000
```

```
mean   2190.000000    57.378341    68.580357   9819.161069
6.078821
std     421.321334    42.746880    22.376841   4955.517327
1.436812
min    1461.000000    20.000000    21.000000   1470.000000
1.000000
25%    1825.500000    20.000000    58.000000   7391.000000
5.000000
50%    2190.000000    50.000000    67.000000   9399.000000
6.000000
75%    2554.500000    70.000000    80.000000  11517.500000
7.000000
max    2919.000000   190.000000   200.000000  56600.000000
10.000000

       OverallCond    YearBuilt  YearRemodAdd   MasVnrArea
BsmtFinSF1  ...  \
count  1459.000000  1459.000000   1459.000000  1444.000000
1458.000000  ...
mean      5.553804  1971.357779   1983.662783   100.709141
439.203704  ...
std       1.113740    30.390071     21.130467   177.625900
455.268042  ...
min       1.000000  1879.000000   1950.000000     0.000000
0.000000  ...
25%       5.000000  1953.000000   1963.000000     0.000000
0.000000  ...
50%       5.000000  1973.000000   1992.000000     0.000000
350.500000  ...
75%       6.000000  2001.000000   2004.000000   164.000000
753.500000  ...
max       9.000000  2010.000000   2010.000000  1290.000000
4010.000000  ...

        GarageArea   WoodDeckSF  OpenPorchSF  EnclosedPorch
3SsnPorch  \
count  1458.000000  1459.000000  1459.000000    1459.000000
1459.000000
mean    472.768861    93.174777    48.313914      24.243317
1.794380
std     217.048611   127.744882    68.883364      67.227765
20.207842
min       0.000000     0.000000     0.000000       0.000000
0.000000
25%     318.000000     0.000000     0.000000       0.000000
0.000000
50%     480.000000     0.000000    28.000000       0.000000
0.000000
75%     576.000000   168.000000    72.000000       0.000000
0.000000
```

```
max    1488.000000  1424.000000   742.000000    1012.000000
360.000000

       ScreenPorch     PoolArea       MiscVal       MoSold
YrSold
count  1459.000000  1459.000000   1459.000000  1459.000000
1459.000000
mean     17.064428     1.744345     58.167923     6.104181
2007.769705
std      56.609763    30.491646    630.806978     2.722432
1.301740
min       0.000000     0.000000      0.000000     1.000000
2006.000000
25%       0.000000     0.000000      0.000000     4.000000
2007.000000
50%       0.000000     0.000000      0.000000     6.000000
2008.000000
75%       0.000000     0.000000      0.000000     8.000000
2009.000000
max     576.000000   800.000000  17000.000000    12.000000
2010.000000

[8 rows x 37 columns]
```

```
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
```

```
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
```

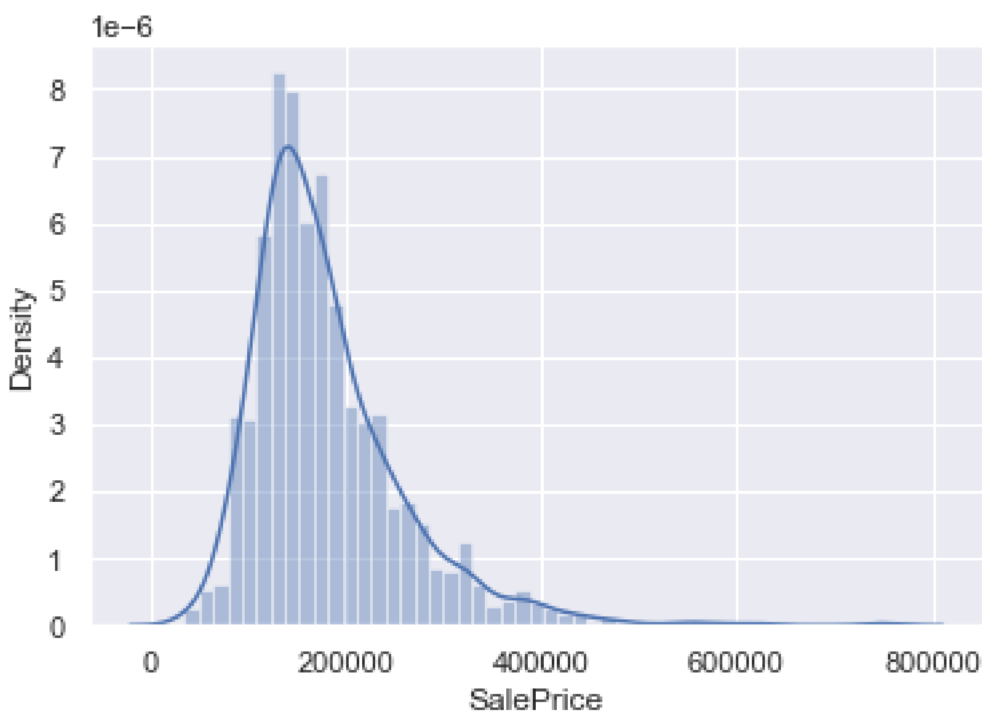
```
df_train['SalePrice'].describe()
```

```
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
```

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

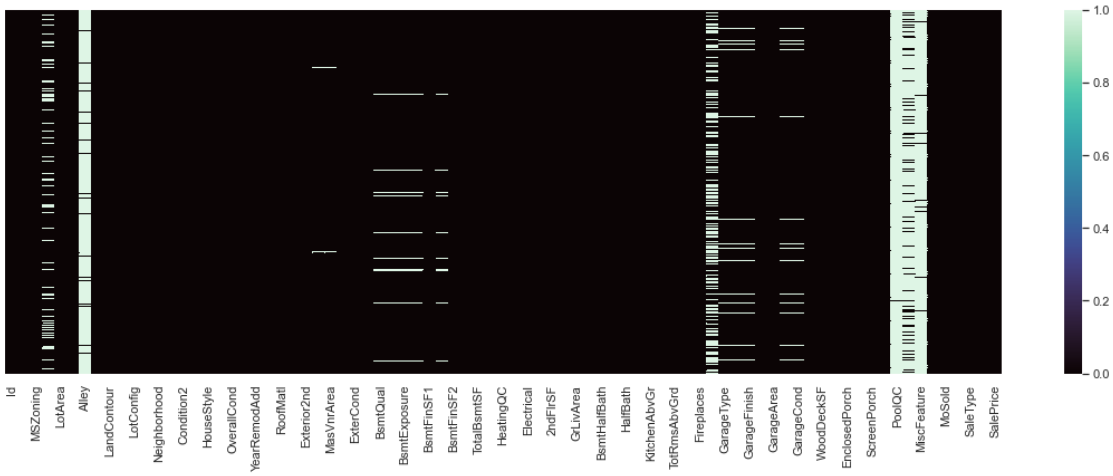
```
C:\Users\16095\anaconda3\lib\site-packages\seaborn\
distributions.py:2557: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your
code to use either `displot` (a figure-level function with similar
flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

```
Skewness: 1.882876
Kurtosis: 6.536282
```



```python
plt.figure(figsize=(20,6))
sns.heatmap(df_train.isnull(),yticklabels=False,cbar=True,cmap='mako')
```

```
<AxesSubplot:>
```




```python
total_null = df_train.isnull().sum().sort_values(ascending=False)
#First sum and order all null values for each variable
percentage =
(df_train.isnull().sum()/df_train.isnull().count()).sort_values(ascend
ing=False) #Get the percentage
missing_data = pd.concat([total_null, percentage], axis=1,
keys=['Total', 'Percentage'])
missing_data.head(20)
```

	Total	Percentage
PoolQC	1453	0.995205
MiscFeature	1406	0.963014
Alley	1369	0.937671
Fence	1179	0.807534
FireplaceQu	690	0.472603
LotFrontage	259	0.177397
GarageYrBlt	81	0.055479
GarageCond	81	0.055479
GarageType	81	0.055479
GarageFinish	81	0.055479
GarageQual	81	0.055479
BsmtFinType2	38	0.026027
BsmtExposure	38	0.026027
BsmtQual	37	0.025342
BsmtCond	37	0.025342
BsmtFinType1	37	0.025342
MasVnrArea	8	0.005479
MasVnrType	8	0.005479
Electrical	1	0.000685
Id	0	0.000000

Categorical

```
categ_vars_ls = ['PoolQC', 'MiscFeature', 'Alley', 'Fence',
                 'FireplaceQu', 'GarageType', 'GarageFinish',
'GarageQual',
                 'GarageCond', 'BsmtQual', 'BsmtCond', 'BsmtExposure',

                 'BsmtFinType1', 'BsmtFinType2', 'MasVnrType']

# Clean train set
for var in categ_vars_ls:
    df_train[var].fillna('None', inplace=True)

# Clean test set
for var in categ_vars_ls:
    df_test[var].fillna('None', inplace=True)
```

Numerical

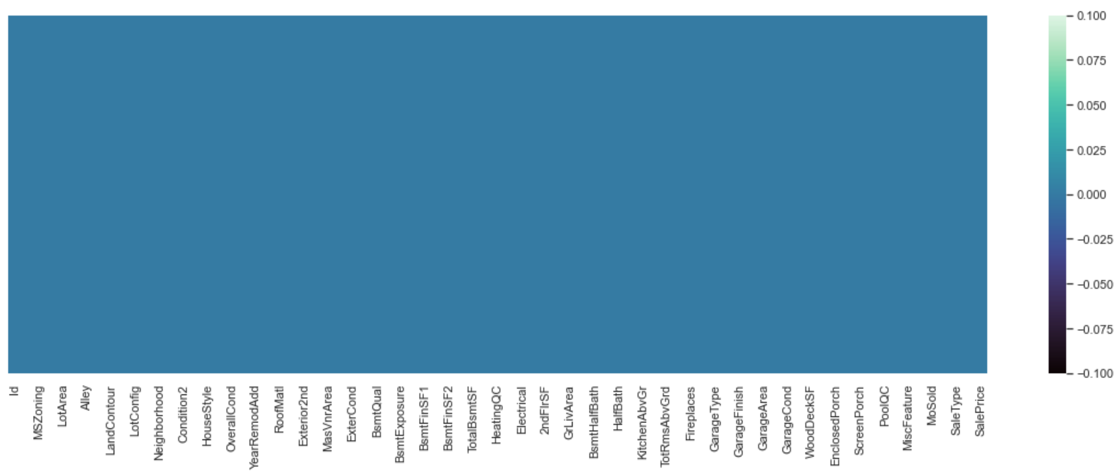
```
num_vars_ls = ['GarageArea', 'GarageCars', 'BsmtFinSF1', 'BsmtFinSF2',

               'BsmtUnfSF', 'TotalBsmtSF', 'BsmtFullBath',
'BsmtHalfBath',
               'MasVnrArea']

# Clean train set
for var in num_vars_ls:
    df_train[var].fillna(0, inplace=True)

# Clean test set
for var in num_vars_ls:
    df_test[var].fillna(0, inplace=True)

vars_ls1 = ['Functional', 'MSZoning', 'Electrical', 'KitchenQual',
'Exterior1st',
            'Exterior2nd', 'SaleType', 'Utilities']

imputer = SimpleImputer(strategy='most_frequent')

# Clean train set
df_train[vars_ls1] =
pd.DataFrame(imputer.fit_transform(df_train[vars_ls1]),
index=df_train.index)

# Clean test set
df_test[vars_ls1] =
pd.DataFrame(imputer.fit_transform(df_test[vars_ls1]),
index=df_test.index)
```

```python
train_average_house_neighb = df_train.groupby('Neighborhood')
['LotFrontage']
test_average_house_neighb = df_test.groupby('Neighborhood')
['LotFrontage']

# Clean train set
df_train['LotFrontage'].fillna(train_average_house_neighb.transform(la
mbda x: x.fillna(x.mean())), inplace=True)

# Clean test set
df_test['LotFrontage'].fillna(test_average_house_neighb.transform(lamb
da x: x.fillna(x.mean())), inplace=True)

# Clean train set
df_train['GarageYrBlt'] =
df_train['GarageYrBlt'].fillna(df_train['YearBuilt'])

# Clean test set
df_test['GarageYrBlt'] =
df_test['GarageYrBlt'].fillna(df_test['YearBuilt'])

## NA Check: Verify that we covered all 'NAs' in our data
print(f'Number of NAs in train df: {sum(df_train.isnull().sum())}')
print(f'Number of NAs in test df: {sum(df_test.isnull().sum())}')
```

```
Number of NAs in train df: 0
Number of NAs in test df: 0
```

```python
plt.figure(figsize=(20,6))
sns.heatmap(df_train.isnull(),yticklabels=False,cbar=True,cmap='mako')
```

```
<AxesSubplot:>
```



Investigate potential features & outliers

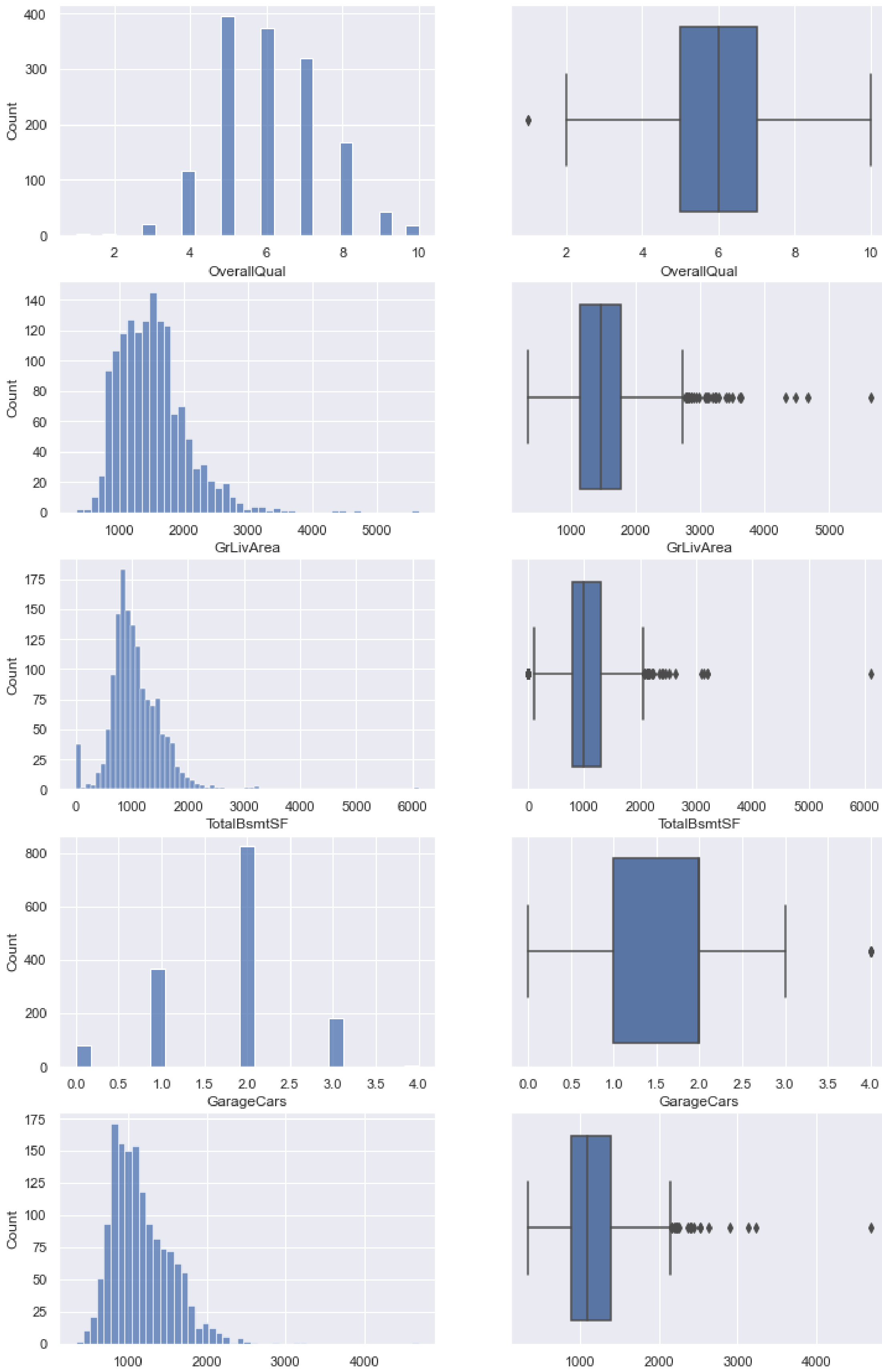
Below, We can see a few of the highest correlating predictors of SalePrice. Based on these features, it is obvious that usable square footage cumulatively amounts to the highest correlation to SalePrice (GrLivArea, TotalBsmtSF, 1stFlrSF, GarageArea). Other discrete and categorical variables (OverallQual, GarageCars, FullBath, TotRmsAdvGrd) influence the dependent variable as well.

```
corr_mat = df_train.corr().SalePrice.sort_values(ascending=False)
corr_mat.head(10)
```

```
SalePrice       1.000000
OverallQual     0.790982
GrLivArea       0.708624
GarageCars      0.640409
GarageArea      0.623431
TotalBsmtSF     0.613581
1stFlrSF        0.605852
FullBath        0.560664
TotRmsAbvGrd    0.533723
YearBuilt       0.522897
Name: SalePrice, dtype: float64
```

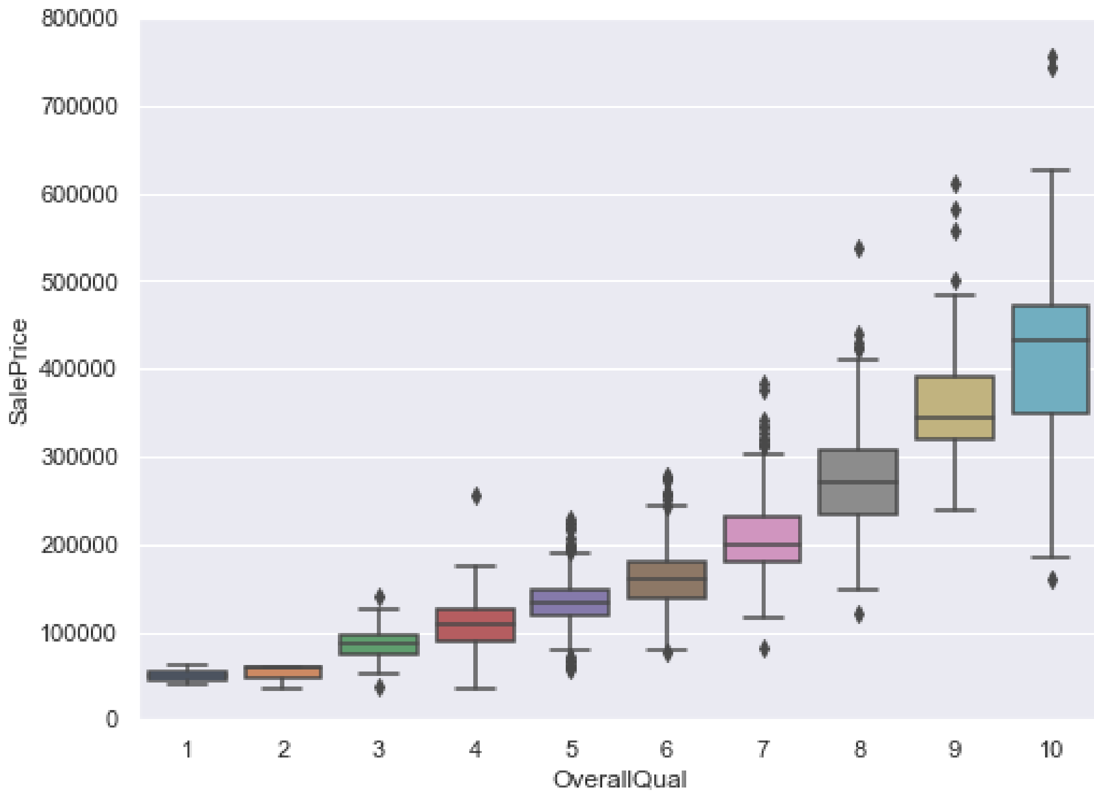
Below we can see the distribution of a few of these variables and assess how outliers may impact the data.

```
cor_features = ['OverallQual', 'GrLivArea', 'TotalBsmtSF',
'GarageCars', '1stFlrSF', 'YearBuilt' ]

n = len(cor_features)

fig = plt.figure(figsize=(6*2, 4*n))
# add 2 graph for each column variable
gs = fig.add_gridspec(n, 2)
ax = [[fig.add_subplot(gs[i, j]) for j in range(2)] for i in range(n)]

for i in range(n):
    sns.histplot(x=cor_features[i], data=df_train, ax=ax[i][0])
    sns.boxplot(x=cor_features[i], data=df_train, ax=ax[i][1])

plt.show()
```



Count
OverallQual
OverallQual
Count
GrLivArea
GrLivArea
Count
TotalBsmtSF
TotalBsmtSF
Count
GarageCars
GarageCars
Count
1stFlrSF
1stFlrSF

```python
# OverallQual and SalePrice
data = pd.concat([df_train['SalePrice'], df_train['OverallQual']],
axis=1)
f, ax = plt.subplots(figsize=(8, 6))
fig = sns.boxplot(x='OverallQual', y="SalePrice", data=data)
fig.axis(ymin=0, ymax=800000);
```




```python
# GrLivArea and SalesPrice
sns.set_style('darkgrid')
plt.figure(figsize=(8, 6))
sns.scatterplot(x='GrLivArea', y='SalePrice', data=df_train)
title = plt.title('House Price vs. Above Ground Living Area')
```

The scatter plot above reveals a few outliers where a larger living area is recorded with a low sale price. These outliers can be removed to ensure they do not influence future models.

```
# Clean df_train (GrLiveArea)
outlier = df_train[(df_train.GrLivArea > 4000) & (df_train.SalePrice < 200000)].index
df_train.drop(outlier, axis=0, inplace=True)

# TotalBsmtSF and SalesPrice
sns.set_style('darkgrid')
plt.figure(figsize=(8, 6))
sns.scatterplot(x='TotalBsmtSF', y='SalePrice', data=df_train)
title = plt.title('House Price vs. Basement (sqft)')
```


```
# 1stFlrSF and SalesPrice
sns.set_style('darkgrid')
plt.figure(figsize=(8, 6))
sns.scatterplot(x='1stFlrSF', y='SalePrice', data=df_train)
title = plt.title('House Price vs. First Floor (sqft)')
```

Feature Creation

Feature creation is likely to be a useful approach to finding more potent predictors in this data set. Based on the list of high correlating variables, it is apparent that features representing usable square feet are strong predictors and can be merged to create a stronger predictive feature. Additionally, the current dataframe seems to categorically discriminate based on above or below ground features. Combining some of high correlation variable, both above and below ground, may yield an overall stronger predictor. Finally, YearBuilt showed up a on the bottom of the correlation list with a comparatively low correlation. However, it remains an interesting feature to explore given some obvious and real world implications. Ideally, it would be nice to see in depth how larger renovations might impact the value of older homes. However, the data makes it difficult to define what renovation may have occurred.

Potentially interesting new predictors include:

-Total Square Feet of living Space (Below and Above ground)

-Total Number of Bathrooms (Below and Above Ground)

-Age of House when sold

```
# Total Square Feet Column
df_train['TotalSqft'] = df_train['TotalBsmtSF'] + df_train['1stFlrSF']
```

```
+ df_train['2ndFlrSF']

df_test['TotalSqft'] = df_test['TotalBsmtSF'] + df_test['1stFlrSF'] +
df_test['2ndFlrSF']

# Total Bathrooms Column
df_train['TotalBath'] = df_train['FullBath'] +
df_train['BsmtFullBath'] + 0.5*(df_train['HalfBath'] +
df_train['BsmtHalfBath'])

df_test['TotalBath'] = df_test['FullBath'] + df_test['BsmtFullBath'] +
0.5*(df_test['HalfBath'] + df_test['BsmtHalfBath'])

# Age of House
df_train['HouseAge'] = df_train['YrSold'] - df_train['YearBuilt']

df_test['HouseAge'] = df_test['YrSold'] - df_test['YearBuilt']

# Check for new columns
df_train.head()
```

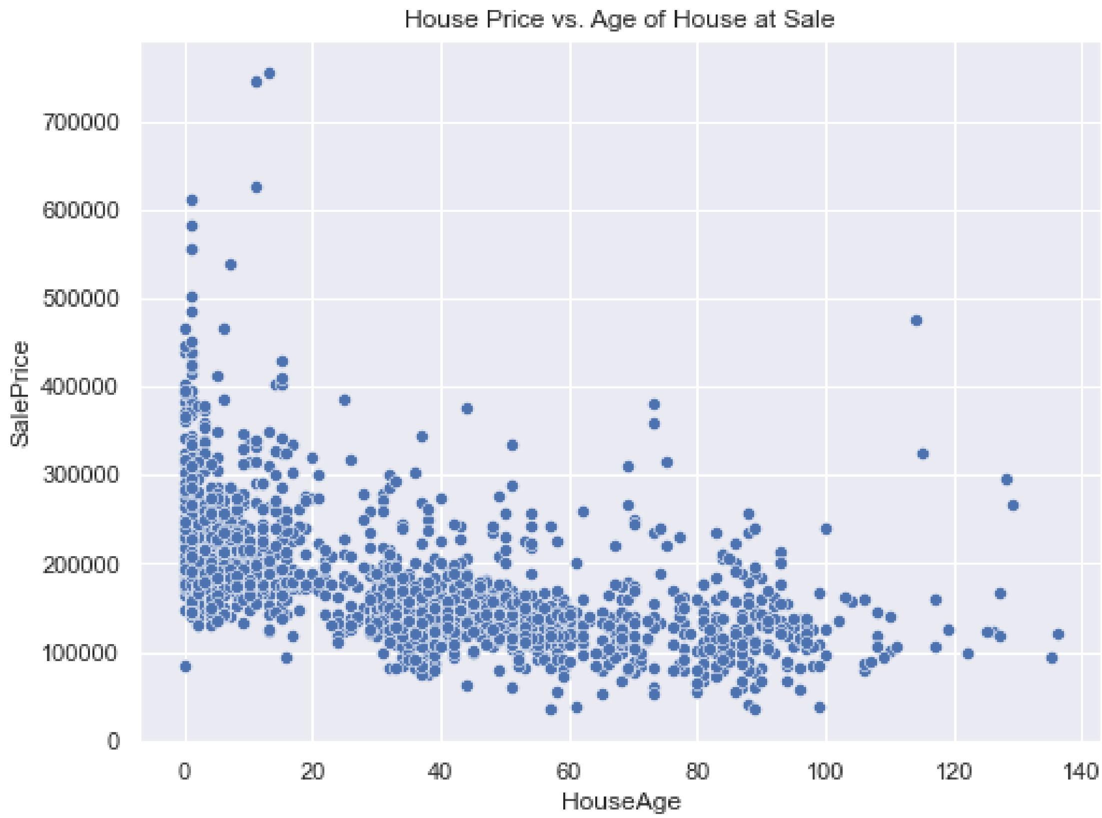
```
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape
\
0   1          60       RL         65.0     8450   Pave  None      Reg

1   2          20       RL         80.0     9600   Pave  None      Reg

2   3          60       RL         68.0    11250   Pave  None      IR1

3   4          70       RL         60.0     9550   Pave  None      IR1

4   5          60       RL         84.0    14260   Pave  None      IR1


  LandContour Utilities  ... MiscFeature MiscVal MoSold YrSold
SaleType  \
0         Lvl    AllPub  ...        None       0      2   2008
WD
1         Lvl    AllPub  ...        None       0      5   2007
WD
2         Lvl    AllPub  ...        None       0      9   2008
WD
3         Lvl    AllPub  ...        None       0      2   2006
WD
4         Lvl    AllPub  ...        None       0     12   2008
WD

  SaleCondition SalePrice  TotalSqft  TotalBath  HouseAge
0        Normal    208500       2566        3.5         5
1        Normal    181500       2524        2.5        31
```

```
2         Normal    223500       2706        3.5          7
3        Abnorml    140000       2473        2.0         91
4         Normal    250000       3343        3.5          8

[5 rows x 84 columns]
```

```
# TotalSqft and SalesPrice
sns.set_style('darkgrid')
plt.figure(figsize=(8, 6))
sns.scatterplot(x='TotalSqft', y='SalePrice', data=df_train)
title = plt.title('House Price vs. Total Living Space')
```


```
# TotalBath and SalesPrice
sns.set_style('darkgrid')
plt.figure(figsize=(8, 6))
sns.scatterplot(x='TotalBath', y='SalePrice', data=df_train)
title = plt.title('House Price vs. Total Bathrooms')
```


```
# HouseAge and SalesPrice
sns.set_style('darkgrid')
plt.figure(figsize=(8, 6))
sns.scatterplot(x='HouseAge', y='SalePrice', data=df_train)
title = plt.title('House Price vs. Age of House at Sale')
```




```
corr_mat2 = df_train.corr().SalePrice.sort_values(ascending=False)
corr_mat2.head(10)
```

```
SalePrice      1.000000
TotalSqft      0.832877
OverallQual    0.795774
GrLivArea      0.734968
TotalBsmtSF    0.651153
GarageCars     0.641047
TotalBath      0.635896
1stFlrSF       0.631530
GarageArea     0.629217
FullBath       0.562165
Name: SalePrice, dtype: float64
```

Skewness Check (train)

```
numerics = ['int16', 'int32', 'int64', 'float16', 'float32',
'float64']

dataset_numeric = df_train.select_dtypes(include=numerics)

dataset_numeric.shape
```

```
(1458, 41)
```

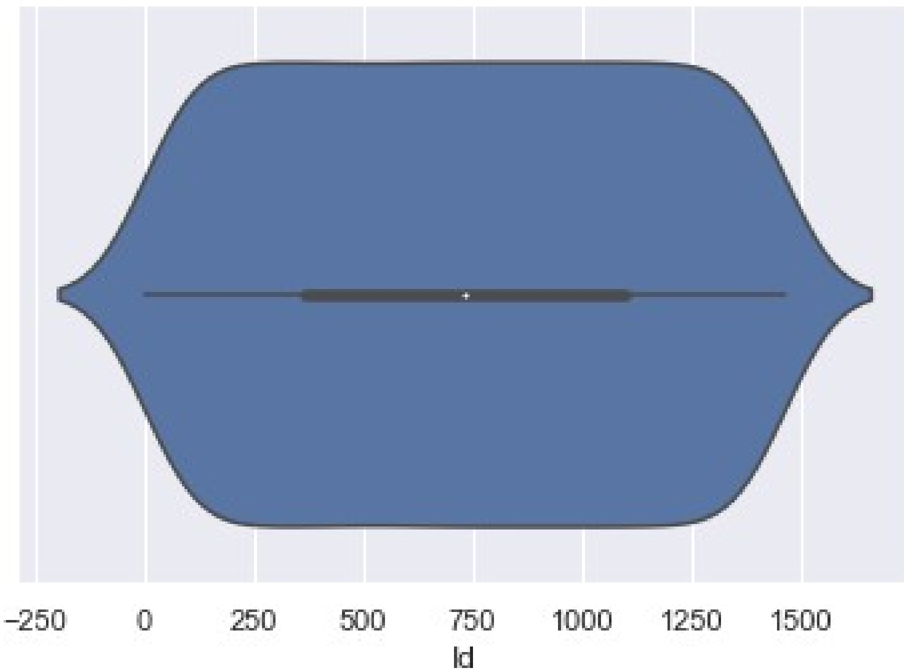
```
dataset_numeric.skew()
```

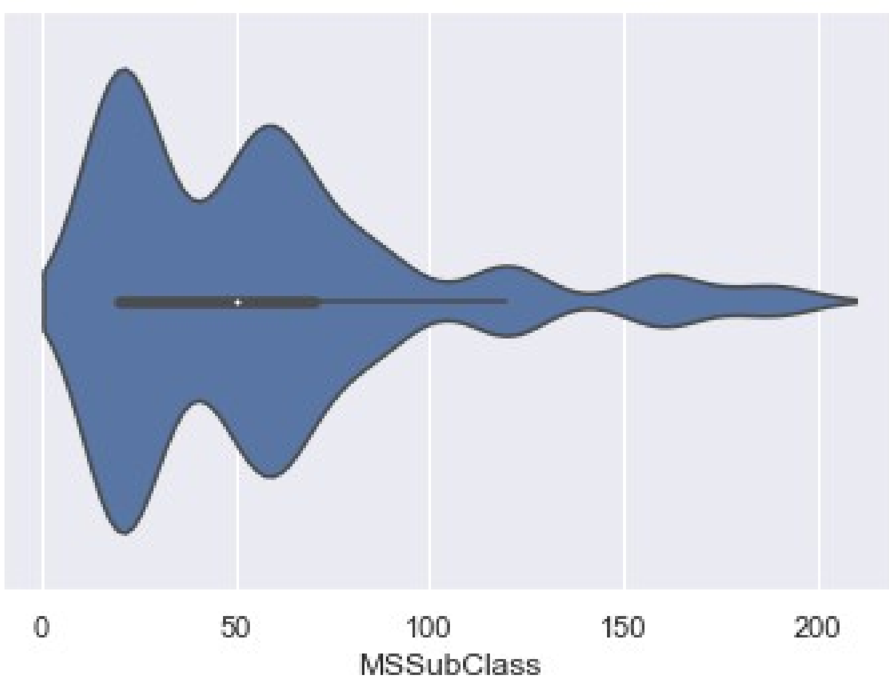
```
Id                0.000165
MSSubClass        1.407011
LotFrontage       1.494021
LotArea          12.573925
OverallQual       0.200786
OverallCond       0.691035
YearBuilt        -0.612295
YearRemodAdd     -0.501838
MasVnrArea        2.696329
BsmtFinSF1        0.764789
BsmtFinSF2        4.251925
BsmtUnfSF         0.920903
TotalBsmtSF       0.511703
1stFlrSF          0.887637
2ndFlrSF          0.812957
LowQualFinSF      9.004955
GrLivArea         1.010992
BsmtFullBath      0.590358
BsmtHalfBath      4.100114
FullBath          0.031271
HalfBath          0.680051
BedroomAbvGr      0.212325
KitchenAbvGr      4.484883
TotRmsAbvGrd      0.660502
Fireplaces        0.632060
GarageYrBlt      -0.693237
GarageCars       -0.342377
GarageArea        0.131748
WoodDeckSF        1.545805
OpenPorchSF       2.339829
EnclosedPorch     3.087164
3SsnPorch        10.297106
ScreenPorch       4.118929
PoolArea         15.948945
MiscVal          24.460085
MoSold            0.215432
YrSold            0.095420
SalePrice         1.881296
TotalSqft         0.804321
TotalBath         0.246687
HouseAge          0.607894
dtype: float64
```

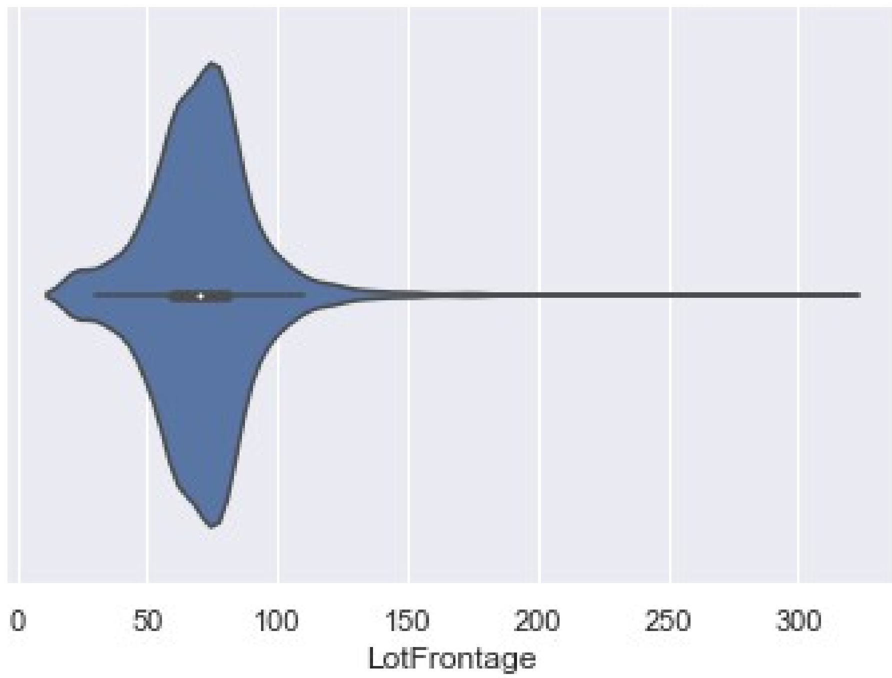
```
cols = dataset_numeric.columns
```

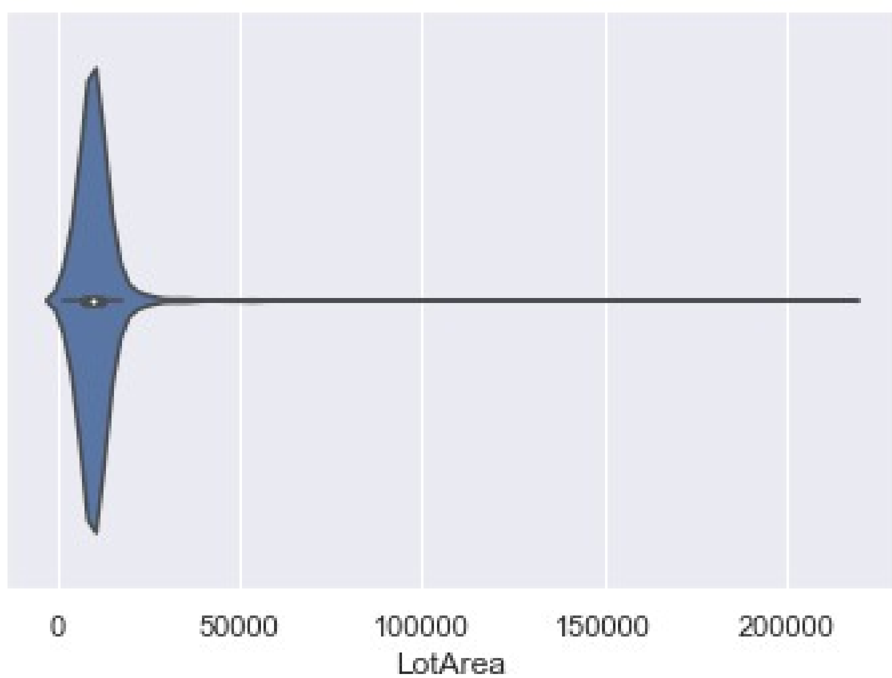
```
for c in cols:
  sns.violinplot(x=dataset_numeric[c])
```

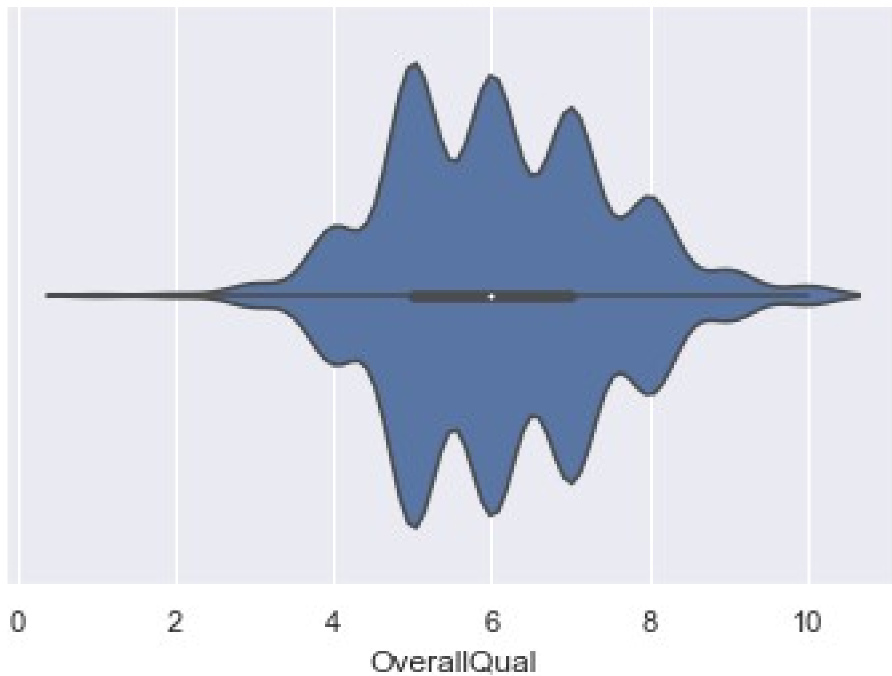
```
plt.xlabel(c)
plt.show()
```

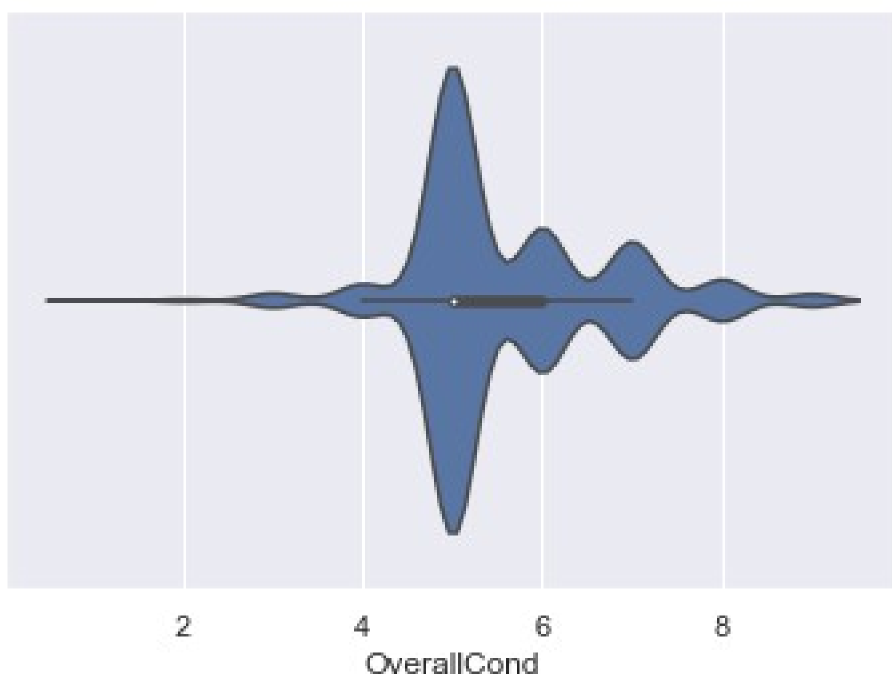


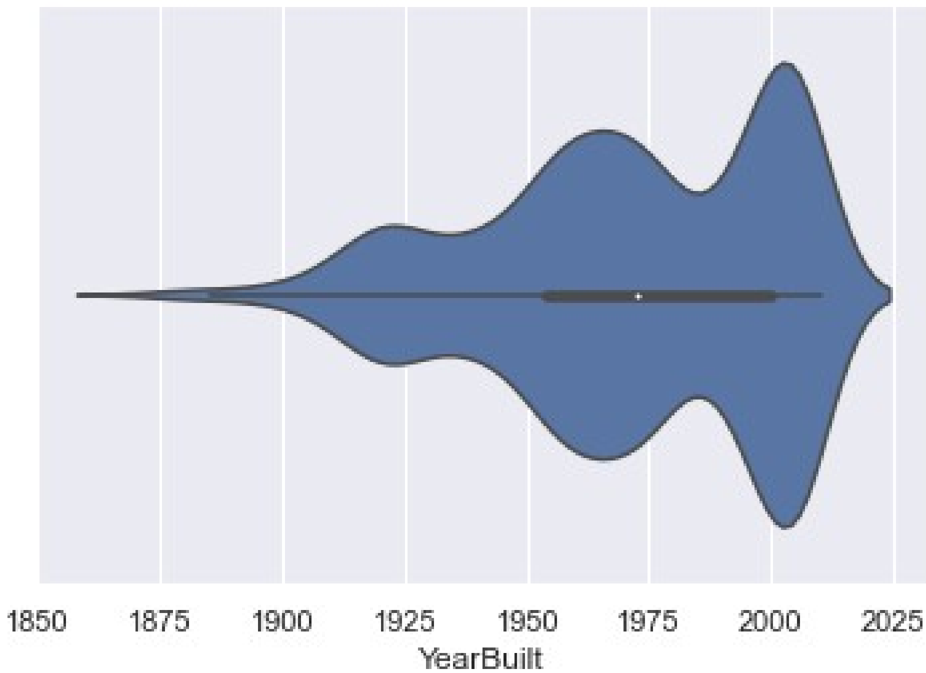


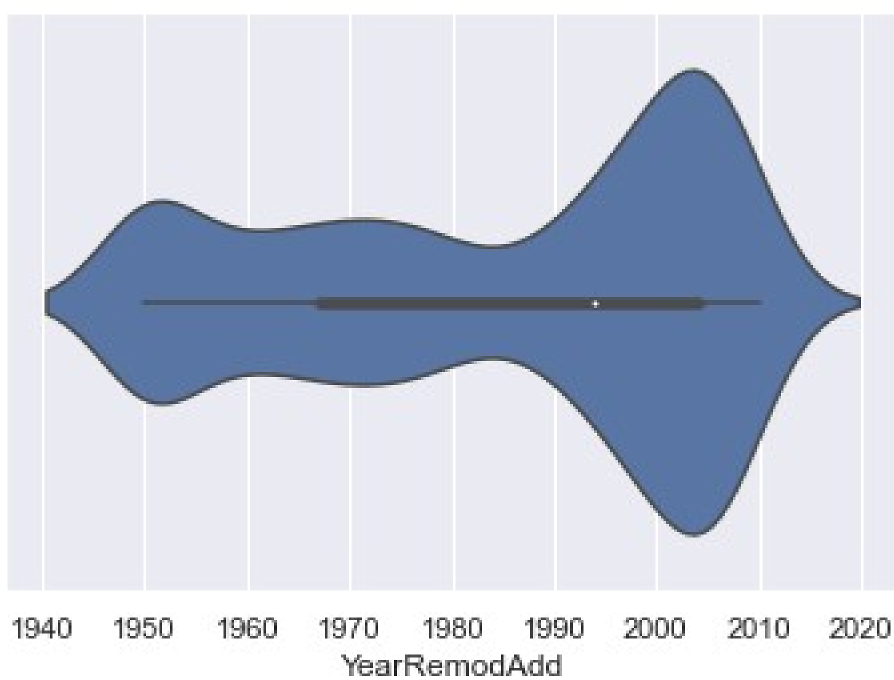

0
50
100
150
200
250
300
LotFrontage

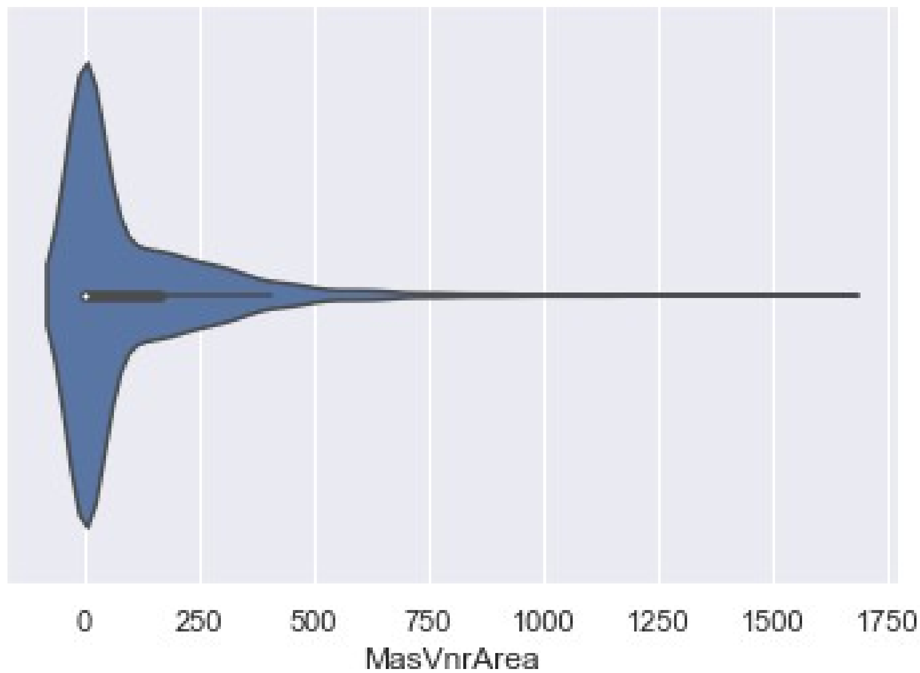

0
50000
100000
150000
200000
LotArea

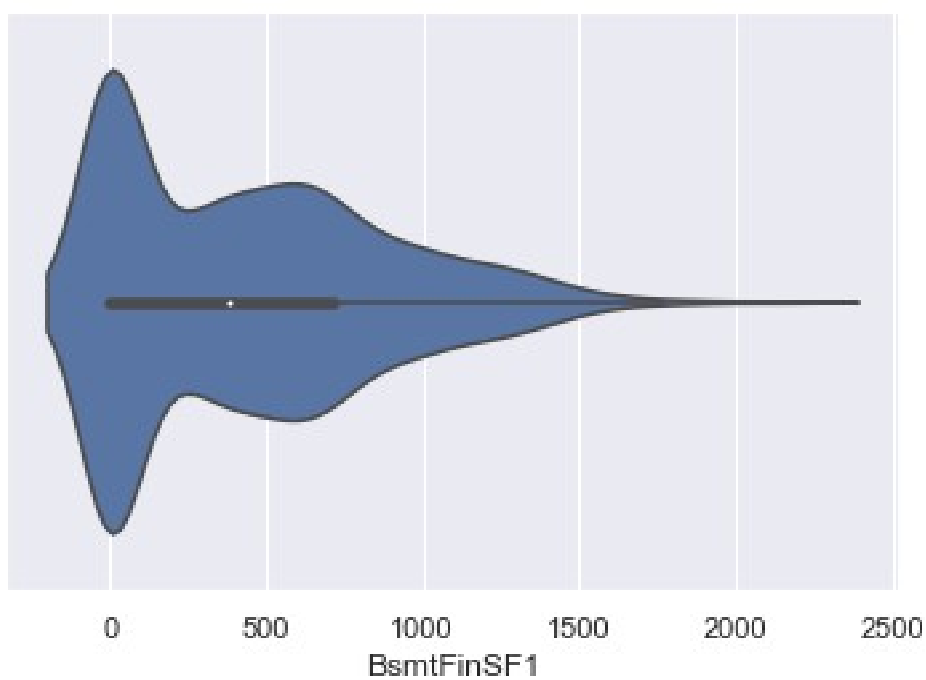

0
2
4
6
8
10
OverallQual

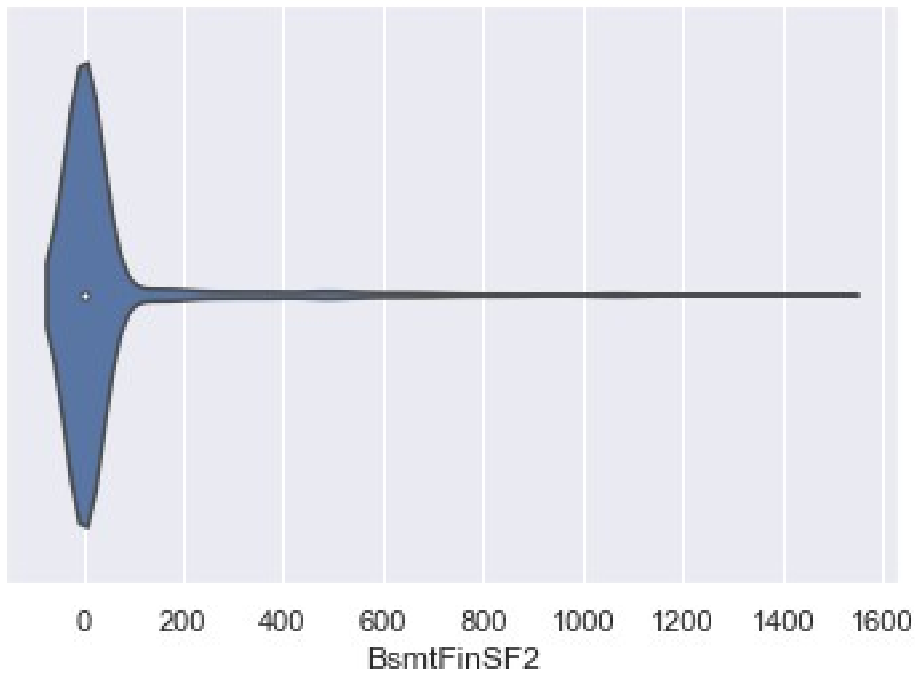

2
4
6
8
OverallCond

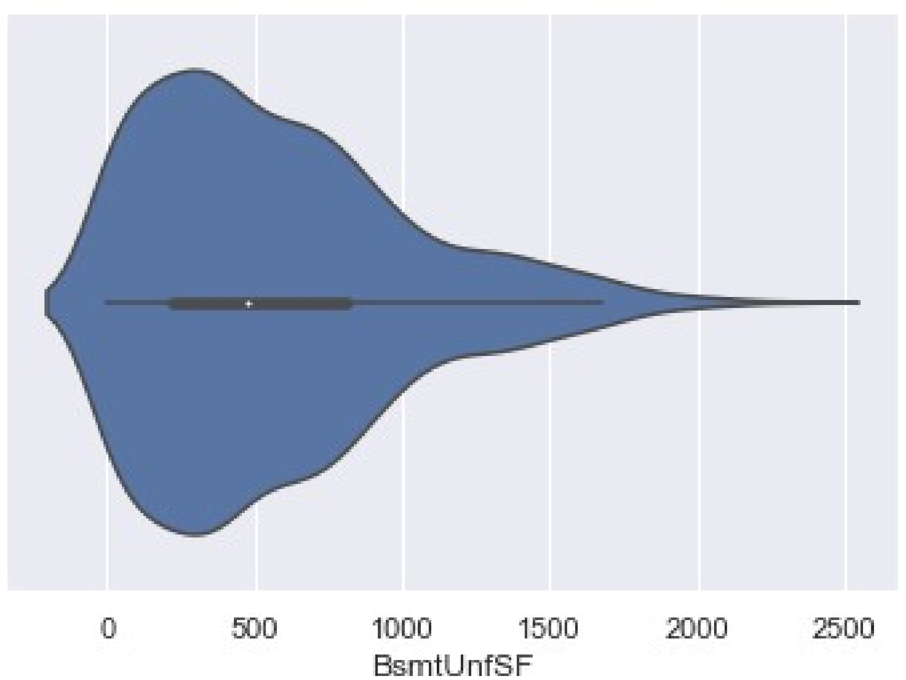

1850
1875
1900
1925
1950
1975
2000
2025
YearBuilt

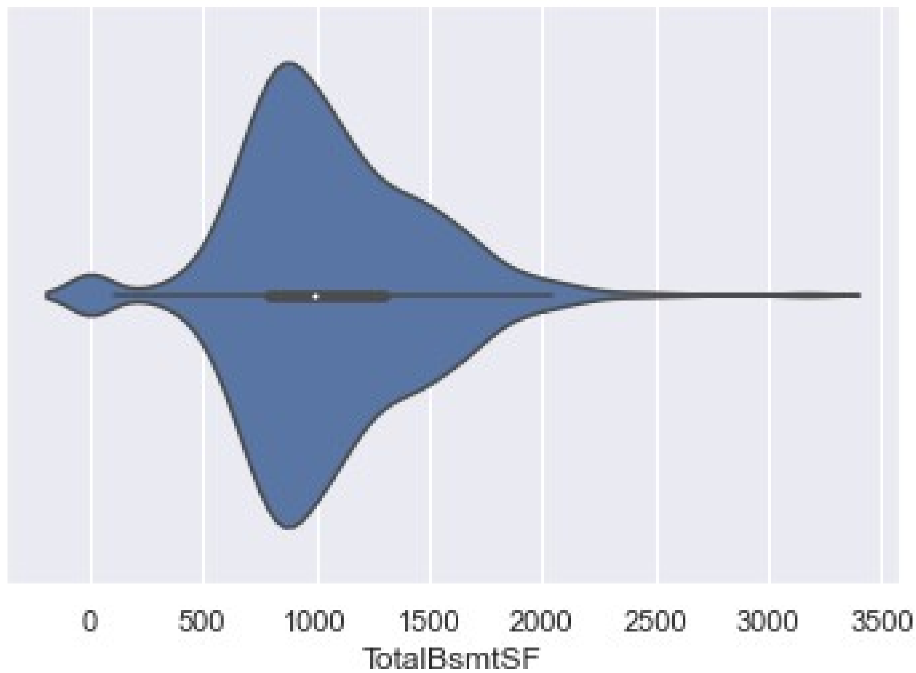

1940
1950
1960
1970
1980
1990
2000
2010
2020
YearRemodAdd

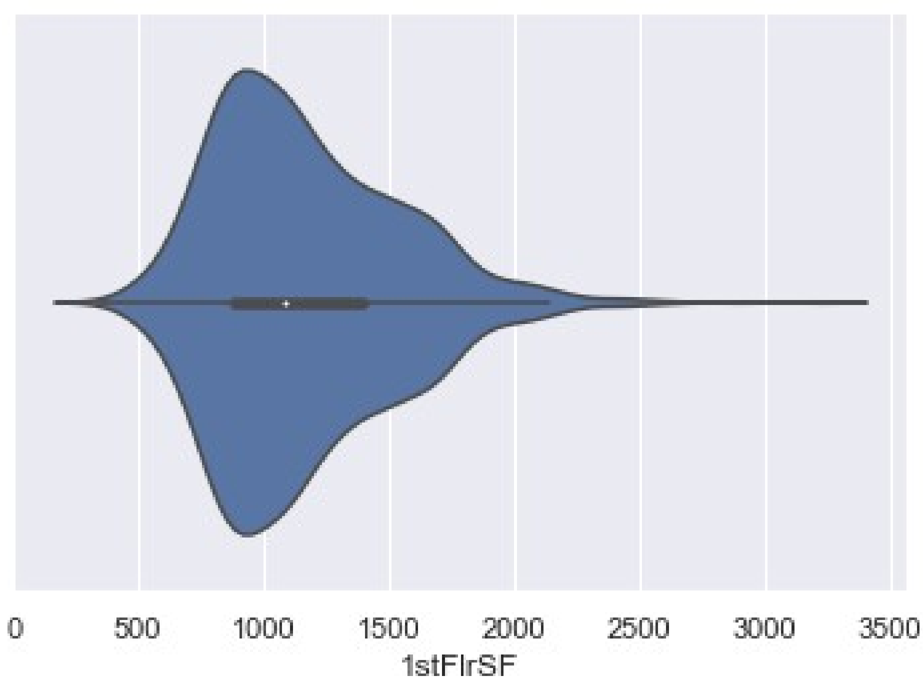

0
250
500
750
1000
1250
1500
1750
MasVnrArea

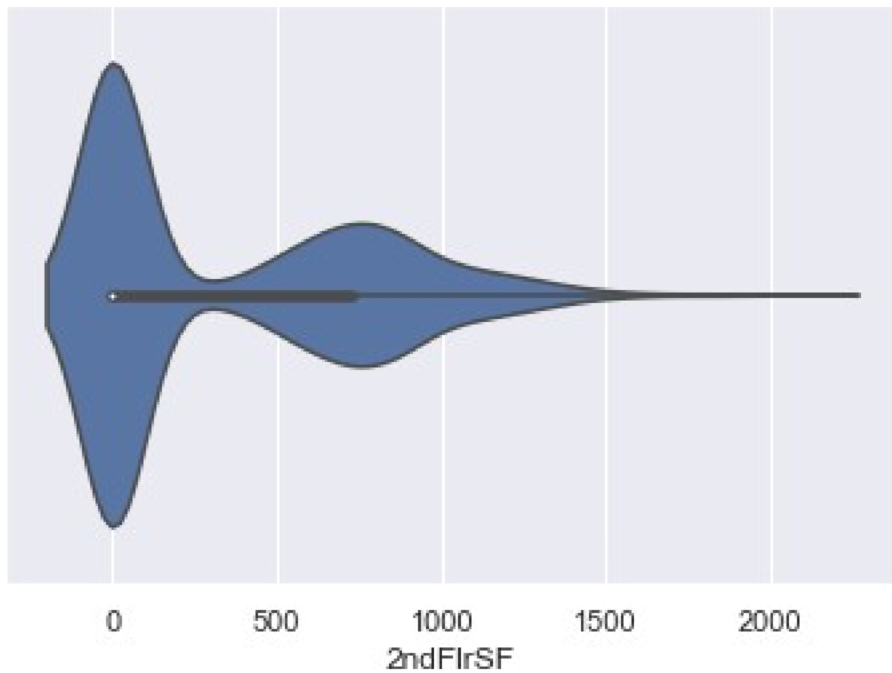

0
500
1000
1500
2000
2500
BsmtFinSF1

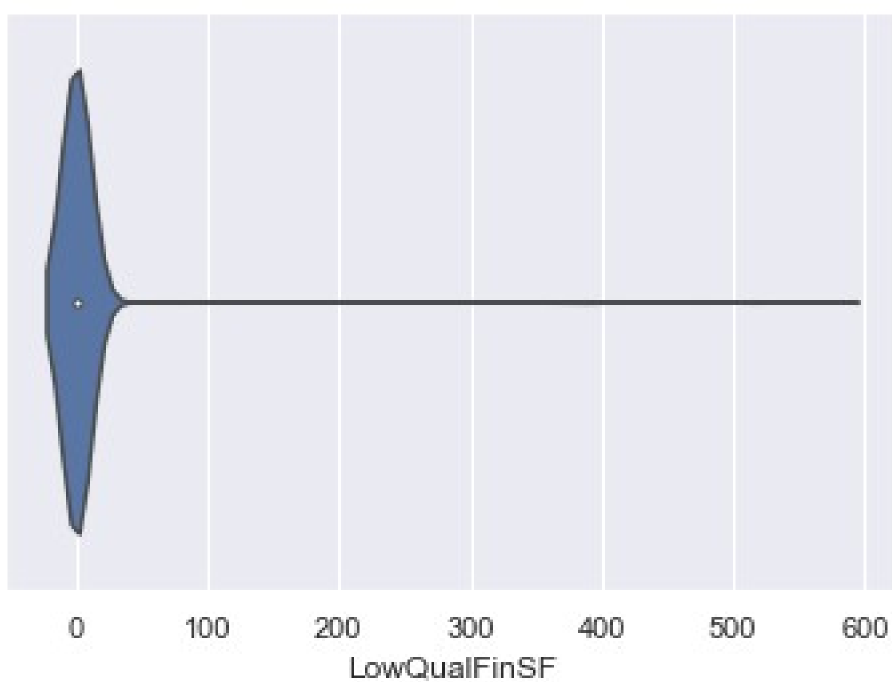

0
200
400
600
800
1000
1200
1400
1600
BsmtFinSF2

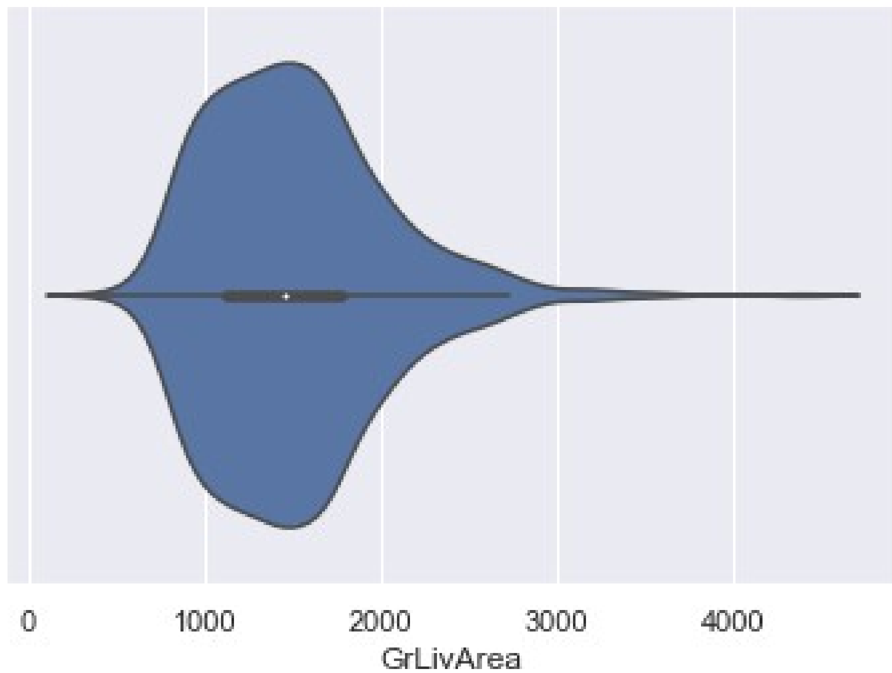

0
500
1000
1500
2000
2500
BsmtUnfSF


0
500
1000
1500
2000
2500
3000
3500
TotalBsmtSF

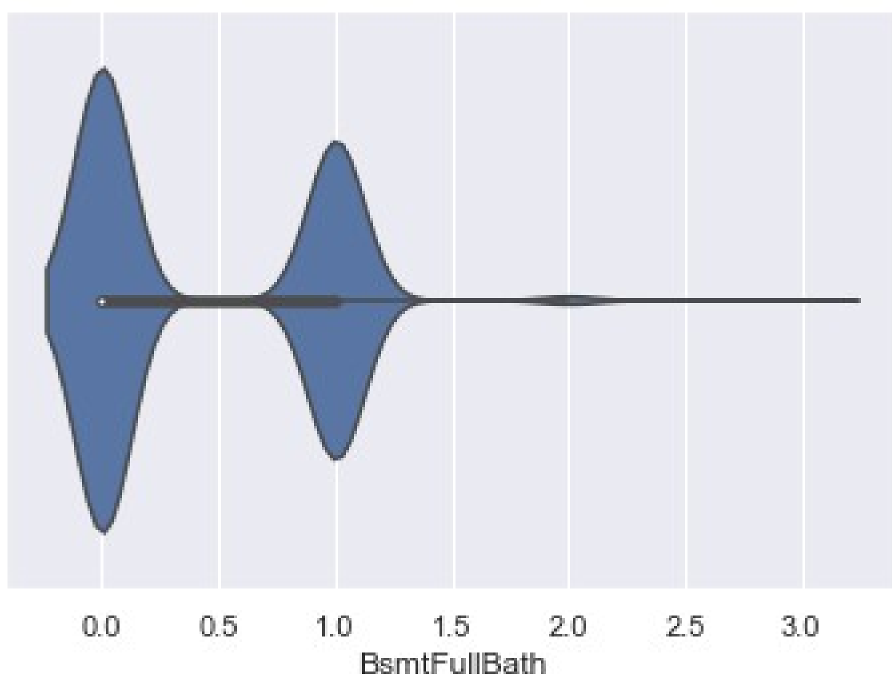

0
500
1000
1500
2000
2500
3000
3500
1stFlrSF


0
500
1000
1500
2000
2ndFlrSF

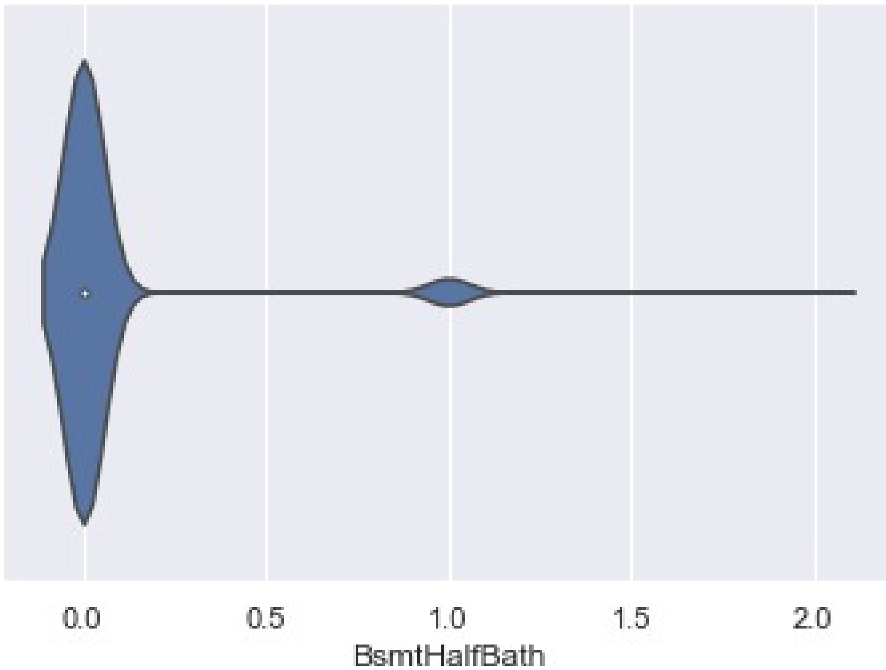

0
100
200
300
400
500
600
LowQualFinSF

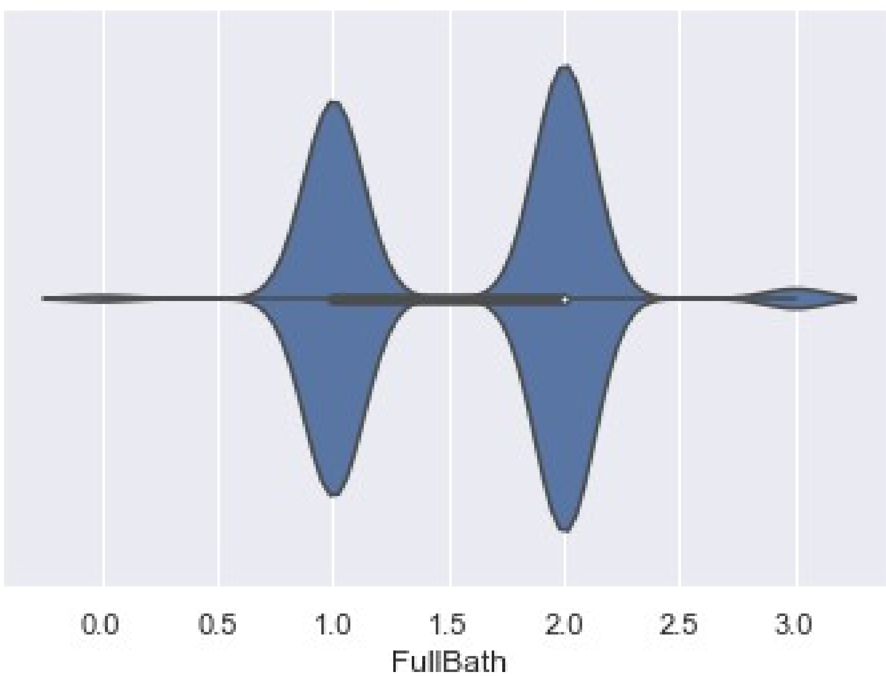


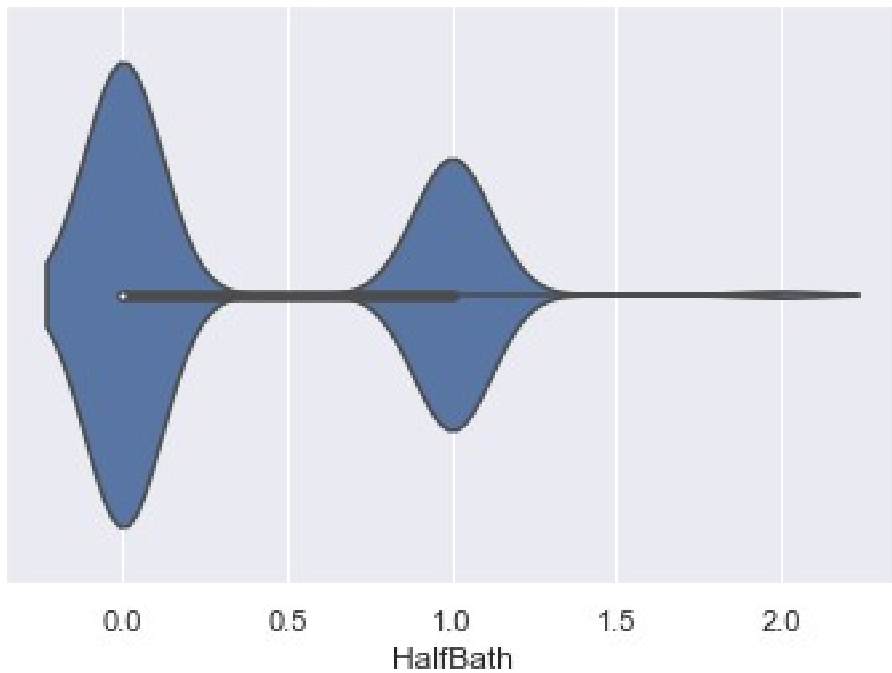
0
1000
2000
3000
4000
GrLivArea



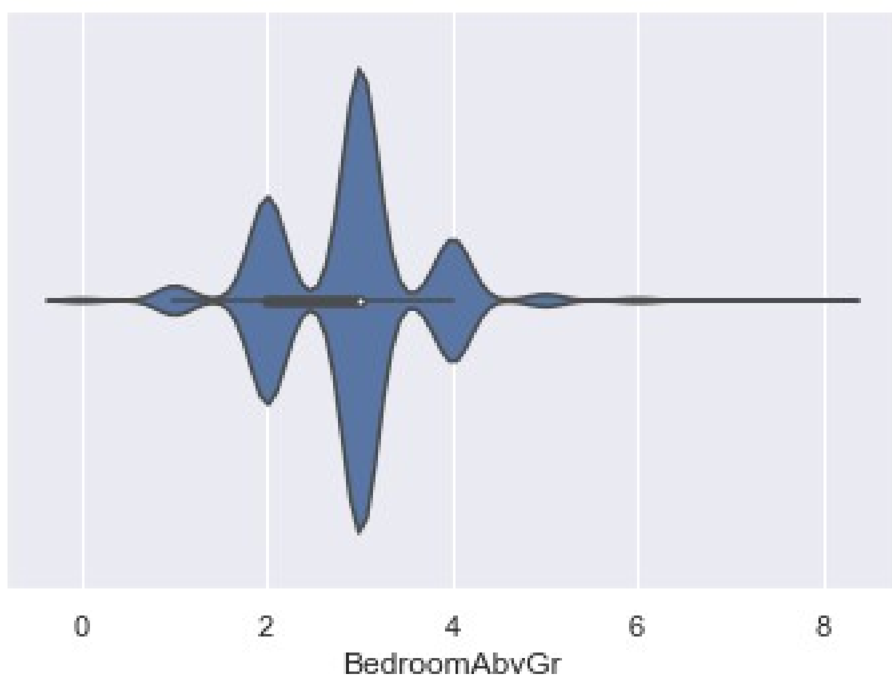
0.0
0.5
1.0
1.5
2.0
2.5
3.0
BsmtFullBath


0.0
0.5
1.0
1.5
2.0
BsmtHalfBath

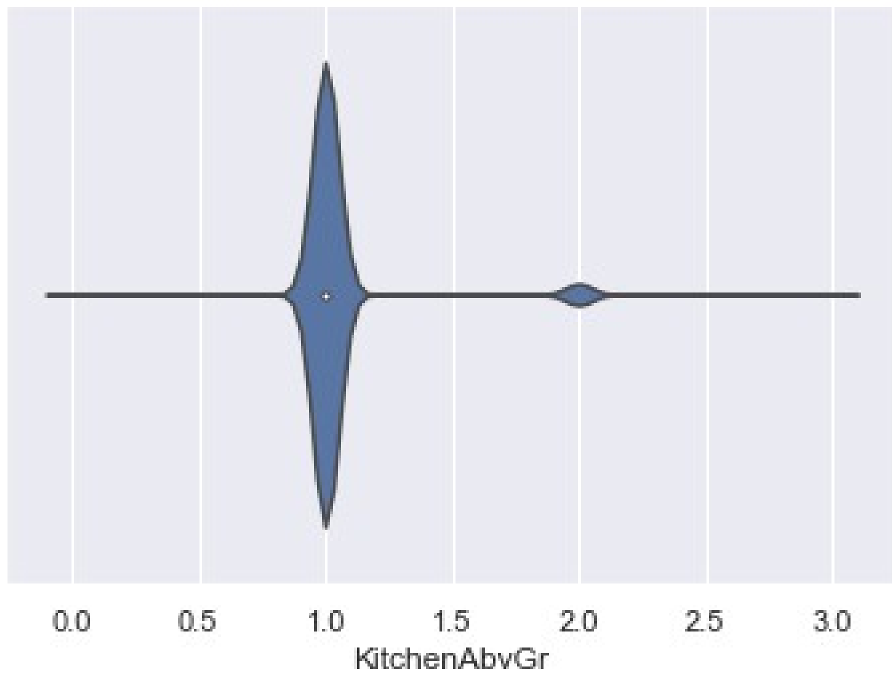

0.0
0.5
1.0
1.5
2.0
2.5
3.0
FullBath



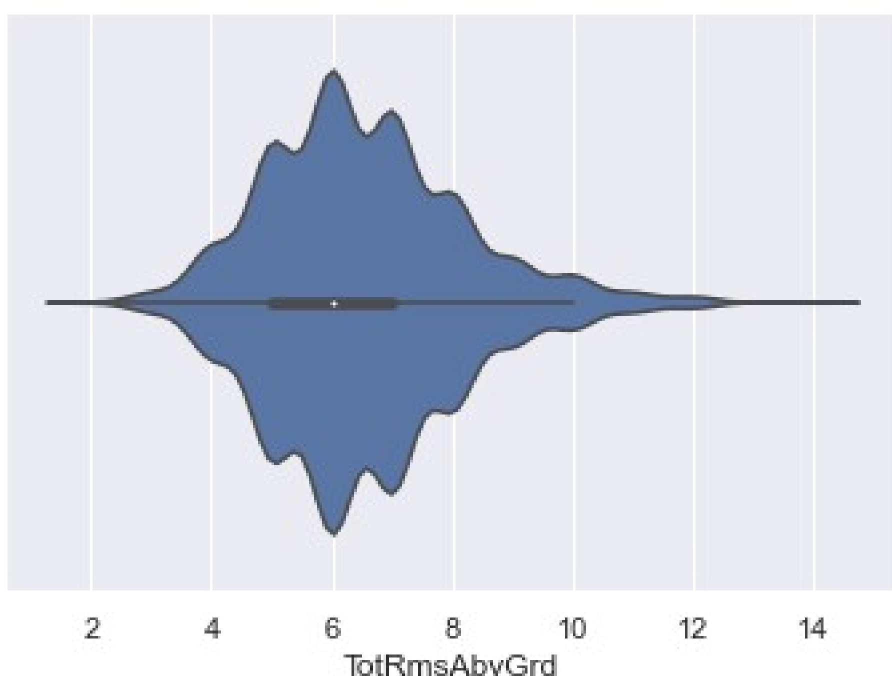
0.0
0.5
1.0
1.5
2.0
HalfBath



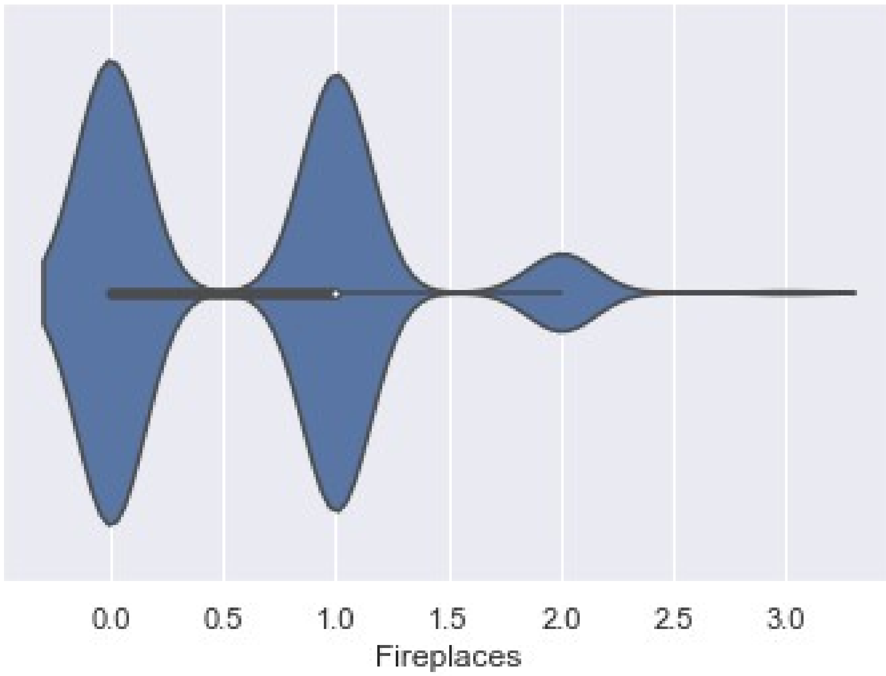
0
2
4
6
8
BedroomAbvGr

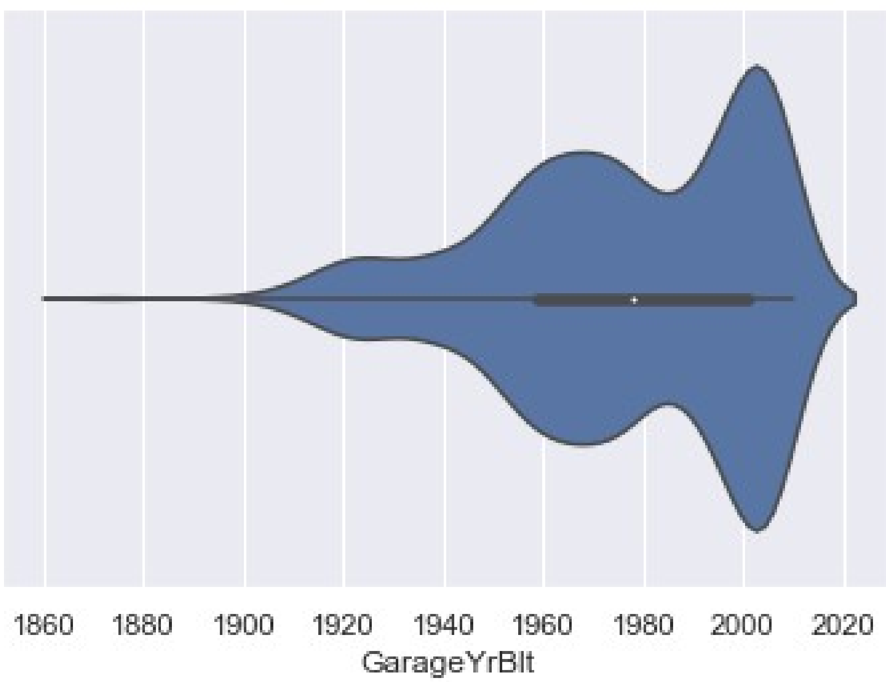


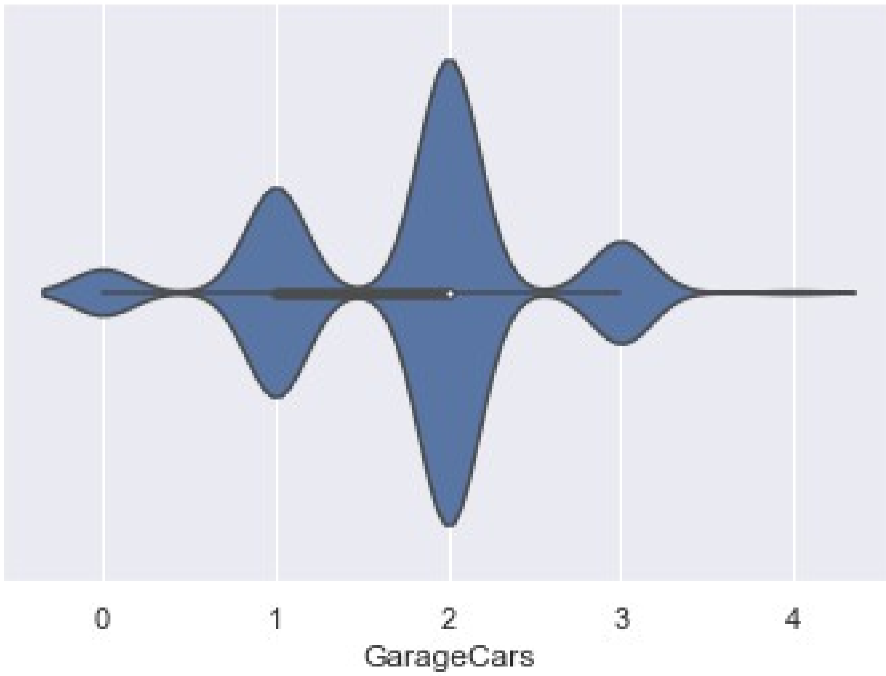
0.0
0.5
1.0
1.5
2.0
2.5
3.0
KitchenAbvGr

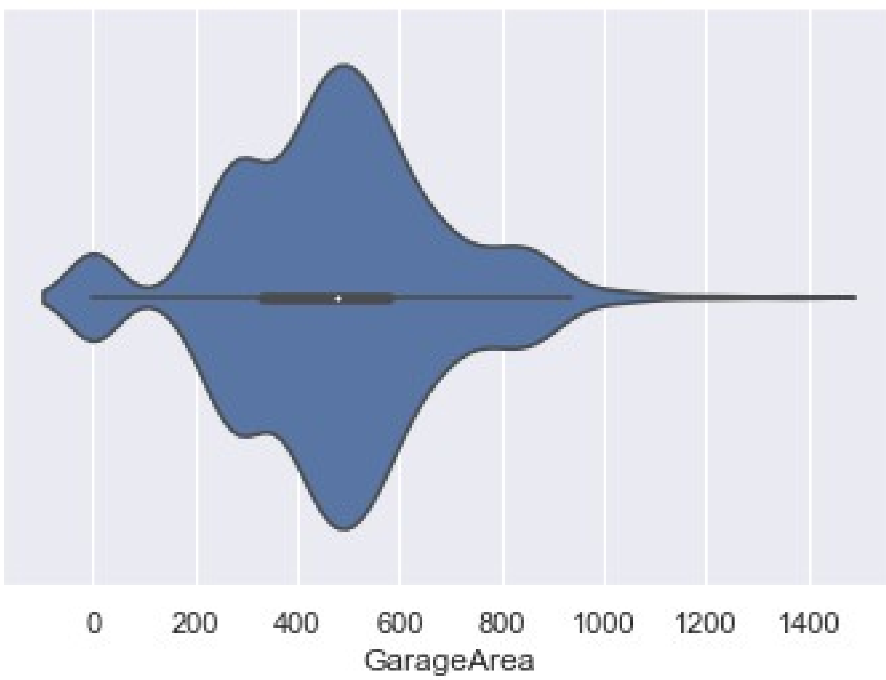


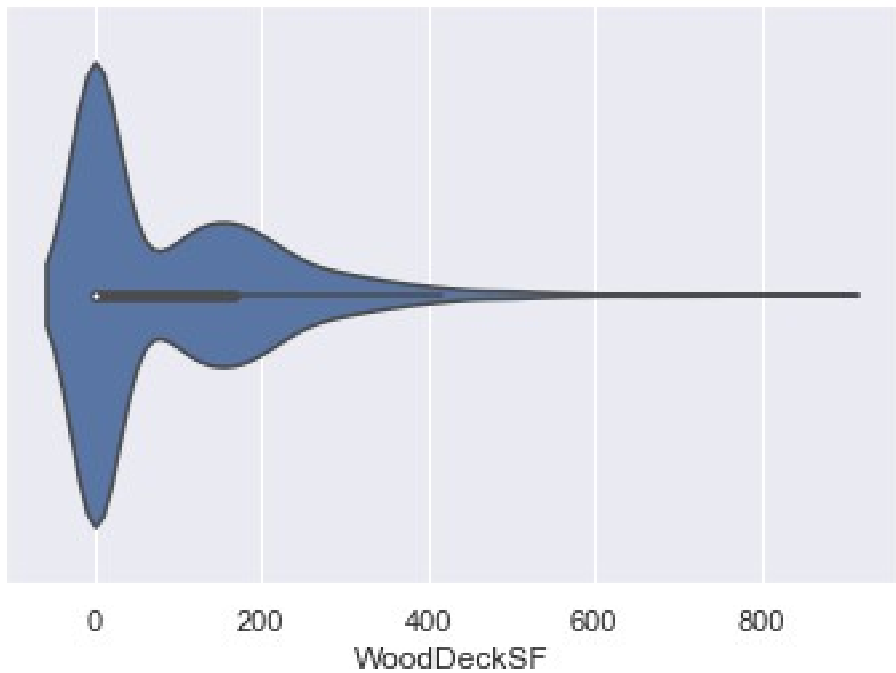
2
4
6
8
10
12
14
TotRmsAbvGrd

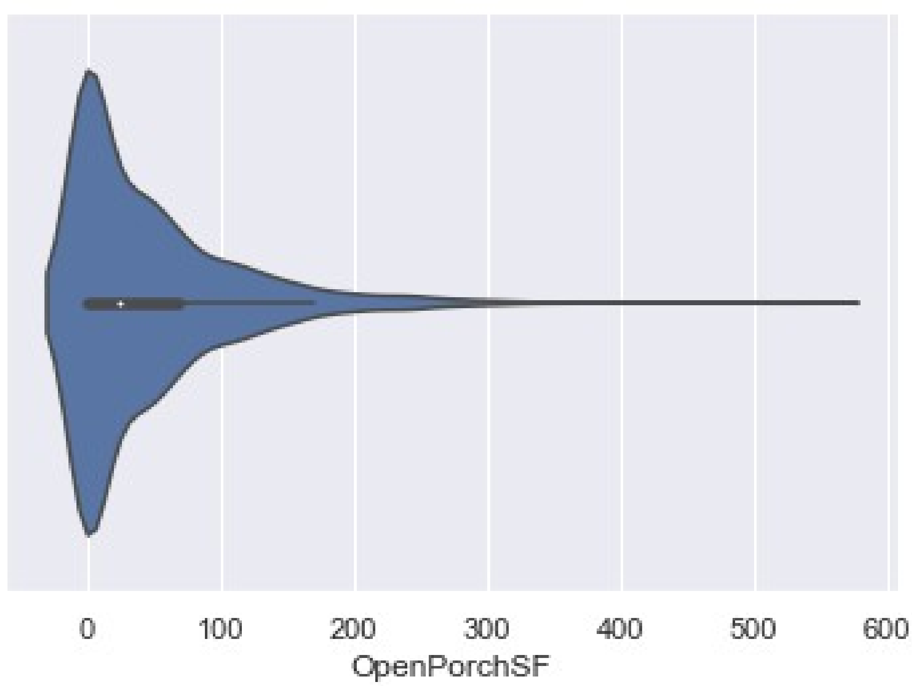

0.0
0.5
1.0
1.5
2.0
2.5
3.0
Fireplaces

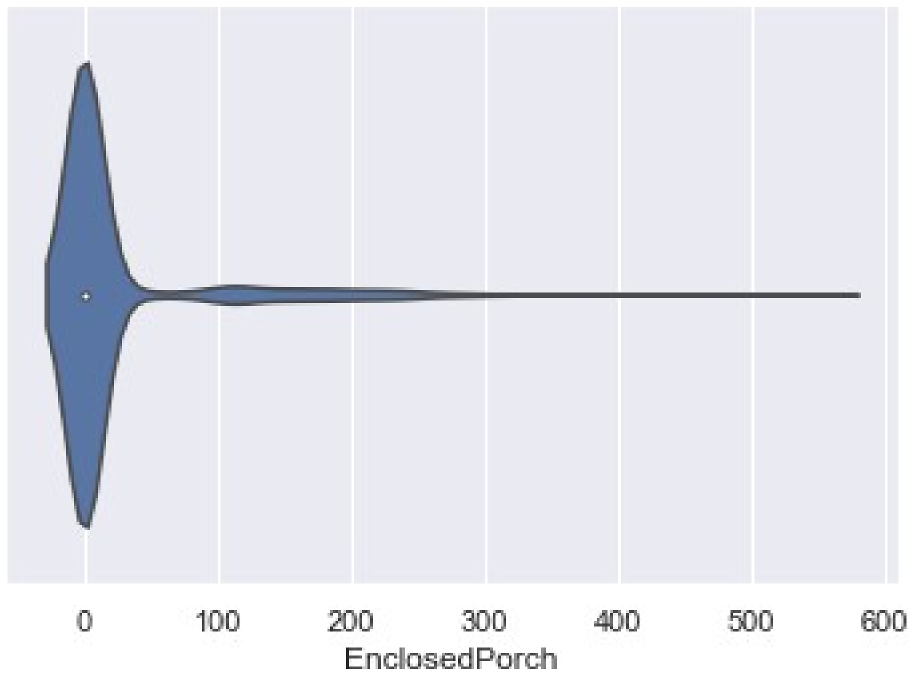

1860
1880
1900
1920
1940
1960
1980
2000
2020
GarageYrBlt

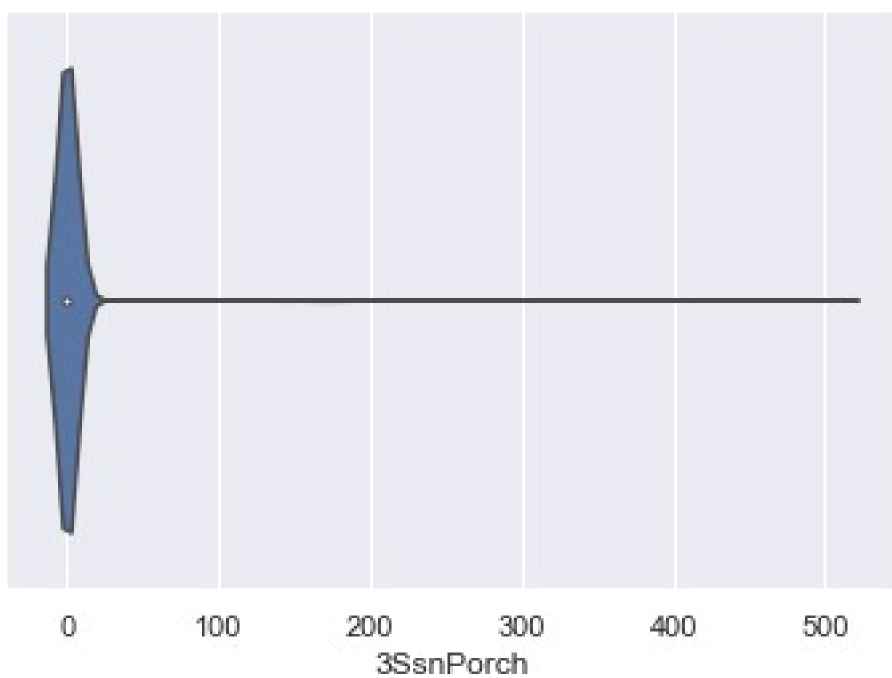

0
1
2
3
4
GarageCars

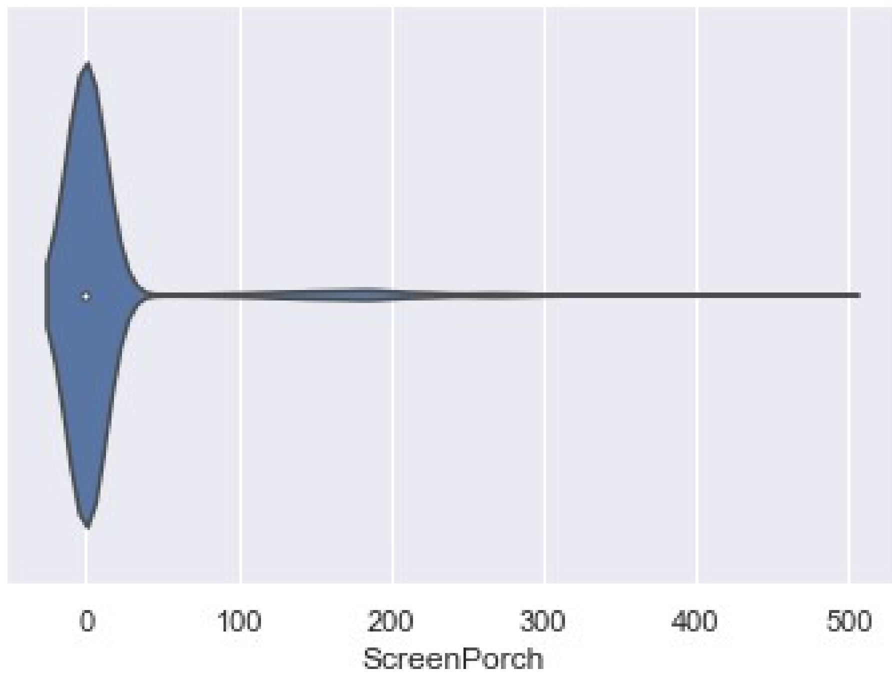

0
200
400
600
800
1000
1200
1400
GarageArea

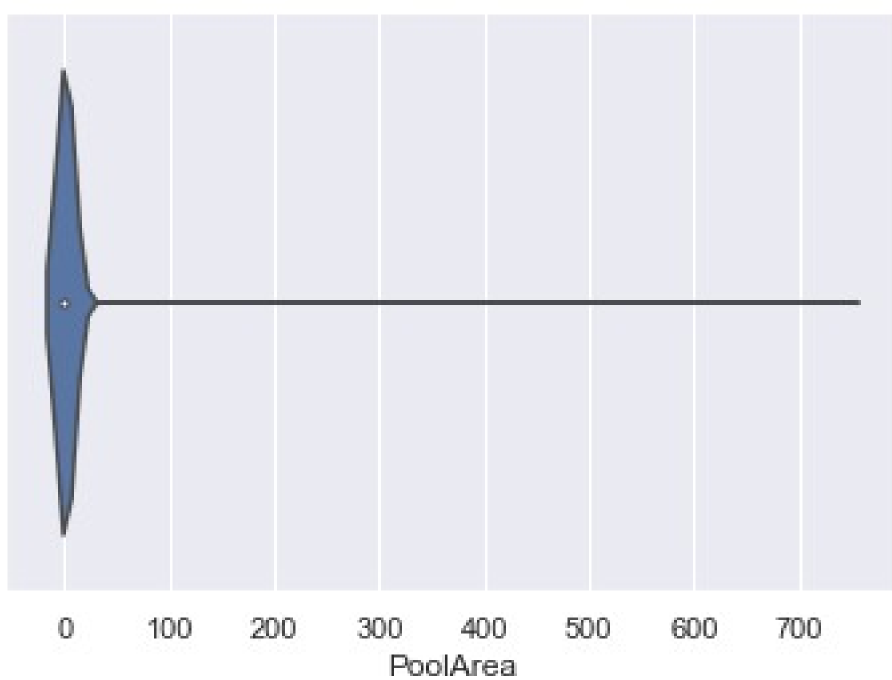

0
200
400
600
800
WoodDeckSF

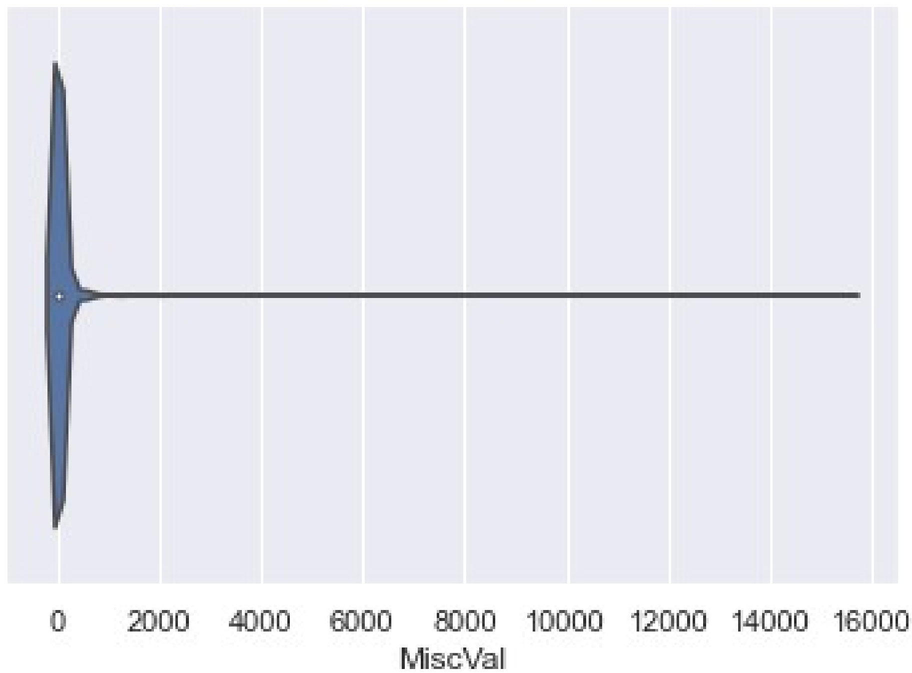

0
100
200
300
400
500
600
OpenPorchSF


0
100
200
300
400
500
600
EnclosedPorch

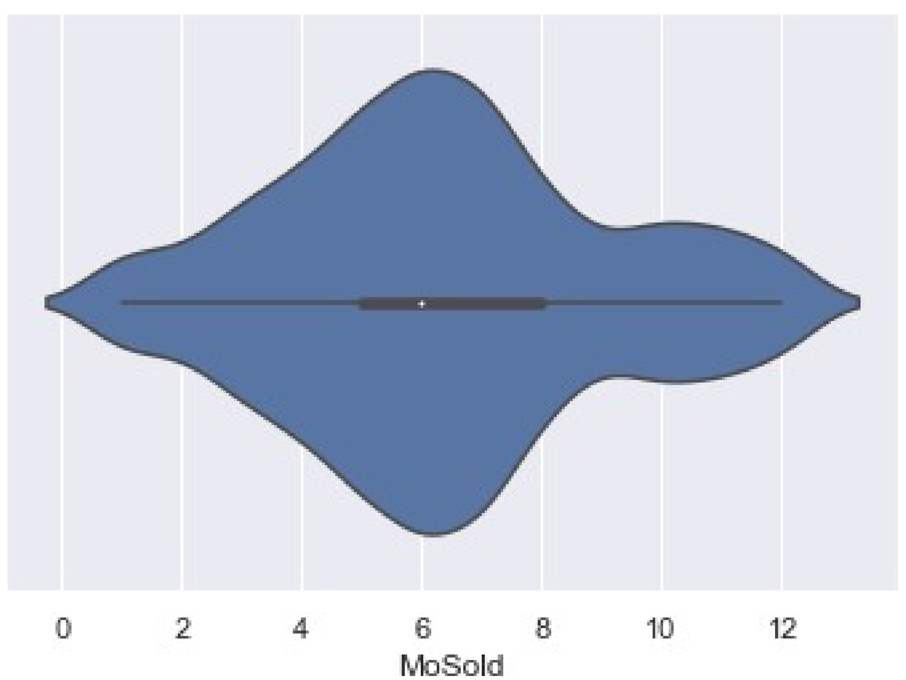

0
100
200
300
400
500
3SsnPorch


0
100
200
300
400
500
ScreenPorch

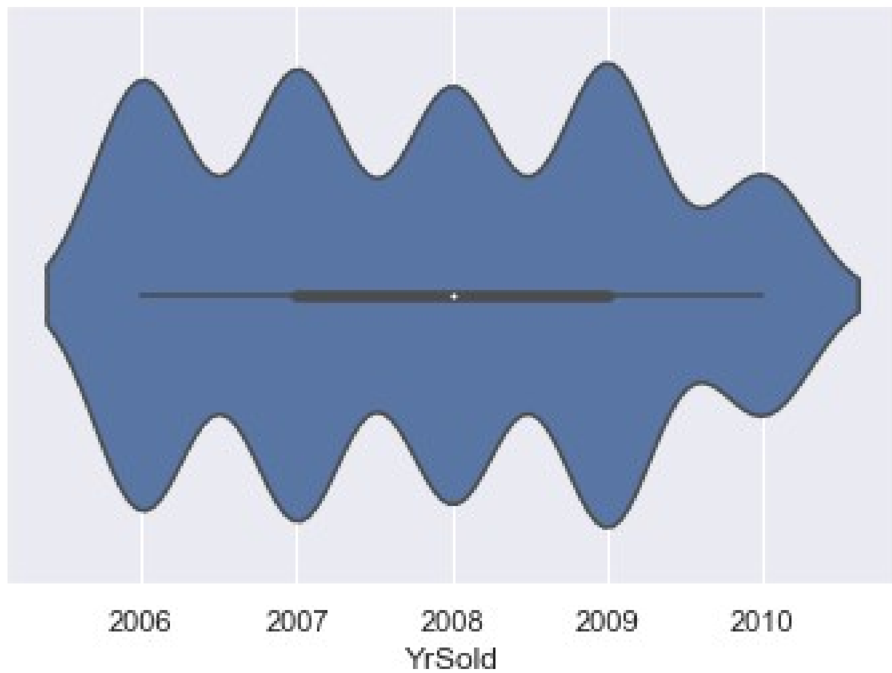

0
100
200
300
400
500
600
700
PoolArea

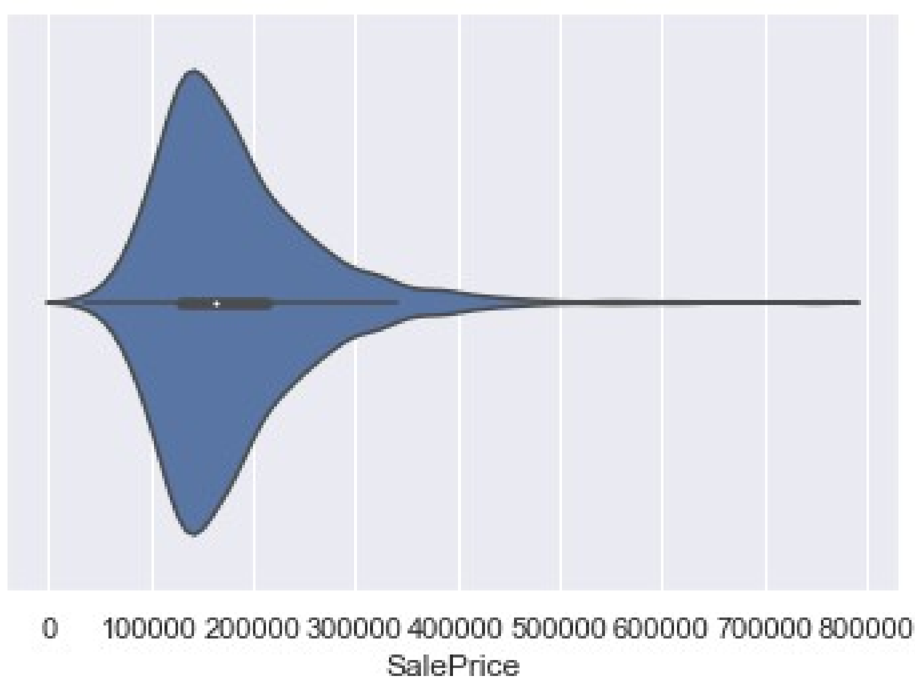


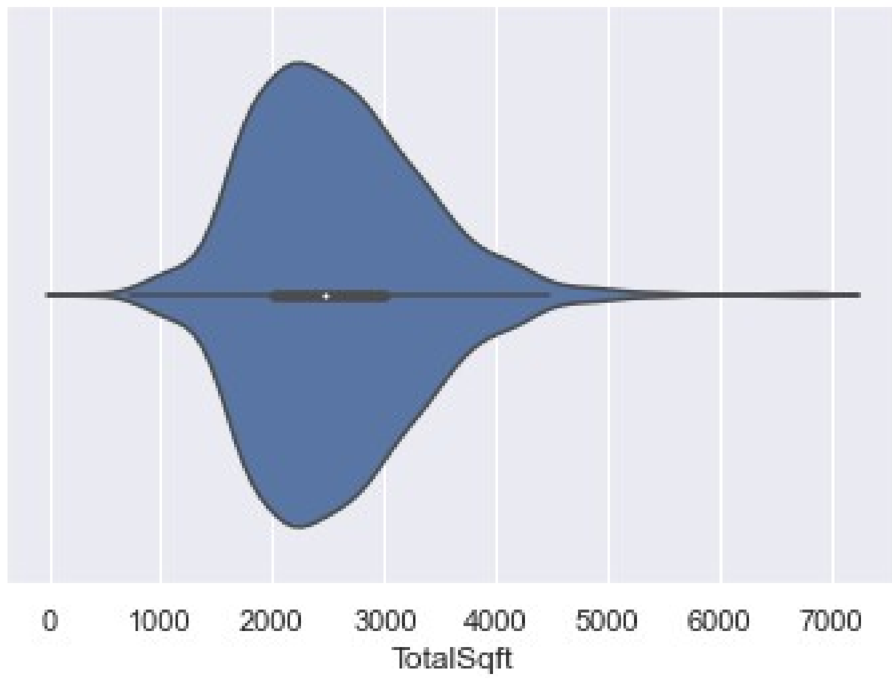
0
2000
4000
6000
8000
10000
12000
14000
16000
MiscVal

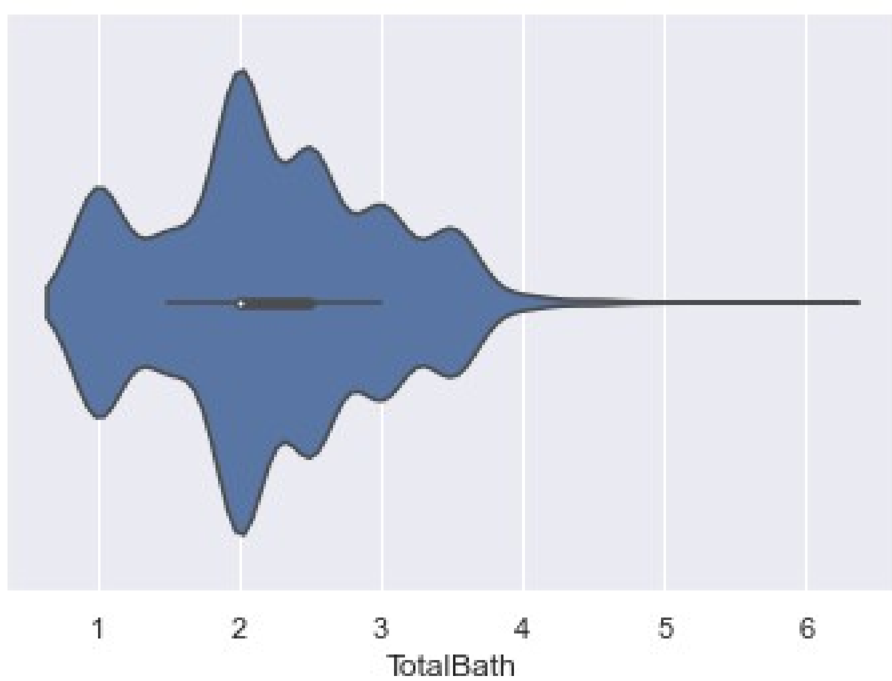


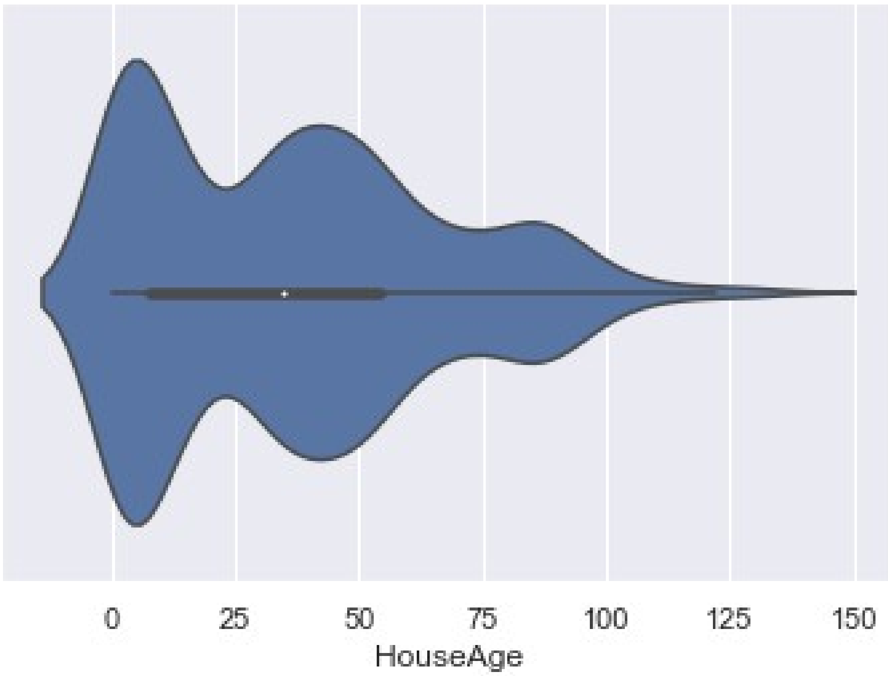
0
2
4
6
8
10
12
MoSold


2006
2007
2008
2009
2010
YrSold


0
100000
200000
300000
400000
500000
600000
700000
800000
SalePrice


0
1000
2000
3000
4000
5000
6000
7000
TotalSqft


1
2
3
4
5
6
TotalBath



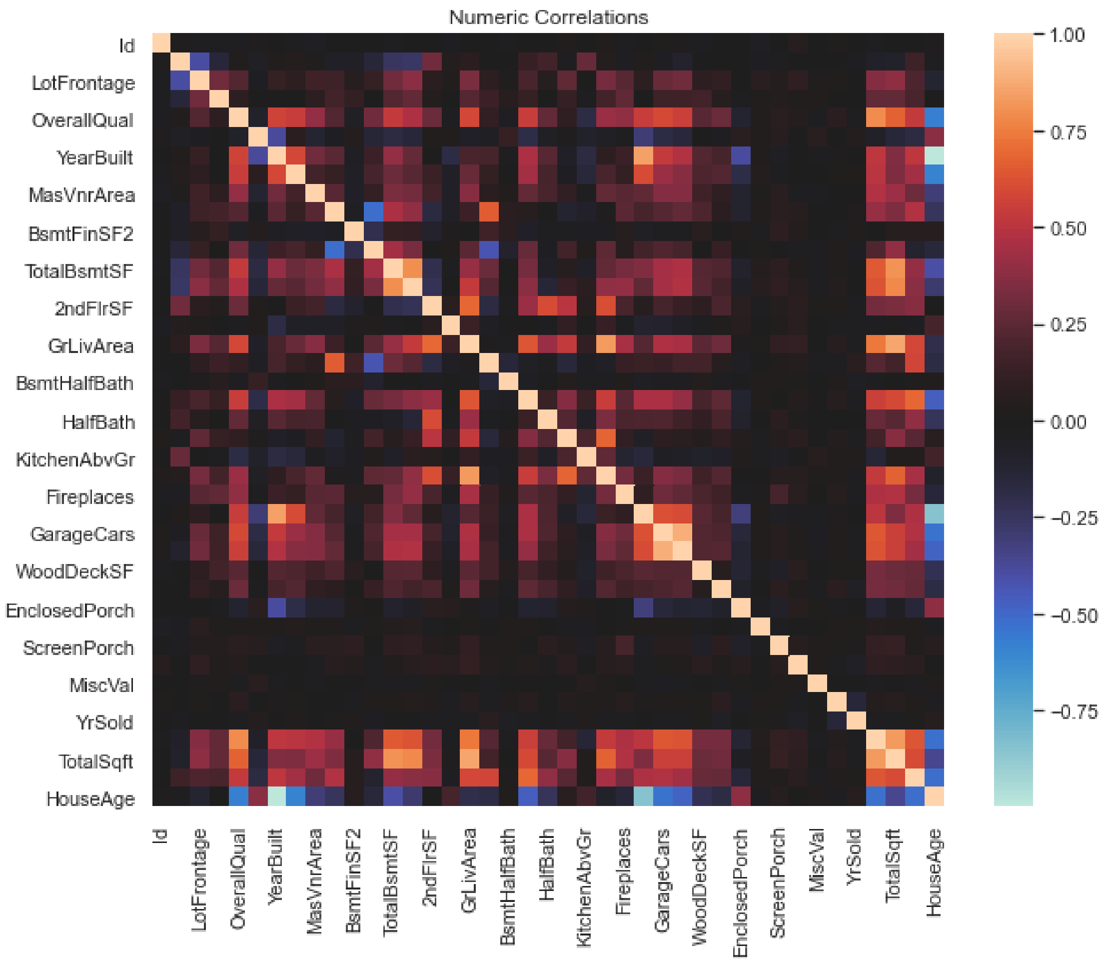
The violin plots above reveal several negative (left-sided) skews. Normalization of these distributions can benefit the model.

```
# Skew Correction
#log1p function applies log(1+x) to all elements of the column
skew = df_train.select_dtypes(include=numerics).skew()

# disabling the pandas warning
pd.options.mode.chained_assignment = None

skewedfeatures = [s for s in skew if(s > 5.0)]
skewedfeatures

for skf in skewedfeatures:
  sk = skew[skew == skf].index[0]
  df_train[sk] = np.log1p(df_train[sk])

# Skew Correction for test set
#log1p function applies log(1+x) to all elements of the column
skew = df_test.select_dtypes(include=numerics).skew()

# disabling the pandas warning
pd.options.mode.chained_assignment = None

skewedfeatures = [s for s in skew if(s > 5.0)]
skewedfeatures

for skf in skewedfeatures:
```

```
    sk = skew[skew == skf].index[0]
    df_test[sk] = np.log1p(df_test[sk])
```

Prep (Numeric Variables)

```
plt.figure(figsize=(10,8))
sns.heatmap(dataset_numeric.corr(), center = 0)
plt.title("Numeric Correlations")
plt.show()
```




```
# Calculates pearson co-efficient for all combinations
data_corr = dataset_numeric.corr()

# Set the threshold to select only highly correlated attributes
threshold = 0.5

# List of pairs along with correlation above threshold
corr_list = []

size = 36
```

```
#Search for the highly correlated pairs
for i in range(0,size): #for 'size' features
    for j in range(i+1,size): #avoid repetition
        if (data_corr.iloc[i,j] >= threshold and data_corr.iloc[i,j] <
1) or (data_corr.iloc[i,j] < 0 and data_corr.iloc[i,j] <= -threshold):
            corr_list.append([data_corr.iloc[i,j],i,j]) #store
correlation and columns index

#Sort to show higher ones first
s_corr_list = sorted(corr_list,key=lambda x: -abs(x[0]))

#Print correlations and column names
for v,i,j in s_corr_list:
    print ("%s and %s = %.2f" % (cols[i],cols[j],v))
```

```
GarageCars and GarageArea = 0.89
YearBuilt and GarageYrBlt = 0.84
GrLivArea and TotRmsAbvGrd = 0.83
TotalBsmtSF and 1stFlrSF = 0.80
2ndFlrSF and GrLivArea = 0.69
BedroomAbvGr and TotRmsAbvGrd = 0.68
BsmtFinSF1 and BsmtFullBath = 0.66
GrLivArea and FullBath = 0.64
GarageYrBlt and GarageCars = 0.62
2ndFlrSF and TotRmsAbvGrd = 0.61
2ndFlrSF and HalfBath = 0.61
YearRemodAdd and GarageYrBlt = 0.60
GarageYrBlt and GarageArea = 0.60
OverallQual and GarageCars = 0.60
YearBuilt and YearRemodAdd = 0.59
OverallQual and GrLivArea = 0.59
OverallQual and YearBuilt = 0.57
OverallQual and GarageArea = 0.56
OverallQual and GarageYrBlt = 0.55
FullBath and TotRmsAbvGrd = 0.55
OverallQual and YearRemodAdd = 0.55
OverallQual and FullBath = 0.55
OverallQual and TotalBsmtSF = 0.54
GrLivArea and BedroomAbvGr = 0.54
YearBuilt and GarageCars = 0.54
1stFlrSF and GrLivArea = 0.53
BsmtFinSF1 and BsmtUnfSF = -0.52
2ndFlrSF and BedroomAbvGr = 0.50
```

Prep (Categorical Variables)

```
dum_vars = ['Neighborhood', 'MSZoning', 'MSSubClass', 'Street',
            'LotShape', 'LotConfig','Utilities', 'LandSlope',
            'BldgType', 'HouseStyle', 'RoofStyle', 'Foundation',
```

```
             'Heating', 'CentralAir', 'PavedDrive', 'MoSold',
             'YrSold', 'SaleType', 'SaleCondition']
```

```
for col in dum_vars:
  print(col)
  train = sorted(df_train[col].unique().tolist())
  test = sorted(df_test[col].unique().tolist())
  total = set(train + test)
  df_train[col] = pd.Categorical(df_train[col], categories=total)
  df_test[col] = pd.Categorical(df_test[col], categories=total)
```

```
Neighborhood
MSZoning
MSSubClass
Street
LotShape
LotConfig
Utilities
LandSlope
BldgType
HouseStyle
RoofStyle
Foundation
Heating
CentralAir
PavedDrive
MoSold
YrSold
SaleType
SaleCondition
```

Models

Lasso Regression

"Least Absolute Shrinkage and Selection Operator" Using Lasso linear regression we will "shrink" values towards the mean to simplify the model. This form of regression will hopefully help reduce the impact of noise on our model.

```
# Feature(s) to look at
f1 = ['MSSubClass', 'LotFrontage', 'LotArea',
    'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
    'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
    'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath',
    'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
    'TotRmsAbvGrd', 'Fireplaces', 'GarageArea', 'WoodDeckSF',
    'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch',
    'PoolArea', 'MiscVal', 'MoSold', 'YrSold',
    'TotalSqft', 'TotalBath', 'HouseAge']
```

```python
# Run a Linear Regression using the feature(s)
x1 = df_train[f1]
y = df_train['SalePrice']

# Split the data
x_train, x_test, y_train, y_test = train_test_split(x1, y)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

```
((1093, 34), (365, 34), (1093,), (365,))
```

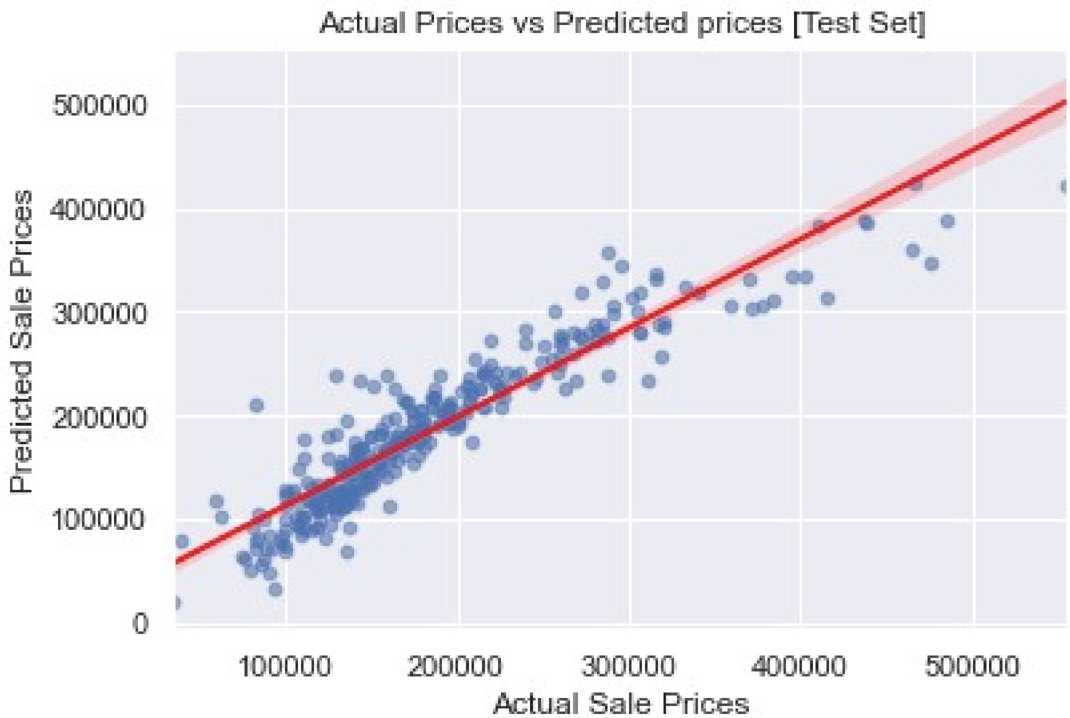
```python
# Set up model
lasso = LassoCV(n_alphas=200, alphas=np.logspace(0, 4, 100), max_iter
= 11000)
kf = KFold(n_splits=7, shuffle=True)

# Standardize the data
ss = StandardScaler()
ss_train = ss.fit_transform(x_train)
ss_test = ss.transform(x_test)

# cross validate
scores = cross_val_score(lasso, ss_train, y_train, cv=kf)
print(scores)
print(f'Mean Score: {scores.mean()}; SD: {scores.std()}')


lasso.fit(ss_train, y_train)
print(f'TRAIN Score: {lasso.score(ss_train, y_train)}')
print(f'TEST Score: {lasso.score(ss_test, y_test)}')

pred = lasso.predict(ss_test)
b, m = np.polynomial.polynomial.polyfit(y_test, pred, 1)
```

```
[0.86678082 0.87698432 0.83635709 0.82759973 0.8937877  0.88209416
 0.79524837]
Mean Score: 0.8541217392587718; SD: 0.03278472706223093
TRAIN Score: 0.864825923973606
TEST Score: 0.8617164697332577
```

My initial attempt at running this model (with max_iter = 1000) yeilded a ConvergenceWarning. To avoid this warning, I decided to drastically increase the iterization. However, this is computationally inefficient and could negatively impact the model. The initial warnings could be a sign that the model is not fitting to the data correctly. That being said, we are left with a bit of confusion considering the mean score seems to reflect a well optimized model.

```python
# Visualize the model results
sns.scatterplot(x=y_test, y=pred, alpha=0.4)
sns.regplot(x=y_test, y=pred, truncate=True, scatter_kws={'s': 20,
```

```
'alpha':0.3},
            line_kws={'color':'red', 'linewidth': 2})
sns.lineplot(x=np.unique(y_test), y=np.unique(np.poly1d(b + m *
np.unique(y_test))), linewidth=0.5, color='r')

plt.xlabel("Actual Sale Prices")
plt.ylabel("Predicted Sale Prices")
plt.title("Actual Prices vs Predicted prices [Test Set]")

plt.show()
```



Prediction Score

```
holdout_df = df_test[f1]

# Standardize the numeric columns
ss = StandardScaler()
ss_holdout = ss.fit_transform(holdout_df)

# predict SalePrice
predict = lasso.predict(ss_holdout)
submit = pd.DataFrame({'Id': df_test['Id'], 'SalePrice': predict})
submit

#export to csv
submit.to_csv('lasso_submission.csv',index=False)
```

```
from IPython import display
display.Image("lasso_score.png")
```

Not a great score, but a worthy first attempt.

Ridge Regression

Ridge regression aims to shrink the dimensions of the data by equal proportions.

```
# Feature(s) to look at
f2 = ['MSSubClass', 'LotFrontage', 'LotArea',
    'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
    'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
    'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath',
    'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
    'TotRmsAbvGrd', 'Fireplaces', 'GarageArea', 'WoodDeckSF',
    'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch',
    'PoolArea', 'MiscVal', 'MoSold', 'YrSold',
    'TotalSqft', 'TotalBath', 'HouseAge']

# Run a Linear Regression using the feature(s)
x2 = df_train[f2]
y = df_train['SalePrice']

# Split the data
x2_train, x2_test, y2_train, y2_test = train_test_split(x2, y)
x2_train.shape, x2_test.shape, y2_train.shape, y2_test.shape
```

```
((1093, 34), (365, 34), (1093,), (365,))
```

```
# Set up model
ridge_reg = Ridge(alpha=1, solver="cholesky")
kf = KFold(n_splits=7, shuffle=True)

# Standardize the data
ss = StandardScaler()
ss_train2 = ss.fit_transform(x2_train)
ss_test2 = ss.transform(x2_test)

# cross validate
scores = cross_val_score(ridge_reg, ss_train2, y2_train, cv=kf)
print(scores)
print(f'Mean Score: {scores.mean()}; SD: {scores.std()}')
```

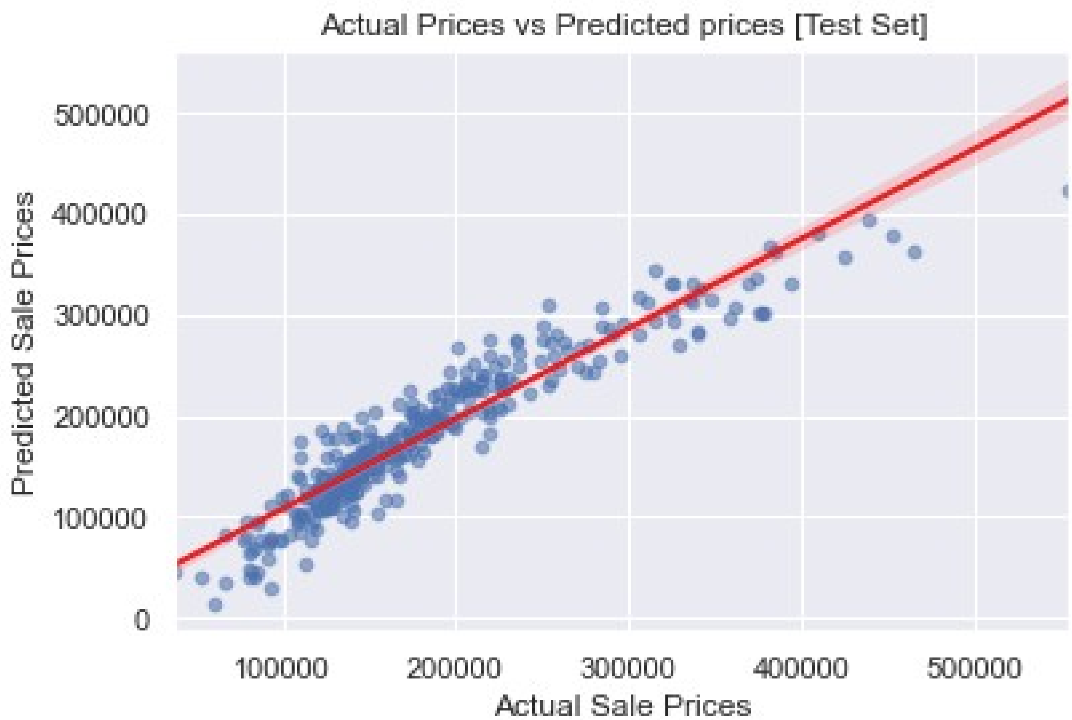
```
ridge_reg.fit(ss_train2, y2_train)
```

```python
print(f'TRAIN Score: {ridge_reg.score(ss_train2, y2_train)}')
print(f'TEST Score: {ridge_reg.score(ss_test2, y2_test)}')

pred2 = ridge_reg.predict(ss_test2)
b, m = np.polynomial.polynomial.polyfit(y2_test, pred2, 1)
```

```
[0.77746177 0.85340441 0.88977111 0.8172674  0.85225053 0.87785206
 0.82413569]
Mean Score: 0.8417347086521818; SD: 0.03565330722620647
TRAIN Score: 0.859717191911943
TEST Score: 0.885335525073244
```

Fortunately, running the ridge regression model did not prompt any warnings. It appears to have fit to the data fairly well, however, the mean score is slightly lower than that of the 11000 iteration Lasso model.

```python
# Visualize the model results
sns.scatterplot(x=y2_test, y=pred2, alpha=0.4)
sns.regplot(x=y2_test, y=pred2, truncate=True, scatter_kws={'s': 20,
'alpha':0.3},
            line_kws={'color':'red', 'linewidth': 2})
sns.lineplot(x=np.unique(y2_test), y=np.unique(np.poly1d(b + m *
np.unique(y2_test))), linewidth=0.5, color='r')

plt.xlabel("Actual Sale Prices")
plt.ylabel("Predicted Sale Prices")
plt.title("Actual Prices vs Predicted prices [Test Set]")

plt.show()
```



Prediction Score

```
holdout_df2 = df_test[f2]

# Standardize the numeric columns
ss = StandardScaler()
ss_holdout2 = ss.fit_transform(holdout_df2)

# predict SalePrice
predict2 = ridge_reg.predict(ss_holdout2)
submit2 = pd.DataFrame({'Id': df_test['Id'], 'SalePrice': predict2})
submit2

#export to csv
submit2.to_csv('ridge_submission.csv',index=False)

display.Image("ridge_submission.png")
```

This is not a great score. It is significantly worse than the Lasso model and leaves me with a lot of questions and concerns about the initial quality of my prep on the dataset.

Elastic Net

Elastic Net regression, in a way, fills the middle ground between lasso and ridge regression. It uses both L1 and L2 regularization techniques to try and capture the benefits while offsetting eachothers potential pitfalls.

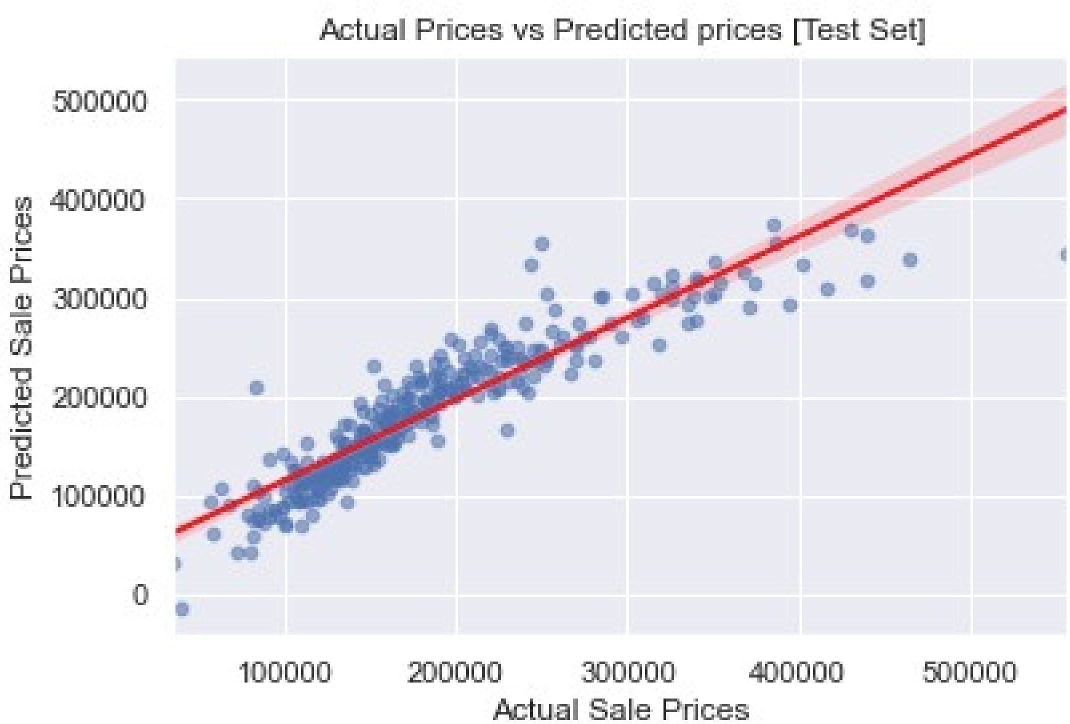
```
# Feature(s) to look at
f3 = ['MSSubClass', 'LotFrontage', 'LotArea',
    'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
    'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
    'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath',
    'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
    'TotRmsAbvGrd', 'Fireplaces', 'GarageArea', 'WoodDeckSF',
    'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch',
    'PoolArea', 'MiscVal', 'MoSold', 'YrSold',
    'TotalSqft', 'TotalBath', 'HouseAge']

# Run a Linear Regression using the feature(s)
x3 = df_train[f3]
y = df_train['SalePrice']

# Split the data
x3_train, x3_test, y3_train, y3_test = train_test_split(x3, y)
x3_train.shape, x3_test.shape, y3_train.shape, y3_test.shape
```

```
((1093, 34), (365, 34), (1093,), (365,))
```

```
# Set up model
e_net = ElasticNetCV(n_alphas=200, alphas=np.logspace(0, 4, 100),
max_iter = 1000)
kf = KFold(n_splits=7, shuffle=True)

# Standardize the data
ss = StandardScaler()
ss_train3 = ss.fit_transform(x3_train)
ss_test3 = ss.transform(x3_test)

# cross validate
scores = cross_val_score(e_net, ss_train3, y3_train, cv=kf)
print(scores)
print(f'Mean Score: {scores.mean()}; SD: {scores.std()}')


e_net.fit(ss_train3, y3_train)
print(f'TRAIN Score: {e_net.score(ss_train3, y3_train)}')
print(f'TEST Score: {e_net.score(ss_test3, y3_test)}')

pred3 = e_net.predict(ss_test3)
b, m = np.polynomial.polynomial.polyfit(y3_test, pred3, 1)
```

```
[0.86790248 0.84501076 0.8735932  0.80718903 0.81116816 0.81723714
 0.82687952]
Mean Score: 0.8355686140825493; SD: 0.025050885363058557
TRAIN Score: 0.8485104942436186
TEST Score: 0.8478841103210351
```

```
# Visualize the model results
sns.scatterplot(x=y3_test, y=pred3, alpha=0.4)
sns.regplot(x=y3_test, y=pred3, truncate=True, scatter_kws={'s': 20,
'alpha':0.3},
            line_kws={'color':'red', 'linewidth': 2})
sns.lineplot(x=np.unique(y3_test), y=np.unique(np.poly1d(b + m *
np.unique(y3_test))), linewidth=0.5, color='r')

plt.xlabel("Actual Sale Prices")
plt.ylabel("Predicted Sale Prices")
plt.title("Actual Prices vs Predicted prices [Test Set]")

plt.show()
```



The Elastic Net model ran without complications. However, based on the mean score and the standard deviation it appears to have fit to the data worse than the previous two models. That being said, it still fits appropriately and overfitting won't seem to be a problem.

Prediction Score

```
holdout_df3 = df_test[f3]
```

```python
# Standardize the numeric columns
ss = StandardScaler()
ss_holdout3 = ss.fit_transform(holdout_df3)

# predict SalePrice
predict3 = e_net.predict(ss_holdout3)
submit3 = pd.DataFrame({'Id': df_test['Id'], 'SalePrice': predict3})
submit3

#export to csv
submit3.to_csv('elasticnet_submission.csv',index=False)

display.Image("elasticnet_score.png")
```

The Elastic Net submission score was by far the best of the three. Despite its initial fit, the model seemed to handle noise well. The combination of L1 and L2 regularization likely played a synergistic role in the performance of this model.

Conclusion

The three models ran fairly well for first attempts. The Kaggle prediction scores fell with in a range of effectivness. Its likely that there performance could have been optimized by better data prep. However, given the circumstances, these models adequately illustrated many of the performance differences between each regression method. Although, it would be an interesting personal project to see exactly how significantly better prep would optimize these regression methods.