Data Science - Capstone Project Review

This project enlists a list of basic and ensemble learning models used to predict Boston House Prices | Advanced Regression Techniques.

Out[3]: The raw code for this IPython notebook is by default hidden for easier reading. To toggle on/off the raw code, click here.

Out[13]:

House Prices: Advanced Regression Techniques

Predict sales prices and practice feature engineering, RFs, and gradient boosting

4,572 teams · Ongoing

Bagging Models

- **Support Vector Machine Regressor**
- **Decision Tree Regressor**
- **ExtraTree Regressor**

Boosting Models

- **Random Forest Regressor**
- **AdaBoost Regressor**
- **Gradient Boost Regressor**
- **XGBoost Regressor**
- **LGBM Regressor**
- **CatBoost Regressor**

Basic Linear Regresison

- **Linear Regression using RFE**

In the pipeline

- **Lasso Linear Regression**
- **Linear Regression using Gradient Descent**
- **Neural Network**

Out[14]:

Standard Boilerplate for Data Science

Out[97]:

Import Scikits Learn Maching Learning Marathon List for Data Science

Background

Ask a home buyer to describe their dream house, and they probably won't begin with the height of the basement ceiling or the proximity to an east-west railroad. But this playground competition's dataset proves that much more influences price negotiations than the number of bedrooms or a white-picket fence.

With 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa, this competition challenges you to predict the final price of each home.

The **Ames Housing** dataset was compiled by Dean De Cock for use in data science education. It's an incredible alternative for data scientists looking for a modernized and expanded version of the often cited Boston Housing dataset.

Step 1:: Import Data Files

On our very first initial step, we will import both train and test datasets.

```
Train data is loaded successfully!
Test data is loaded successfully!

There are 1460 observations and 80 features in the TRAIN dataset:
         Of which, 37 numeric features
         And, 43 non-numeric/object features.

There are 1459 observations and 79 features in the TEST dataset:
         Of which, 36 numeric features
         And, 43 non-numeric/object features.
```

We will review the train dataset first and making sure everything is reviewed before we move to the test dataset. One thing to note is test data has one lesser column and that is the dependent variable (i.e. target variable)

Step 2:: Exploratory Data Analysis

Let's review how the data looks like.

Out[21]:

Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotCon
1	60	RL	65.00	8450	Pave	NaN	Reg	Lvl	AllPub	Insi
2	20	RL	80.00	9600	Pave	NaN	Reg	Lvl	AllPub	F
3	60	RL	68.00	11250	Pave	NaN	IR1	Lvl	AllPub	Insi
4	70	RL	60.00	9550	Pave	NaN	IR1	Lvl	AllPub	Corr
5	60	RL	84.00	14260	Pave	NaN	IR1	Lvl	AllPub	F
6	50	RL	85.00	14115	Pave	NaN	IR1	Lvl	AllPub	Insi
7	20	RL	75.00	10084	Pave	NaN	Reg	Lvl	AllPub	Insi
8	60	RL	nan	10382	Pave	NaN	IR1	Lvl	AllPub	Corr
9	50	RM	51.00	6120	Pave	NaN	Reg	Lvl	AllPub	Insi
10	190	RL	50.00	7420	Pave	NaN	Reg	Lvl	AllPub	Corr

10 rows × 80 columns

A quick look of it, there are some missing data / NaN values eg. Alley, PoolQC, Fence, MiscFeature. And some columns are numeric. Likewise others are categorical.

So, what data types do we have with this list of columns?

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1460 entries, 1 to 1460
Data columns (total 80 columns):
MSSubClass       1460 non-null int64
MSZoning         1460 non-null object
LotFrontage      1201 non-null float64
LotArea          1460 non-null int64
Street           1460 non-null object
Alley            91 non-null object
LotShape         1460 non-null object
LandContour      1460 non-null object
Utilities        1460 non-null object
LotConfig        1460 non-null object
LandSlope        1460 non-null object
Neighborhood     1460 non-null object
Condition1       1460 non-null object
Condition2       1460 non-null object
BldgType         1460 non-null object
HouseStyle       1460 non-null object
OverallQual      1460 non-null int64
OverallCond      1460 non-null int64
YearBuilt        1460 non-null int64
YearRemodAdd     1460 non-null int64
RoofStyle        1460 non-null object
RoofMatl         1460 non-null object
Exterior1st      1460 non-null object
Exterior2nd      1460 non-null object
MasVnrType       1452 non-null object
MasVnrArea       1452 non-null float64
ExterQual        1460 non-null object
ExterCond        1460 non-null object
Foundation       1460 non-null object
BsmtQual         1423 non-null object
BsmtCond         1423 non-null object
BsmtExposure     1422 non-null object
BsmtFinType1     1423 non-null object
BsmtFinSF1       1460 non-null int64
BsmtFinType2     1422 non-null object
BsmtFinSF2       1460 non-null int64
BsmtUnfSF        1460 non-null int64
TotalBsmtSF      1460 non-null int64
Heating          1460 non-null object
HeatingQC        1460 non-null object
CentralAir       1460 non-null object
Electrical       1459 non-null object
1stFlrSF         1460 non-null int64
2ndFlrSF         1460 non-null int64
LowQualFinSF     1460 non-null int64
GrLivArea        1460 non-null int64
BsmtFullBath     1460 non-null int64
BsmtHalfBath     1460 non-null int64
FullBath         1460 non-null int64
HalfBath         1460 non-null int64
BedroomAbvGr     1460 non-null int64
KitchenAbvGr     1460 non-null int64
KitchenQual      1460 non-null object
TotRmsAbvGrd     1460 non-null int64
Functional       1460 non-null object
Fireplaces       1460 non-null int64
FireplaceQu      770 non-null object
GarageType       1379 non-null object
GarageYrBlt      1379 non-null float64
GarageFinish     1379 non-null object
GarageCars       1460 non-null int64
GarageArea       1460 non-null int64
GarageQual       1379 non-null object
GarageCond       1379 non-null object
PavedDrive       1460 non-null object
WoodDeckSF       1460 non-null int64
```

We will split categorical and numerical features and address each of them separately.

```
37 numeric features:
These numeric features are:
 Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
       'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2',
       'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
       'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces',
       'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
       'MoSold', 'YrSold', 'SalePrice'],
      dtype='object').
```

A quick glance on the numerical features distribution.

Out[24]:

	count	mean	std	min	25%	50%	75%	max
MSSubClass	1460.00	56.90	42.30	20.00	20.00	50.00	70.00	190.00
LotFrontage	1201.00	70.05	24.28	21.00	59.00	69.00	80.00	313.00
LotArea	1460.00	10516.83	9981.26	1300.00	7553.50	9478.50	11601.50	215245.00
OverallQual	1460.00	6.10	1.38	1.00	5.00	6.00	7.00	10.00
OverallCond	1460.00	5.58	1.11	1.00	5.00	5.00	6.00	9.00
YearBuilt	1460.00	1971.27	30.20	1872.00	1954.00	1973.00	2000.00	2010.00
YearRemodAdd	1460.00	1984.87	20.65	1950.00	1967.00	1994.00	2004.00	2010.00
MasVnrArea	1452.00	103.69	181.07	0.00	0.00	0.00	166.00	1600.00
BsmtFinSF1	1460.00	443.64	456.10	0.00	0.00	383.50	712.25	5644.00
BsmtFinSF2	1460.00	46.55	161.32	0.00	0.00	0.00	0.00	1474.00
BsmtUnfSF	1460.00	567.24	441.87	0.00	223.00	477.50	808.00	2336.00
TotalBsmtSF	1460.00	1057.43	438.71	0.00	795.75	991.50	1298.25	6110.00
1stFlrSF	1460.00	1162.63	386.59	334.00	882.00	1087.00	1391.25	4692.00
2ndFlrSF	1460.00	346.99	436.53	0.00	0.00	0.00	728.00	2065.00
LowQualFinSF	1460.00	5.84	48.62	0.00	0.00	0.00	0.00	572.00
GrLivArea	1460.00	1515.46	525.48	334.00	1129.50	1464.00	1776.75	5642.00
BsmtFullBath	1460.00	0.43	0.52	0.00	0.00	0.00	1.00	3.00
BsmtHalfBath	1460.00	0.06	0.24	0.00	0.00	0.00	0.00	2.00
FullBath	1460.00	1.57	0.55	0.00	1.00	2.00	2.00	3.00
HalfBath	1460.00	0.38	0.50	0.00	0.00	0.00	1.00	2.00
BedroomAbvGr	1460.00	2.87	0.82	0.00	2.00	3.00	3.00	8.00
KitchenAbvGr	1460.00	1.05	0.22	0.00	1.00	1.00	1.00	3.00
TotRmsAbvGrd	1460.00	6.52	1.63	2.00	5.00	6.00	7.00	14.00
Fireplaces	1460.00	0.61	0.64	0.00	0.00	1.00	1.00	3.00
GarageYrBlt	1379.00	1978.51	24.69	1900.00	1961.00	1980.00	2002.00	2010.00
GarageCars	1460.00	1.77	0.75	0.00	1.00	2.00	2.00	4.00
GarageArea	1460.00	472.98	213.80	0.00	334.50	480.00	576.00	1418.00
WoodDeckSF	1460.00	94.24	125.34	0.00	0.00	0.00	168.00	857.00
OpenPorchSF	1460.00	46.66	66.26	0.00	0.00	25.00	68.00	547.00
EnclosedPorch	1460.00	21.95	61.12	0.00	0.00	0.00	0.00	552.00
3SsnPorch	1460.00	3.41	29.32	0.00	0.00	0.00	0.00	508.00
ScreenPorch	1460.00	15.06	55.76	0.00	0.00	0.00	0.00	480.00
PoolArea	1460.00	2.76	40.18	0.00	0.00	0.00	0.00	738.00
MiscVal	1460.00	43.49	496.12	0.00	0.00	0.00	0.00	15500.00
MoSold	1460.00	6.32	2.70	1.00	5.00	6.00	8.00	12.00
YrSold	1460.00	2007.82	1.33	2006.00	2007.00	2008.00	2009.00	2010.00
SalePrice	1460.00	180921.20	79442.50	34900.00	129975.00	163000.00	214000.00	755000.00

Very obvious, there are definitely some outliers **(eg. LotArea, MiscVal)** which we need to address in the next step. Keep an eye on this!

What we probably can gauge from these numeric features are:

- are they missing values? if so, what treatment(s) can we apply?
- are they highly correlated? if so, what can we do to de-correlate them?
- are they outliers which needs to be managed?

Now, let's look at the same information on categorical features.

```
43 non-numeric/object features:
These non-numeric features are Index(['MSZoning', 'Street', 'Alley', 'LotShape
', 'LandContour', 'Utilities',
       'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2',
       'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st',
       'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation',
       'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',
       'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual',
       'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual
',
       'GarageCond', 'PavedDrive', 'PoolQC', 'Fence', 'MiscFeature',
       'SaleType', 'SaleCondition'],
      dtype='object').
```

For Categorical, perhaps we can see what are some of the unique values.

```
Unique values in each non-numeric column:
MSZoning : ['RL' 'RM' 'C (all)' 'FV' 'RH']
Street : ['Pave' 'Grvl']
Alley : [nan 'Grvl' 'Pave']
LotShape : ['Reg' 'IR1' 'IR2' 'IR3']
LandContour : ['Lvl' 'Bnk' 'Low' 'HLS']
Utilities : ['AllPub' 'NoSeWa']
LotConfig : ['Inside' 'FR2' 'Corner' 'CulDSac' 'FR3']
LandSlope : ['Gtl' 'Mod' 'Sev']
Neighborhood : ['CollgCr' 'Veenker' 'Crawfor' 'NoRidge' 'Mitchel' 'Somerst' 'N
WAmes'
 'OldTown' 'BrkSide' 'Sawyer' 'NridgHt' 'NAmes' 'SawyerW' 'IDOTRR'
 'MeadowV' 'Edwards' 'Timber' 'Gilbert' 'StoneBr' 'ClearCr' 'NPkVill'
 'Blmngtn' 'BrDale' 'SWISU' 'Blueste']
Condition1 : ['Norm' 'Feedr' 'PosN' 'Artery' 'RRAe' 'RRNn' 'RRAn' 'PosA' 'RRNe
']
Condition2 : ['Norm' 'Artery' 'RRNn' 'Feedr' 'PosN' 'PosA' 'RRAn' 'RRAe']
BldgType : ['1Fam' '2fmCon' 'Duplex' 'TwnhsE' 'Twnhs']
HouseStyle : ['2Story' '1Story' '1.5Fin' '1.5Unf' 'SFoyer' 'SLvl' '2.5Unf' '2.
5Fin']
RoofStyle : ['Gable' 'Hip' 'Gambrel' 'Mansard' 'Flat' 'Shed']
RoofMatl : ['CompShg' 'WdShngl' 'Metal' 'WdShake' 'Membran' 'Tar&Grv' 'Roll'
 'ClyTile']
Exterior1st : ['VinylSd' 'MetalSd' 'Wd Sdng' 'HdBoard' 'BrkFace' 'WdShing' 'Ce
mntBd'
 'Plywood' 'AsbShng' 'Stucco' 'BrkComm' 'AsphShn' 'Stone' 'ImStucc'
 'CBlock']
Exterior2nd : ['VinylSd' 'MetalSd' 'Wd Shng' 'HdBoard' 'Plywood' 'Wd Sdng' 'Cm
entBd'
 'BrkFace' 'Stucco' 'AsbShng' 'Brk Cmn' 'ImStucc' 'AsphShn' 'Stone'
 'Other' 'CBlock']
MasVnrType : ['BrkFace' 'None' 'Stone' 'BrkCmn' nan]
ExterQual : ['Gd' 'TA' 'Ex' 'Fa']
ExterCond : ['TA' 'Gd' 'Fa' 'Po' 'Ex']
Foundation : ['PConc' 'CBlock' 'BrkTil' 'Wood' 'Slab' 'Stone']
BsmtQual : ['Gd' 'TA' 'Ex' nan 'Fa']
BsmtCond : ['TA' 'Gd' nan 'Fa' 'Po']
BsmtExposure : ['No' 'Gd' 'Mn' 'Av' nan]
BsmtFinType1 : ['GLQ' 'ALQ' 'Unf' 'Rec' 'BLQ' nan 'LwQ']
BsmtFinType2 : ['Unf' 'BLQ' nan 'ALQ' 'Rec' 'LwQ' 'GLQ']
Heating : ['GasA' 'GasW' 'Grav' 'Wall' 'OthW' 'Floor']
HeatingQC : ['Ex' 'Gd' 'TA' 'Fa' 'Po']
CentralAir : ['Y' 'N']
Electrical : ['SBrkr' 'FuseF' 'FuseA' 'FuseP' 'Mix' nan]
KitchenQual : ['Gd' 'TA' 'Ex' 'Fa']
Functional : ['Typ' 'Min1' 'Maj1' 'Min2' 'Mod' 'Maj2' 'Sev']
FireplaceQu : [nan 'TA' 'Gd' 'Fa' 'Ex' 'Po']
GarageType : ['Attchd' 'Detchd' 'BuiltIn' 'CarPort' nan 'Basment' '2Types']
GarageFinish : ['RFn' 'Unf' 'Fin' nan]
GarageQual : ['TA' 'Fa' 'Gd' nan 'Ex' 'Po']
GarageCond : ['TA' 'Fa' nan 'Gd' 'Po' 'Ex']
PavedDrive : ['Y' 'N' 'P']
PoolQC : [nan 'Ex' 'Fa' 'Gd']
Fence : [nan 'MnPrv' 'GdWo' 'GdPrv' 'MnWw']
MiscFeature : [nan 'Shed' 'Gar2' 'Othr' 'TenC']
SaleType : ['WD' 'New' 'COD' 'ConLD' 'ConLI' 'CWD' 'ConLw' 'Con' 'Oth']
SaleCondition : ['Normal' 'Abnorml' 'Partial' 'AdjLand' 'Alloca' 'Family']
```

Same goes to the categorical features, there are definitely NaN values which need to be managed if we want to bring into the model.

From the above, what we can summarise are as follows:

- There are **1460** data rows in the train dataset
- **37** numerical and **43** categorical columns

Our next step is getting into some data construction process:

- **Step 3a** **Handling Missing Values**
- **Step 3b** **Removing Collinearity**
- **Step 3c** **Managing Outliers**

Step 3a:: Handling Missing Values

On this section, we probably need to find out what is the magnitude of missing values in the dataset and what treatments do we want to apply to overcome that. For now, let's take alook at which features do we need to address and how.

```
There are 19 columns with missing data.
These columns are as follows:

Features        No of Obs Missing #    No of Obs Missing %
------------  ---------------------  ---------------------
PoolQC                         1453                  99.52
MiscFeature                    1406                  96.3
Alley                          1369                  93.77
Fence                          1179                  80.75
FireplaceQu                     690                  47.26
LotFrontage                     259                  17.74
GarageFinish                     81                   5.55
GarageQual                       81                   5.55
GarageType                       81                   5.55
GarageYrBlt                      81                   5.55
GarageCond                       81                   5.55
BsmtFinType2                     38                   2.6
BsmtExposure                     38                   2.6
BsmtFinType1                     37                   2.53
BsmtQual                         37                   2.53
BsmtCond                         37                   2.53
MasVnrArea                        8                   0.55
MasVnrType                        8                   0.55
Electrical                        1                   0.07
```

There are **19 columns with missing data**, with **PoolQC** being the top feature with highest missing values in the data. For a simpler or straightforward approach, it would be easier to remove these rows in total but not comprising on the overall purpose of this exercise. Alternatively, we can impute these missing values with median/mode.

In summary, we will carry out two approaches when handling missing values.

- **Solution #1: Removing features with more than 40% of missing data** . The proportion of missing data can be risky if we try to impute any logic to feed them back into the analysis.
- **Solution #2:** Impute using **np.median or np.mode** on those remaining features with some levels of missing values

Solution #1 - Remove features with high proportion of missing data

Filter those features with more than 40% of proportion are missing. As a general rule of thumb, we probably remove them in totality.

Out[29]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	La
Id										
1	60	RL	65.00	8450	Pave	Reg	Lvl	AllPub	Inside	
2	20	RL	80.00	9600	Pave	Reg	Lvl	AllPub	FR2	
3	60	RL	68.00	11250	Pave	IR1	Lvl	AllPub	Inside	
4	70	RL	60.00	9550	Pave	IR1	Lvl	AllPub	Corner	
5	60	RL	84.00	14260	Pave	IR1	Lvl	AllPub	FR2	

5 rows × 75 columns

```
These 5 features are removed: ['PoolQC', 'MiscFeature', 'Alley', 'Fence', 'Fir
eplaceQu'].
```

After excluding all missing data, the new dataset consists of:

```
After excluded these features, there are 1460 observations and 75 features.
         Of which, 37 numeric features
         And, 38 non-numeric/object features.
```

You probably have noticed **80** features are now left with **75**. And for the rest of features with some missing values, we will impute using median or mode depending on the features

Solution #2 - Impute data with median or mode

In this alternate solution in address missing data, we will filter those features with less than 40% of proportion are missing. This part requires some level of data impute (i.e. applying median or mode depending on the data type).

```
After removed 5 features, we still need to review the remaining 14 features.
         Of which, 3 numeric features
         And, 11 non-numeric/object features.
```

Since there are 3 numeric and 11 categorical features, we will apply median on those numeric and model on categorical features where there are missing values

Out[33]:

Id	1	2	3	4	5	6	7	8	9	10
LotFrontage	65.00	80.00	68.00	60.00	84.00	85.00	75.00	NaN	51.00	50.00
GarageFinish	RFn	RFn	RFn	Unf	RFn	Unf	RFn	RFn	Unf	RFn
GarageQual	TA	TA	TA	TA	TA	TA	TA	TA	Fa	Gd
GarageType	Attchd	Attchd	Attchd	Detchd	Attchd	Attchd	Attchd	Attchd	Detchd	Attchd
GarageYrBlt	2003.00	1976.00	2001.00	1998.00	2000.00	1993.00	2004.00	1973.00	1931.00	1939.00
GarageCond	TA	TA	TA	TA	TA	TA	TA	TA	TA	TA
BsmtFinType2	Unf	Unf	Unf	Unf	Unf	Unf	Unf	BLQ	Unf	Unf
BsmtExposure	No	Gd	Mn	No	Av	No	Av	Mn	No	No
BsmtFinType1	GLQ	ALQ	GLQ	ALQ	GLQ	GLQ	GLQ	ALQ	Unf	GLQ
BsmtQual	Gd	Gd	Gd	TA	Gd	Gd	Ex	Gd	TA	TA
BsmtCond	TA	TA	TA	Gd	TA	TA	TA	TA	TA	TA
MasVnrArea	196.00	0.00	162.00	0.00	350.00	0.00	186.00	240.00	0.00	0.00
MasVnrType	BrkFace	None	BrkFace	None	BrkFace	None	Stone	Stone	None	None
Electrical	SBrkr	SBrkr	SBrkr	SBrkr	SBrkr	SBrkr	SBrkr	SBrkr	FuseF	SBrkr

Looks like it, we might need to handle categorical and numeric features separately. For a start, let's take alook at the categorical features.

Out[34]:

	GarageFinish	GarageQual	GarageType	GarageCond	BsmtFinType2	BsmtExposure	BsmtFinType1	Bsm
0	Unf	TA	Attchd	TA	Unf	No	Unf	

Out[38]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	La
Id										
1	60	RL	65.00	8450	Pave	Reg	Lvl	AllPub	Inside	
2	20	RL	80.00	9600	Pave	Reg	Lvl	AllPub	FR2	
3	60	RL	68.00	11250	Pave	IR1	Lvl	AllPub	Inside	
4	70	RL	60.00	9550	Pave	IR1	Lvl	AllPub	Corner	
5	60	RL	84.00	14260	Pave	IR1	Lvl	AllPub	FR2	
6	50	RL	85.00	14115	Pave	IR1	Lvl	AllPub	Inside	
7	20	RL	75.00	10084	Pave	Reg	Lvl	AllPub	Inside	
8	60	RL	nan	10382	Pave	IR1	Lvl	AllPub	Corner	
9	50	RM	51.00	6120	Pave	Reg	Lvl	AllPub	Inside	
10	190	RL	50.00	7420	Pave	Reg	Lvl	AllPub	Corner	

10 rows × 75 columns

Now, for the numerical features, we will apply **MEDIAN** function to the remaining numeric features

Out[40]:

	count	mean	std	min	25%	50%	75%	max	median	Q1	(
LotFrontage	1201.00	70.05	24.28	21.00	59.00	69.00	80.00	313.00	69.00	59.00	80.(
GarageYrBlt	1379.00	1978.51	24.69	1900.00	1961.00	1980.00	2002.00	2010.00	1980.00	1961.00	2002.(
MasVnrArea	1452.00	103.69	181.07	0.00	0.00	0.00	166.00	1600.00	0.00	0.00	166.(

Upon checking against the basic statistics on these numeric features, it look reasonable to apply **MEDIAN** on these features

Out[42]:

```
LotFrontage      69.00
GarageYrBlt    1980.00
MasVnrArea        0.00
dtype: float64
```

Out[45]:

Id	MSSubClass	MSZoning	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neig
1	60	RL	8450	Pave	Reg	Lvl	AllPub	Inside	Gtl	
2	20	RL	9600	Pave	Reg	Lvl	AllPub	FR2	Gtl	
3	60	RL	11250	Pave	IR1	Lvl	AllPub	Inside	Gtl	
4	70	RL	9550	Pave	IR1	Lvl	AllPub	Corner	Gtl	
5	60	RL	14260	Pave	IR1	Lvl	AllPub	FR2	Gtl	

5 rows × 75 columns

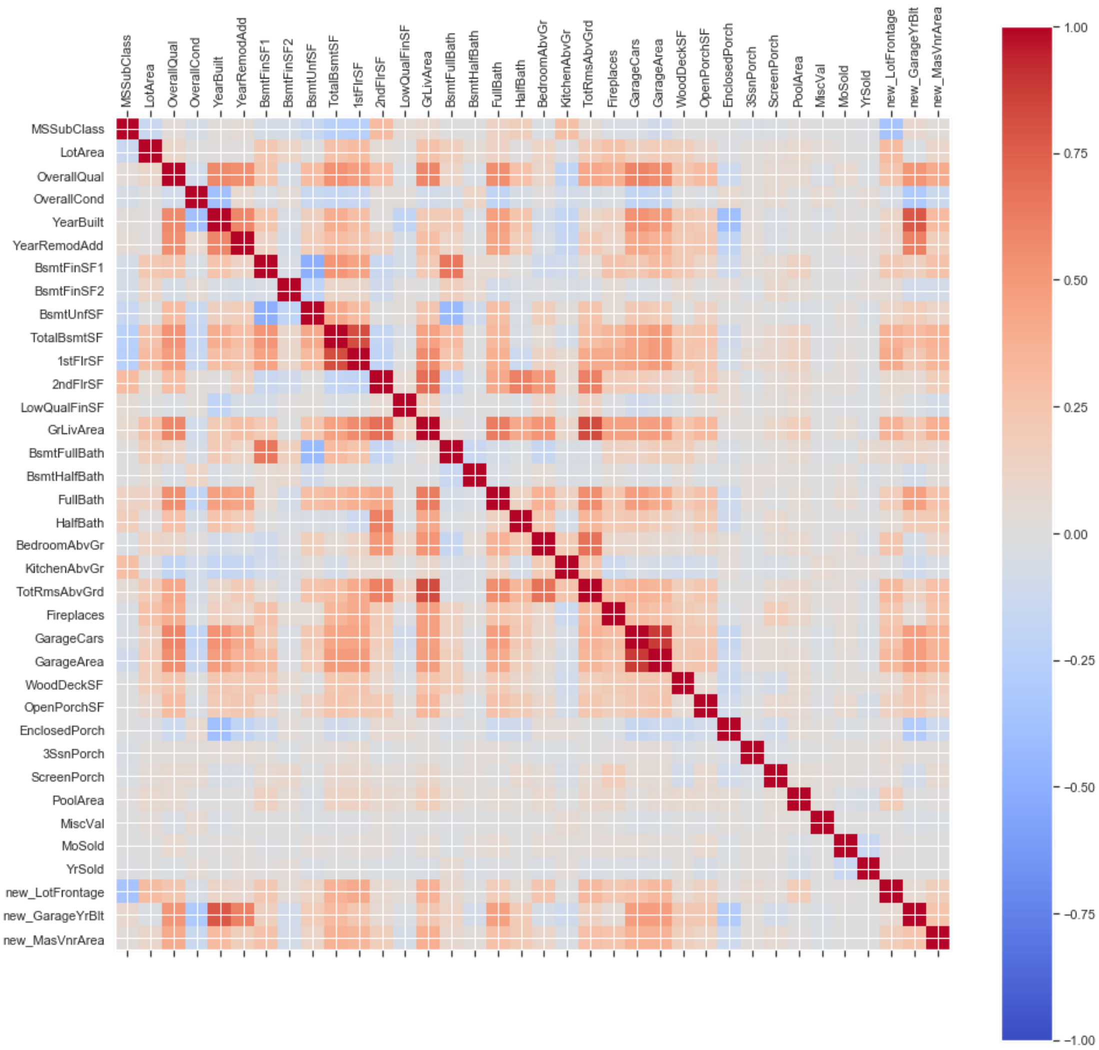
```
There are 0 columns with missing data.
These columns are as follows:

Features     No of Obs Missing #    No of Obs Missing %
----------   ---------------------  ---------------------


Since there are 0 columns with missing data, which means we have succesfully r
emoved or
imputed with mode/median accordingly.
```

Step3b:: Collinearity

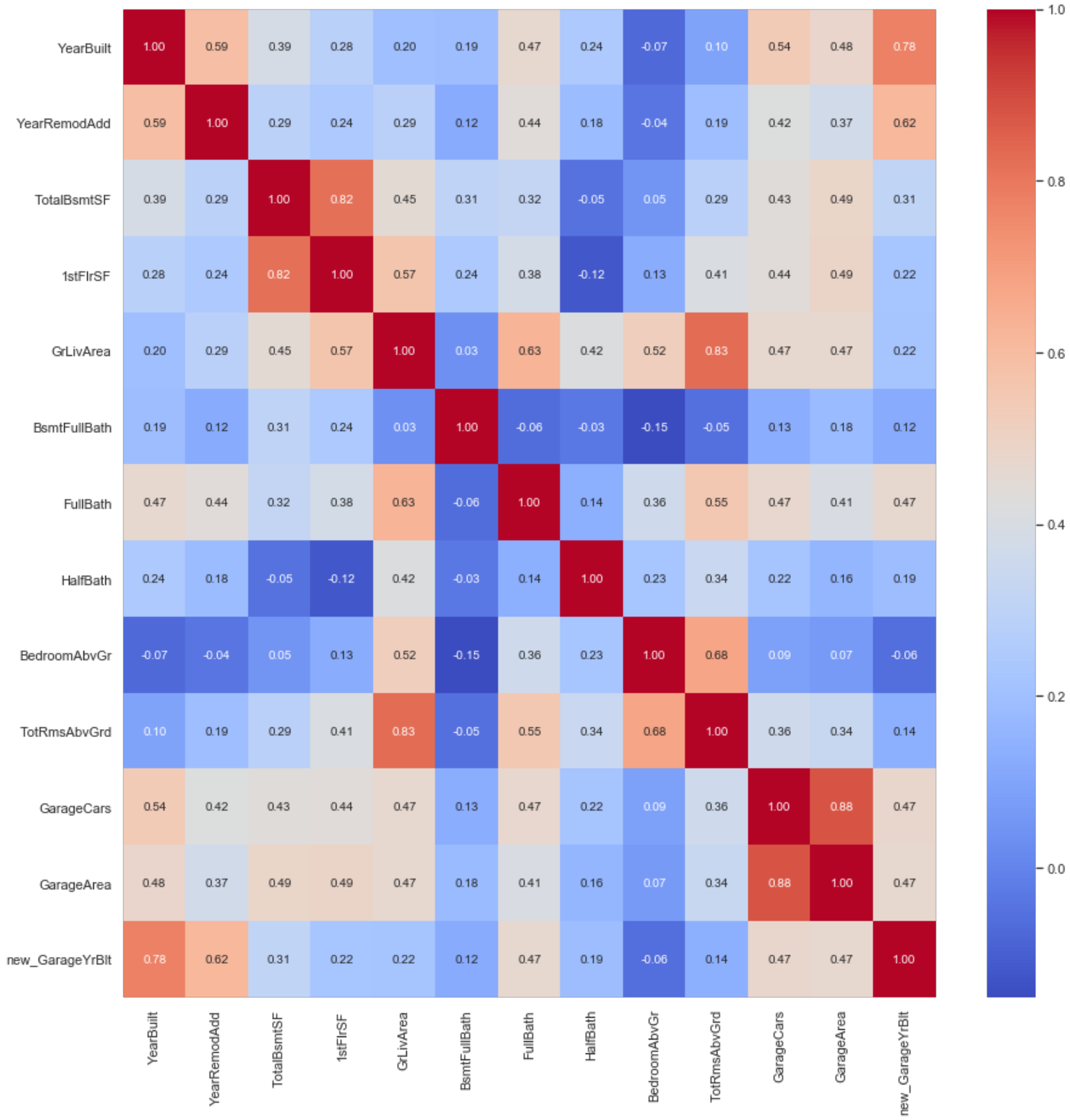
After fixing the missing values, the next thing we will look at is collinearity. As collinearity is very important before we start building any learning models.



Based on the heatmap above, the darker red/blue tone across the grid denote highly positively/negatively correlated. Which means some of these columns need to be removed from the model selection process

```
There are 13 features/columns with high correlation, which need to be removed.
==============================================================================
==
           Features
0         YearBuilt
1      YearRemodAdd
2       TotalBsmtSF
3          1stFlrSF
4         GrLivArea
5      BsmtFullBath
6          FullBath
7          HalfBath
8      BedroomAbvGr
9      TotRmsAbvGrd
10       GarageCars
11       GarageArea
12   new_GarageYrBlt
```

Out[50]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2bd28150>



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1460 entries, 1 to 1460
Data columns (total 62 columns):
MSSubClass          1460 non-null int64
MSZoning            1460 non-null object
LotArea             1460 non-null int64
Street              1460 non-null object
LotShape            1460 non-null object
LandContour         1460 non-null object
Utilities           1460 non-null object
LotConfig           1460 non-null object
LandSlope           1460 non-null object
Neighborhood        1460 non-null object
Condition1          1460 non-null object
Condition2          1460 non-null object
BldgType            1460 non-null object
HouseStyle          1460 non-null object
OverallQual         1460 non-null int64
OverallCond         1460 non-null int64
RoofStyle           1460 non-null object
RoofMatl            1460 non-null object
Exterior1st         1460 non-null object
Exterior2nd         1460 non-null object
ExterQual           1460 non-null object
ExterCond           1460 non-null object
Foundation          1460 non-null object
BsmtFinSF1          1460 non-null int64
BsmtFinSF2          1460 non-null int64
BsmtUnfSF           1460 non-null int64
Heating             1460 non-null object
HeatingQC           1460 non-null object
CentralAir          1460 non-null object
2ndFlrSF            1460 non-null int64
LowQualFinSF        1460 non-null int64
BsmtHalfBath        1460 non-null int64
KitchenAbvGr        1460 non-null int64
KitchenQual         1460 non-null object
Functional          1460 non-null object
Fireplaces          1460 non-null int64
PavedDrive          1460 non-null object
WoodDeckSF          1460 non-null int64
OpenPorchSF         1460 non-null int64
EnclosedPorch       1460 non-null int64
3SsnPorch           1460 non-null int64
ScreenPorch         1460 non-null int64
PoolArea            1460 non-null int64
MiscVal             1460 non-null int64
MoSold              1460 non-null int64
YrSold              1460 non-null int64
SaleType            1460 non-null object
SaleCondition       1460 non-null object
SalePrice           1460 non-null int64
new_GarageFinish    1460 non-null object
new_GarageQual      1460 non-null object
new_GarageType      1460 non-null object
new_GarageCond      1460 non-null object
new_BsmtFinType2    1460 non-null object
new_BsmtExposure    1460 non-null object
new_BsmtFinType1    1460 non-null object
new_BsmtQual        1460 non-null object
new_BsmtCond        1460 non-null object
new_MasVnrType      1460 non-null object
new_Electrical      1460 non-null object
new_LotFrontage     1460 non-null float64
new_MasVnrArea      1460 non-null float64
dtypes: float64(2), int64(22), object(38)
memory usage: 718.6+ KB
```

With collinearity check, these is our latest dataset which we can take to the next step - data visualisation. This time round, we removed a fair bit of numerical columns.

In summary, we have:

- **removed 5 features** due to high proportion of data with missing values,
- applied mode function on 11 categorical features,
- applied median function on 3 numeric features and
- **removed 13 columns** which are highly correlated.

```
After collinearity step, there are now 1460 observations and 62 features.
         Of which, 24 numeric features
         And, 38 non-numeric/object features.
```

Step 3c:: Managing Outliers

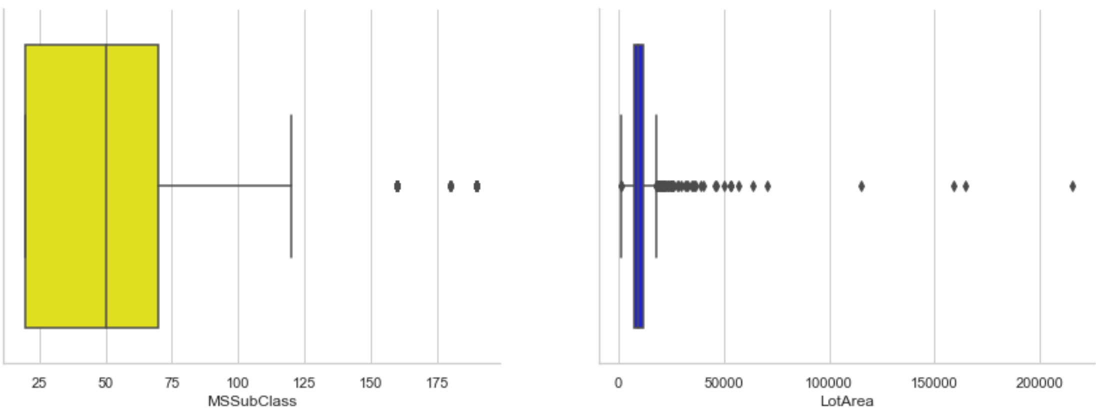
Come to the last part of data construction, managing outliers, I find boxplot does a really good job and applying Z-score (if you know how to do that).

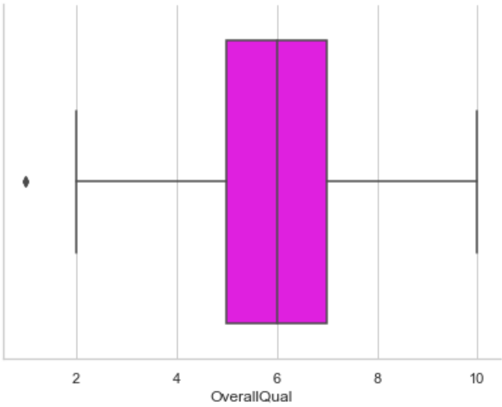
I have written a function below to identify outliers.

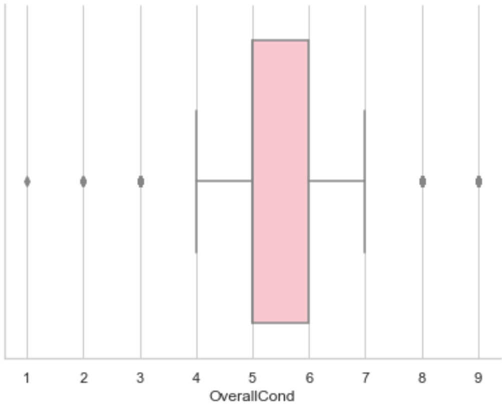
Out[54]:

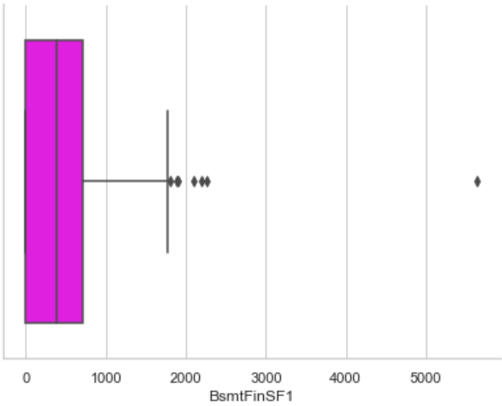
	MSSubClass	LotArea	OverallQual	OverallCond	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	2ndFlr
min	20.00	1300.00	1.00	1.00	0.00	0.00	0.00	0.
max	190.00	215245.00	10.00	9.00	5644.00	1474.00	2336.00	2065.
Q1	20.00	7553.50	5.00	5.00	0.00	0.00	223.00	0.
Q3	70.00	11601.50	7.00	6.00	712.25	0.00	808.00	728.
IQR	50.00	4048.00	2.00	1.00	712.25	0.00	585.00	728.
low_xtrem	20.00	1481.50	2.00	3.50	0.00	0.00	0.00	0.
hi_xtrem	145.00	17673.50	10.00	7.50	1780.62	0.00	1685.50	1820.

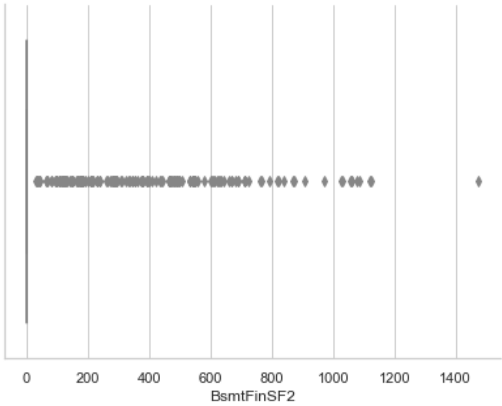
From above, we can see there are definitely outliers which need to be removed. But before we do that, it is always good to visual some of these boxplots for better illustration.

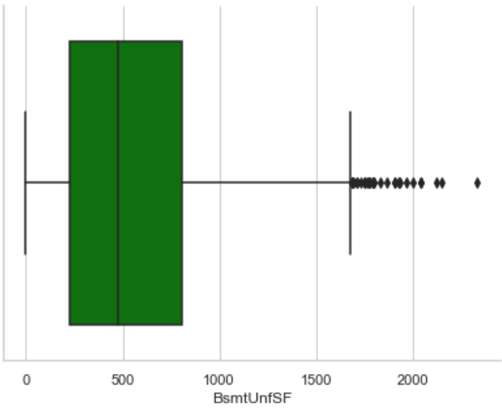


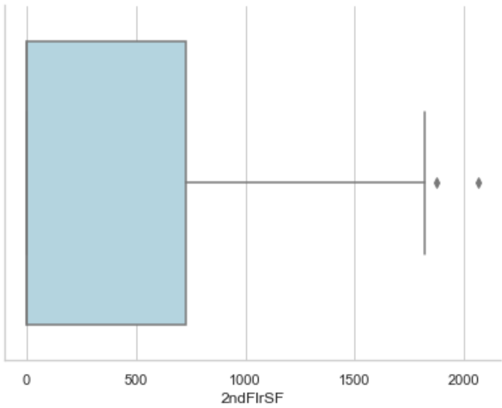

2
4
6
8
10
OverallQual

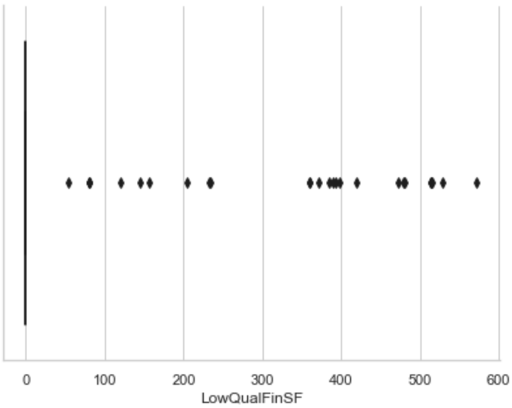

1
2
3
4
5
6
7
8
9
OverallCond

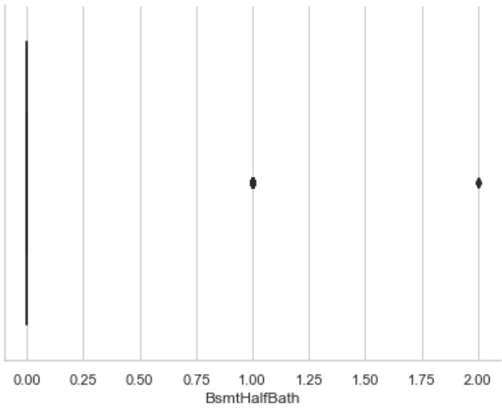

0
1000
2000
3000
4000
5000
BsmtFinSF1

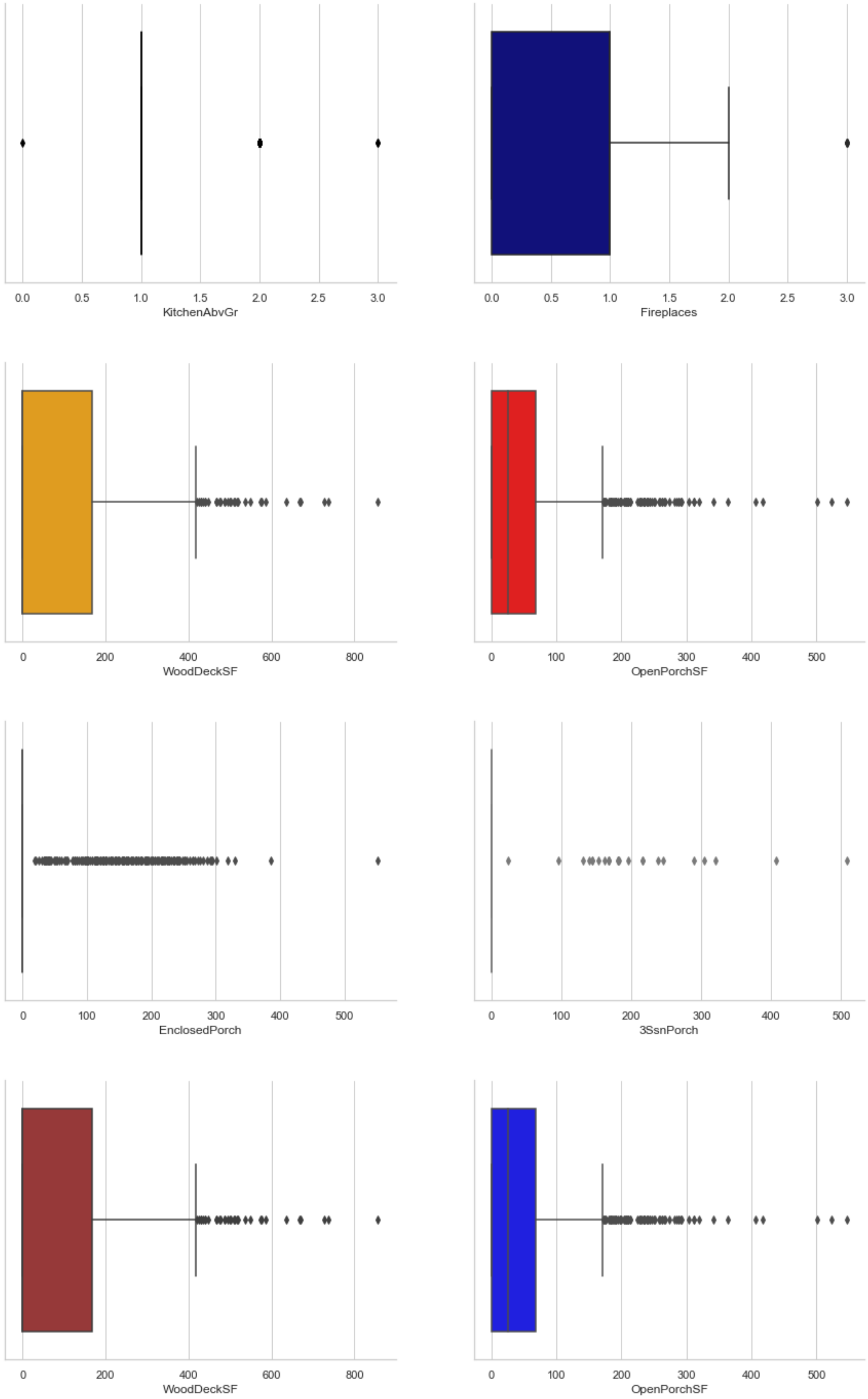

0
200
400
600
800
1000
1200
1400
BsmtFinSF2

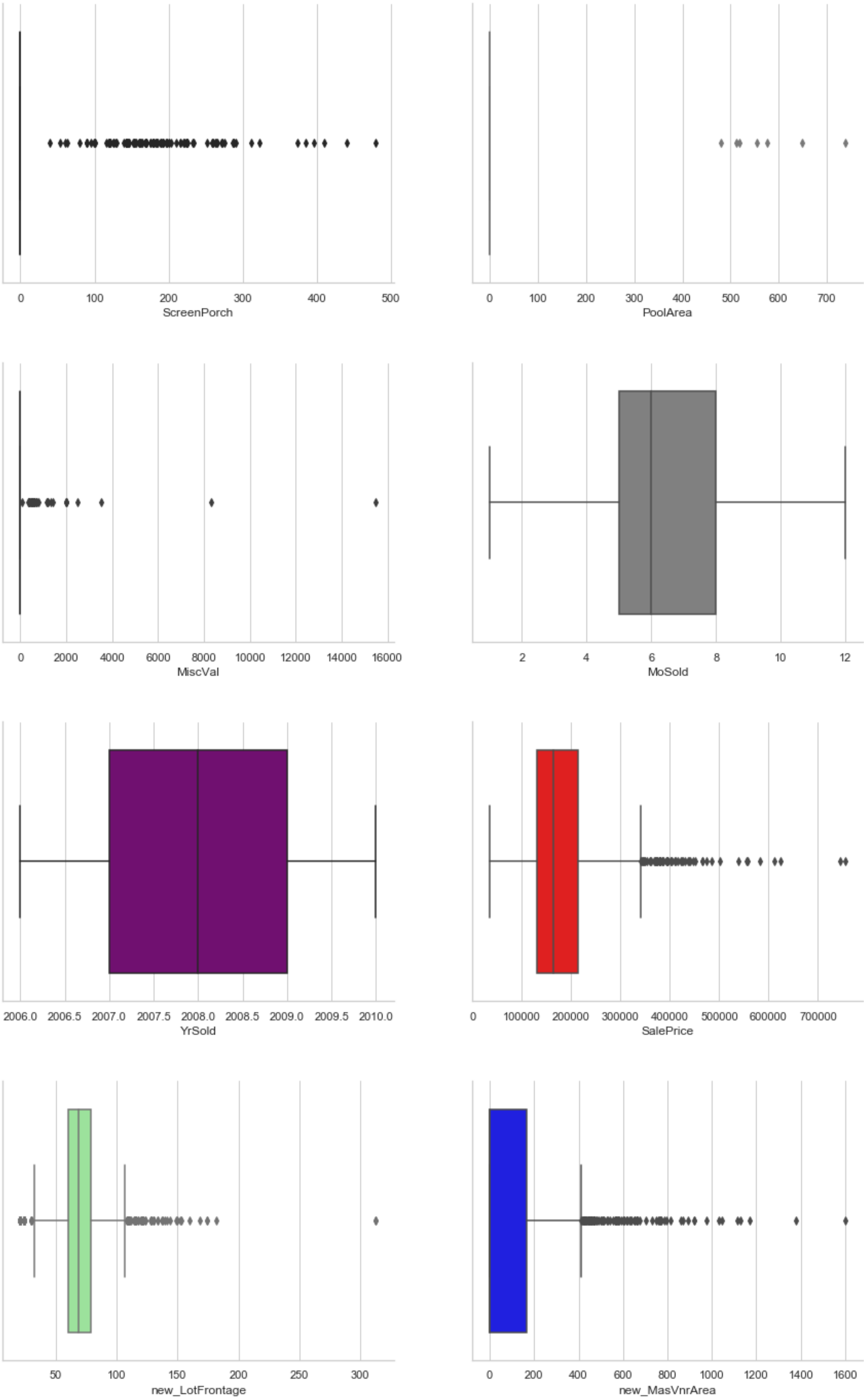

0
500
1000
1500
2000
BsmtUnfSF


0
500
1000
1500
2000
2ndFlrSF


0
100
200
300
400
500
600
LowQualFinSF


0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
BsmtHalfBath


KitchenAbvGr
Fireplaces
WoodDeckSF
OpenPorchSF
EnclosedPorch
3SsnPorch
WoodDeckSF
OpenPorchSF



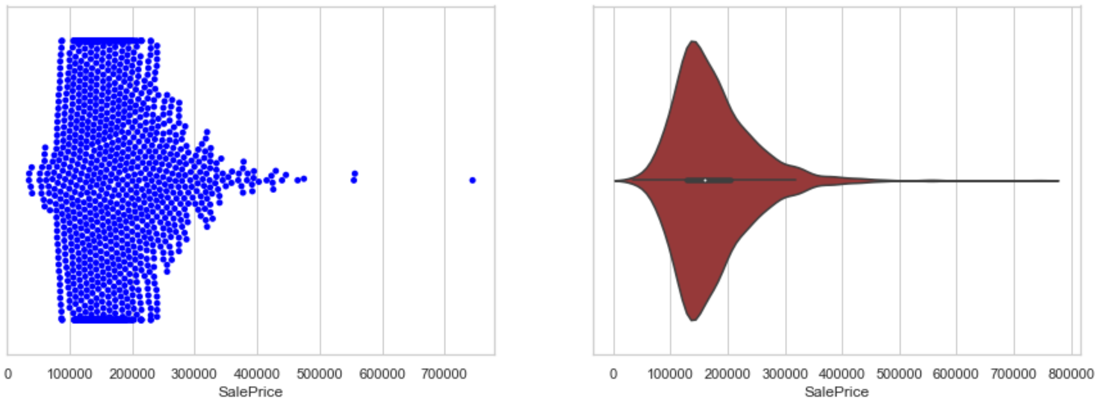
So after having a look at the distribution and boxplot graphs, we will proceed to remove these outliers for better accuracy in our model development.

```
After eliminating outliers, there are now 1362 observations and 62 features.
In summary, we still have 93.29% of the data retained, which is still reasonab
le.
```

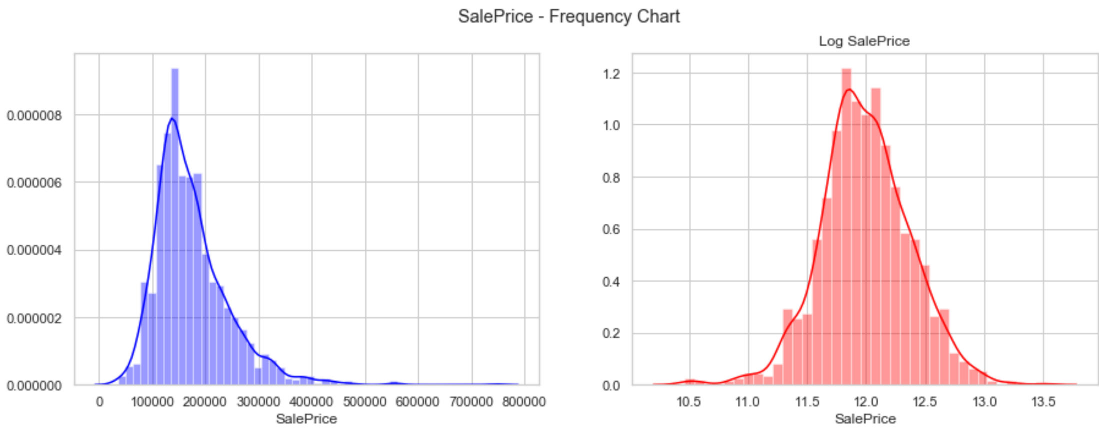
Step 4: Data Visualisation

As this exercise is to predict on the dependent variable, in this case, the SalePrice. We will visualise how the original data is distributed

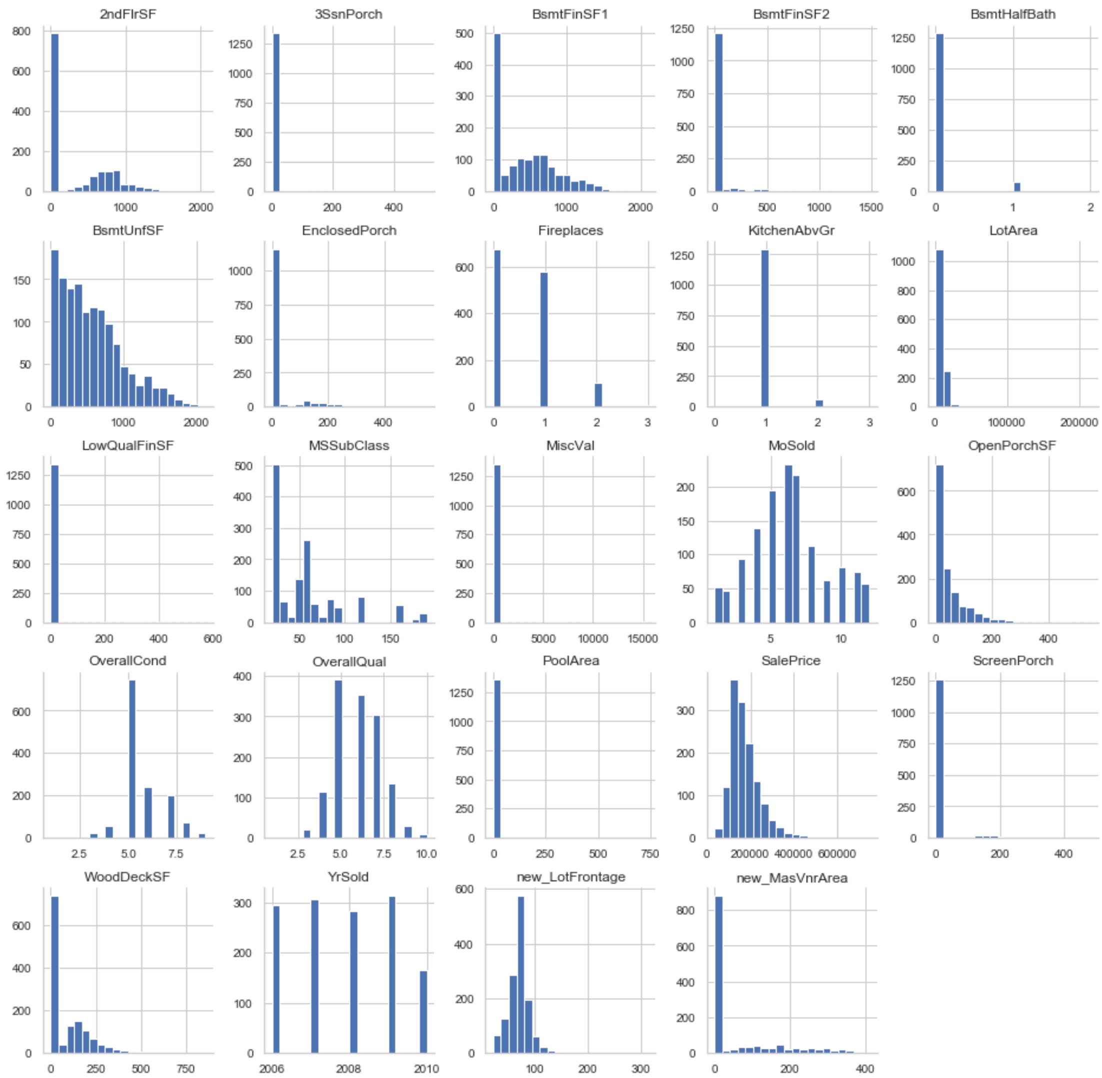
Out[62]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1e212350>



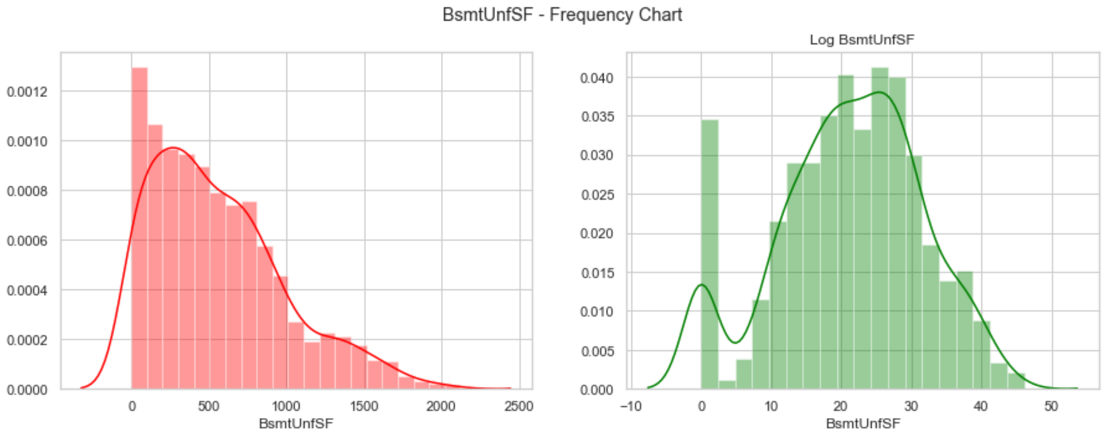
Out[63]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1e0294d0>



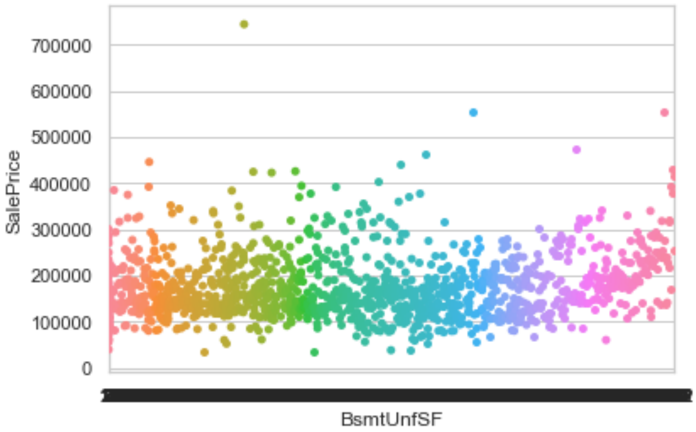
It looks like SalePrice is leftly skewed (see histogram on the left above) and after applying log transformation, SalePrice is normally distributed. This is important if we are using regression technique.



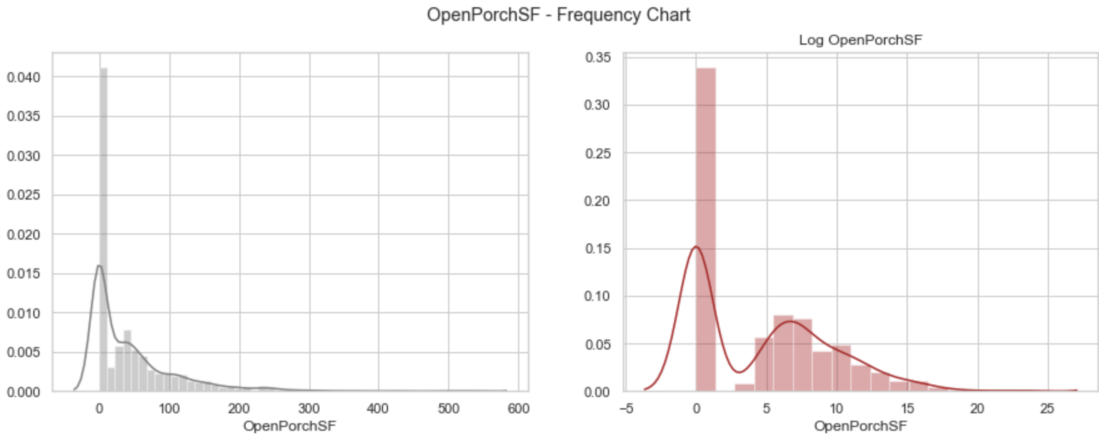
Out[65]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1f184e10>

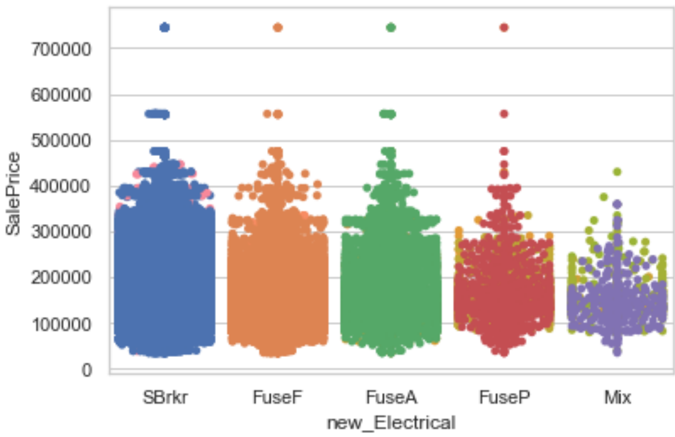


Out[66]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1e655610>

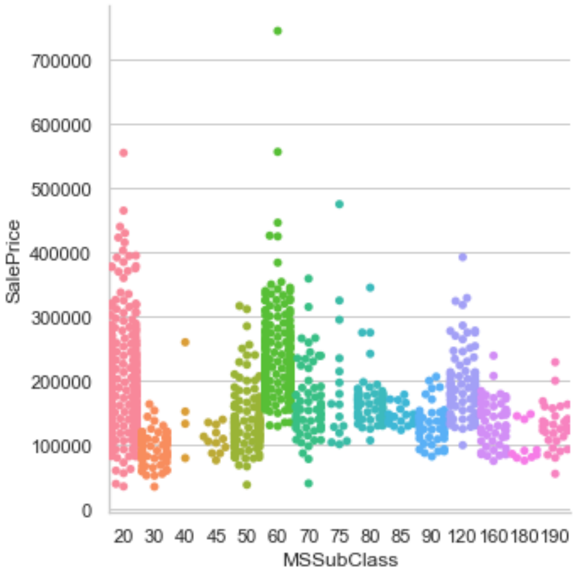


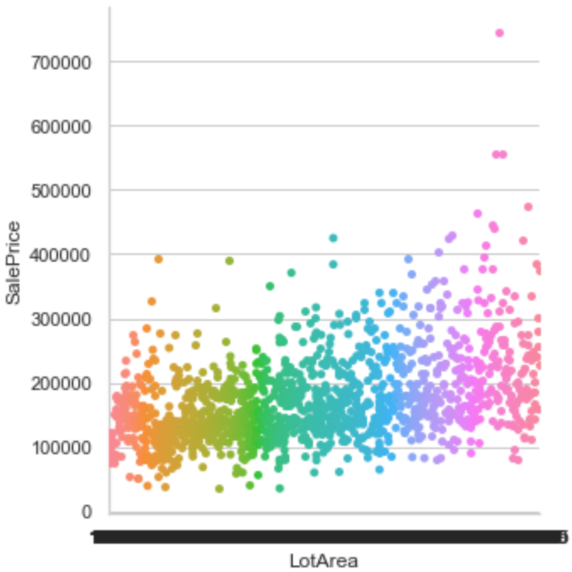
Out[67]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1e42c350>

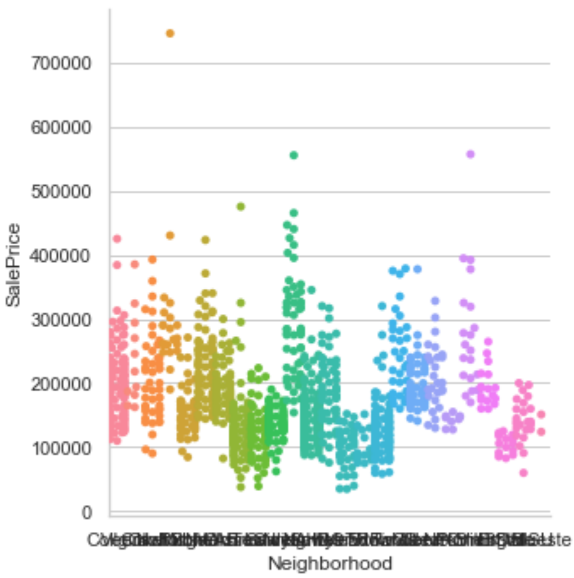




Out[69]: <seaborn.axisgrid.FacetGrid at 0x1a1fc54c10>








```
In this section, there is no change. There are still have 1362 observations wi
th 62 features.
```

Step 5 :: Features Engineering

The process of constructing additional features is very time-consuming because each new feature usually requires several steps to build, especially when using information from more than one table. We can group the operations of feature creation into two categories: transformations and aggregations. Let’s look at a few examples to see these concepts in action.

```
Response Code : 200 which means all good.
```

Now that the web data has been scrapped successfully, we will now massage the data to get the table information from the web.

```
Out[74]: bs4.element.ResultSet
```

```
Out[75]: bs4.element.Tag
```

This part can be easily done by assessing the exact tag location in the webpage.

Out[77]:
```
array([['1914', '$20.00', '0'],
       ['1915', '$20.03', '0.16%'],
       ['1916', '$20.60', '2.85%'],
       ['1917', '$23.38', '13.48%'],
       ['1918', '$27.54', '17.79%'],
       ['1919', '$33.40', '21.27%'],
       ['1920', '$39.16', '17.25%'],
       ['1921', '$35.65', '-8.96%'],
       ['1922', '$32.49', '-8.86%'],
       ['1923', '$32.98', '1.51%'],
       ['1924', '$33.21', '0.67%'],
       ['1925', '$33.57', '1.10%'],
       ['1926', '$34.81', '3.69%'],
       ['1927', '$34.44', '-1.05%'],
       ['1928', '$33.78', '-1.94%'],
       ['1929', '$33.62', '-0.47%'],
       ['1930', '$33.38', '-0.71%'],
       ['1931', '$30.79', '-7.75%'],
       ['1932', '$27.76', '-9.85%'],
       ['1933', '$25.92', '-6.63%'],
       ['1934', '$26.73', '3.12%'],
       ['1935', '$27.29', '2.08%'],
       ['1936', '$27.44', '0.58%'],
       ['1937', '$28.06', '2.26%'],
       ['1938', '$27.35', '-2.55%'],
       ['1939', '$26.94', '-1.51%'],
       ['1940', '$27.19', '0.94%'],
       ['1941', '$28.40', '4.44%'],
       ['1942', '$31.43', '10.68%'],
       ['1943', '$33.08', '5.25%'],
       ['1944', '$33.44', '1.10%'],
       ['1945', '$34.13', '2.04%'],
       ['1946', '$36.92', '8.19%'],
       ['1947', '$42.03', '13.84%'],
       ['1948', '$45.49', '8.23%'],
       ['1949', '$44.94', '-1.22%'],
       ['1950', '$45.52', '1.31%'],
       ['1951', '$48.57', '6.69%'],
       ['1952', '$49.63', '2.19%'],
       ['1953', '$49.76', '0.26%'],
       ['1954', '$49.95', '0.38%'],
       ['1955', '$50.19', '0.48%'],
       ['1956', '$51.57', '2.75%'],
       ['1957', '$53.38', '3.51%'],
       ['1958', '$55.10', '3.21%'],
       ['1959', '$55.52', '0.78%'],
       ['1960', '$56.57', '1.89%'],
       ['1961', '$57.43', '1.52%'],
       ['1962', '$58.67', '2.16%'],
       ['1963', '$59.86', '2.03%'],
       ['1964', '$60.67', '1.35%'],
       ['1965', '$61.81', '1.88%'],
       ['1966', '$63.71', '3.08%'],
       ['1967', '$65.38', '2.62%'],
       ['1968', '$67.95', '3.93%'],
       ['1969', '$71.71', '5.54%'],
       ['1970', '$76.10', '6.11%'],
       ['1971', '$80.14', '5.32%'],
       ['1972', '$83.00', '3.57%'],
       ['1973', '$87.62', '5.57%'],
       ['1974', '$96.67', '10.33%'],
       ['1975', '$105.38', '9.01%'],
       ['1976', '$113.95', '8.13%'],
       ['1977', '$119.76', '5.10%'],
       ['1978', '$126.16', '5.34%'],
       ['1979', '$138.86', '10.07%'],
       ['1980', '$156.48', '12.69%'],
       ['1981', '$174.25', '11.36%'],
       ['1982', '$181.71', '4.28%'],
```

Great! Looks like it the data is almost ready except a little of touch-ups before we can fully use it and then merge this information back into our dataset.

Out[78]:
```
Year       int64
CPI      float64
dtype: object
```

Out[84]:
```
dtype('int64')
```

Before merging the data, make sure both columns carried the same datatype.

Out[87]:

	MSSubClass	MSZoning	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neigl
0	60	RL	8450	Pave	Reg	Lvl	AllPub	Inside	Gtl	
1	60	RL	11250	Pave	IR1	Lvl	AllPub	Inside	Gtl	
2	60	RL	14260	Pave	IR1	Lvl	AllPub	FR2	Gtl	
3	50	RM	6120	Pave	Reg	Lvl	AllPub	Inside	Gtl	
4	190	RL	7420	Pave	Reg	Lvl	AllPub	Corner	Gtl	

5 rows × 66 columns

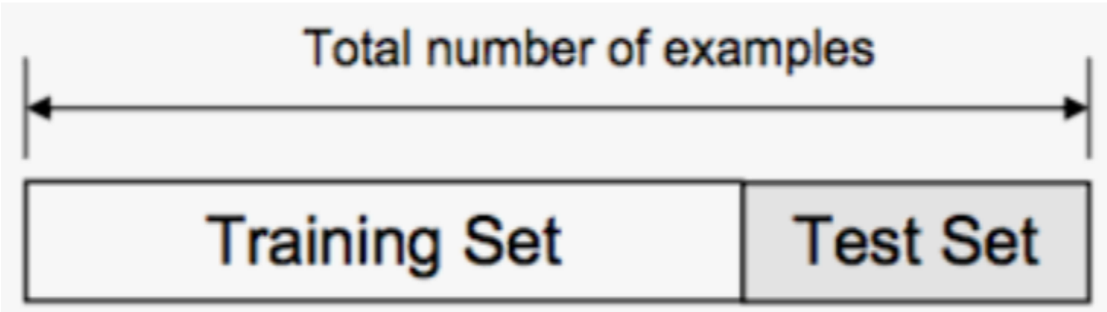
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1362 entries, 0 to 1361
Data columns (total 65 columns):
MSSubClass          1362 non-null int64
MSZoning            1362 non-null object
LotArea             1362 non-null int64
Street              1362 non-null object
LotShape            1362 non-null object
LandContour         1362 non-null object
Utilities           1362 non-null object
LotConfig           1362 non-null object
LandSlope           1362 non-null object
Neighborhood        1362 non-null object
Condition1          1362 non-null object
Condition2          1362 non-null object
BldgType            1362 non-null object
HouseStyle          1362 non-null object
OverallQual         1362 non-null int64
OverallCond         1362 non-null int64
RoofStyle           1362 non-null object
RoofMatl            1362 non-null object
Exterior1st         1362 non-null object
Exterior2nd         1362 non-null object
ExterQual           1362 non-null object
ExterCond           1362 non-null object
Foundation          1362 non-null object
BsmtFinSF1          1362 non-null int64
BsmtFinSF2          1362 non-null int64
BsmtUnfSF           1362 non-null int64
Heating             1362 non-null object
HeatingQC           1362 non-null object
CentralAir          1362 non-null object
2ndFlrSF            1362 non-null int64
LowQualFinSF        1362 non-null int64
BsmtHalfBath        1362 non-null int64
KitchenAbvGr        1362 non-null int64
KitchenQual         1362 non-null object
Functional          1362 non-null object
Fireplaces          1362 non-null int64
PavedDrive          1362 non-null object
WoodDeckSF          1362 non-null int64
OpenPorchSF         1362 non-null int64
EnclosedPorch       1362 non-null int64
3SsnPorch           1362 non-null int64
ScreenPorch         1362 non-null int64
PoolArea            1362 non-null int64
MiscVal             1362 non-null int64
MoSold              1362 non-null int64
YrSold              1362 non-null int64
SaleType            1362 non-null object
SaleCondition       1362 non-null object
SalePrice           1362 non-null int64
new_GarageFinish    1362 non-null object
new_GarageQual      1362 non-null object
new_GarageType      1362 non-null object
new_GarageCond      1362 non-null object
new_BsmtFinType2    1362 non-null object
new_BsmtExposure    1362 non-null object
new_BsmtFinType1    1362 non-null object
new_BsmtQual        1362 non-null object
new_BsmtCond        1362 non-null object
new_MasVnrType      1362 non-null object
new_Electrical      1362 non-null object
new_LotFrontage     1362 non-null float64
new_MasVnrArea      1362 non-null float64
CPI                 1362 non-null float64
new_BsmtUnfSF       1362 non-null float64
new_OpenPorchSF     1362 non-null float64
dtypes: float64(5), int64(22), object(38)
```

```
With some features engineering, we are still keeping 1362 observations but hav
ing 66 features now.
In summary, we added 4 new data columns (i.e. CPI, transformed BsmtUnfSF & Ope
nPorchSF).
```

Step 6:: Train/Test Split and Cross Validation with Hyperparams Tuning

Now that we have completed cleansing the dataset, removed/imputed missing values, added new/engineered features and transformed target variables, we will now split the dataset for cross validation and model fitting.

Out[89]:




```
So in X dataframe, there are 65 independent variables and in Y dataframe, only
1 target variable.
Also there are 1362 number of observations.
```

Drawback of Train/Test split It provides high variance estimate since changing which observations or examples happens to be in testing dataset can significantly change testing accuracy. Now you may say what if we split the datasets into bunch of train/test splits, calculate their training accuracy and average the results together. That is where cross-validation comes to play. The common type of cross validation is k-fold cross validation.

Out[91]:

	Street_Grvl	Street_Pave	LotShape_IR1	LotShape_IR2	LotShape_IR3	LotShape_Reg	LandContour_Bnk
0	0	1	0	0	0	1	0
1	0	1	1	0	0	0	0
2	0	1	1	0	0	0	0
3	0	1	0	0	0	1	0
4	0	1	0	0	0	1	0

5 rows × 225 columns

```
Training examples: 953
Validation examples:  409
```

Step 7:: Model Development

This section, we will run some basic regression learning models to identify the most important features to be used in our prediction model.

- **Support Vector Regressor**
    - Support Vector Machine can also be used as a regression method, maintaining all the main features that characterize the algorithm (maximal margin). The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences. First of all, because output is a real number it becomes very difficult to predict the information at hand, which has infinite possibilities. In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which would have already requested from the problem. But besides this fact, there is also a more complicated reason, the algorithm is more complicated therefore to be taken in consideration. However, the main idea is always the same: to minimize error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated.
        - **Rule**: Standardise all independent variables & target variable before deploying SVR
- **DecisionTree Regressor**
    - Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. Decision tree is arriving at an estimate by asking a series of questions to the data, each question narrowing our possible values until the model get confident enough to make a single prediction. The order of the question as well as their content are being determined by the model. In addition, the questions asked are all in a True/False form.
        - **Rule**: Decision trees can handle both categorical and numerical data.
- **ExtraTree Regressor**
    - This class implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Rule: The maximum depth of the tree. But sometimes, it is also called as extreme random forests. In my personal opinion, this model sits in between decision tree and random forest. And when it comes to performance wise, extra trees seem to keep a higher performance in presence of noisy features. Extra Tree lies in the fact that, instead of computing the locally optimal feature/split combination (for the random forest), for each feature under consideration, a random value is selected for the split (for the extra trees).
- **Random Forest Regressor**
    - Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.Random decision forests correct for decision trees' habit of overfitting to their training set. In summary, random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.
        - **Rule**: The number of trees in the forest.
- **AdaBoost Regressor**
    - AdaBoost is the first stepping stone in the world of Boosting. In contrast to bagging techniques like Random Forest, in which trees are grown to their maximum extent, boosting makes use of trees with fewer splits. This model is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction. AdaBoost is sensitive to noisy data and outliers. In some problems it can be less susceptible to the overfitting problem than other learning algorithms.
- **Gradient Boosting Regressor**
    - It (GBM for short) ensembles of weak prediction models, typically decision trees. The objective of any supervised learning algorithm is to define a loss function and minimize it. So, the intuition behind gradient boosting algorithm is to repetitively leverage the patterns in residuals and strengthen a model with weak predictions and make it better. Once we reach a stage that residuals do not have any pattern that could be modeled, we can stop modeling residuals (otherwise it might lead to overfitting). Algorithmically, we are minimizing our loss function, such that test loss reach its minima. In summary, this follows Gradient Descent concept for optimizing the loss function.
- **XGBoost Regressor**
    - XGBoost is an algorithm that has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. XGBoost stands for eXtreme Gradient Boosting. Both GBM & XGBoost are the same, XGBoost and GBM, both works on the same principle. In XGBoost parallel computation is possible, means in XGBoost parallelly many GBM's are working. In XGBoost tunning parameters are more. Any of them can be

Out[99]:

Support Vector Machine

```
R2 Score    : 0.7381
MSE         : 0.0427
RMSE        : 0.2066
Optimal hyperparameter(s): {'C': 1000.0, 'epsilon': 0.001, 'kernel': 'rbf'}.
Optimal Estimator:
SVR(C=1000.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.001, gamma='scale
',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

Decision Tree

```
R2 Score    : 0.7342
MSE         : 0.0433
RMSE        : 0.2082
Optimal hyperparameter(s): {'criterion': 'mse', 'max_depth': 10, 'min_samples_
split': 40}.
Optimal Estimator:
DecisionTreeRegressor(criterion='mse', max_depth=10, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=40, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=24, splitter='best')
```

Extra Tree

```
R2 Score   : 0.8101
MSE        : 0.031
RMSE       : 0.176
Optimal hyperparameter(s): {'max_features': 30, 'n_estimators': 1500}.
Optimal Estimator:
ExtraTreesRegressor(bootstrap=False, criterion='mse', max_depth=None,
                    max_features=30, max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=1500,
                    n_jobs=None, oob_score=False, random_state=24, verbose=0,
                    warm_start=False)
```

Random Forests

```
R2 Score   : 0.8459
MSE        : 0.0251
RMSE       : 0.1585
Optimal hyperparameter(s): {'n_estimators': 3000}.
Optimal Estimator:
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=3000,
                      n_jobs=None, oob_score=False, random_state=24, verbose=
0,
                      warm_start=False)
```

Gradient Boosting

```
R2 Score   : 0.8875
MSE        : 0.0183
RMSE       : 0.1354
Optimal hyperparameter(s): {'learning_rate': 0.03, 'max_features': 'sqrt', 'n_
estimators': 3000}.
Optimal Estimator:
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                          learning_rate=0.03, loss='ls', max_depth=3,
                          max_features='sqrt', max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=3000,
                          n_iter_no_change=None, presort='auto',
                          random_state=24, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=Fals
e)
```

AdaBoost

```
R2 Score   : 0.8046
MSE        : 0.0318
RMSE       : 0.1784
Optimal hyperparameter(s): {'learning_rate': 0.5, 'n_estimators': 3000}.
Optimal Estimator:
AdaBoostRegressor(base_estimator=None, learning_rate=0.5, loss='linear',
                  n_estimators=3000, random_state=24)
```

XGBoost

```
[20:11:47] WARNING: src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[20:12:03] WARNING: src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[20:12:20] WARNING: src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[20:12:36] WARNING: src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[20:12:52] WARNING: src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[20:13:08] WARNING: src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[20:13:24] WARNING: src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[20:13:40] WARNING: src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[20:13:56] WARNING: src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[20:14:12] WARNING: src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
[20:14:28] WARNING: src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
R2 Score   : 0.8789
MSE        : 0.0197
RMSE       : 0.1405
Optimal hyperparameter(s): {'learning_rate': 0.02, 'max_depth': 5, 'n_estimato
rs': 1000}.
Optimal Estimator:
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.02, max_delta_step=0,
             max_depth=5, min_child_weight=1, missing=None, n_estimators=1000,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=24,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

Light GBM

CatBoost

```
0:	learn: 0.3698692	total: 11.4ms	remaining: 22.8s
1:	learn: 0.3678352	total: 23.1ms	remaining: 23.1s
2:	learn: 0.3660123	total: 45ms	remaining: 30s
3:	learn: 0.3640926	total: 50.8ms	remaining: 25.4s
4:	learn: 0.3621144	total: 64.4ms	remaining: 25.7s
5:	learn: 0.3601741	total: 71.6ms	remaining: 23.8s
6:	learn: 0.3584215	total: 80.3ms	remaining: 22.9s
7:	learn: 0.3566557	total: 86.6ms	remaining: 21.6s
8:	learn: 0.3548704	total: 92.9ms	remaining: 20.5s
9:	learn: 0.3530245	total: 98.9ms	remaining: 19.7s
10:	learn: 0.3513392	total: 105ms	remaining: 19s
11:	learn: 0.3495341	total: 111ms	remaining: 18.4s
12:	learn: 0.3477495	total: 117ms	remaining: 18s
13:	learn: 0.3461279	total: 123ms	remaining: 17.5s
14:	learn: 0.3443992	total: 130ms	remaining: 17.3s
15:	learn: 0.3426235	total: 136ms	remaining: 16.9s
16:	learn: 0.3409551	total: 142ms	remaining: 16.6s
17:	learn: 0.3392668	total: 148ms	remaining: 16.3s
18:	learn: 0.3377111	total: 154ms	remaining: 16.1s
19:	learn: 0.3359597	total: 160ms	remaining: 15.8s
20:	learn: 0.3342743	total: 166ms	remaining: 15.6s
21:	learn: 0.3326281	total: 172ms	remaining: 15.4s
22:	learn: 0.3311236	total: 178ms	remaining: 15.3s
23:	learn: 0.3294708	total: 184ms	remaining: 15.1s
24:	learn: 0.3280725	total: 190ms	remaining: 15s
25:	learn: 0.3266595	total: 196ms	remaining: 14.8s
26:	learn: 0.3251069	total: 202ms	remaining: 14.7s
27:	learn: 0.3235301	total: 211ms	remaining: 14.9s
28:	learn: 0.3219792	total: 227ms	remaining: 15.5s
29:	learn: 0.3205190	total: 234ms	remaining: 15.3s
30:	learn: 0.3190128	total: 239ms	remaining: 15.2s
31:	learn: 0.3175946	total: 253ms	remaining: 15.6s
32:	learn: 0.3164112	total: 260ms	remaining: 15.5s
33:	learn: 0.3148448	total: 266ms	remaining: 15.4s
34:	learn: 0.3133569	total: 274ms	remaining: 15.4s
35:	learn: 0.3119571	total: 280ms	remaining: 15.3s
36:	learn: 0.3106652	total: 287ms	remaining: 15.2s
37:	learn: 0.3092358	total: 293ms	remaining: 15.1s
38:	learn: 0.3078719	total: 299ms	remaining: 15s
39:	learn: 0.3065867	total: 305ms	remaining: 14.9s
40:	learn: 0.3052142	total: 311ms	remaining: 14.9s
41:	learn: 0.3038410	total: 317ms	remaining: 14.8s
42:	learn: 0.3024747	total: 323ms	remaining: 14.7s
43:	learn: 0.3011759	total: 329ms	remaining: 14.6s
44:	learn: 0.2998979	total: 335ms	remaining: 14.6s
45:	learn: 0.2985808	total: 342ms	remaining: 14.5s
46:	learn: 0.2972943	total: 347ms	remaining: 14.4s
47:	learn: 0.2960470	total: 354ms	remaining: 14.4s
48:	learn: 0.2946958	total: 360ms	remaining: 14.3s
49:	learn: 0.2935066	total: 364ms	remaining: 14.2s
50:	learn: 0.2922166	total: 370ms	remaining: 14.1s
51:	learn: 0.2909577	total: 376ms	remaining: 14.1s
52:	learn: 0.2896621	total: 382ms	remaining: 14s
53:	learn: 0.2884248	total: 388ms	remaining: 14s
54:	learn: 0.2871877	total: 394ms	remaining: 13.9s
55:	learn: 0.2858988	total: 400ms	remaining: 13.9s
56:	learn: 0.2846304	total: 406ms	remaining: 13.9s
57:	learn: 0.2833973	total: 414ms	remaining: 13.8s
58:	learn: 0.2822967	total: 422ms	remaining: 13.9s
59:	learn: 0.2811390	total: 430ms	remaining: 13.9s
60:	learn: 0.2799584	total: 444ms	remaining: 14.1s
61:	learn: 0.2789152	total: 450ms	remaining: 14.1s
62:	learn: 0.2776149	total: 459ms	remaining: 14.1s
63:	learn: 0.2765137	total: 465ms	remaining: 14.1s
64:	learn: 0.2753440	total: 471ms	remaining: 14s
65:	learn: 0.2743085	total: 477ms	remaining: 14s
66:	learn: 0.2731885	total: 483ms	remaining: 13.9s
67:	learn: 0.2721525	total: 490ms	remaining: 13.9s
68:	learn: 0.2710530	total: 495ms	remaining: 13.9s
```

Which Model Has the Best Performance?

From the hyperparameters tuning process below, we have gotten most of the models with their best parameters. So in this part, we will call out each and every of them and compare their performance.

```
[12:08:41] WARNING: src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
0:	learn: 0.3631696	total: 78.5ms	remaining: 2m 36s
1:	learn: 0.3611579	total: 84.7ms	remaining: 1m 24s
2:	learn: 0.3593385	total: 90.8ms	remaining: 1m
3:	learn: 0.3572012	total: 96.7ms	remaining: 48.3s
4:	learn: 0.3554998	total: 103ms	remaining: 41s
5:	learn: 0.3536831	total: 109ms	remaining: 36.1s
6:	learn: 0.3517982	total: 114ms	remaining: 32.5s
7:	learn: 0.3499999	total: 120ms	remaining: 29.9s
8:	learn: 0.3483170	total: 126ms	remaining: 27.9s
9:	learn: 0.3466251	total: 132ms	remaining: 26.2s
10:	learn: 0.3448163	total: 138ms	remaining: 24.9s
11:	learn: 0.3430775	total: 144ms	remaining: 23.8s
12:	learn: 0.3414047	total: 151ms	remaining: 23.1s
13:	learn: 0.3397186	total: 157ms	remaining: 22.3s
14:	learn: 0.3379550	total: 164ms	remaining: 21.8s
15:	learn: 0.3363028	total: 171ms	remaining: 21.2s
16:	learn: 0.3345596	total: 177ms	remaining: 20.6s
17:	learn: 0.3328670	total: 183ms	remaining: 20.1s
18:	learn: 0.3311341	total: 189ms	remaining: 19.7s
19:	learn: 0.3295795	total: 197ms	remaining: 19.5s
20:	learn: 0.3279493	total: 205ms	remaining: 19.3s
21:	learn: 0.3264573	total: 211ms	remaining: 19s
22:	learn: 0.3249050	total: 217ms	remaining: 18.7s
23:	learn: 0.3233423	total: 223ms	remaining: 18.4s
24:	learn: 0.3216840	total: 230ms	remaining: 18.2s
25:	learn: 0.3201139	total: 246ms	remaining: 18.7s
26:	learn: 0.3185966	total: 252ms	remaining: 18.4s
27:	learn: 0.3169656	total: 258ms	remaining: 18.2s
28:	learn: 0.3154448	total: 264ms	remaining: 17.9s
29:	learn: 0.3140814	total: 270ms	remaining: 17.7s
30:	learn: 0.3125907	total: 279ms	remaining: 17.7s
31:	learn: 0.3111431	total: 288ms	remaining: 17.7s
32:	learn: 0.3097749	total: 294ms	remaining: 17.5s
33:	learn: 0.3085077	total: 300ms	remaining: 17.3s
34:	learn: 0.3069630	total: 308ms	remaining: 17.3s
35:	learn: 0.3054645	total: 314ms	remaining: 17.1s
36:	learn: 0.3041763	total: 321ms	remaining: 17.1s
37:	learn: 0.3029609	total: 327ms	remaining: 16.9s
38:	learn: 0.3016048	total: 335ms	remaining: 16.8s
39:	learn: 0.3003799	total: 341ms	remaining: 16.7s
40:	learn: 0.2991962	total: 347ms	remaining: 16.6s
41:	learn: 0.2978391	total: 354ms	remaining: 16.5s
42:	learn: 0.2965498	total: 362ms	remaining: 16.5s
43:	learn: 0.2952134	total: 369ms	remaining: 16.4s
44:	learn: 0.2940328	total: 377ms	remaining: 16.4s
45:	learn: 0.2926954	total: 388ms	remaining: 16.5s
46:	learn: 0.2914741	total: 395ms	remaining: 16.4s
47:	learn: 0.2902896	total: 402ms	remaining: 16.3s
48:	learn: 0.2890199	total: 409ms	remaining: 16.3s
49:	learn: 0.2878247	total: 417ms	remaining: 16.2s
50:	learn: 0.2865303	total: 424ms	remaining: 16.2s
51:	learn: 0.2852432	total: 433ms	remaining: 16.2s
52:	learn: 0.2841812	total: 440ms	remaining: 16.2s
53:	learn: 0.2829225	total: 447ms	remaining: 16.1s
54:	learn: 0.2818122	total: 454ms	remaining: 16.1s
55:	learn: 0.2806020	total: 463ms	remaining: 16.1s
56:	learn: 0.2794183	total: 470ms	remaining: 16s
57:	learn: 0.2781683	total: 481ms	remaining: 16.1s
58:	learn: 0.2770785	total: 490ms	remaining: 16.1s
59:	learn: 0.2759602	total: 507ms	remaining: 16.4s
60:	learn: 0.2748951	total: 517ms	remaining: 16.4s
61:	learn: 0.2737439	total: 525ms	remaining: 16.4s
62:	learn: 0.2726395	total: 532ms	remaining: 16.4s
63:	learn: 0.2715255	total: 540ms	remaining: 16.3s
64:	learn: 0.2702957	total: 547ms	remaining: 16.3s
65:	learn: 0.2691468	total: 554ms	remaining: 16.2s
66:	learn: 0.2681354	total: 562ms	remaining: 16.2s
```

So, this model scoring board consolidates all R2 Score and RMSE for ease of comparison.

Out[98]:

	Models	R2 Score	RMSE
7	LGBMRegressor	0.89	0.13
4	GradientBoostRegressor	0.89	0.13
8	CatBoostRegressor	0.88	0.14
6	XGBRegressor	0.88	0.14
3	RandomForestRegressor	0.85	0.16
2	ExtraTreeRegressor	0.81	0.18
5	AdaBoostRegressor	0.80	0.18
0	SupportVectorRegressor	0.74	0.21
1	DecisionTreeRegressor	0.74	0.21

Looks like it, LGBM & GradientBoost algorithms provide the highest R2 score and lowest RMSE values.

Selecting two best models and reviewing the top 30 features.

```
                  Features  Importance
2              OverallQual        0.09
11              Fireplaces        0.06
151           ExterQual_TA        0.06
1                  LotArea        0.05
4               BsmtFinSF1        0.04
243        new_BsmtQual_TA        0.04
7                 2ndFlrSF        0.04
207   new_GarageFinish_Unf        0.04
159         Foundation_PConc        0.03
26         new_OpenPorchSF        0.03
```

```
            Features  Importance
6         BsmtUnfSF         274
1           LotArea         265
2       OverallQual         196
7          2ndFlrSF         179
4        BsmtFinSF1         176
0        MSSubClass         174
14    EnclosedPorch         150
13      OpenPorchSF         127
12       WoodDeckSF         109
21  new_LotFrontage         104
```

One interesting note, although Gradient Boost and Light GBM generated the highest score of 0.89 but their top 10 most important features are somewhat different.

Is this normal? It is not surprising. First, I am using different measures of feature importance. It's like measuring the importance of people (or simply sorting them) using their a) weight, b) height, c) wealth and d) IQ. With a and b you might get quite similar results, but these results are likely to be different from results obtained with c and d.

Second, the performance of my models is likely to be different. In extreme case, output on one model could be completely not making sense at all. Then the feature importance metrics produced with such model is not credible. In less extreme scenarios when the difference in models' performance is not so dramatic the trustworthiness of importances produced by two different models is more comparable. Still the importances might be quite different due to the first argument, i.e. different language used to capture the importance.

In this play, I would suggest we need to apply some business or general knowledge when looking at situation as such.

Whilst you are putting some thoughts to this, it would also be good to know how R2 Score will change if we take these features into a linear regression model. Now, let's take alook at how linear regression would have performed independently and how that will differ if we some of the features we identified through the two top ensemble models above.

Linear Regression Model using recursive feature elimination

For a start, we probably need to work out what is the best number of features from a raw linear model. To do that, we can use recursive feature elimination proces. Recursive Feature Elimination (RFE) method works by recursively removing attributes and building a model on those attributes that remain. It uses accuracy metric to rank the feature according to their importance. The RFE method takes the model to be used and the number of required features as input. It then gives the ranking of all the variables, 1 being most important. It also gives its support, True being relevant feature and False being irrelevant feature.

```
Optimal number of features : 56

R2 Score   : 0.632
MSE        : 0.06
RMSE       : 0.2449
```

If we take 56 independent variables into linear regression model, we could reach an R2 Score of 0.632 which is still lower than all models above.

What about we only apply the features we identified through Gradient Boost and Light GBM and see if these linear model outperform the linear model using 56 variables above.

Linear Regression Model using top 56 features from Light GBM

```
R2 Score   : 0.8703
MSE        : 0.0211
RMSE       : 0.1454
```

Linear Regression Model using top 56 features from Gradient Boost

```
R2 Score   : 0.8826
MSE        : 0.0191
RMSE       : 0.1383
```

Consolidate both scoring and put that into a dataframe.

Merge these scores with the original Model Scoring Board above.

Out[122]:

	Models	R2 Score	RMSE
7	LGBMRegressor	0.89	0.13
4	GradientBoostRegressor	0.89	0.13
11	LinearReg_w_GBM	0.88	0.14
8	CatBoostRegressor	0.88	0.14
6	XGBRegressor	0.88	0.14
10	LinearReg_w_LightGBM	0.87	0.14
3	RandomForestRegressor	0.85	0.16
2	ExtraTreeRegressor	0.81	0.18
5	AdaBoostRegressor	0.80	0.18
0	SupportVectorRegressor	0.74	0.21
1	DecisionTreeRegressor	0.74	0.21
9	LinearReg_w_RFE	0.63	0.24

Note: Next step, I will be exploring a few more models eg linear regression model with gradient descent & neural network. Stay tuned! Hope you find this kernel useful and enjoyable. Your comments and feedbacks are most welcome.

THE END