FLIP ROBO

Housing Price Prediction

ACKNOWLEDGMENT

First of all I would like to thank all my mentors in Data Trained and FlipRobo Technologies for this opportunity.

Most of the concepts used to predict the Housing Price Prediction are learned from Data Trained Institute and below documentations.

- https://scikit-learn.org/stable/
- https://seaborn.pydata.org/
- https://www.scipy.org/
- Stack-overflow
- https://imbalanced-learn.org/stable/

INTRODUCTION

Houses are one of the necessary need of each and every person around the globe and therefore housing and real estate market is one of the markets which is one of the major contributors in the world's economy. It is a very large market and there are various companies working in the domain. Data science comes as a very important tool to solve problems in the domain to help the companies increase their overall revenue, profits, improving their marketing strategies and focusing on changing trends in house sales and purchases. Predictive modelling, Market mix modelling, recommendation systems are some of the machine learning techniques used for achieving the business goals for housing companies. Our problem is related to one such housing company.

A US-based housing company named Surprise Housing has decided to enter the Australian market. The company uses data analytics to purchase houses at a price below their actual values and flip them at a higher price. For the same purpose, the company has collected a data set from the sale of houses in Australia. The data is provided in the CSV file below.

The company is looking at prospective properties to buy houses to enter the market. You are required to build a model using Machine Learning in order to predict the actual value of the prospective properties and decide whether to invest in them or not. For this company wants to know:

- Which variables are important to predict the price of variable?
- How do these variables describe the price of the house?

Analytical Problem Framing

The given dataset for training the Machine Learning model consists of 1168 rows and 81 columns. Using this dataset we will be training the Machine Learning models on 70% of the data and the models will be validated on 30% data. Finally we will predict the prices for the testing dataset consisting of 292 rows and 80 columns.

The provided dataset has null values and we will be imputing the same carefully before we proceed with any pre-processing steps.

The Dataset consists of 81 variables and their explanation is given below:

- MSSubClass: Identifies the type of dwelling involved in the sale.
- MSZoning: Identifies the general zoning classification of the sale.
- LotFrontage: Linear feet of street connected to property
- LotArea: Lot size in square feet
- Street: Type of road access to property
- Alley: Type of alley access to property
- LotShape: General shape of property
- LandContour: Flatness of the property
- Utilities: Type of utilities
- LotConfig: Lot configuration
- LandSlope: Slope of property
- Neighborhood: Physical locations within Ames city limits
- Condition1: Proximity to various conditions
- Condition2: Proximity to various conditions (if more than one is present)
- BldgType: Type of dwelling
- HouseStyle: Style of dwelling
- OverallQual: Rates the overall material and finish of the house
- OverallCond: Rates the overall condition of the house
- YearBuilt: Original construction date
- YearRemodAdd: Remodel date (same as construction date if no remodeling or additions)
- RoofStyle: Type of roof
- RoofMatl: Roof material
- Exterior1st: Exterior covering on house
- Exterior2nd: Exterior covering on house (if more than one material)
- MasVnrType: Masonry veneer type
- MasVnrArea: Masonry veneer area in square feet
- ExterQual: Evaluates the quality of the material on the exterior
- ExterCond: Evaluates the present condition of the material on the exterior
- Foundation: Type of foundation
- BsmtQual: Evaluates the height of the basement
- BsmtCond: Evaluates the general condition of the basement

- BsmtExposure: Refers to walkout or garden level walls
- BsmtFinType1: Rating of basement finished area
- BsmtFinSF1: Type 1 finished square feet
- BsmtFinType2: Rating of basement finished area (if multiple types)
- BsmtFinSF2: Type 2 finished square feet
- BsmtUnfSF: Unfinished square feet of basement area
- TotalBsmtSF: Total square feet of basement area
- Heating: Type of heating
- HeatingQC: Heating quality and condition
- CentralAir: Central air conditioning
- Electrical: Electrical system
- 1stFlrSF: First Floor square feet
- 2ndFlrSF: Second floor square feet
- LowQualFinSF: Low quality finished square feet (all floors)
- GrLivArea: Above grade (ground) living area square feet
- BsmtFullBath: Basement full bathrooms
- BsmtHalfBath: Basement half bathrooms
- FullBath: Full bathrooms above grade
- HalfBath: Half baths above grade
- Bedroom: Bedrooms above grade (does NOT include basement bedrooms)
- Kitchen: Kitchens above grade
- KitchenQual: Kitchen quality
- TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)
- Functional: Home functionality (Assume typical unless deductions are warranted)
- Fireplaces: Number of fireplaces
- FireplaceQu: Fireplace quality
- GarageType: Garage location
- GarageYrBlt: Year garage was built
- GarageFinish: Interior finish of the garage
- GarageCars: Size of garage in car capacity
- GarageArea: Size of garage in square feet
- GarageQual: Garage quality
- GarageCond: Garage condition
- PavedDrive: Paved driveway
- WoodDeckSF: Wood deck area in square feet
- OpenPorchSF: Open porch area in square feet
- EnclosedPorch: Enclosed porch area in square feet
- 3SsnPorch: Three season porch area in square feet
- ScreenPorch: Screen porch area in square feet
- PoolArea: Pool area in square feet
- PoolQC: Pool quality
- Fence: Fence quality

- MiscFeature: Miscellaneous feature not covered in other categories
- MiscVal: $Value of miscellaneous feature
- MoSold: Month Sold (MM)
- YrSold: Year Sold (YYYY)
- SaleType: Type of sale
- SaleCondition: Condition of sale

Importing the necessary libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder, power_transform
from scipy.stats import zscore
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
import warnings
warnings.filterwarnings('ignore')
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer, KNNImputer
```

Looking at the glimpse of the dataset

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition
0	127	120	RL	NaN	4928	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	NPkVill	Norm	Norr
1	889	20	RL	95.0	15865	Pave	NaN	IR1	Lvl	AllPub	Inside	Mod	NAmes	Norm	Norr
2	793	60	RL	92.0	9920	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl	NoRidge	Norm	Norr
3	110	20	RL	105.0	11751	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	NWAmes	Norm	Norr
4	422	20	RL	NaN	16635	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	NWAmes	Norm	Norr

	BldgType	HouseStyle	OverallQual	OverallCond	YearBuilt	YearRemodAdd	RoofStyle	RoofMatl	Exterior1st	Exterior2nd	MasVnrType	MasVnrArea	ExterQual
0	TwnhsE	1Story	6	5	1976	1976	Gable	CompShg	Plywood	Plywood	None	0.0	TA
1	1Fam	1Story	8	6	1970	1970	Flat	Tar&Grv	Wd Sdng	Wd Sdng	None	0.0	Gd
2	1Fam	2Story	7	5	1996	1997	Gable	CompShg	MetalSd	MetalSd	None	0.0	Gd
3	1Fam	1Story	6	6	1977	1977	Hip	CompShg	Plywood	Plywood	BrkFace	480.0	TA
4	1Fam	1Story	6	7	1977	2000	Gable	CompShg	CemntBd	CmentBd	Stone	126.0	Gd

	BsmtQual	BsmtCond	BsmtExposure	BsmtFinType1	BsmtFinSF1	BsmtFinType2	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	Heating	HeatingQC	CentralAir	Electri
0	Gd	TA	No	ALQ	120	Unf	0	958	1078	GasA	TA	Y	SE
1	TA	Gd	Gd	ALQ	351	Rec	823	1043	2217	GasA	Ex	Y	SE
2	Gd	TA	Av	GLQ	862	Unf	0	255	1117	GasA	Ex	Y	SE
3	Gd	TA	No	BLQ	705	Unf	0	1139	1844	GasA	Ex	Y	SE
4	Gd	TA	No	ALQ	1246	Unf	0	356	1602	GasA	Gd	Y	SE

	LowQualFinSF	GrLivArea	BsmtFullBath	BsmtHalfBath	FullBath	HalfBath	BedroomAbvGr	KitchenAbvGr	KitchenQual	TotRmsAbvGrd	Functional	Fireplaces	F
0	0	958	0	0	2	0	2	1	TA	5	Typ	1	
1	0	2217	1	0	2	0	4	1	Gd	8	Typ	1	
2	0	2013	1	0	2	1	3	1	TA	8	Typ	1	
3	0	1844	0	0	2	0	3	1	TA	7	Typ	1	
4	0	1602	0	1	2	0	3	1	Gd	8	Typ	1	

	GarageFinish	GarageCars	GarageArea	GarageQual	GarageCond	PavedDrive	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch	PoolAre
0	RFn	2	440	TA	TA	Y	0	205	0	0	0	
1	Unf	2	621	TA	TA	Y	81	207	0	0	224	
2	Unf	2	455	TA	TA	Y	180	130	0	0	0	
3	RFn	2	546	TA	TA	Y	0	122	0	0	0	
4	Fin	2	529	TA	TA	Y	240	0	0	0	0	

	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	0	2	2007	WD	Normal	128000
1	0	10	2007	WD	Normal	268000
2	0	6	2007	WD	Normal	269790
3	0	1	2010	COD	Normal	190000
4	0	6	2009	WD	Normal	215000

Pre-Processing:

Before we can proceed let's check for null values in the dataset, so that it could be handled.

```
pd.set_option('display.max_rows',None)
dataset.isnull().sum()
```

```
Id                 0
MSSubClass         0
MSZoning           0
LotFrontage      214
LotArea            0
Street             0
Alley           1091
LotShape           0
LandContour        0
Utilities          0
LotConfig          0
LandSlope          0
Neighborhood       0
Condition1         0
Condition2         0
BldgType           0
HouseStyle         0
OverallQual        0
OverallCond        0
YearBuilt          0
YearRemodAdd       0
RoofStyle          0
RoofMatl           0
Exterior1st        0
Exterior2nd        0
MasVnrType         7
MasVnrArea         7
ExterQual          0
ExterCond          0
Foundation         0
BsmtQual          30
BsmtCond          30
BsmtExposure      31
BsmtFinType1      30
BsmtFinSF1         0
BsmtFinType2      31
BsmtFinSF2         0
BsmtUnfSF          0
TotalBsmtSF        0
Heating            0
HeatingQC          0
CentralAir         0
Electrical         0
1stFlrSF           0
2ndFlrSF           0
LowQualFinSF       0
GrLivArea          0
BsmtFullBath       0
BsmtHalfBath       0
FullBath           0
HalfBath           0
BedroomAbvGr       0
KitchenAbvGr       0
KitchenQual        0
TotRmsAbvGrd       0
Functional         0
Fireplaces         0
FireplaceQu      551
GarageType        64
GarageYrBlt       64
GarageFinish      64
GarageCars         0
GarageArea         0
GarageQual        64
GarageCond        64
PavedDrive         0
WoodDeckSF         0
OpenPorchSF        0
EnclosedPorch      0
3SsnPorch          0
ScreenPorch        0
PoolArea           0
PoolQC          1161
Fence            931
MiscFeature     1124
MiscVal            0
MoSold             0
YrSold             0
SaleType           0
SaleCondition      0
```

We can see that the variables like LotFrontage, Alley, MasVnrType, MasVnrArea, BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2, FireplaceQu, GarageType, GarageYrBlt, GarageFinish, GarageQual, GarageCond, PoolQC, Fence and MiscFeature have missing data. Now that we have identified the variables containing 'missing data', I'm proceeding with handling the missing data.

Firstly, I'm removing the entire variables that contains 'missing data' more than 80%.

We can clearly see that for the variables 'Alley', 'MiscFeature' and 'PoolQC', there are more than 80% data is missing, hence removing the same. Further I'm also removing 'Id' because it is unique for each row and will not help in the Prediction of sale price.

```
data = dataset.drop(columns = ['Alley','MiscFeature','PoolQC','Id'])
```

Since we have removed the variables, we can proceed with the missing data imputation for other variables.

Firstly I'm imputing the LotFrontage variable using the KNN Imputer.

```
knim = KNNImputer()
dataset[['LotFrontage','LotArea']] = knim.fit_transform(dataset[['LotFrontage','LotArea']])
```

Proceeding with MasVnrType (Masonry veneer type). Here we are verifying its value counts to check the major category and found that most of the houses doesn't have Masonry veneer (the value is None). Hence replacing the null value with None.

```
dataset['MasVnrType'] = dataset['MasVnrType'].fillna(dataset['MasVnrType'].mode()[0])
```

For the variable MasVnrArea (Masonry veneer Area), I'm imputing the missing variable with 0, because the variables MasVnrType and MasVnrArea has same number of missing value, also the data is missing in the same row. So, I imputed MasVnrType with None (Which means no Masonry veneer present), hence the MasVnrArea will be 0.

```
dataset['MasVnrArea'] = dataset['MasVnrArea'].fillna(0)
```

I'm imputing the basement related variables, all the variables (BsmtQual, BsmtExposure, BsmtCond, BsmtFinType1 and BsmtFinType2) are missing in the same row and the corresponding basement area is 0. Therefore imputing these features with NA (assuming that these properties doesn't have any basement)

```
dataset['BsmtQual'] = dataset['BsmtQual'].fillna('NA')
dataset['BsmtCond'] = dataset['BsmtCond'].fillna('NA')
dataset['BsmtExposure'] = dataset['BsmtExposure'].fillna('NA')
dataset['BsmtFinType1'] = dataset['BsmtFinType1'].fillna('NA')
dataset['BsmtFinType2'] = dataset['BsmtFinType2'].fillna('NA')
```

Now the remaining variables are related to the fireplace and garage. Upon review, I found that the there is a missing data for the FireplaceQu because, there is no fireplace for those properties. Hence, imputing the same with NA

Further, It's the same with garage related attributes (GarageType, GarageYrBlt, GarageFinish, GarageQual and GarageCond), they are missing because there was no garage available for these properties and we can confirm the same from the corresponding garage area (which is '0'). Similarly for the Fence

```
dataset['FireplaceQu'] = dataset['FireplaceQu'].fillna('NA')
```

```
dataset['GarageType'] = dataset['GarageType'].fillna('NA')
dataset['GarageYrBlt'] = dataset['GarageYrBlt'].fillna(0)
dataset['GarageFinish'] = dataset['GarageFinish'].fillna('NA')
dataset['GarageQual'] = dataset['GarageQual'].fillna('NA')
dataset['GarageCond'] = dataset['GarageCond'].fillna('NA')
```

```
dataset['Fence'] = dataset['Fence'].fillna('NA')
```

Now that we have handled the null values in the dataset, I'm encoding the date before taking in to any further analysis. I'm using ordinal encoder to perform the same

```
encoder = OrdinalEncoder()
for i in data.columns:
    if data[i].dtypes == 'object':
        data[i] = encoder.fit_transform(data[i].values.reshape(-1,1))
```

I have used the simple 'for' loop to encode the data variables which encodes the data with datatype object.

Now we can proceed with finding the correlation between the dependent variable and independent variables.

In order to achieve that I can use .corr method in python.

```
data_corr = data.corr()
data_corr['SalePrice'].sort_values(ascending = False)
```

Below are the correlation coefficients:

SalePrice	1.000000	Condition1	0.105820
OverallQual	0.789185	PoolArea	0.103280
GrLivArea	0.707300	ScreenPorch	0.100284
GarageCars	0.628329	Exterior2nd	0.097541
GarageArea	0.619000	BsmtCond	0.084121
TotalBsmtSF	0.595042	MoSold	0.072764
1stFlrSF	0.587642	BsmtFinType2	0.069657
FullBath	0.554988	3SsnPorch	0.060119
TotRmsAbvGrd	0.528363	Street	0.044753
YearBuilt	0.514408	Condition2	0.033956
YearRemodAdd	0.507831	LandContour	0.032836
MasVnrArea	0.460535	LandSlope	0.015485
Fireplaces	0.459611	MasVnrType	0.007732
Foundation	0.374169	BsmtFinSF2	-0.010151
BsmtFinSF1	0.362874	BsmtHalfBath	-0.011109
OpenPorchSF	0.339500	MiscVal	-0.013071
2ndFlrSF	0.330386	LowQualFinSF	-0.032381
LotFrontage	0.319416	YrSold	-0.045508
WoodDeckSF	0.315444	SaleType	-0.050851
HalfBath	0.295592	LotConfig	-0.060452
GarageYrBlt	0.265622	MSSubClass	-0.060775
LotArea	0.249499	OverallCond	-0.065642
GarageCond	0.249340	BldgType	-0.066028
CentralAir	0.246754	FireplaceQu	-0.076951
Electrical	0.234621	BsmtFinType1	-0.099860
PavedDrive	0.231707	Heating	-0.100021
SaleCondition	0.217687	EnclosedPorch	-0.115004
BsmtUnfSF	0.215724	KitchenAbvGr	-0.132108
BsmtFullBath	0.212924	MSZoning	-0.133221
HouseStyle	0.205502	LotShape	-0.248171
Neighborhood	0.198942	BsmtExposure	-0.267635
RoofStyle	0.192654	HeatingQC	-0.406604
GarageQual	0.192392	GarageType	-0.415370
RoofMatl	0.159865	GarageFinish	-0.424922
BedroomAbvGr	0.158281	KitchenQual	-0.592468
Fence	0.143922	BsmtQual	-0.601307
Functional	0.118673	ExterQual	-0.624820
ExterCond	0.115167	Utilities	NaN
Exterior1st	0.108451		

These are the highly correlated variables with respect to the sale price (Target). I'm considering the variables with the correlation coefficient of greater than or equal to 0.25.

Highly Correlated variables with the SalePrice

OverallQual	0.789185
GrLivArea	0.707300
GarageCars	0.628329
GarageArea	0.619000
TotalBsmtSF	0.595042
1stFlrSF	0.587642
FullBath	0.554988
TotRmsAbvGrd	0.528363
YearBuilt	0.514408
YearRemodAdd	0.507831
MasVnrArea	0.460535
Fireplaces	0.459611
Foundation	0.374169
BsmtFinSF1	0.362874
OpenPorchSF	0.339500
2ndFlrSF	0.330386
LotFrontage	0.319416
WoodDeckSF	0.315444
HalfBath	0.295592
GarageYrBlt	0.265622
LotArea	0.249499
GarageCond	0.249340
CentralAir	0.246754
LotShape	-0.248171
BsmtExposure	-0.267635
HeatingQC	-0.406604
GarageType	-0.415370
GarageFinish	-0.424922
KitchenQual	-0.592468
BsmtQual	-0.601307
ExterQual	-0.624820

Visualizing the type of relation between the highly correlated independent variable and the target variable.

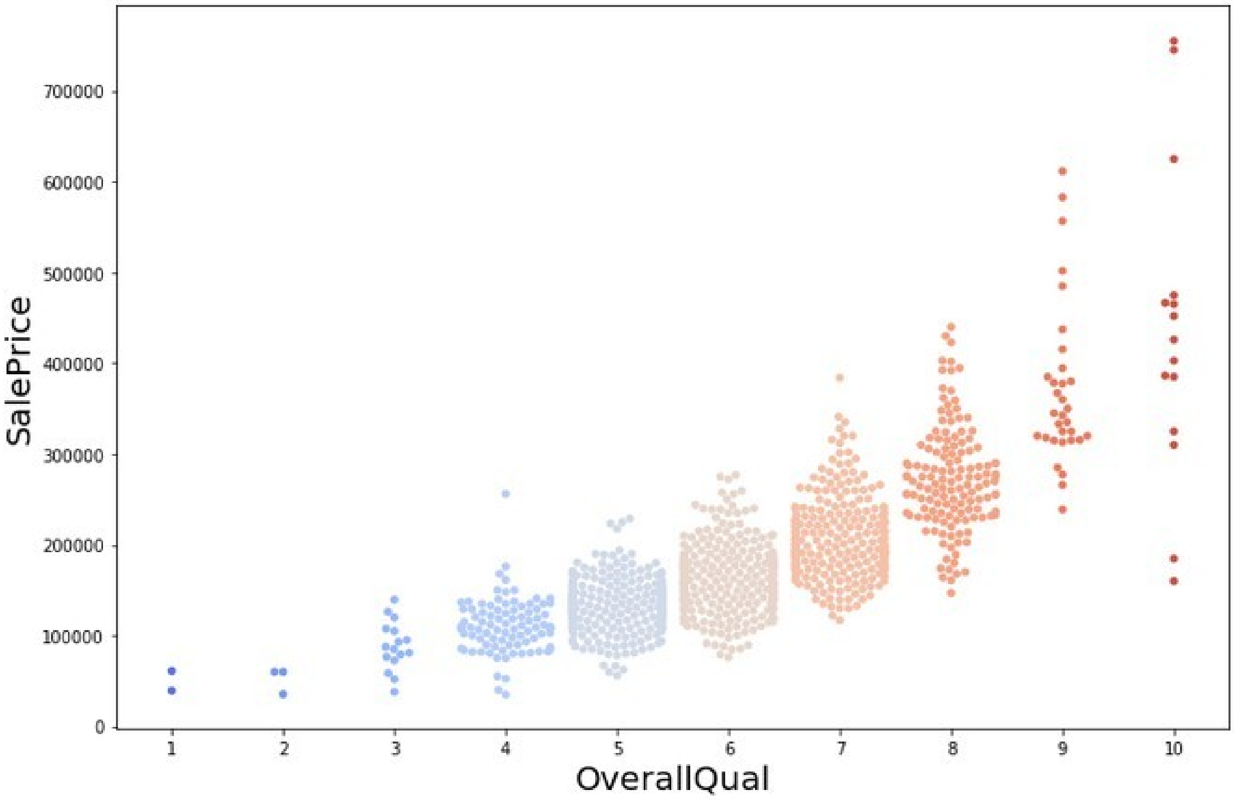
I'm using swarm plot to view the correlation of categorical type variable, using scatterplot to visualize the continuous type variable and using line plot to visualize the ordinal type variables like date.

- Overall Quality and Sale Price.

  I can say that the higher the overall quality of the house, higher the sale price and they and directly proportional to each other.

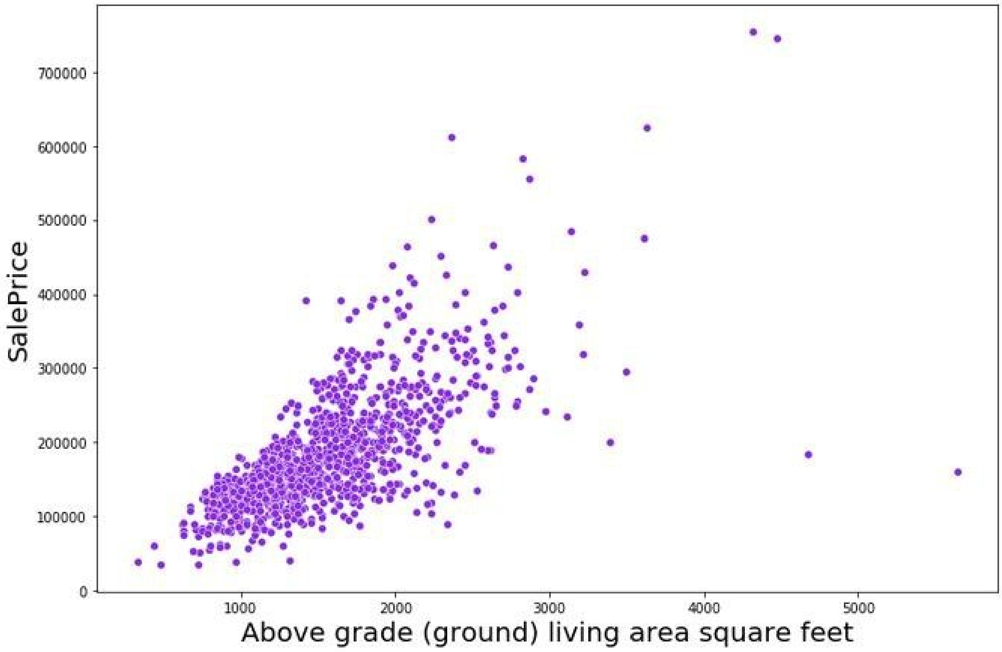
```
plt.figure(figsize = (12,8))
sns.swarmplot(x = 'OverallQual',y = 'SalePrice', data = dataset, palette = 'coolwarm')
plt.xlabel('OverallQual', fontsize = 20)
plt.ylabel('SalePrice', fontsize = 20)
```

```
Text(0, 0.5, 'SalePrice')
```



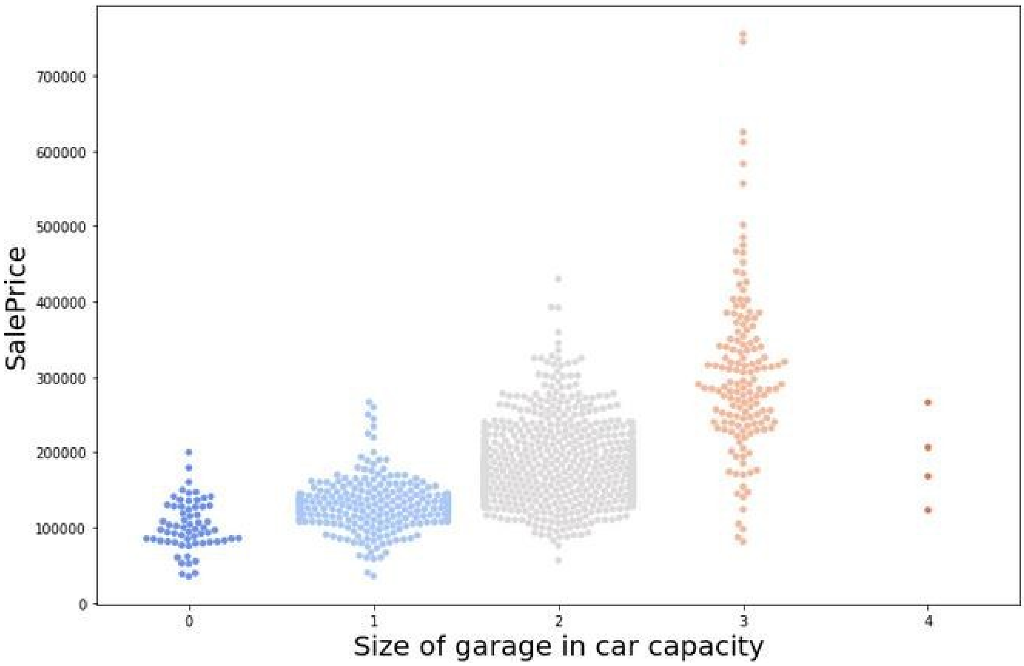
- GrLivArea and SalePrice.

I can say that the larger the space of living area, the higher the price of the property.



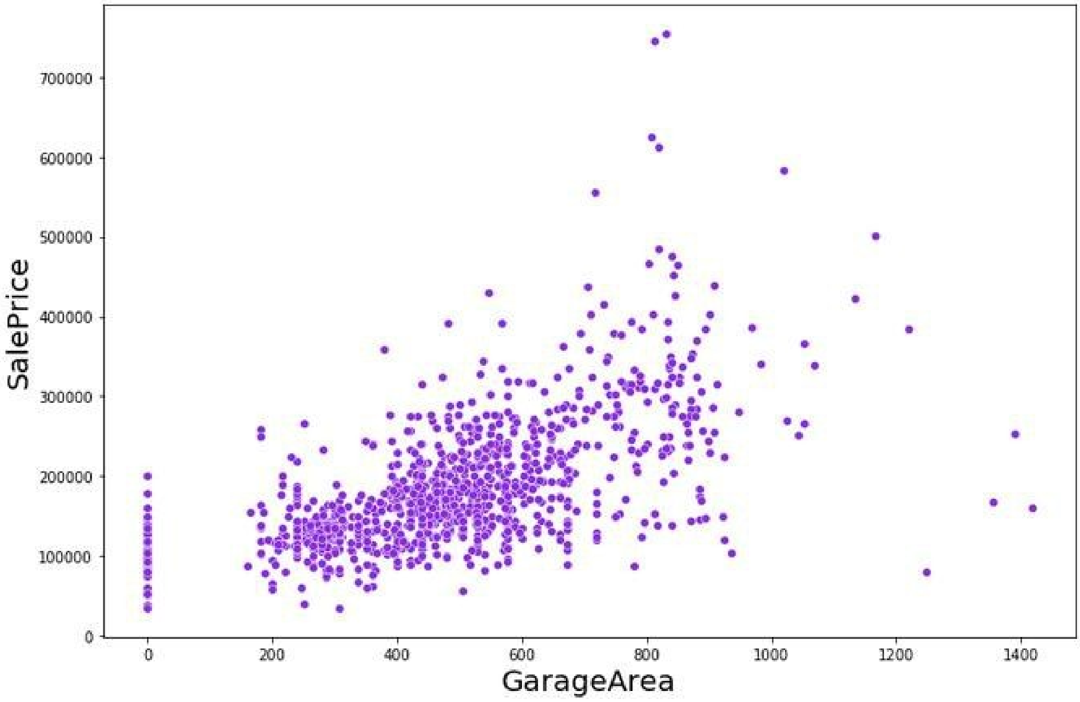
- GarageCars and SalePrice

Bigger the garage size in car parking capacity, higher the price of the property.



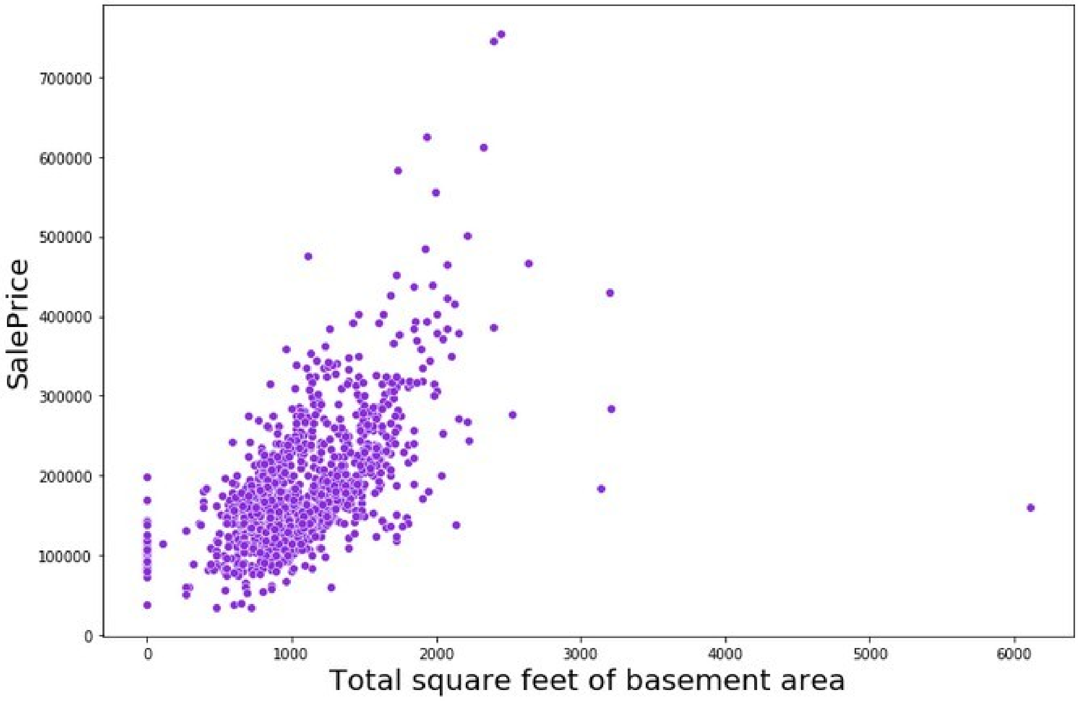
- GarageArea and SalePrice

There is an average positive relationship with Garage Area and Sale Price, the Sale price of a property increases with the increase in garage area



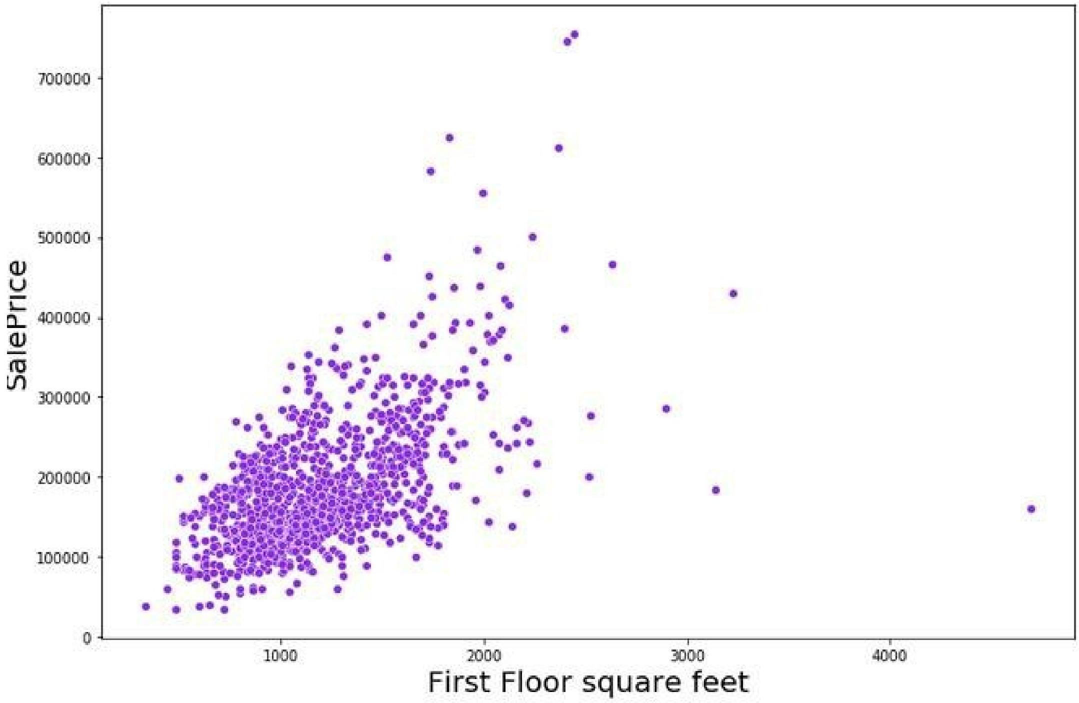
- TotalBsmtSF and SalePrice

There is an average positive relationship with Basement Area and Sale Price, the Sale price of a property increases with the bigger basement area.



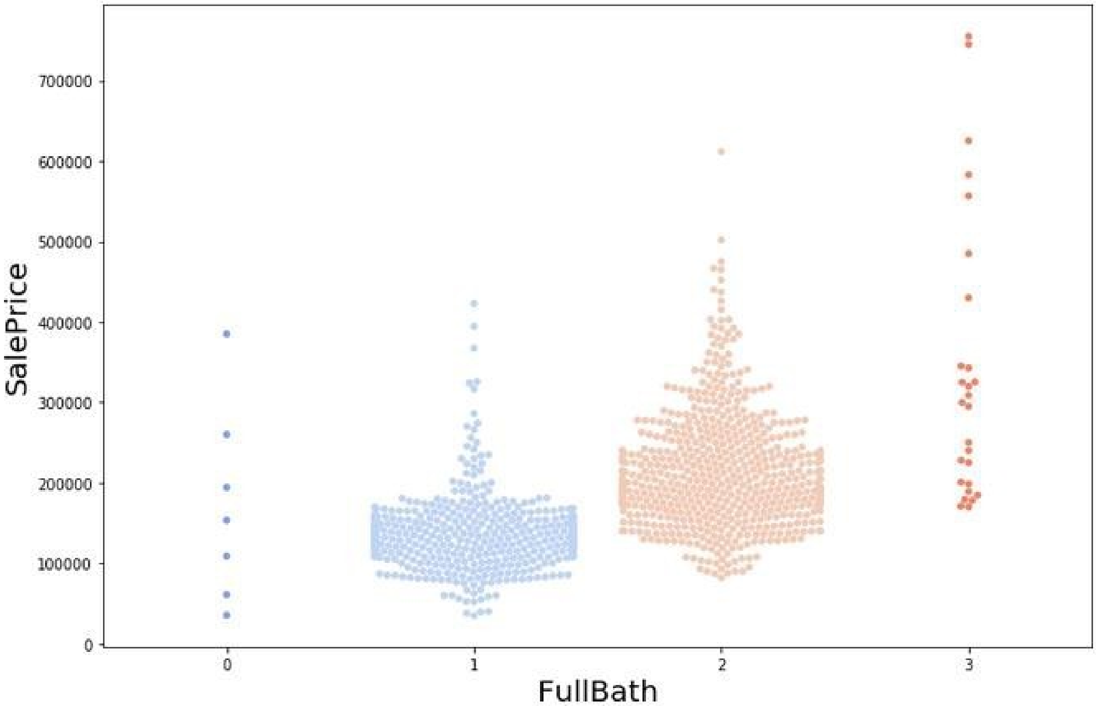
- 1stFlrSF and SalePrice

The square feet of the first floor is positively correlated with the sale price. I can say that the bigger the first floor the higher the sale price



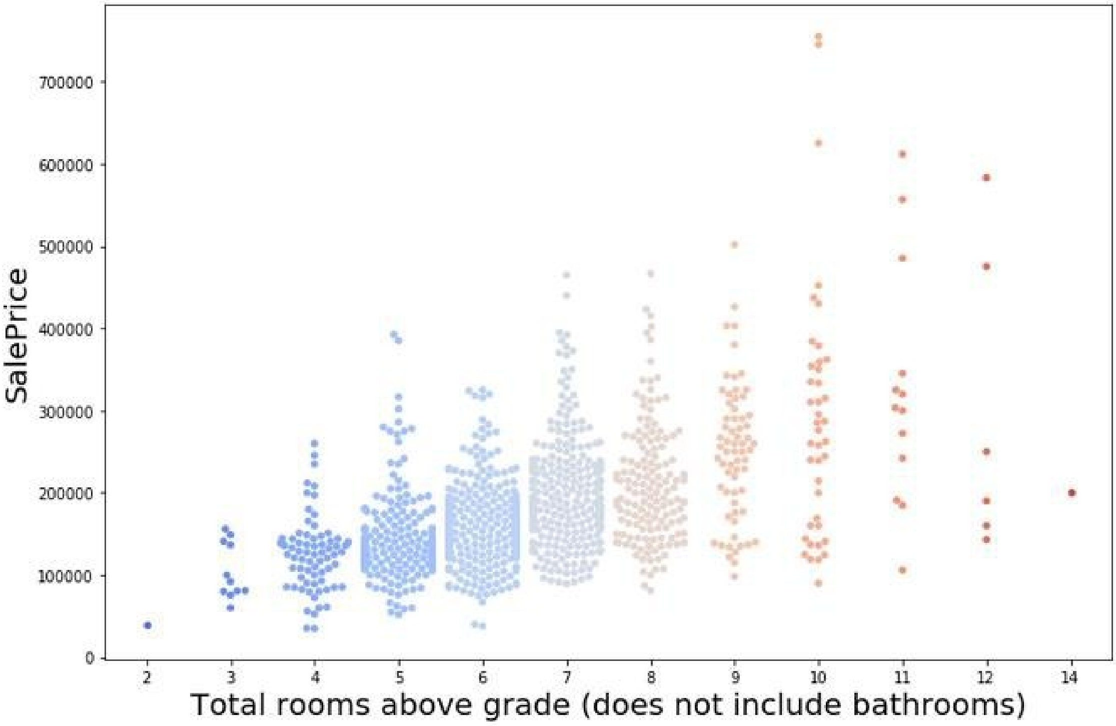
- FullBath and Saleprice

The properties with more number of full size bathrooms have higher sale price



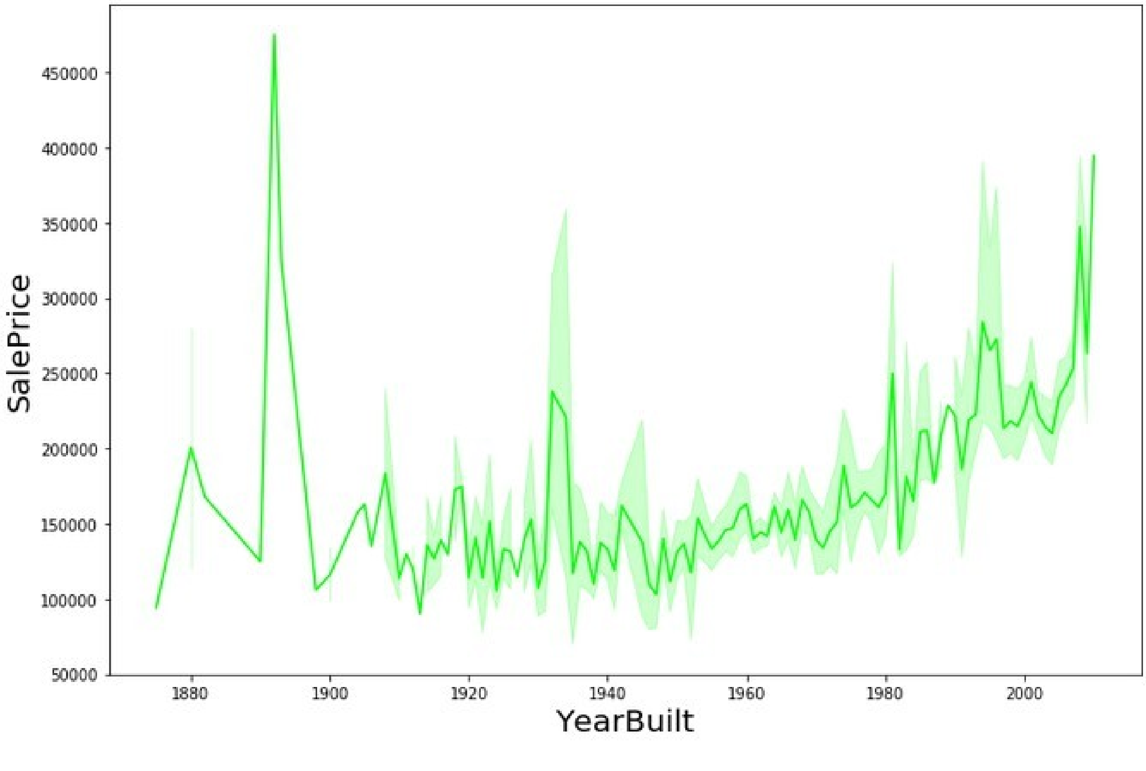
- TotRmsAbvGrd and SalePrice

More number of high grade rooms in a house, higher its price. We can view the same from the below figure



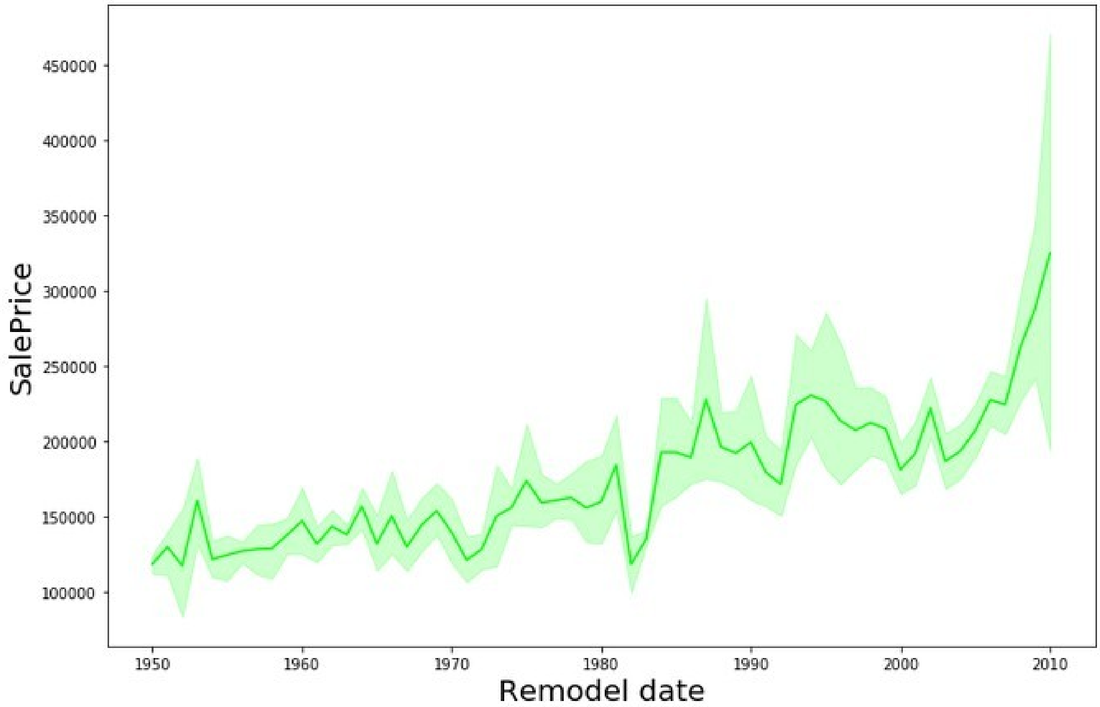
- YearBuilt and SalePrice

From the overall analysis, newer the house, the higher its value. Further, we can see that houses built just before 1900s were sold for unusually higher price.



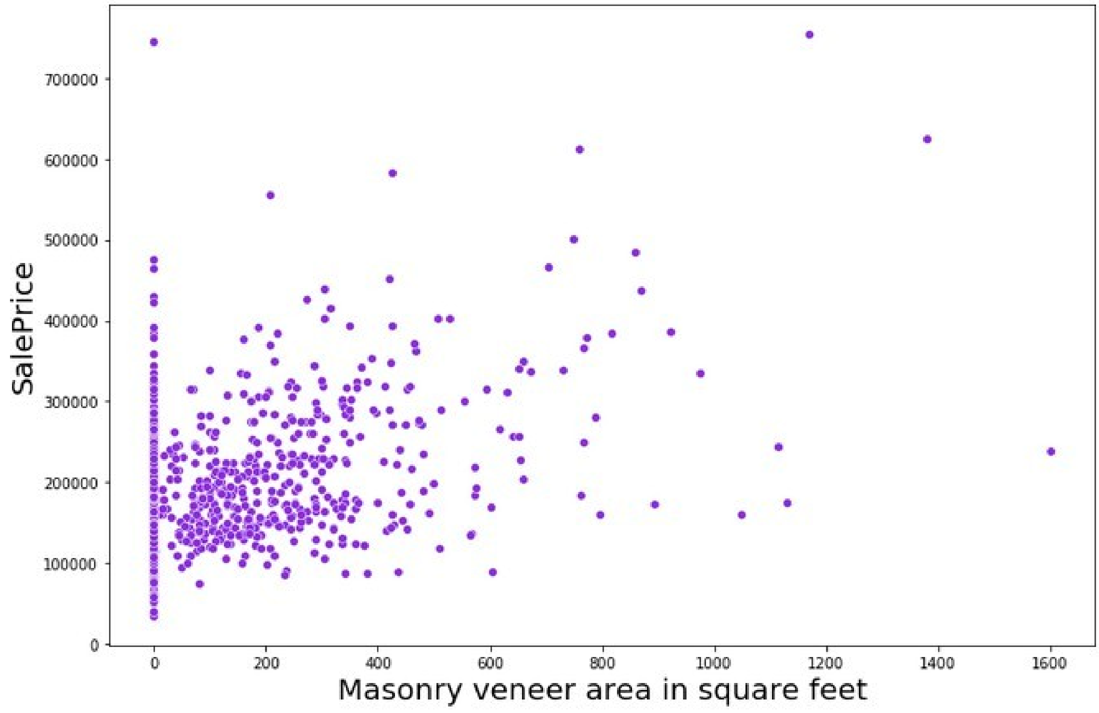
- YearRemodAdd and SalePrice

Most of the houses weren't remodelled, however we are looking at the sale prices of the remodelled houses. The Sale price was higher for houses when the remodelling was done recently



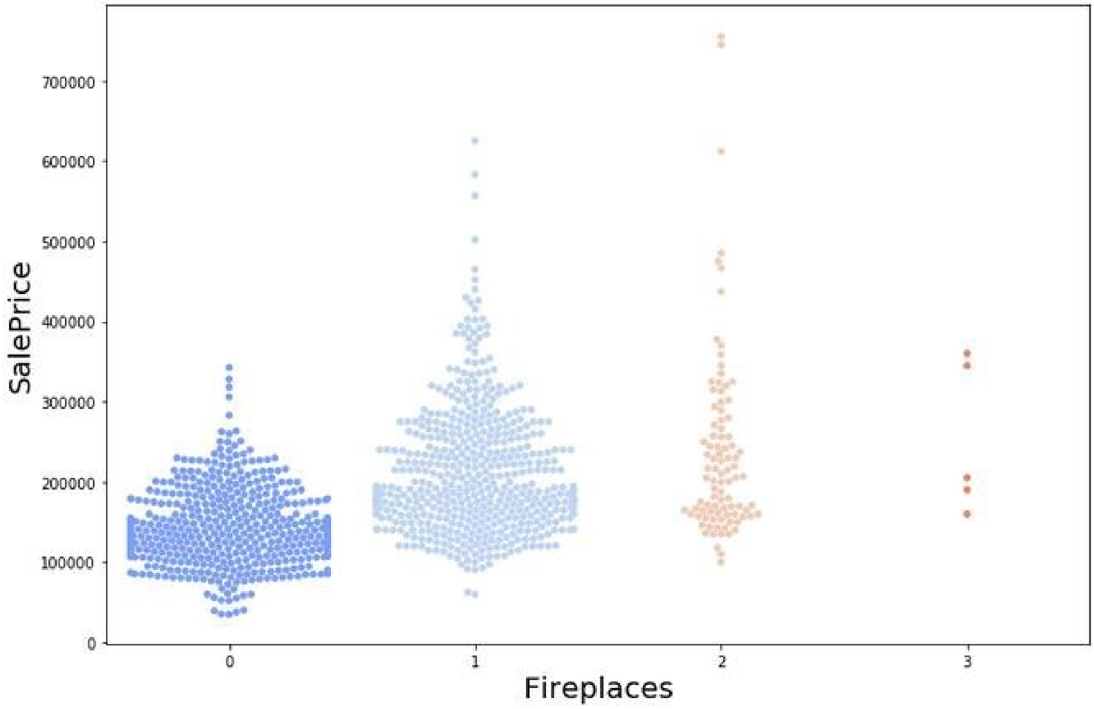
- MasVnrArea and SalePrice

Bigger the Masonry veener area, higher were the prices of the houses. We can't be sure of the relationship because the correlation is just fair.



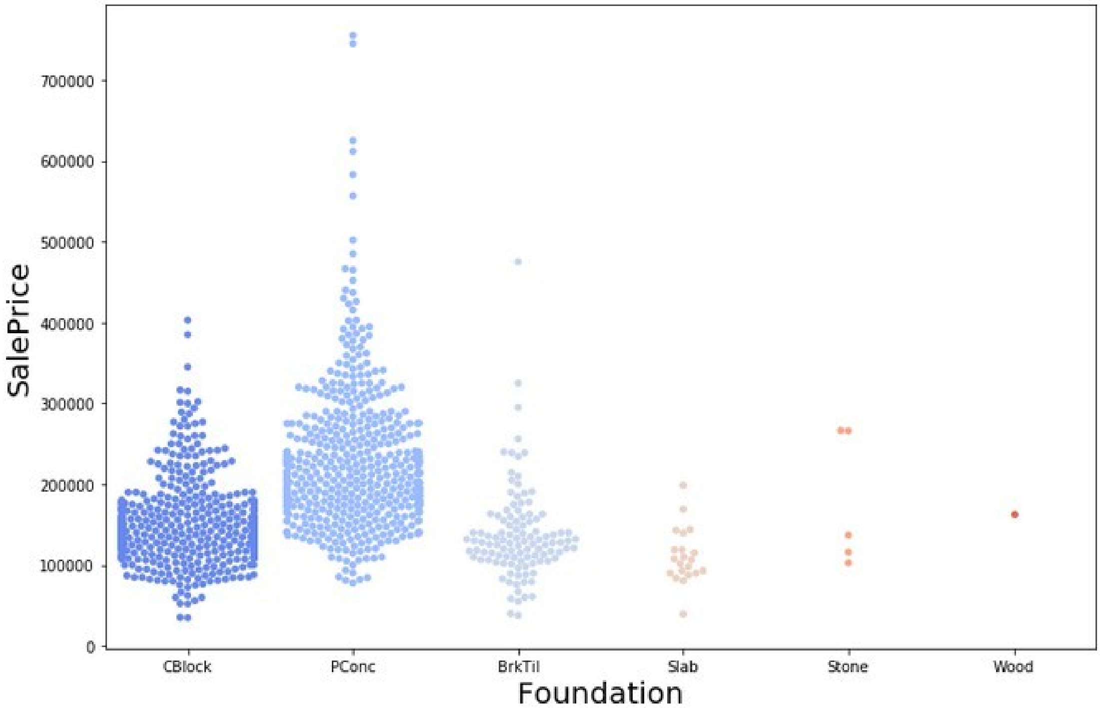
- Fireplaces and SalePrice

Presence of sale price was preferred by the buyers and they paid a little higher price for the properties containing a fireplace.

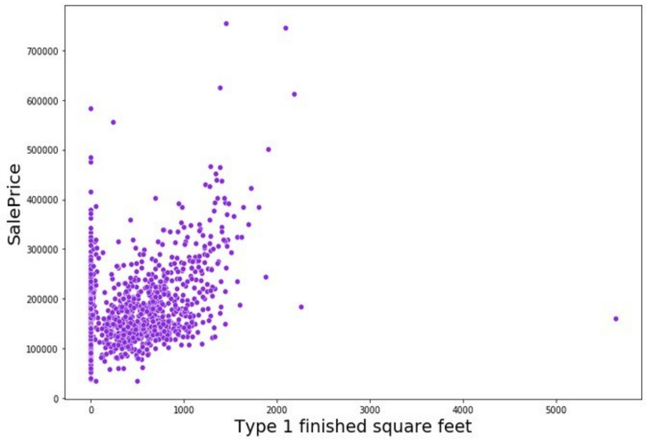


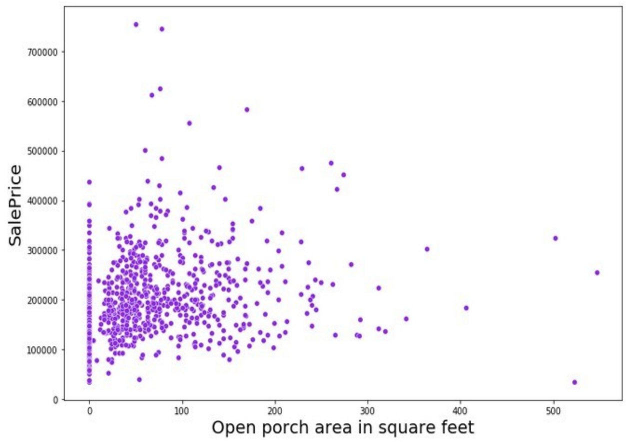
- Foundation and SalePrice

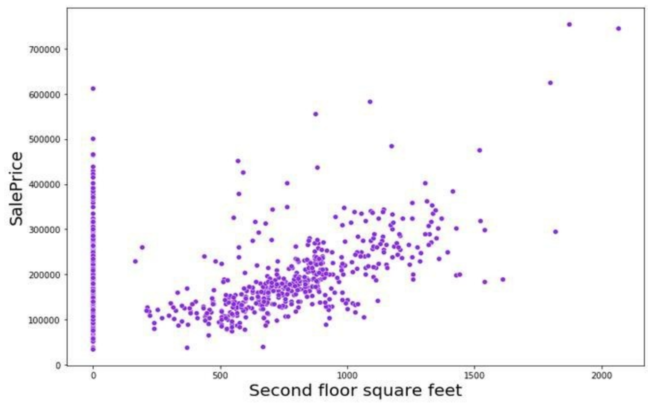
Type of foundation for a house also decided the property (house) price. I can say that the houses built with poured concrete foundation were of higher value and the sale price was higher for the same.

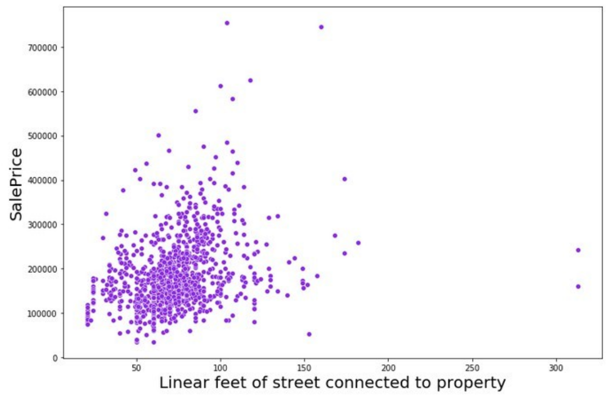


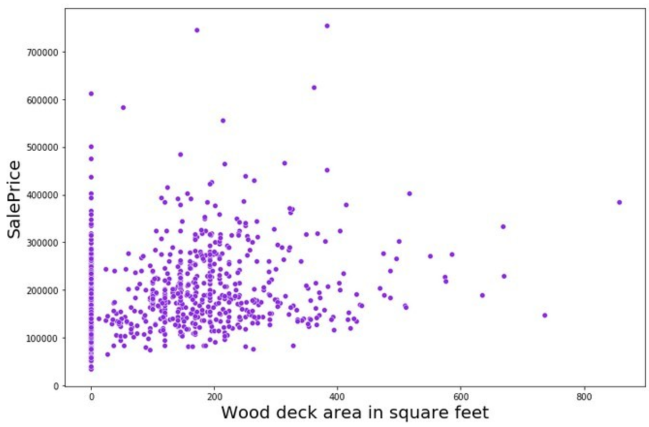
- Below are the few positively correlated variable with the target variable and they affected the target variable positively

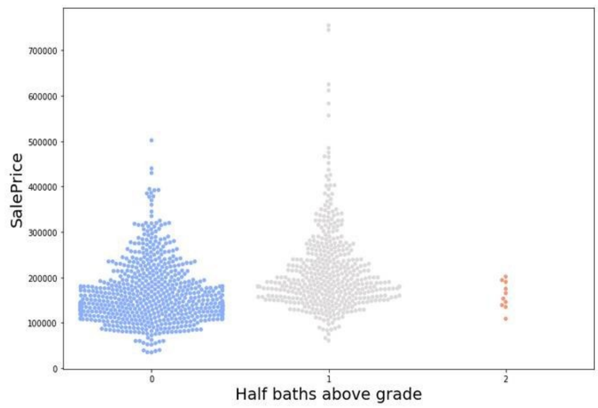


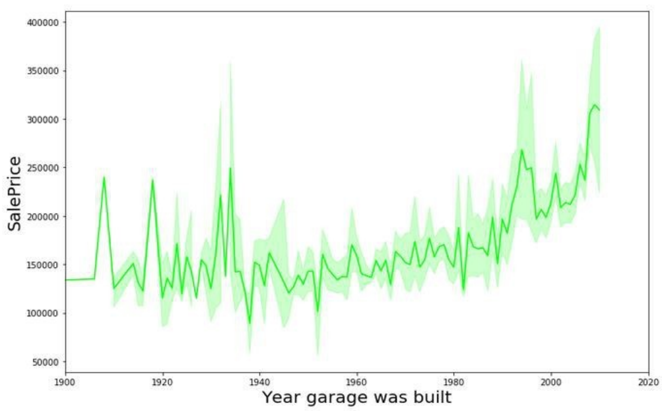


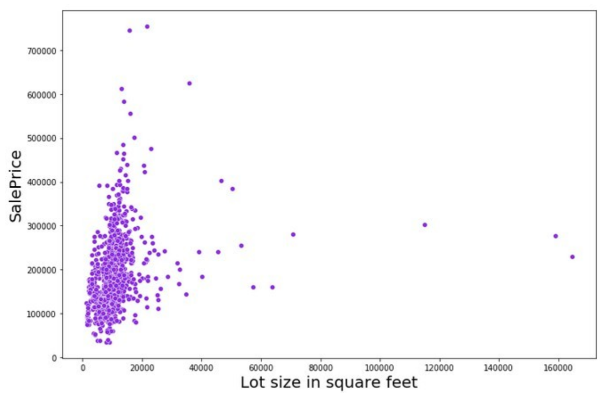


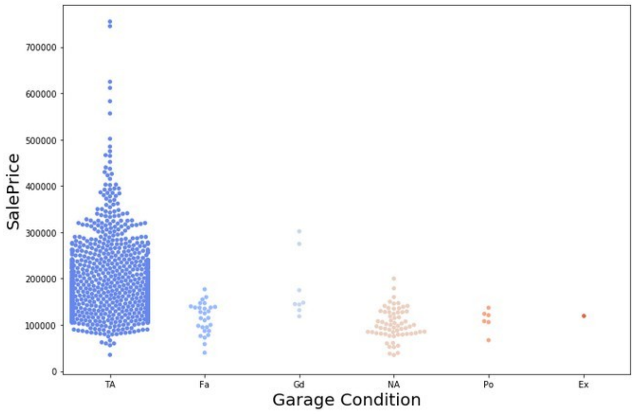


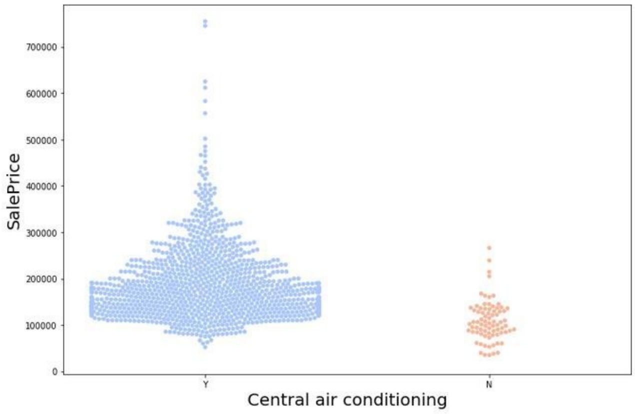








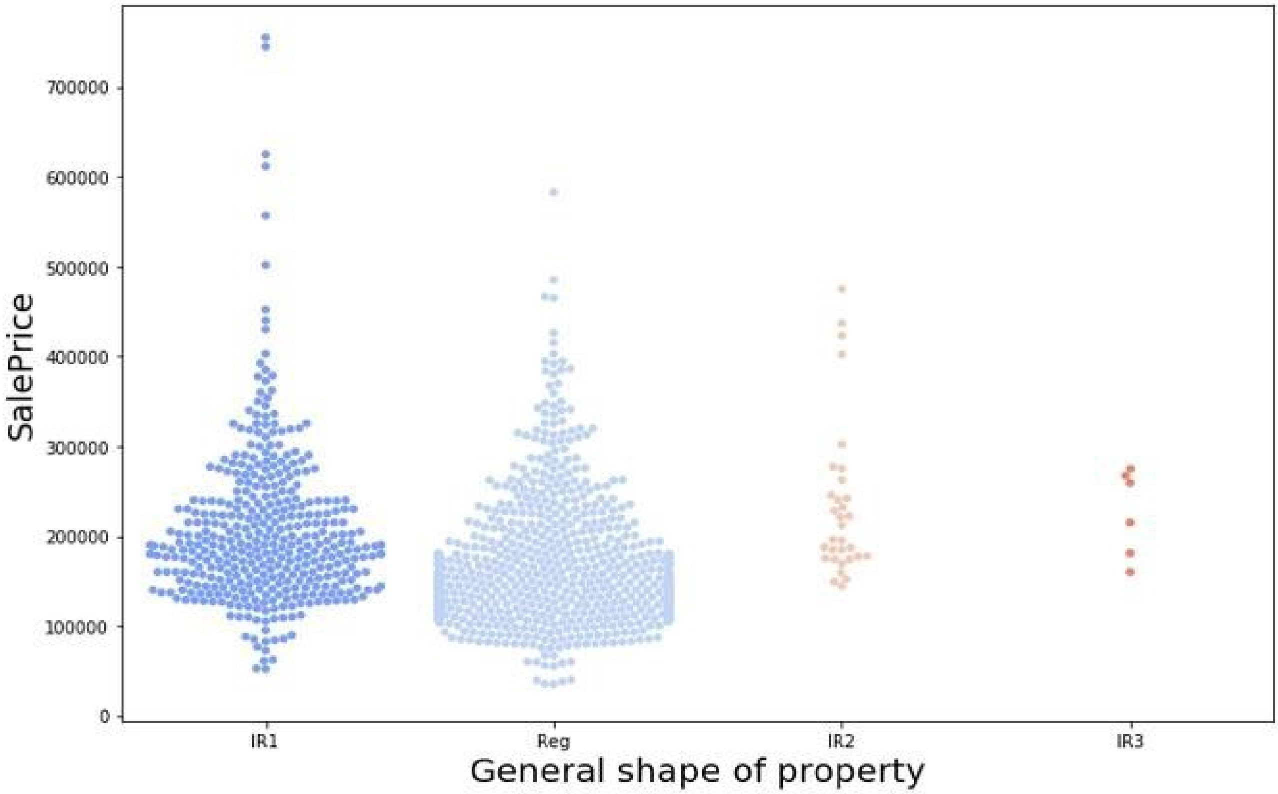




- Let’s look at few of the highly negatively correlated variables

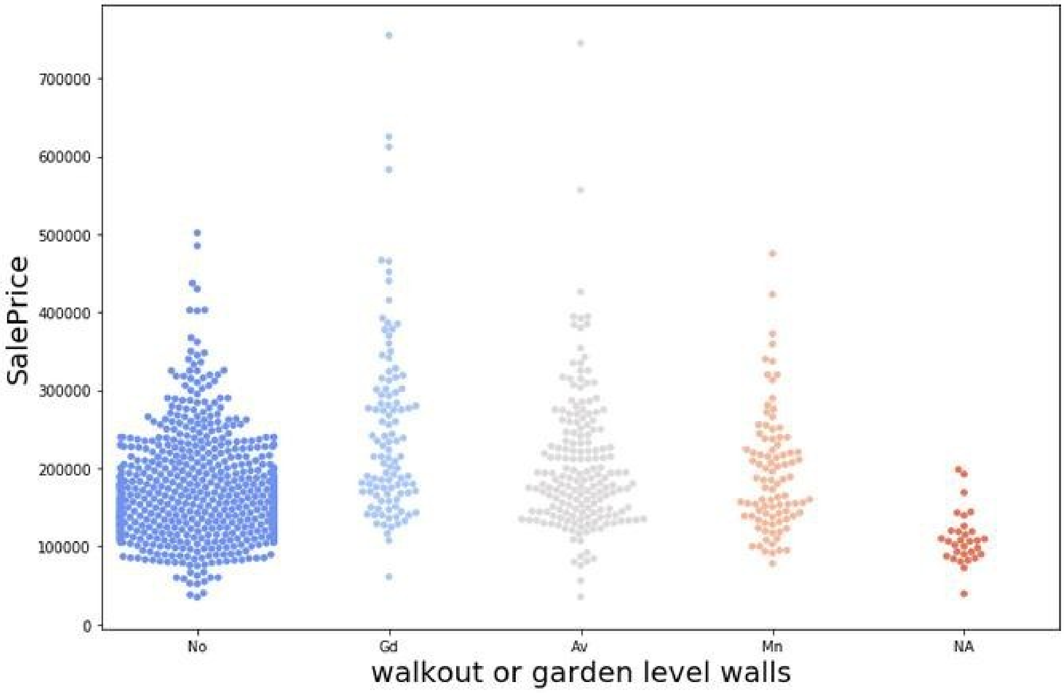
LotShape and Sale Price

I can see that although there is a negative correlation coefficient. I can see that the variable is categorical. The houses with slightly irregular shape were sold for higher prices when compared to regular shaped houses.



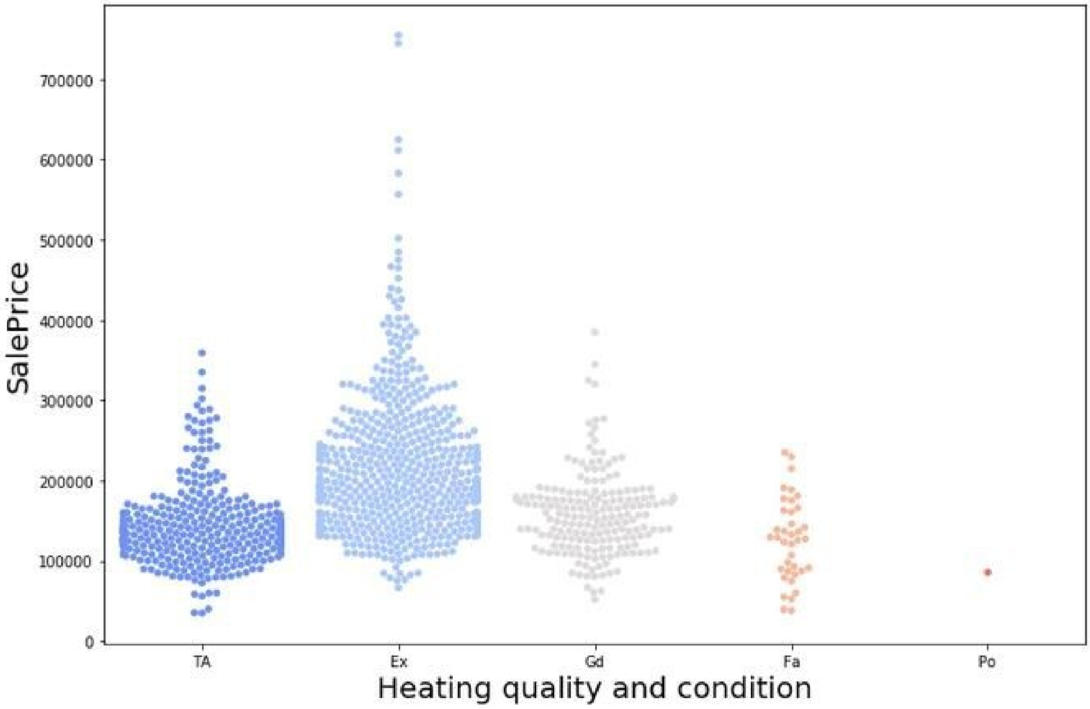
- BsmtExposure and SalePrice

The houses with Good walkout walls were sold with slightly higher prices when compared to others



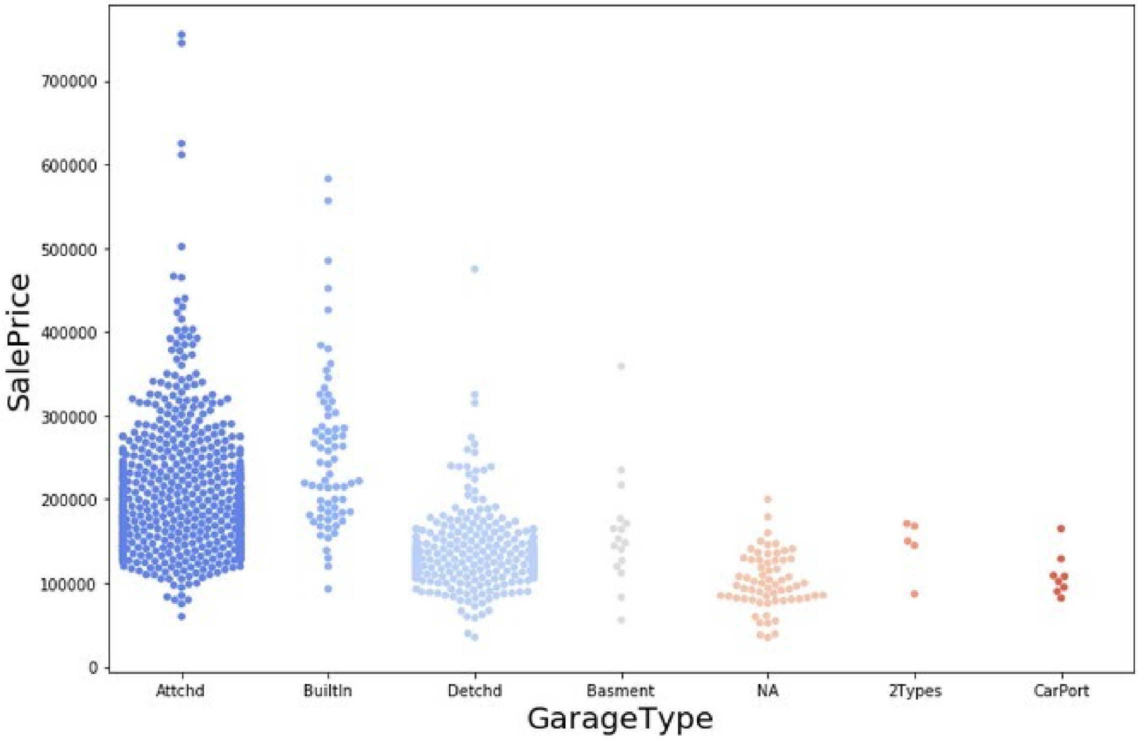
- HeatingQC and Saleprice

Houses with excellent heating quality were sold at higher prices when compared to others.



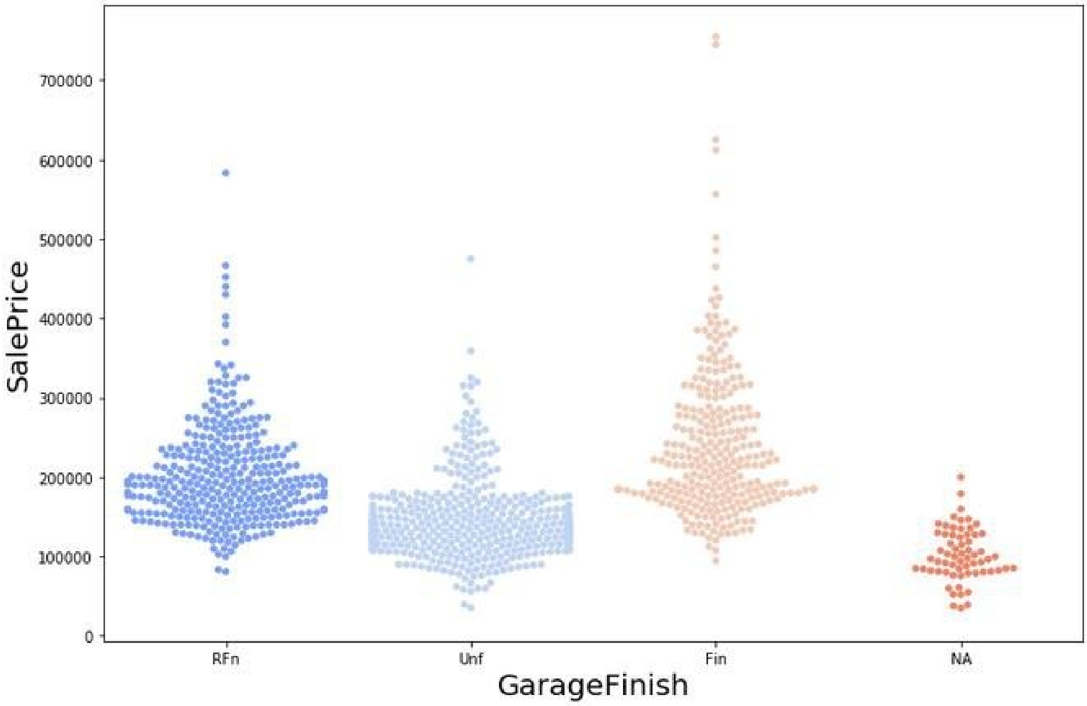
- GarageType and SalePrice

Houses with attached and the built-in garages were sold at higher prices when compared to others.



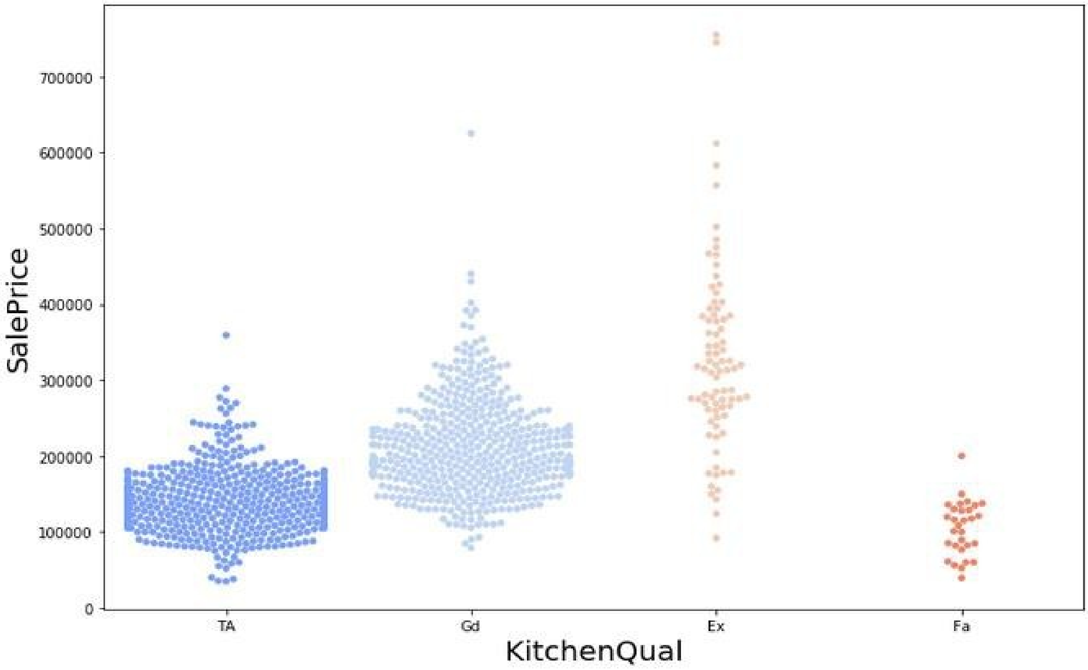
- GarageFinish and SalePrice

Houses with finished and roughly finished garages were sold at higher prices when compared to others



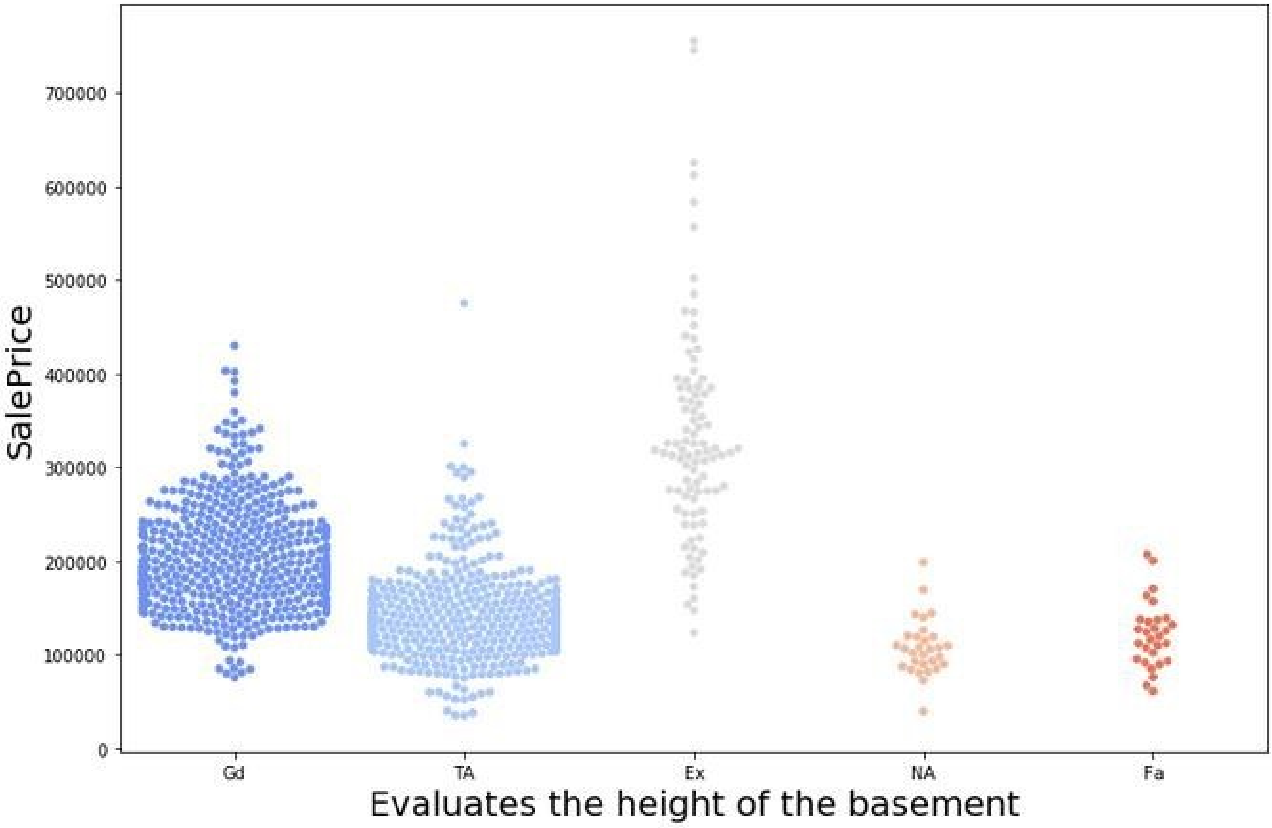
- KitchenQual and SalePrice

Higher the quality of kitchen, higher the price of the house



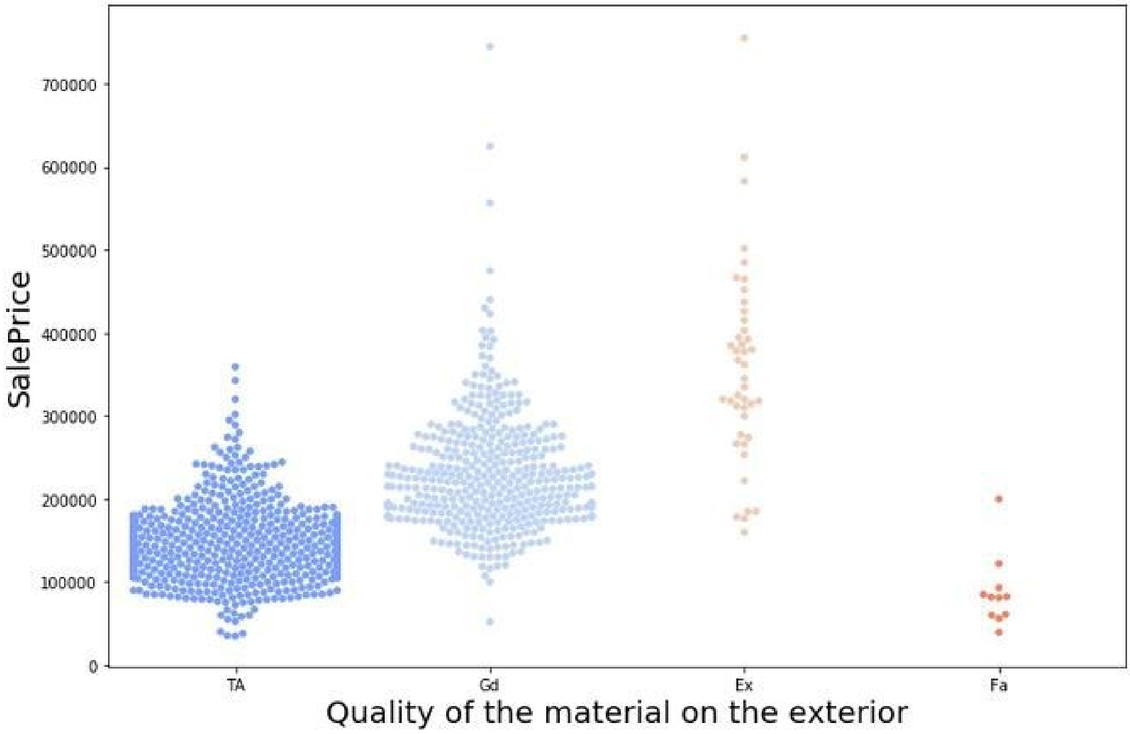
- BsmtQual and SalePrice

Houses with the higher basement quality were sold at higher prices



- ExterQual and SalePrice

I can say that the higher the quality of the material on exterior, higher the sale price of the house



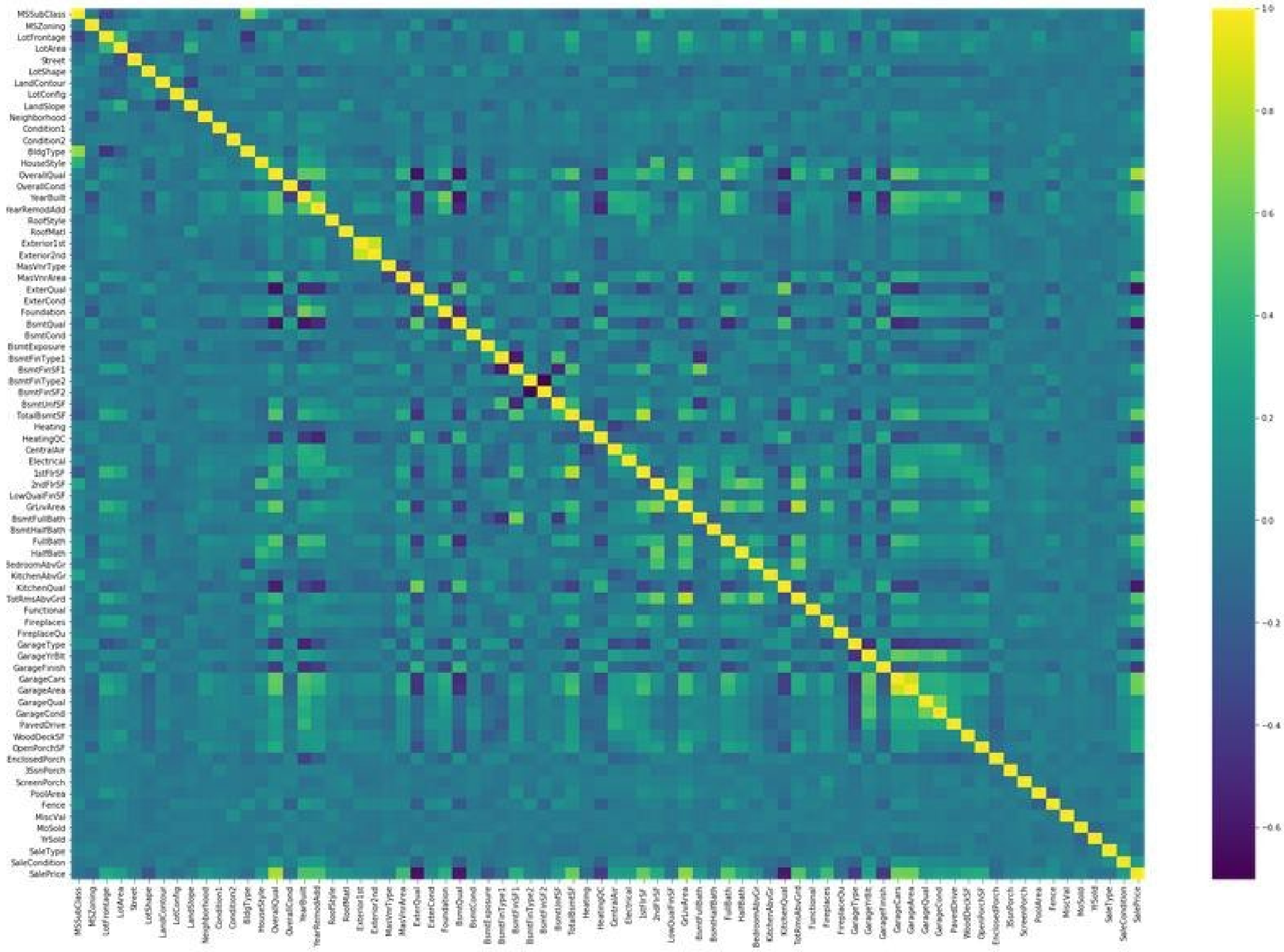
Now that we have visualized the variables with high correlation with the target (Sale Price). I can proceed further with visualizing the multi-collinearity

I'm removing the variable utilities from the dataset it has no correlation with the target

```
data = data.drop(columns = 'Utilities')
```

To determine the multi-collinearity, I'm using heat map from seaborn library to visualize the same

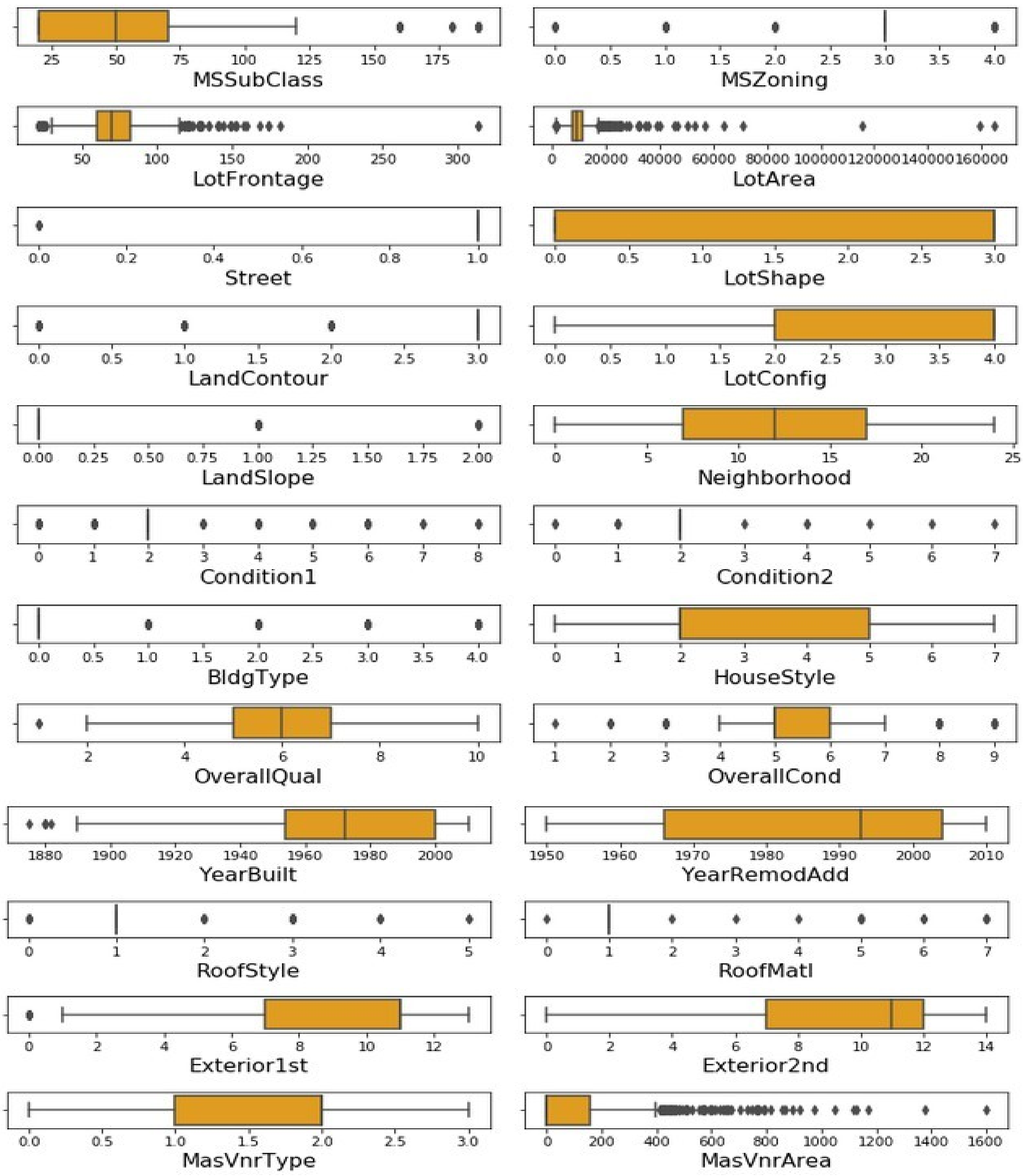
```
heat = data.corr()
plt.figure(figsize = (30,20))
sns.heatmap(heat, cmap = 'viridis')
```

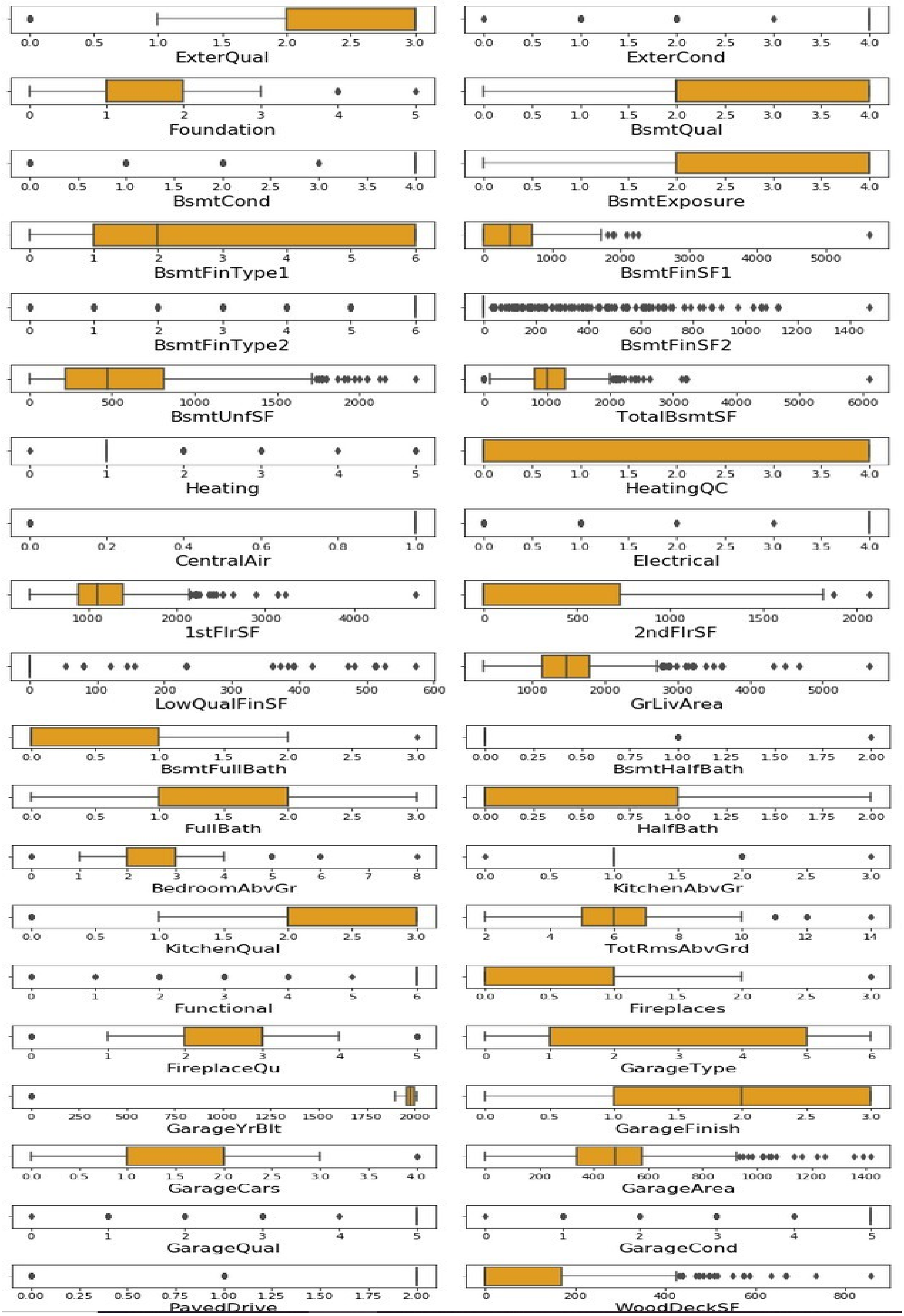


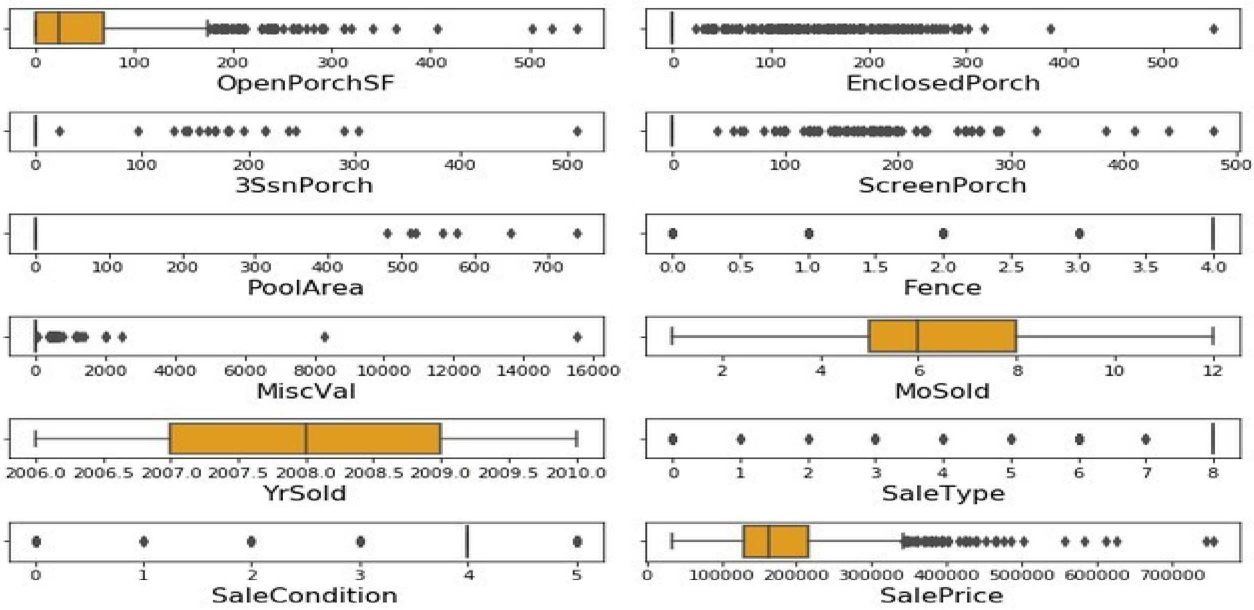
I can see that there is no multi-collinearity issues with the dataset and I can proceed with the further pre-processing of the data

Checking for outliers in the dataset using the boxplot method

```
plt.figure(figsize = (10,90))
pltnum = 1
for i in data:
    if pltnum <= 80:
        plt.subplot(80,2, pltnum)
        sns.boxplot(data[i], color = 'orange')
        plt.xlabel(i, fontsize = 15)
    pltnum+=1
plt.tight_layout()
```




ExterQual
ExterCond
Foundation
BsmtQual
BsmtCond
BsmtExposure
BsmtFinType1
BsmtFinSF1
BsmtFinType2
BsmtFinSF2
BsmtUnfSF
TotalBsmtSF
Heating
HeatingQC
CentralAir
Electrical
1stFlrSF
2ndFlrSF
LowQualFinSF
GrLivArea
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
KitchenQual
TotRmsAbvGrd
Functional
Fireplaces
FireplaceQu
GarageType
GarageYrBlt
GarageFinish
GarageCars
GarageArea
GarageQual
GarageCond
PavedDrive
WoodDeckSF



I can see that most of the continuous data variables has outliers. I'm using the zscore method to control the outliers. I'm removing the data with z-score of more than 3.

```
z = np.abs(zscore(data[['LotFrontage','LotArea','MasVnrArea','GarageArea','OpenPorchSF']]))
z.head()
```

	LotFrontage	LotArea	MasVnrArea	GarageArea	OpenPorchSF
0	1.060531	0.620616	0.558343	0.171944	2.387850
1	0.936882	0.600903	0.558343	0.672371	2.417992
2	0.813585	0.063075	0.558343	0.101973	1.257525
3	1.347872	0.141424	2.076985	0.322517	1.136957
4	0.369715	0.686902	0.133430	0.243217	0.701705

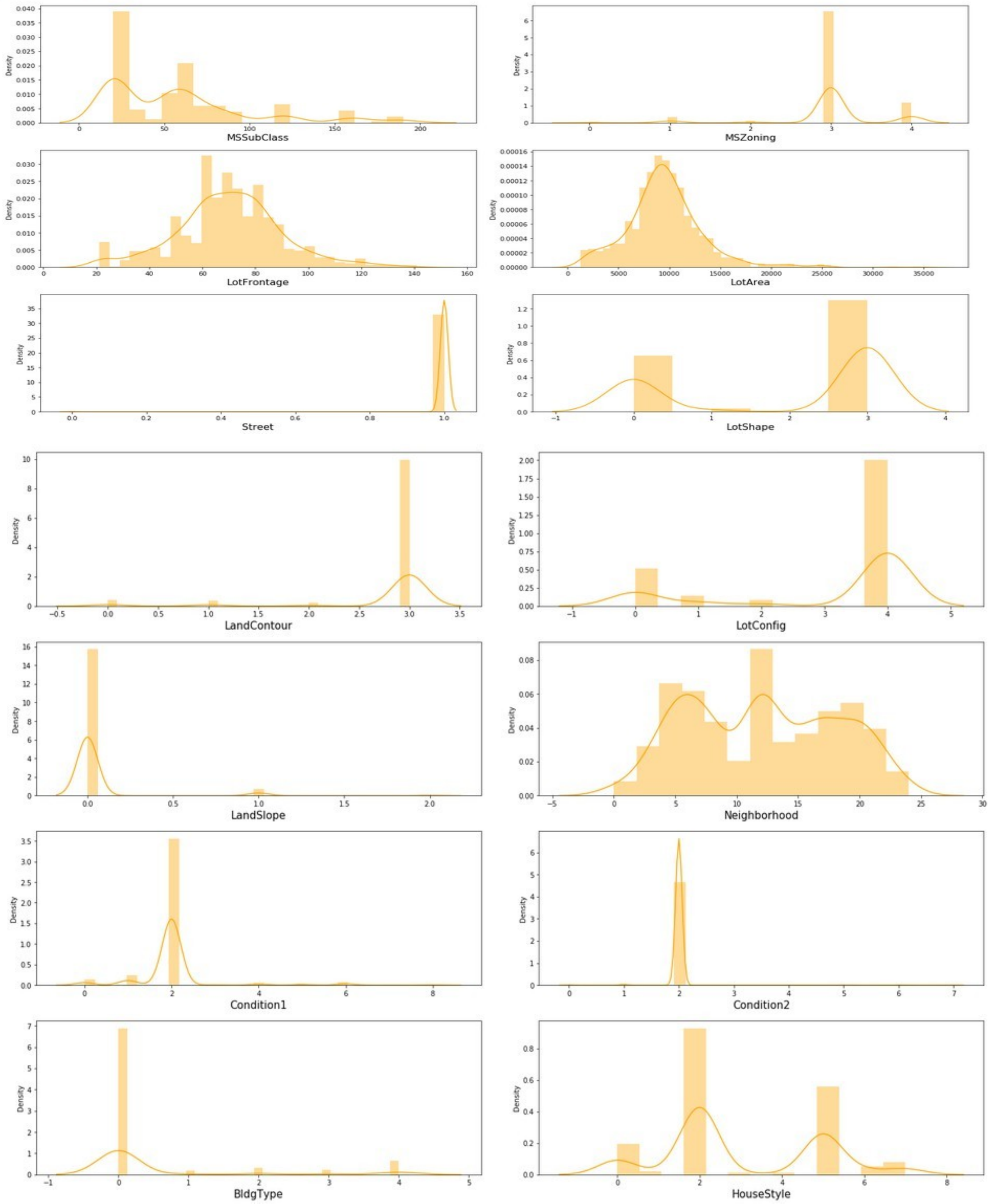
```
new_data = data[(z<3).all(axis = 1)]
print(data.shape)
print(new_data.shape)
```

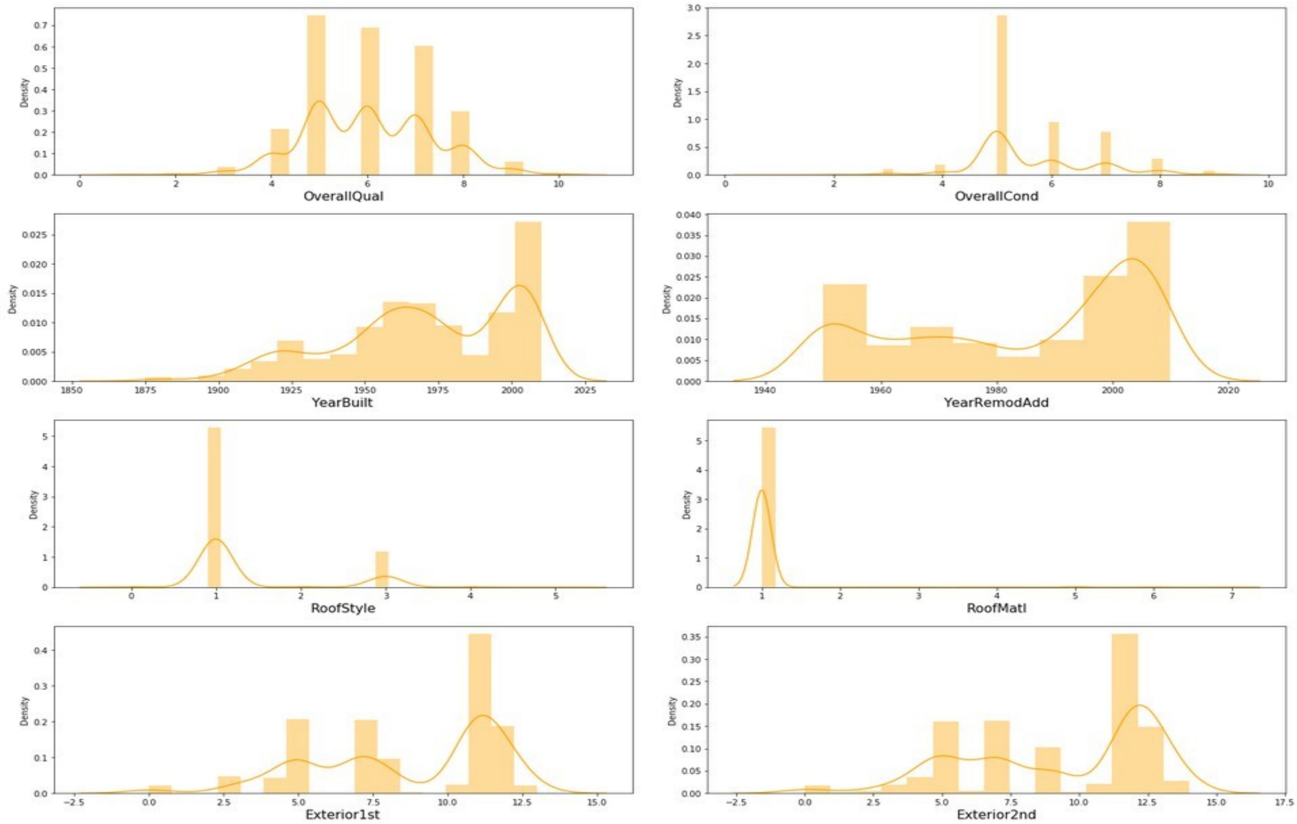
```
(1168, 76)
(1098, 76)
```

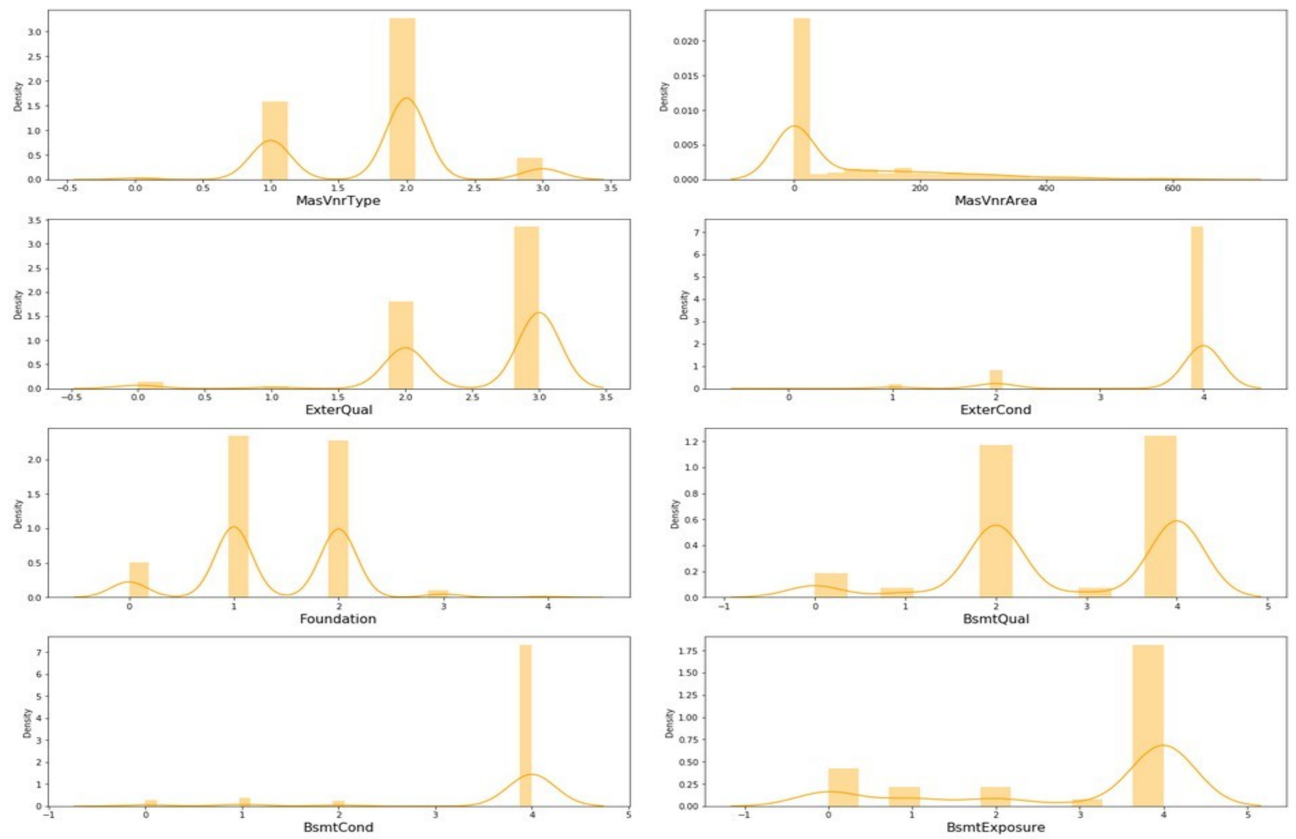
Using the above method to remove outliers we are only removing 6% of the data, which is under limit. Therefore proceeding with outlier removal.

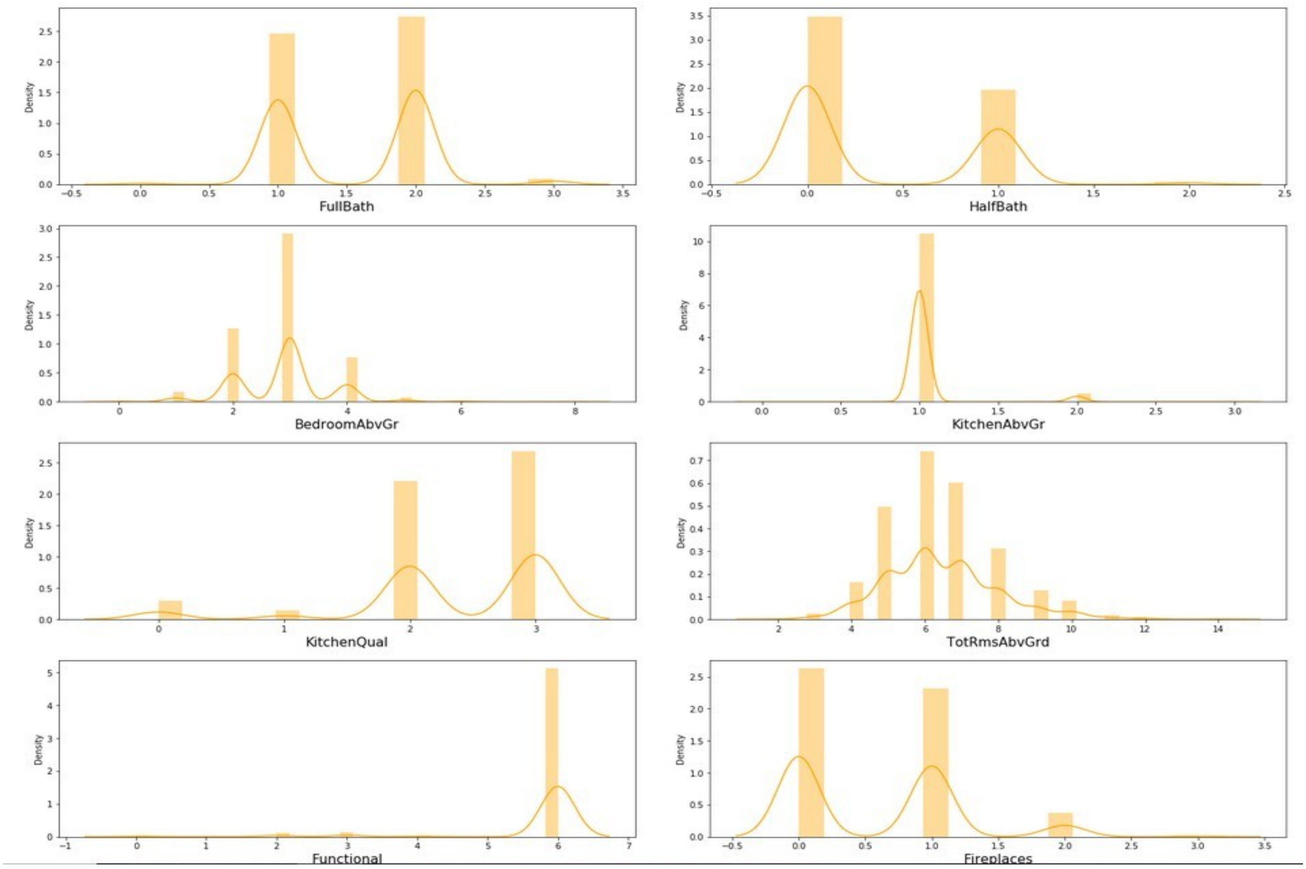
Checking for the data distribution using distribution plot from Seaborn library to check and control skewness

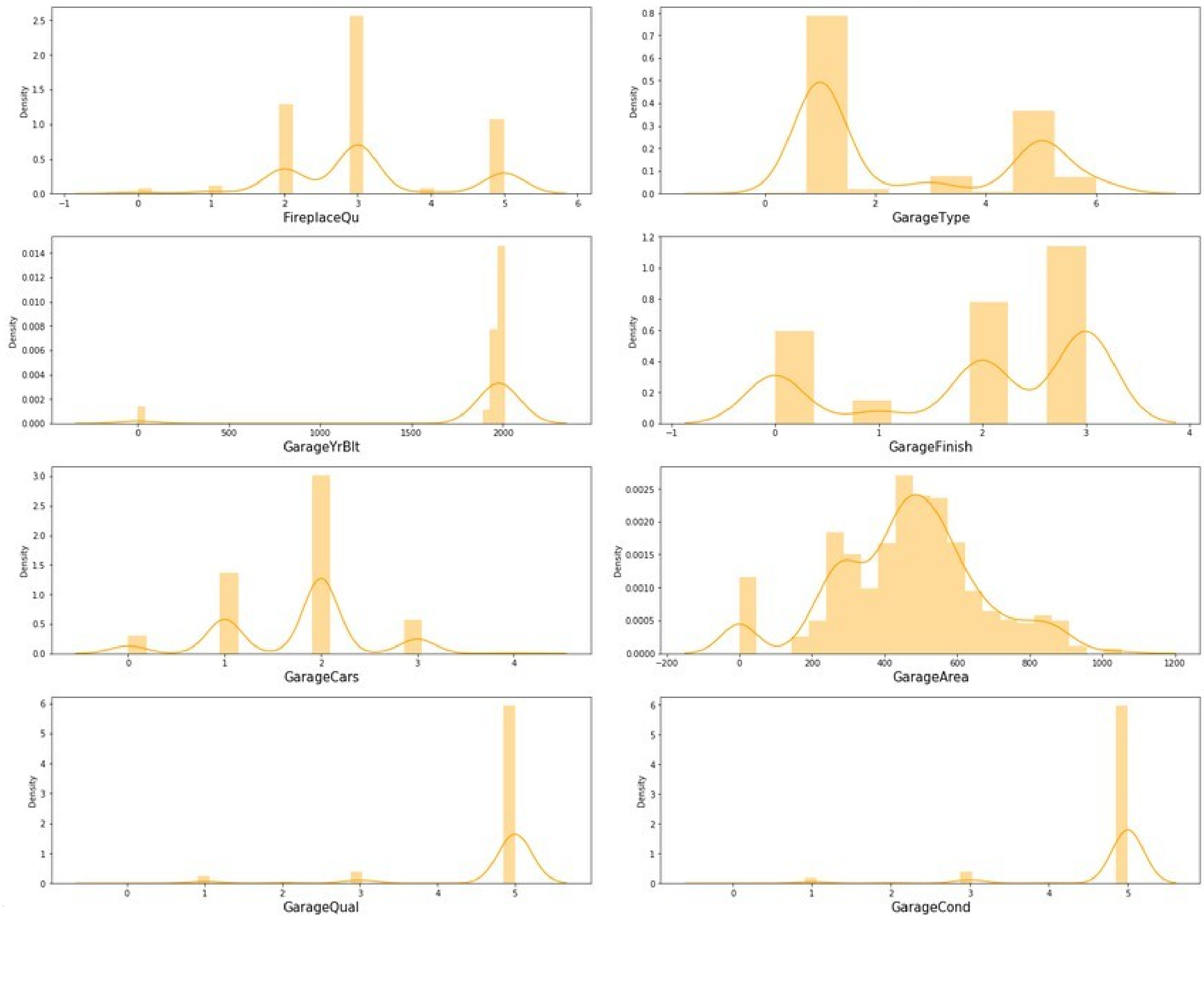
```
plt.figure(figsize = (20,150))
pltnum = 1
for i in new_data:
    if pltnum <= 80:
        plt.subplot(40,2, pltnum)
        sns.distplot(new_data[i], color = 'orange')
        plt.xlabel(i, fontsize = 15)
    pltnum+=1
plt.tight_layout()
```

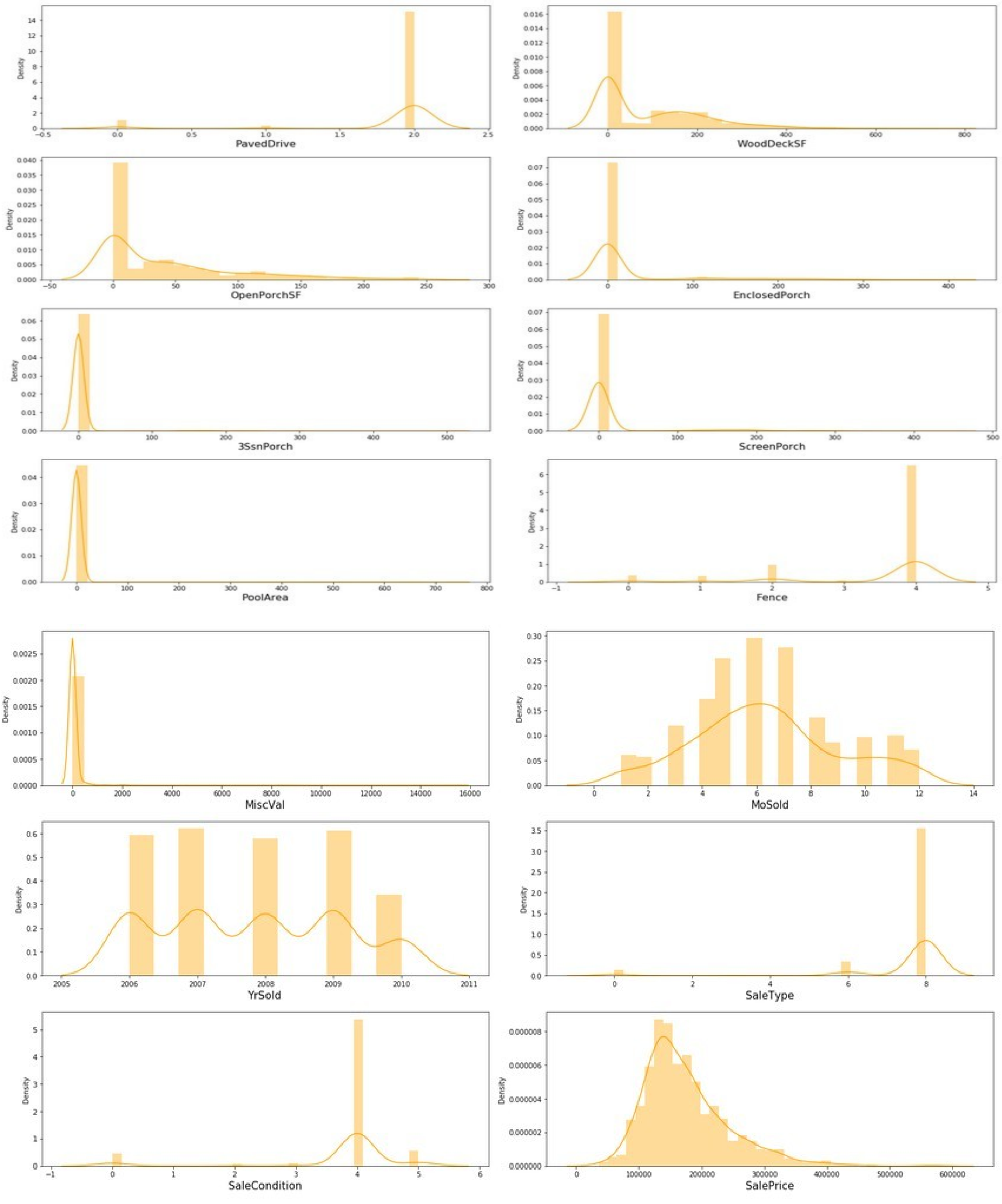



Density
OverallQual
OverallCond
YearBuilt
YearRemodAdd
RoofStyle
RoofMatl
Exterior1st
Exterior2nd


Density
MasVnrType
MasVnrArea
ExterQual
ExterCond
Foundation
BsmtQual
BsmtCond
BsmtExposure


Density
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
KitchenQual
TotRmsAbvGrd
Functional
Fireplaces


Density
FireplaceQu
GarageType
GarageYrBlt
GarageFinish
GarageCars
GarageArea
GarageQual
GarageCond



Upon viewing the above distribution plots, I can see that there is good amount of skewness in the continuous variables. In order to control them I'm using the transformation techniques to transform the data and control skewness.

In this dataset, I'm using power transformation technique to achieve the same.

The data has negative values and in order to apply power transformation we can use yeo-johnson method.

I'm splitting the target and independent variable before applying transformation technique on the independent variables

```
x = new_data.drop(columns = 'SalePrice')
y = new_data['SalePrice']
```

Once the data has been split to x and y, I'm scaling the dataset and applying transformation to get the skewness within the range of -0.5 to +0.5 (acceptable range).

```
from sklearn.preprocessing import StandardScaler
scal = StandardScaler()
sc = scal.fit_transform(x)
x = pd.DataFrame(sc, columns = x.columns)
```

```
tr = power_transform(x, method = 'yeo-johnson')
x = pd.DataFrame(tr, columns = x.columns)
```

We can verify the skewness in the dataset, post transforming the data

```
x.skew()
```

MSSubClass	0.221887
MSZoning	0.032840
LotFrontage	0.018485
LotArea	0.008158
Street	-23.398659
LotShape	-0.639342
LandContour	-2.771882
LotConfig	-1.047930
LandSlope	4.316133
Neighborhood	0.009453
Condition1	-0.488883
Condition2	-1.326000
BldgType	1.811786
HouseStyle	-0.021076
OverallQual	-0.041239
OverallCond	-0.292863
YearBuilt	-0.119437
YearRemodAdd	-0.184431
RoofStyle	-0.969075
RoofMatl	9.038621
Exterior1st	-0.185874
Exterior2nd	-0.190573
MasVnrType	0.165466
MasVnrArea	0.681389
ExterQual	-0.578356
ExterCond	-2.286746
Foundation	0.022249
BsmtQual	-0.182205
BsmtCond	-2.463674
BsmtExposure	-0.736718
BsmtFinType1	0.004043
BsmtFinSF1	0.174261
BsmtFinType2	-2.054602
BsmtFinSF2	2.434390
BsmtUnfSF	0.097727
TotalBsmtSF	-0.063782
Heating	-9.508290
HeatingQC	0.250463
CentralAir	-3.455825
Electrical	-2.925186
1stFlrSF	0.020715
2ndFlrSF	0.430262
LowQualFinSF	7.031376
GrLivArea	0.038383
BsmtFullBath	0.416963
BsmtHalfBath	3.963567
FullBath	0.057587
HalfBath	0.564088
BedroomAbvGr	0.120683
KitchenAbvGr	-7.394896
KitchenQual	-0.314050
TotRmsAbvGrd	-0.012789
Functional	-3.399013
Fireplaces	0.236908
FireplaceQu	-0.170242
GarageType	0.449208
GarageYrBlt	-1.273654
GarageFinish	-0.232185
GarageCars	0.130759
GarageArea	0.008223
GarageQual	-2.650201
GarageCond	-2.771882
PavedDrive	-3.008481
WoodDeckSF	0.384833
OpenPorchSF	0.381232
EnclosedPorch	2.004681
3SsnPorch	6.859889
ScreenPorch	3.096878
PoolArea	16.499917
Fence	-1.425834
MiscVal	5.100044
MoSold	-0.017917
YrSold	0.032931
SaleType	-2.086172
SaleCondition	0.615633

I can see that most of the skewness of the continuous variable is under control, however the variables 'MiscVal', 'PoolArea', 'ScreenPorch', '3SsnPorch', 'EnclosedPorch', 'BsmtFinSF2' and 'Exterior2nd' still has lot of skewness which will affect the prediction of the Sale price. Therefore I'm removing them from the dataset and proceeding with the model building.

```
x = x.drop(columns = ['MiscVal','PoolArea','ScreenPorch','3SsnPorch','EnclosedPorch','BsmtFinSF2','Exterior2nd'])
```

Further, before build the model we will have to split the data to test and train. The best possible way to split the data is by finding the best random state to split and the benefit is that we can control over fitting up to certain extent before even building the model.

We are trying to match the accuracy score of the training data set and the test dataset, which ever split (random state) satisfies the condition (R2 score of training dataset = R2 score of testing dataset). We'll take the same random state to split the dataset and build the model.

We are using a simple for loop to achieve the same.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
rs = 0
for i in range(0,3000):
    x_train,x_val, y_train,y_val = train_test_split(x,y,test_size = 0.3, random_state = i)
    lg = LinearRegression()
    lg.fit(x_train,y_train)
    val_pred = lg.predict(x_val)
    tr_score = lg.score(x_train,y_train)
    val_score = lg.score(x_val,y_val)
    if round(tr_score*100,1)==round(val_score*100,1):
        if i>rs:
            rs = i
print('the best random state for the data set is', rs)
```

```
the best random state for the data set is 2744
```

Now, I can say that the best random state for the split is 2744 and we will be splitting the dataset 70% train and 30% test with the random state 2744.

I'm testing the results with the below algorithms.

1. Logistic Regression
2. Random Forest Classifier
3. Extra Trees Classifier
4. XG Boost Classifier
5. K-Nearest Neighbors Classifier

In order to test the model, I'm using Mean Absolute Error and R2 score, further in order to verify the model's fit, I'm using cross val score to identify the best model.

Model 1: Linear Regression

The first Machine Learning model I'm using to predict the sale price is Linear Regression, this gives us with better understanding of the dataset and it's a simple model to build

```
lin = LinearRegression()
lin.fit(x_train,y_train)
lin_pred = lin.predict(x_val)
lin_score = lin.score(x_val,y_val)
lin_score
```

```
0.8758945634569286
```

```
lin_rmse = mean_absolute_error(y_val, lin_pred)
print('The recoreded mean absolute error for the Linear Regression is: ', lin_rmse)
```

```
The recoreded root mean absolute error for the Linear Regression is:  16863.62382546969
```

Using the Linear Regression, we were able to get the R2 score of 0.88, and the mean absolute error is 16863.62

Further, I'm verifying the fit using cross_val_score with cross validation of 5 and see that the model is not overfitting.

```
cv = cross_val_score(lin,x,y,scoring ='r2', cv = 5)
cv =cv.mean()
cv
```

```
0.8508168665375377
```

Model 2: Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
rfr  = RandomForestRegressor()
rfr.fit(x_train,y_train)
rfr_pred = rfr.predict(x_val)
rfr_score = rfr.score(x_val,y_val)
rfr_score
```

```
0.8931618363914138
```

```
rfr_rmse = mean_absolute_error(y_val, rfr_pred)
print('The recoreded mean absolute error for the Random Forest Regression is: ', rfr_rmse)
```

```
The recoreded root mean absolute error for the Random Forest Regression is:  14959.242303030303
```

Using the Random Forest Regression, we were able to get the R2 score of 0.89, and the mean absolute error is 14959.24

Further, I'm verifying the fit using cross_val_score with cross validation of 5 and see that the model is not overfitting.

```
cv1 = cross_val_score(rfr,x,y,scoring ='r2', cv = 5)
cv1 =cv1.mean()
cv1
```

```
0.8517103023510184
```

Model 3: Extra Trees Regressor

```
from sklearn.ensemble import ExtraTreesRegressor
et  = ExtraTreesRegressor()
et.fit(x_train,y_train)
et_pred = et.predict(x_val)
et_score = et.score(x_val,y_val)
et_score
```

```
0.8803535406846894
```

```
et_rmse = mean_absolute_error(y_val, et_pred)
print('The recoreded  mean absolute error for the ExtraTrees Regression is: ', et_rmse)
```

```
The recoreded root mean absolute error for the ExtraTrees Regression is:  14911.914333333334
```

Using the Extra Trees Regression, we were able to get the R2 score of 0.88, and the mean absolute error is 14911.91

Further, I'm verifying the fit using cross_val_score with cross validation of 5 and see that the model is not overfitting.

```
cv2 = cross_val_score(et,x,y,scoring ='r2', cv = 5)
cv2 =cv2.mean()
cv2
```

```
0.8483706516988508
```

Model 4: Ridge Regression

```
from sklearn.linear_model import Ridge, RidgeCV
ridgecv = RidgeCV(alphas = np.arange(0.001,0.1,0.01), normalize = True)
ridgecv.fit(x_train, y_train)
```

```
RidgeCV(alphas=array([0.001, 0.011, 0.021, 0.031, 0.041, 0.051, 0.061, 0.071, 0.081,
       0.091]),
        normalize=True)
```

```
alpha = ridgecv.alpha_
alpha
```

```
0.09099999999999998
```

```
ridge_reg = Ridge(alpha)
ridge_reg.fit(x_train, y_train)
```

```
Ridge(alpha=0.09099999999999998)
```

```
ridge_reg.score(x_val, y_val)
```

```
0.8759152195629079
```

```
rid_pred = ridge_reg.predict(x_val)
```

```
rid_mae = mean_absolute_error(y_val, rid_pred)
print('The recoreded mean absolute error for the Ridge Regression is: ', rid_mae)
```

```
The recoreded mean absolute error for the Ridge Regression is:  16861.681447121322
```

Using the Ridge Regression, we were able to get the R2 score of 0.88, and the mean absolute error is 16861.91

Further, I'm verifying the fit using cross_val_score with cross validation of 5 and see that the model is not overfitting.

```
cv3 = cross_val_score(ridge_reg,x,y,scoring ='r2', cv = 5)
cv3 =cv3.mean()
cv3
```

```
0.8508419726849377
```

Model 5: XG Boost Regressor

```
from xgboost import XGBRegressor
xg = XGBRegressor()
xg.fit(x_train,y_train)
xg_pred = xg.predict(x_val)
xg_score = xg.score(x_val,y_val)
xg_score
```

```
0.8691153864127427
```

```
xg_mae = mean_absolute_error(y_val, xg_pred)
print('The recoreded mean absolute error for the XG Boost Regressor is: ', xg_mae)
```

```
The recoreded mean absolute error for the XG Boost Regressor is:  16416.311576704546
```

Using the Linear Regression, we were able to get the R2 score of 0.87, and the mean absolute error is 16861.91

Further, I'm verifying the fit using cross_val_score with cross validation of 5 and see that the model is not overfitting.

```
cv4 = cross_val_score(xg,x,y,scoring ='r2', cv = 5)
cv4 =cv4.mean()
cv4
```

```
0.8412766241398156
```

Finding the best model by subtracting the model's R2 score with the cross validation scores.

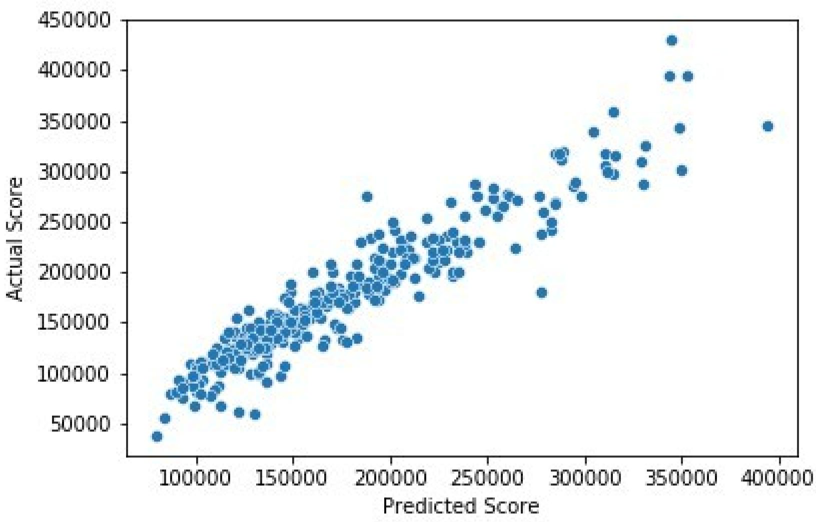
```
model = [lin_score,rfr_score,et_score,rid_pred,xg_score]
cv = [cv,cv1,cv2,cv3,cv4]
model_sel = pd.DataFrame({})
model_sel['model'] = model
model_sel['cv'] = cv
model_sel['difference'] = model_sel['model'] - model_sel['cv']
model_sel
```

	model	cv	difference
0	0.875895	0.850817	0.025078
1	0.893162	0.851710	0.041452
2	0.880354	0.848371	0.031983
3	[150102.65875060426, 79503.52708491248, 121639...	0.850842	[150101.80790863157, 79502.6762429398, 121638....
4	0.869115	0.841277	0.027839

Here I can see that the Random Forest model is giving good r2 score and the mean absolute error is less. Further the model is also not overfitting.

```
sns.scatterplot(x = rfr_pred, y = y_val)
plt.xlabel('Predicted Score')
plt.ylabel('Actual Score')
```

```
Text(0, 0.5, 'Actual Score')
```



Performing the Hyper Parameter Tuning on the Random Forest regressor

```
params ={'n_estimators':[100,200,300,400],
        'max_depth':[13,15,17,19],
        'min_samples_split':[3,4,5,6],
        'criterion':['mse','mae']}
```

```
gcv = GridSearchCV(RandomForestRegressor(), params, cv =5, n_jobs = -1)
gcv.fit(x_train,y_train)
```

```
GridSearchCV(cv=5, estimator=RandomForestRegressor(), n_jobs=-1,
             param_grid={'criterion': ['mse', 'mae'],
                         'max_depth': [13, 15, 17, 19],
                         'min_samples_split': [3, 4, 5, 6],
                         'n_estimators': [100, 200, 300, 400]})
```

```
gcv.best_params_
```

```
{'criterion': 'mae',
 'max_depth': 15,
 'min_samples_split': 3,
 'n_estimators': 100}
```

```
fin = RandomForestRegressor(criterion = 'mae',max_depth = 15, min_samples_split = 3, n_estimators = 100)
fin.fit(x_train,y_train)
fin_pred = fin.predict(x_val)
fin_score = fin.score(x_val,y_val)
fin_score
```

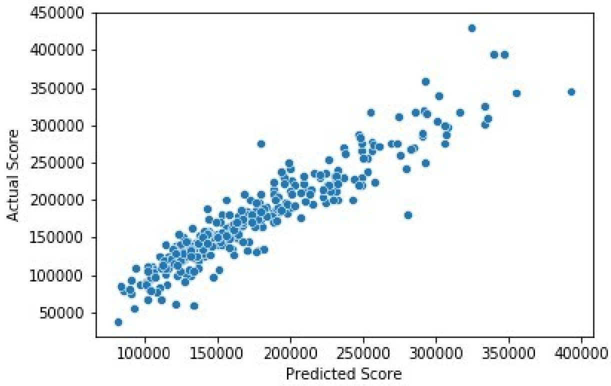
```
0.8832173043391425
```

```
fin_mae = mean_absolute_error(y_val,fin_pred)
print("The mean absolute error for the final model is ", fin_mae)
```

```
The mean absolute error for the final model is  15534.975681818181
```

```
sns.scatterplot(x = fin_pred, y = y_val)
plt.xlabel('Predicted Score')
plt.ylabel('Actual Score')
```

```
Text(0, 0.5, 'Actual Score')
```



Performing the hyper parameter tuning doesn't improve the scores, therefore finalizing the base Random Forest model because it is providing the R2 score of 0.89.

The Key Metric used to finalize the model was R2 score, cross_val_score and the Mean Absolute Error. And the Random Forest is the best model at predicting the selling price of a house.

Now using the same pre-processing method to predict the test data provided and we are using basic Random Forest Regressor to predict the same.

```
predicted_data = rfr.predict(test_new)
```

Saving the best model

```
import joblib
joblib.dump(fin,'RealEstatePrediction.pkl')
```

```
['RealEstatePrediction.pkl']
```

Conclusion

We have successfully built a model using multiple models and found that the Random Forest Regressor model.

Below are the details of the model's metrics predicting the dataset

1. R2 score of 0.89
2. Mean Absolute error of 14959.24

Major variables which are correlated with the target variable and is important in predicting the sale price of a house are

- OverallQual = 0.789185
- GrLivArea = 0.707300
- GarageCars = 0.628329
- GarageArea = 0.619000
- TotalBsmtSF = 0.595042
- 1stFlrSF = 0.587642
- FullBath = 0.554988
- TotRmsAbvGrd = 0.528363
- YearBuilt = 0.514408
- YearRemodAdd = 0.507831
- MasVnrArea = 0.460535
- Fireplaces = 0.459611
- Foundation = 0.374169
- BsmtFinSF1 = 0.362874
- OpenPorchSF = 0.339500
- 2ndFlrSF = 0.330386
- LotFrontage = 0.319416
- WoodDeckSF = 0.315444
- HalfBath = 0.295592
- HeatingQC = -0.406604
- GarageType = -0.415370
- GarageFinish = -0.424922
- KitchenQual = -0.592468
- BsmtQual = -0.601307
- ExterQual = -0.624820

All the above mentioned variable affect the sales price of a house which we discussed in the pre-processing section and visualized each variable's relation with the target

Limitations of this work and Scope for Future Work

- The amount of data is very less, it would be better to have more data to predict the sale price more accurately.
- There are more outliers in the provided data and I was unable to remove all the outliers because I could lose data. With more data more outliers can be removed from the dataset.

Other than these above limitations, I couldn't find more scope for improvement.