HOUSING PRICE PRIDICTION PROJECT

Submitted by:
Manjunath Aparoji

ACKNOWLEDGMENT

Firstly, I would like to thank FlipRobo Technologies for giving me this opportunity to work on this project. Also, I would like to thank the DataTrained team, especially Shankargouda Tegginmani sir for providing me the knowledge and guidance which helped me a lot to work on this project.

References:

https://stackoverflow.com/

https://scikit-learn.org/stable/

https://seaborn.pydata.org/

INTRODUCTION

- Business Problem Framing

  The main objective of this project is to model the price of houses with the available independent variables. This model will then be used by the management to understand how exactly the prices vary with the variables. They can accordingly manipulate the strategy of the firm and concentrate on areas that will yield high returns. Further, the model will be a good way for the management to understand the pricing dynamics of a new market.

- Conceptual Background of the Domain Problem

  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.

- Technical Requirements
  - Data contains 1460 entries each having 81 variables.
  - Data contains Null values. We need to treat them using the domain knowledge and your own understanding.
  - Extensive EDA has to be performed to gain relationships of important variable and price.

- Data contains numerical as well as categorical variable. We need to handle them accordingly.
- We have to build Machine Learning models, apply regularization and determine the optimal values of Hyper Parameters.
- We need to find important features which affect the price positively or negatively.
- Two datasets are being provided to us (test.csv, train.csv).

Motivation for the Problem Undertaken

1. The objective behind to take this project is to harness the required data science skills.
2. Improve the analytical thinking.
3. 3. Get into the real world problem solving mechanics.

Analytical Problem Framing

- Mathematical/ Analytical Modeling of the Problem

This is a Regression problem, where our end goal is to predict the Prices of House based on given data provided in the dataset. We have divided the provided dataset into Training and Testing parts.

A Regression Model will be built and trained using the Training data and the Test data will be used to predict the outcomes. This will be compared with available test results to find how well our model has performed.

We are using Mean Absolute Error, Root Mean Square Error, and 'R2_Score' to determine the best model among,

- ❖ Linear Regression
- ❖ Lasso
- ❖ Decision Tree Regression
- ❖ K Neighbors Regression
- ❖ Random Forest Regression

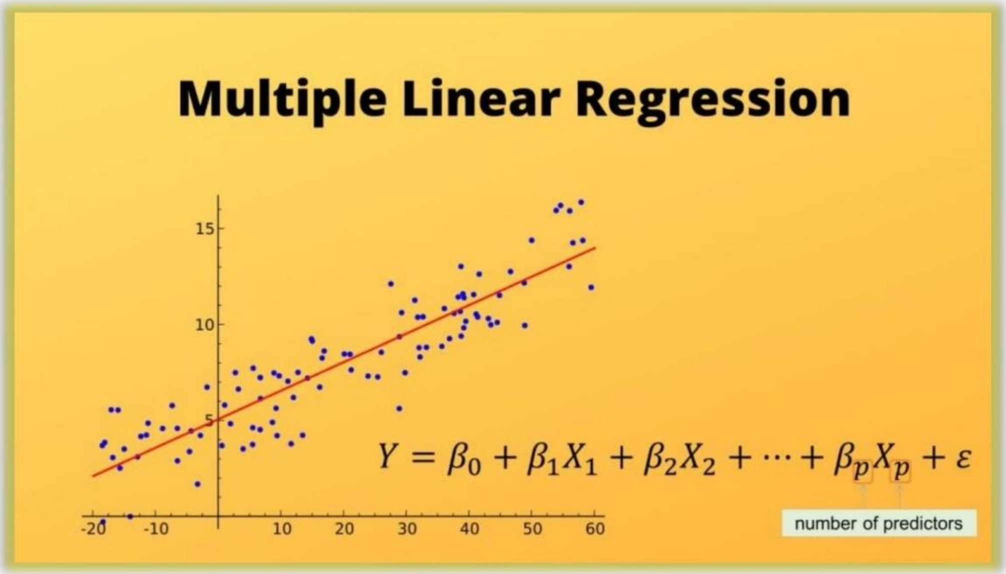
The best results were obtained using Lasso Regression. So, let's discuss a little bit about it. In a simple regression problem (a single x and a single y), the form of the model would be:

y = B0 + B1*x

where **B0** —intercept, **B1** —coefficient, **x** —independent variable
y — output or the dependent variable.

In higher dimensions when we have more than one input (x), The General equation for a Multiple linear regression with p — independent variables:

Y=B0 + B1 * X1 + B2 * X2 + ... + Bp * Xp + E(Random Error or Noise)



(Image Source: https://morioh.com/p/0d9b2bedf683)

Let's consider a regression scenario where 'y' is the predicted vector and 'x' is the feature matrix. Basically in any regression problem, we try to minimize the squared error. Let 'β' be the vector of parameters (weights of importance of features) and 'p' be the number of features.

Now, let's discuss the case of lasso regression, which is also called L1 regression since it uses the L1 norm for regularization. In lasso regression, we try to solve the below minimization problem:

$$Min_\beta \, L_1 = (y - x\beta)^2 + \lambda \sum_{i=1}^{p} |\beta_i|$$

To simplify, suppose p =1, βi = β. Then,

$$L_1 = (y - x\beta)^2 + \lambda|\beta|$$
$$= \mathrm{y}^2 - 2\mathrm{xy}\beta + x^2\beta^2 + \lambda|\beta|$$

In Lasso Regression, the L1 penalty will look like,

L1p = |β1| + |β2|

Shrinking β1 to 8 and β2 to 100 would minimize the penalty to 108 from 1010, which means in this case the change is not so significant just by shrinking the larger quantity. So, in the case of the L1 penalty, both the coefficients have to be shrunk to extremely small values, in order to achieve regularization. And in this whole process, some coefficients may shrink to zero.

(Reference: https://www.analyticsvidhya.com/blog/2020/11/lasso-regression-causes-sparsity-while-ridge-regression-doesnt-unfolding-the-math/)

Assumptions:

I. **Linearity:** The relationship between X & mean of Y is linear.

II. **Homoscedasticity:** The variance of residual is the same for any value of X.

III. **Independence:** Observations are independent of each other.

IV. **Normality:** For any fixed value of X, Y is normally distributed.

- ## Data Sources and their formats

A US-based housing company named Surprise Housing has decided to enter the Australian market. The company uses data analytic 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 dataset contains 1460 rows and 81 columns (including the train dataset and test dataset).

The top 5 rows of the dataset are:

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

Condition2	BldgType	HouseStyle	OverallQual	OverallCond	YearBuilt	YearRemodAdd	RoofStyle	RoofMatl	Exterior1st	Exterior2nd	MasVnrType	MasVnrArea
Norm	TwnhsE	1Story	6	5	1976	1976	Gable	CompShg	Plywood	Plywood	None	0.0
Norm	1Fam	1Story	8	6	1970	1970	Flat	Tar&Grv	Wd Sdng	Wd Sdng	None	0.0
Norm	1Fam	2Story	7	5	1996	1997	Gable	CompShg	MetalSd	MetalSd	None	0.0
Norm	1Fam	1Story	6	6	1977	1977	Hip	CompShg	Plywood	Plywood	BrkFace	480.0
Norm	1Fam	1Story	6	7	1977	2000	Gable	CompShg	CemntBd	CmentBd	Stone	126.0

ExterQual	ExterCond	Foundation	BsmtQual	BsmtCond	BsmtExposure	BsmtFinType1	BsmtFinSF1	BsmtFinType2	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF
TA	TA	CBlock	Gd	TA	No	ALQ	120	Unf	0	958	1078
Gd	Gd	PConc	TA	Gd	Gd	ALQ	351	Rec	823	1043	2217
Gd	TA	PConc	Gd	TA	Av	GLQ	862	Unf	0	255	1117
TA	TA	CBlock	Gd	TA	No	BLQ	705	Unf	0	1139	1844
Gd	TA	CBlock	Gd	TA	No	ALQ	1246	Unf	0	356	1602

Heating	HeatingQC	CentralAir	Electrical	1stFlrSF	2ndFlrSF	LowQualFinSF	GrLivArea	BsmtFullBath	BsmtHalfBath	FullBath	HalfBath	BedroomAbvGr
GasA	TA	Y	SBrkr	958	0	0	958	0	0	2	0	2
GasA	Ex	Y	SBrkr	2217	0	0	2217	1	0	2	0	4
GasA	Ex	Y	SBrkr	1127	886	0	2013	1	0	2	1	3
GasA	Ex	Y	SBrkr	1844	0	0	1844	0	0	2	0	3
GasA	Gd	Y	SBrkr	1602	0	0	1602	0	1	2	0	3

KitchenAbvGr	KitchenQual	TotRmsAbvGrd	Functional	Fireplaces	FireplaceQu	GarageType	GarageYrBlt	GarageFinish	GarageCars	GarageArea	GarageQual
1	TA	5	Typ	1	TA	Attchd	1977.0	RFn	2	440	TA
1	Gd	8	Typ	1	TA	Attchd	1970.0	Unf	2	621	TA
1	TA	8	Typ	1	TA	Attchd	1997.0	Unf	2	455	TA
1	TA	7	Typ	1	TA	Attchd	1977.0	RFn	2	546	TA
1	Gd	8	Typ	1	TA	Attchd	1977.0	Fin	2	529	TA

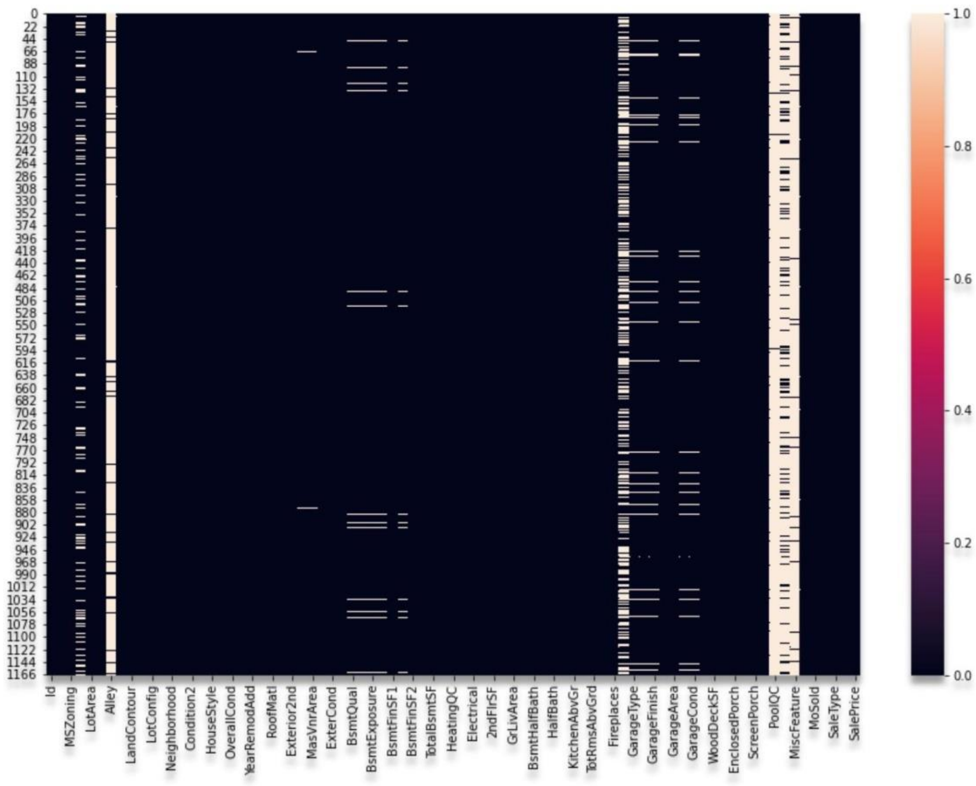
GarageCond	PavedDrive	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold
TA	Y	0	205	0	0	0	0	NaN	NaN	NaN	0	2
TA	Y	81	207	0	0	224	0	NaN	NaN	NaN	0	10
TA	Y	180	130	0	0	0	0	NaN	NaN	NaN	0	6
TA	Y	0	122	0	0	0	0	NaN	MnPrv	NaN	0	1
TA	Y	240	0	0	0	0	0	NaN	NaN	NaN	0	6

YrSold	SaleType	SaleCondition	SalePrice
2007	WD	Normal	128000
2007	WD	Normal	268000
2007	WD	Normal	269790
2010	COD	Normal	190000
2009	WD	Normal	215000

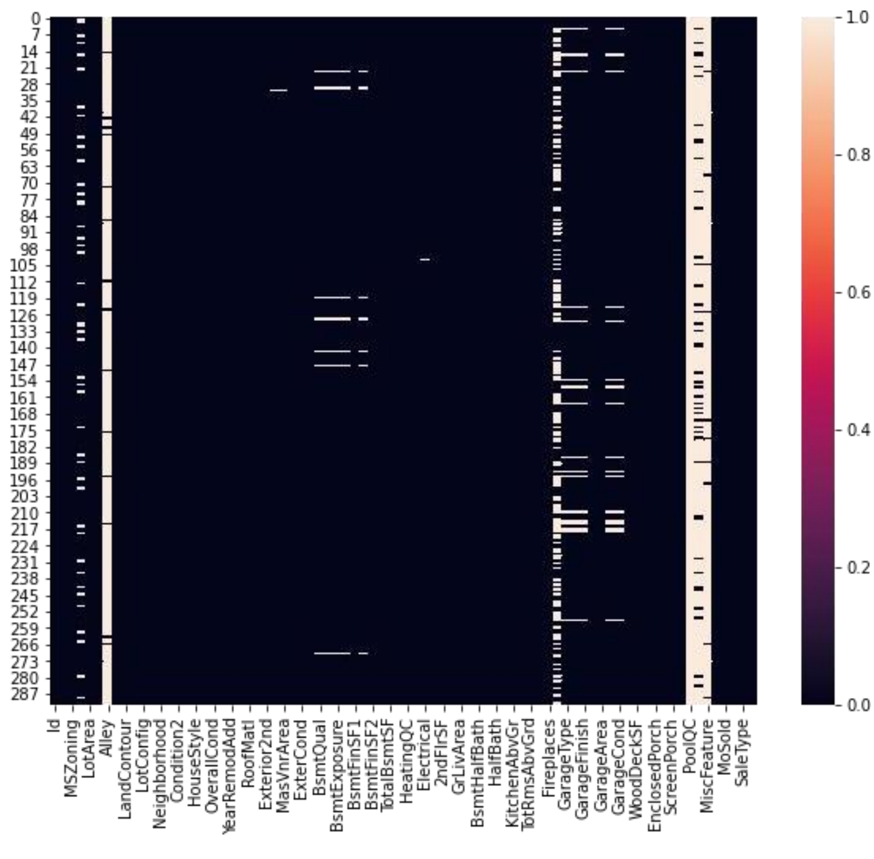
- The column **'SalePrice'** is the target column. We need to predict the sale price of the houses.

- Data Preprocessing Done

As our dataset contains null values (missing values) so we have replace the missing values with the required values. Details are mentioned below:



Heatmap to show the null values of Train Dataset



Heatmap to show the null values of Test Dataset.

We car see that most of the rows of the column " ooIQC" is empty so v/e'fe considering the empty values as there is no pool available in the house. So, u'e're filling the null values with 'NA'

```
# Checking for the values counts of the column "PoolQC"

df- ' PooIQR ' j . va1 ue_counts ( )
```

```
Gd    3
Fa    2
Ex    2
Name: PoolQC, dtype: int64
```

```
# Peqlori«g tñe null votues i:'.th '‹t4'

df['PoolQC' j.fillna('NA', inplace=True)
```

2. M/scFeature (Af/sce//aneous feafzfre not covered in other *categories)*

\Ve can see that most of the rows of the column "tdiscFeature" is empty so considering it as None we are replacing the missing values with 'NA'

```
# Checking for the value counts of the column "MiscFeature"

df['Miscreature’].value counts()
```

```
She d    49
Othr     2
Gar2     2
TenC     1
Name: MiscFeature, dtyQe: int64
```

```
# Replacing the null values with 'NA'

df['lisc-eatune’].fillna(’NA’, inplace=True)
```

3. Alley j7ype or a/ley access fo *property)*

We can see that in the case of alley column also mDst of the rDws are empty. So consioering it as no alley option was avaiable we‘re replacing the missing values with 'NA'

```
# Checking for the values counts of the column "Alley"

df[ ' ' ley ' . va1ue_count s ( )
```

```
Grvl    50
Pave    41
Name: Alley, dtyQe: int64
```

```
é Peplorzng the null votues with 'N/1'

df/ ' AJiey ' ] . f i l lna ( ' flA, ' 1np1ace=True)
```

4. hence *(Fence quality}*

From our observation we found that most of the rows are empty of hence column also. So. we're repJac ng the missing values with 'NA' to show that no fence was available.

```
d*['Ferce'].value_counts()
```

```
MnPrv    157

GdNo     54
f'1nMa   11
Name:  Fence,  dtype'  int64
```

```
df['Fence'j.*illna('NA', inQlace=True)
```

5. FineplaoeQu (fireplace qualify)

 # Checb:ing fi-or the voLues counts

```
df['FireplaceQu'}.value counts()
```

```
Gd      38e

Fa      33
Ex      Z4
Po      20
Name: FireplaceQu, dtype: intd4
```

We're considering the empty values as no fireplaCe is avaiable & replacing the empty values with 'NA'.

P *RepLac* ng* Site empty voLues ›vith M

```
df['FireplaceQu'}.fillna('NA', inplace=True)
```

6. LofFrontage {Linear feet ofsfreet *connecfad to property).-*

We'll rep1ace the missing values of this column with the mean value.

it Check ing f-or the nean va Lue of- the coLnon LotFrontage

```
df[ LotFrantage ' ] .nean()
```

```
70.04995836802665
```

4 RepLac!ng the oi ss ing vaL ues v! th the mean of the coL u/rn.

```
df['LotFrontage'}.fillna(df['LotFrontage'].mean(), inplace=True)
```

T. Sarage7)fpe Garage iecafionj

P Checking fi-or the vaL ue counts :

```
d -I[ ' G a rag eType ' ] . va lue_count s ( )
```

```
Attchd      8T0
Oetchd      387
BuiltIn      88
Basrent      t9
CarPort       9
2Types        6
Name : GanageType, dtype : int64
```

Considering the Garage option is not avaiable for the houses tLiat have empty rows for 'GarageType' column. So, we're replcing it with 'NA'

e Rep tack eg the ni ss ! rig vat ues or th “ NA ’

```
df['GarageType'].fillna('NA', inplace=True)
```

8. Garage*YrBtt tYear ga*ra *ge* was Aunty

```
# Replacing the missing values with 'NA' to show that Garage is not avaiable
df['GarageYrBlt'I.*illnat'°*', inplace = True)
```

9. CarageFrnisfi //nterior 7rnisfi of fhe garage}

```
# Checking for the vaLue counts

df['GarageFinish'].value counts()
```

```
Unf     605
RFn     422
Fin     352
Name: Gar'ageF1nish, dtype: Int64
```

```
4 Rep Laci eg the ni ssi rig vo I ues ivi th 'NA' to show opt i on i s not oval L abL e

df[ 'GarageFi ni sh'].Fi 11 na('NA' 1np1ace=True)
```

10. *GarageQual* /Garage qualify}

```
# Pepeoctng the elissing vaLues ›vlth "filA ' to show Garage is not avoi Lob L e

df['GarageQual'].fillna('NA', inplace=True)
```

If. GarageCond ¿Garage cond/tionJ

```
# Replacing the missing values with 'NA'

df['GarageCond'].fillna('NA', inplace=True)
```

12. BsmfFinType2 (Rafing of easement finished area (it mulâple types))

```
# Replacing the missing values with NA

df['BsmtfinType2'].fillna('NA’, inplace=True)
```

---

13. BsmfErposure {ReFers la walKout or garden /evef vra/is}

```
# Rep Lac ing the e.i ss i ng vaLues \ i th  ' h'A

df['BsmtExposune'].fillna('NA', inplace=True)
```

14. Bsmloual { vafuates the 6efghf of ttie basement)

```
# Replacing the missing values with 'NA'

df['BsmtQual'].fillna('NA’, inplace=True)
```

IS. BsmfConz/ {EVa/cafes the general conz/i#on of the basement)

```
# Replacing the missing  values with 'NA'

df['BsmtCond’].fillna(’NA’, inplace=True)
```

16. BsmtFinType1 (Rating of basement finished area)

```
0 Rep Lac i rig th e mi s s i n g vo I we s ivi th ' h'A '

df['BsmtfinType1 ].fillna('NA’, inplace=True)
```

17. âdasVnrType (Masonry veneer fyye}

```
it Ch e ck i rig  for  ltte  va L ue corn ts

df[ ' f'tasVnr7ype ' ] . va1ue_count s()
```

```
None        864
BrkFace     445
Stone       1Z8
Br'kCrin     1S
Name: MasVnrType, dtype: int64
```

- As the most occtmng masonary venner type is None so, we are replacing the missing values with 'None'

```
# Replacing the missing values with 'None'

df['MasVnrType’}.fillna(’None', inplace=True)
```

18. MasVnrArea (Masonry veneer area in square feet)

```
# Calculating the mean value

df['MasVnrArea'].mean()
```

```
103.68526170798899
```

```
# Replacing the missing values with the mean value

df['MasVnrArea'].fillna(df['MasVnrArea'].mean(), inplace=True)
```

19. Electrical (Electrical system)

```
# Checking for the value counts

df['Electrical'].value_counts()
```

```
SBrkr    1334
FuseA      94
FuseF      27
FuseP       3
Mix         1
Name: Electrical, dtype: int64
```

- Circuit Breakers & Romex electrical system is mostly used so we are replacing the missing value with SBrkr

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

- Data Inputs- Logic- Output Relationships

EDA was performed by creating valuable insights using various visualization libraries.

Importing the required libraries:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split

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

- Hardware and Software Requirements and Tools Used

Hardware Configuration:

Operating System: Windows 10 System

Type: 64-bit operating system, x64-based processor

Processor: Intel® Core™ i3-5005U @ 2.00 GHz 2.00 GHz

RAM: 4GB

Software & Tools:

a) Jupyter Notebook (used as a notebook to code)

b) Python (used for scientific computation)

c) Pandas (used for scientific computation)

d) Numpy (used for scientific computation)

e) Matplotlib (used for visualization)

f) Seaborn (used for visualization)

g) Scikit-learn (used as algorithmic libraries)

Model/s Development and Evaluation

- Identification of possible problem-solving approaches (methods)
  - ✓ Performed EDA (Exploratory Data Analysis).
  - ✓ Data Cleaning and dropping the columns which were not contributing to the dataset.
  - ✓ Handled the missing values.
  - ✓ Checked for the outliers and tried to remove the outliers of the dataset.
  - ✓ Checked for the skewness in the dataset and removed the skewness for better model building.
  - ✓ Train- Test the dataset into independent and dependent variables.
  - ✓ Model Building.
- Testing of Identified Approaches (Algorithms)

  Below are the algorithms used for the training and testing:

  1) Linear Regression.
  2) Lasso
  3) Decision Tree Regression.
  4) K Neighbour Regression.
  5) Random Forest Regression.

- Run and Evaluate selected models

1. LinearRegression: ¶

```
from sklearn.linear_model import LinearRegression
LR = LinearRegression( fit_intercept = True)

LR.fit(x_train, y_train)
print(f"Linear coefficients : {LR.coef_}")
print(f"Intercept : {LR.intercept_}")
```

```
Linear coefficients : [ 8.03328015e+02 -1.19340146e+02 -4.37179268e+02 -4.82330058e+02
  5.33507676e+03  1.45874067e+03  1.07144277e+03  2.81630650e+03
 -1.00174878e+03  1.46346716e+03  1.56220891e+03 -2.44967596e+02
 -1.56530022e+03 -4.76366819e+03 -1.56558207e+03  1.62527912e+04
  5.34617811e+03 -1.52133719e+03  2.03824578e+02  6.36276198e+03
  1.29204204e+04 -3.31310097e+02 -2.05575965e+03  8.89946375e+02
  1.59810157e+02 -4.86914853e+03  8.81079125e+02  1.72894848e+03
 -5.18318324e+03 -3.38694984e+02 -4.62829878e+03  1.16696580e+03
  6.45614865e+03 -9.19678376e+03 -9.52719330e+03 -4.00054986e+03
  1.41440436e+04 -1.11943311e+03 -2.24885057e+03  7.81524720e+02
 -1.71043063e+03 -1.30389313e+03 -1.49218814e+02 -2.30666691e+03
  1.71694947e+04  1.62517313e+03 -1.68605436e+03  3.25099799e+03
  5.01700714e+03 -1.63394307e+03 -1.76613177e+03 -5.88504560e+03
  3.37937524e+03  3.24367604e+03  4.17102505e+03 -3.15206622e+03
  2.47775321e+03  3.21989047e+03 -2.70524274e+03  8.83444623e+03
  3.75437007e+02 -2.88203704e+03  3.16546924e+03  1.03268148e+03
  1.23663546e+03 -4.34562341e+02 -1.98628237e+02 -2.56343863e+02
  7.22343174e+02 -6.92337710e+15 -6.92337710e+15  5.87022656e+02
  5.40287087e+02  7.55021211e+00 -1.06136434e+03 -2.31728522e+02
 -1.41132880e+03  1.86576023e+03]
Intercept : 180956.48107936294
```

```
# Predicting the new result

LR pned = LR.predict(x  test)
L R p red
```

```
arra)([256716.48107936, 2e4z 4.23107936, 109856.16857936, 24G401.2.^* 57936,
       107879.7fl357936, 184985.3560793f, 337G96.35o67.^36, 1z.^660.s 6e7sz6,
       167ol8.41857936, 225302.10807936, 186966.79357936, 31381s.s8ie7.°36,
       140021.73107936, 1992B7.6060793f, I9 260.‹sie79z6, 78335.54357°36,
       1Z68B7.I0607936, 1402B0.60o07936, 317393.85667936, 140Z01.79357936,
       83317.41857936, 155316.48107936, 168171.64357.^36, 198882.10607936,
       202724.48107936, 299457.8560793f, 117977.48167.^*6, 106937.5435T936,
       139.^34.79337936, 165186.60607936, 228213.16857.^35, 94140.6060T936,
       248363.66837936, 232501.48107936, 196442.85667936, 173321.66857936,
       176085.48107936, 146047.85607936, 242080.48167.^36, 144643.41657036,
       115072.23107936, 104007.04357936, 266 17.35667936, 12.^163.sse7.°36,
       150529.54357936, 58012.4183793€, 329317.vzie79z6, 218173.7.^3 57.°35 ,
       152245.°B107936, 189484.9185793C, 238531.54357936, zi:46 .i06e7 z6 ,
       286204.23107936, 343814.29357936, 113150.48167.^36, 254671.04357936,
       144170.1B607936, 162390.48107936, 16347.^ .50607.^*6, 8P990.98i0T936,
       10464P.23107?36, 370119.35607936, 293967.54357.^36, 17.^652.98167.^ 36,
       240843.85607936, 163529.043>7936, 18o592.29357936, 97813.41857936,
       241524.98107936, 141450.918?793f, 86340.66o67.^36, 293613.29357036,
       2B610P.79357936, 5a39B.35607936, 373G36 .79357936, 177345.2P357°36,
       217045.23107936, 234855.2935793f, 16341o.16857936, 372280.10607°36,
       ia&sBe.°B107936, 162035.65o07936, 218271.29357936, 102B70.79357936,
       42.^86.79 337936, 75225.91857936, 120956.60667.^36, 129P75.91857936,
       143231.15857936, 94655.7933793f, 21756.^ .79357.^*6, 53121.1685T936,
       138960.6f857936, i49655.34337936, 193651.64357.^ 35, 124832.6060T936,
       169725.16857936, 1302fld.91857536, 252270.B56B7936, Z17P28.98107936,
       112347.08 10793G, 265156.41857936, 206056.10o07P36, 92962.z3ie7s36,
```

2. Lasso

```
fnom sklearn.linear model import Lasso

1s = Lasso()
1s.fit(x train, y train)

  Preo'iczFns tne ne' nesoi?y

1s pned = 1s.predict(x test)
1s pned
```

```
array([256688. 0881713s,   204223. 08os9      ,   16.^690.38811304,   24G467 . 9417358 3,
        107883.22300311,   18498 3. 4613 30G1,   3 37732.2689803 ,   139640 . 78262227,
        1€76S3. 442230.^4,   22527 3.71254572,   186063 . BE792203,   3 13803 . 20665669,
        140009 . 7128871s ,   199365. 45289377,   193258.14272.^47,   78343 . 80172.^11,
        126658 . 7624441 ,   1403 30 . G1824404,   3 17392.78030.^31,   140199 . 67S43347,
        8 3326. 89017777,   195280. G521€3     ,   168188.21786071,   1988 54.3636583 ,
        262786. 118.^2 162,   299447 . 88839209,   117.^83.21397872,   196944 . 833 19664,
        139+35- . S14o7239,   1€ 5210. 86449453,   228238.5 5930373,   94138.70949859,
        248377. 39 114329,   23 25 2 2. 80439988,   19o442 . 51447149,   173344 . 9243439 5,
        176073.212.^254S,   146084. 07 o46641,   242039 . 3 1062706,   144658 . 3449 3572,
        11 5106. 14668428,   164041.26742747,   26o912 . 54877247,   12.^153 . 9.^982369,
        156312.22586634,   58026. 8633C672,   32.^360.60986379,   218163 . 83B7 1348,
        15 2259 . *67.^42.^S ,   18945 5. 87647037,   238566.75817475,   213453 . 34542988,
        286183. S 3435609,   343797. 48648858,   113113 . 68282487,   2 54£82. 9277769 ,
        1441G7 . 962BE 903,   152388.29327 1 2,   163453.5885 1096,   90653 . 58654316,
        164648.7€27392G,   37613 5. 94196307,   293968 . 0474833 ,   17.^681 . 52620127 ,
        240855 . 4241166G,   163332. G7.^.^8818,   18oG69.47.^111G1,   97820. 9.^4673 18,
        2413»7.27696 387,   141484. 78249667,   86350. 93215834,   293647 . 97 18466,
        266117. G 2 52S470,   58368. 353014.^ 3,   37359.^ . 51825468,   177342 . 5017959 3,
        217949 . 75045909,   234843.26172602,   163441.233783G4,   372324 . 6078 2761,
        125628. 177.^7 224,   1G1980 . 34758642,   218*63 . 97387601,   102898 . 5876719 1,
        42.^74. 38636831,   7 5238.22184147,   120981. 6165 50G1,   12.°9 30 . 88694G 2 5,
        143234. 0317533 G,   94G39. 19237423,   217331 . 92854144,   53696.92017722,
        138.^54.28729472,   149714. 38014656,   193642 . 4933 3234,   1248 38.71934031,
        169727 . 88 56623 G,   1303 19.23048415,   2522C7 . 60817366,   217916. 3889 303 ,
        112342. 45 532326   265171. 79389115   206046.18.^77374   92911.10781167
```

3. DecisionTreeRegressor:

```
from sk1earnIree import Decision TreeRegressor          T.*';o. *" T§ ?/7* i i Gi°°i.^ '

DT = DecisionTneeRegressor()
DT.fit(x train, y train)

# Predicting the new result

DT pned = DT.predict(x test)
DT p red
```

```
arnay([175000., 173000., 140000., 2B3000., 135900., 155000., 24oZ78.,
        89471., 215000., 205000., 206900., 317000., 120500., 201000.,
       138800., 12P000., 133000., 123000., 281000., 108000.,  98600.,
       202900., 140900., 172500., 235000., 317000., 135000., 140000.,
       115000., 181000., 227000.,  78000., 236500., 194000., 181000.,
       192000., 172500., 16?900., 262280., 133900., 128000., 192000.,
       250000., 141000., 139000.,  P2000., 325624., 176000., 136-00.,
       200100., 222500., 250580., 311872., 306000., 116050., 238300.,
       139000., 154000., 124P00., 120300., 128500., 6116S7., 185000.,
       167500., 226000., 1T>000., 132000., 109500., 224000., 155000.,
       i>a00e., aa5000., >40000., ie80e0., 611657., M3000., 60000.,
       154000., 137900., 437154.,  89471.,  82500., 272000., 121600.,
        94000., 128000., 149000., 125000., 175000., 100000., 268000.,
        85400., 142500., 168000., 115300., 129500., 169000., 142300.,
       383970., 227000., 139090., 249700., 226000., 109P00., 282922.,
       27800a, 12500., 190eo0.,  72Z09., 144000., 205000., 188000.,
       325300., 175500.,  79500., 275000., 176432., 159000., 192300.,
       191000., 140000., 192500., 191000., 169000., 201000., 250000.,
       124500., 171750., 277500., 135000., 23B000., 119500., 202B00.,
       140000., 203900., 120500.,  P3000., 147400., 402000., 192000.,
       133000., 127000., 130000., 148000., 117000., 191000., 127000.,
       237000., 230000., 150500., ;e2000., i32000., 160000., 290300.,
       175000., 103000.,  83500., 135500., 128000., 257000., 230000.,
       211000., 1P1000., 171750., 415298., 325000., 213500., 165Z00.,
       145000., 12P000., 1?2500.,  89471., 132500., 114500., 134500.,
```

4. KNeighborsReg ressor:

```
fnom sklearn.neighbors impont KNeighborsRegressor

KNfJ = Kf'Je1ghborsRegressor (n neighbors = 2)
KNN.fit(x train, y tnain)

  msdirztng the ne' resul?

btlN peed - k FiFl . pr ed i c I (x te    st )
KNN pred
```

```
arnay([4P7S00. ,190637.5, 128509., lB5859. , 108500. , 135250. ,
        332500. , 1129B0., 155430., 196200. , 1B5250. , 2P9875. ,
        147230. , 208000., 309965., 104500. , 144800. ,  956f1l.>,
        2620E0. , 112000. , 103000., 273900. , 151750. , 2543ee. ,
        195250. , 305000. , 142000. , 114259. , 113000. , 196000. ,
        293375. , 123S00. , 225000. , 238250. , 146959.,  222250. ,
        158500. , 142950. , 145000. , l447Z0. , 115000. , 135950. ,
        30108e. , i33s0e. , i442 0. ,  86000. , 307000. , 210000. ,
        143450. , 146250., 2382Z9., 237799. , 2140eB. , i91495. ,
         95000. , 221500., 1252Z0., 155000. , 1450B0. , 117750. ,
        144450. , 431966.B, 205700. , 1€5750. , 234759. , 157475. ,
        1P3125. , 111230. , 21d500., 165500. , 117750.,  331875. ,
        23150B. ,  86000. , 503044.5, 1ld954. , 174700. , l6T975. ,
        174000. , 3B0S00. , 126700. , 126450. , 1BP700.,  153500. ,
         98600. ,  867Z0. , 169230. , 110750. , 118000. , 100600. ,
        i o°re. ,  s:00e. , i552 0. , i50125. , I3t750. , I6l>00. ,
        178750. , 13?4>O., 282875., 2e4725. ,  P369l.3, 261000. ,
        192500. ,  P7200., 230425., 270000. , 190450. , l13P50. ,
         60500. , 145000., 183950., 166550. , 2P35B0. , 205250. ,
        l387E0. , 248946.5, ie9000., 239700. , lB7600. , 228359. ,
        11945B. , 230500.  , 159250. , 114009. , 254038.5, 209800. ,
        127750. , 189000.  , 318980.5, 109500. , 252000., 151125. ,
        198600. , 1382BO.  , 179500. , 1307B0. , 109000. , 145775. ,
        274°5e. , i77216.  , 118504. , 122004. , 124750. , 148250. ,
        109000. , 133475.,  175109., 255759. , 233500. , 145000. ,
        2312Z0. , 1487Z0.,  1499Z0., 192950. , 222250. , l19P50. ,
```

5. RandomForestRegressor:

```
from sklearn.ensemble import RandomForestRegressor
RF = RandomForestRegressor(max_depth=2, random_state=42)

RF.fit(x_train,y_train)

# Predicting the new result

RF_pred = RF.predict(x_test)
RF_pred
```

```
array([152018.811756  , 203612.72279716, 143205.2902323 , 205982.26328349,
       143763.05656626, 209563.64111188, 205502.72175165, 150485.11194221,
       163303.81032812, 209537.30656935, 163303.81032812, 274393.6371864 ,
       148900.23334221, 206605.90343334, 150437.75412853, 128827.01487053,
       129961.13870096, 150081.71498632, 280165.71772777, 130411.89347053,
       130764.11164022, 264053.32885347, 133416.24018354, 164774.45722734,
       207448.12727862, 269602.50783888, 141930.58497411, 132727.20324515,
       162003.80344578, 164774.45722734, 262705.55632178, 129230.41182642,
       263860.1437495 , 210773.39263101, 164774.45722734, 205982.26328349,
       164774.45722734, 152800.42171683, 211870.96106693, 144582.71351743,
       162003.80344578, 144535.35570375, 273214.57011654, 129629.98780979,
       131125.93633315, 128827.01487053, 286416.32538585, 206605.90343334,
       148900.23334221, 204198.72499706, 207448.12727862, 205982.26328349,
       212920.09530163, 280966.7469931 , 150081.71498632, 263860.1437495 ,
       131546.01730096, 130764.11164022, 143763.05656626, 129629.98780979,
       128827.01487053, 333541.59011704, 206605.90343334, 152018.811756  ,
       211397.03278086, 151619.23577264, 205502.72175165, 129646.67182171,
       165790.05581954, 131142.62034507, 129961.13870096, 278634.30287429,
       165790.05581954, 128827.01487053, 327258.96545754, 150884.68792557,
       209724.63907297, 152393.24151634, 145716.83734786, 316248.10545572,
       130811.4694539 , 150485.11194221, 165790.05581954, 129961.13870096,
       128827.01487053, 128827.01487053, 202509.54111547, 130411.89347053,
       131882.54036975, 128827.01487053, 209537.30656935, 130364.53565685,
       134933.06936497, 143112.06661822, 128827.01487053, 141538.62653175,
       165790.05581954, 130008.49651464, 216662.09041445, 262705.55632178,
USING_PRIC130364.53565685, 267496.6856693 , 205982.26328349, 129188.83956347,
```

- Key Metrics for success in solving problem under consideration

Calculating Mean Absolute Error:

```
from sklearn.metrics import mean_absolute_error

print(' Mean Absolute Error for LinearRegression is ', mean_absolute_error(y_test, LR_pred),
      '\n Mean Absolute Error for the Lasso is ', mean_absolute_error(y_test, ls_pred),
     '\n Mean Absolute Error for DecisionTreeRegressor is ', mean_absolute_error(y_test, DT_pred),
     '\n Mean Absolute Error for KNeighborsRegressor is ', mean_absolute_error(y_test, KNN_pred),
      '\n Mean Absolute Error for RandomForestRegressor is ', mean_absolute_error(y_test, RF_pred))
```

```
 Mean Absolute Error for LinearRegression is  22158.142691832993
 Mean Absolute Error for the Lasso is  22154.59984041892
 Mean Absolute Error for DecisionTreeRegressor is  25394.410256410258
 Mean Absolute Error for KNeighborsRegressor is  29492.096153846152
 Mean Absolute Error for RandomForestRegressor is  30094.539950999737
```

- We can see that the Mean Absilute error is least for Lasso (22154.599), so this can be considered as good model.
- Also the Mean Absolute Error for LinearRegression is (22158.14), which is almost equal to the Lasso. So, let's check for Root Mean Squared Error and R2_Score to decide the best model.

Root Mean Square Error:

```
from sklearn import metrics

rmse_LR = np.sqrt(metrics.mean_squared_error(y_test, LR_pred))
rmse_ls = np.sqrt(metrics.mean_squared_error(y_test, ls_pred))
rmse_DT = np.sqrt(metrics.mean_squared_error(y_test, DT_pred))
rmse_KNN = np.sqrt(metrics.mean_squared_error(y_test, KNN_pred))
rmse_RF = np.sqrt(metrics.mean_squared_error(y_test, RF_pred))

print('Root Mean Squared Error for LinearRegression is ', rmse_LR)
print('Root Mean Squared Error for Lasso is ', rmse_ls)
print('Root Mean Squared Error for DecisionTreeRegressor is ', rmse_DT)
print('Root Mean Squared Error for KNeighborsRegressor is ', rmse_KNN)
print('Root Mean Squared Error for RandomForestRegressor is ', rmse_RF)
```

```
Root Mean Squared Error for LinearRegression is  32900.06525455999
Root Mean Squared Error for Lasso is  32896.55457436603
Root Mean Squared Error for DecisionTreeRegressor is  38982.791146512965
Root Mean Squared Error for KNeighborsRegressor is  54358.71132123481
Root Mean Squared Error for RandomForestRegressor is  44644.049167381185
```

- We can see that the root mean square error is minimum for Lasso. So, we can say that Lasso is the best fit model. Let's check r2 score for more accurate decision.

R-Squared:

```
from sklearn.metrics import r2_score

print(' R2_Score for LinearRegression is ', r2_score(y_test, LR_pred),
      '\n R2_Score for the Lasso is', r2_score(y_test, ls_pred),
      '\n R2_Score for DecisionTreeRegressor is ', r2_score(y_test, DT_pred),
      '\n R2_Score for KNeighborsRegressor is ', r2_score(y_test, KNN_pred),
      '\n R2_Score for RandomForestRegressor is ', r2_score(y_test, RF_pred))
```

```
 R2_Score for LinearRegression is  0.8419165475343597
 R2_Score for the Lasso is 0.8419502830805364
 R2_Score for DecisionTreeRegressor is  0.7780583779481173
 R2_Score for KNeighborsRegressor is  0.5684499232056501
 R2_Score for RandomForestRegressor is  0.7089148507145988
```

- - R2_Score closest to 1.0 is considered as best. From the above observations we can see that the best R2_Score is for LinearRegression and Lasso (0.84).
- So, as per our observations we can say that the best fit model for our dataset is Lasso.

- Visualizations

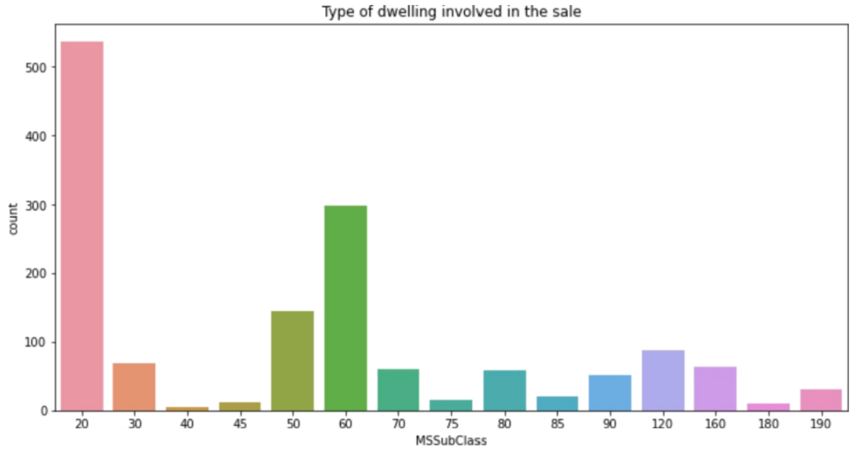
Data Visualization:

```
# Checking for the value counts of column 'MSSubClass'

df['MSSubClass'].value_counts()
```

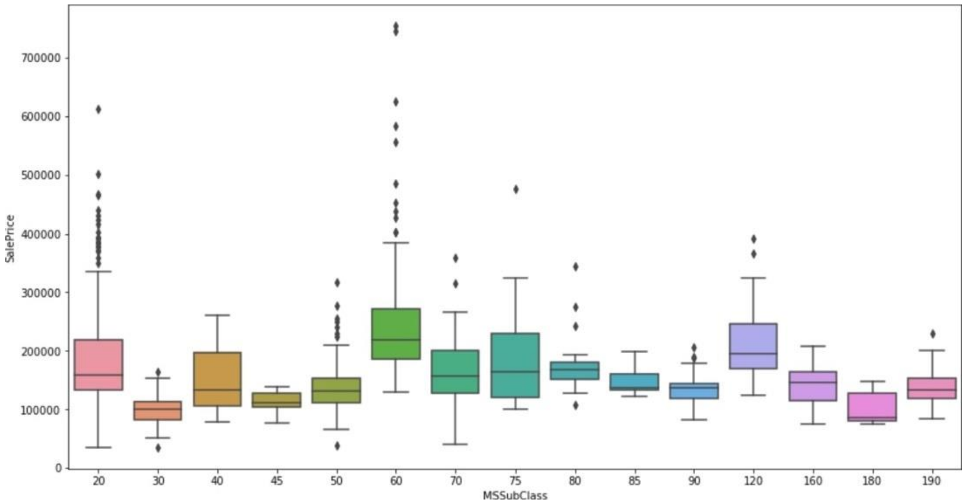
```
20     536
60     299
50     144
120     87
30      69
160     63
70      60
80      58
90      52
190     30
85      20
75      16
45      12
180     10
40       4
Name: MSSubClass, dtype: int64
```

e vi sma £ i z i rig the uat ue coun ts o f- rhe co£ uein 'hlSSubC Los s '

```
plt.figure(figsize• t'*,6])
sns.countplot(df.NSSubClass)
plt.title('Type of dwelling involved in the sale')
plt.show()
```

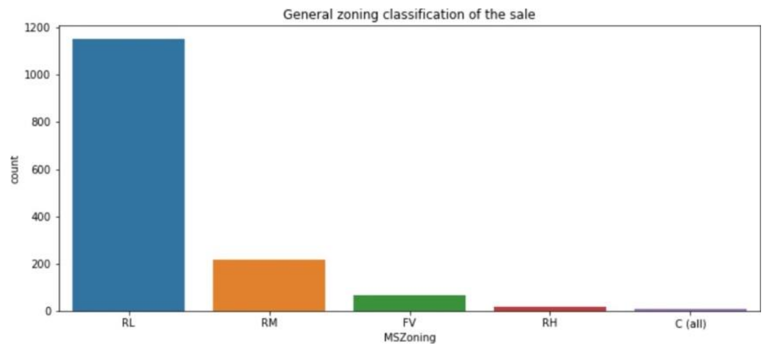


sns b:xpIot(:-:‘ lSSub1 as s,” y- ' Sa1ePr i ce ' , data-d F. sort_va1ues (' Sa1ePri ce” , ascending-Fa1se))



Cfiec/z ing /or tfie ooh ue counts a/ the co £ umn ' *NS2on i ng ' (ident ifi es* tfie gener’ot zoni ng c Iossi/i cotI on O/ the sob e}

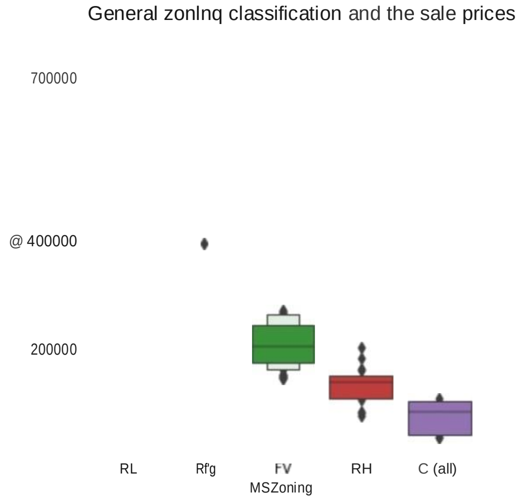
```
plt.figure(figsize=[12,5])
sns . count p1at (df·. MSZon1ng)
pit . t 1tIe ( ' Genera I zoni ng c la ss zf i ca I ion of the s ate ' )
```



- we can see tnat the maximum numDer of generai zoning ciassification of the sale is Residential Low Density (RL) and tne minimum is for the commerciai.

```
# Let 's chec fi the ef-f-eel of- zon i ng c Lass ifi cat i on on the so Le pri ce.

plt.figure(figsize=[12,8])
sns.catplot(x='MSZoning', y='SaleQrice',data=df.sort_values('SalePrice',ascending=Palse), kind='boxen')
plt.title('General zoning classification and the sale prices')
plt.show()
```

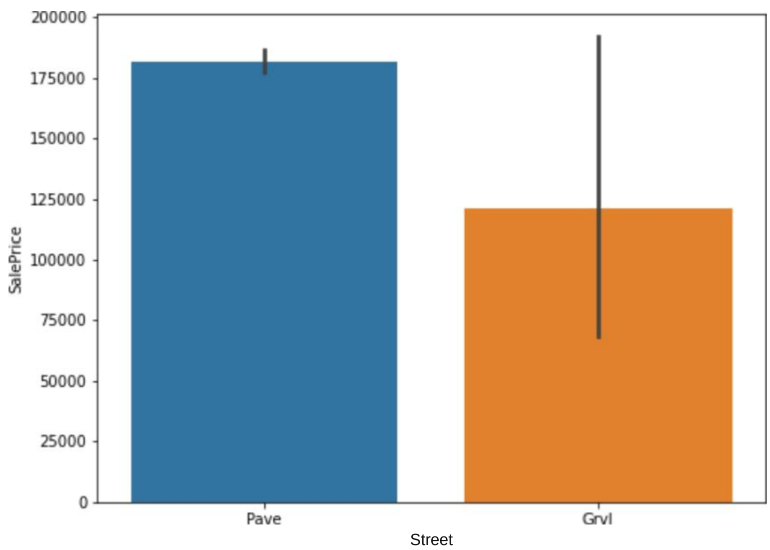
<figure size 864xS76 with B Axes>



Observations:

- For Residentiai Low Density (RL), the maximum prices are ranging between 50,000 to 4,00,000.
- For Floating Village Residential (FV), the maximum prices are ranging between 150000 to 250000.

```
e Check ing for the sat e pri ce on the bas i s o/ road access to the property

plt.figure(figsize- 8,61)
sns.barplot(x•'Street', y•'SalePrice', data    df.sort_values('SalePrice', ascending-Calse))

plt.show()
```



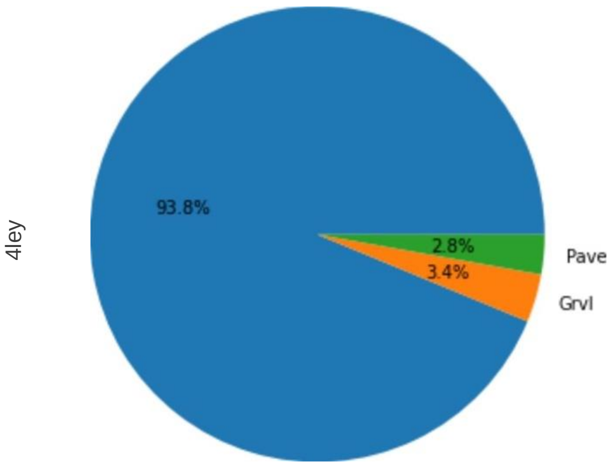
- we can observe that the property with the road access of Pave is in more demand and so its price is aiso high

---

e Let 's chech -or the aL Ley access to property

```
p1t. f1gure(f1gs1ze=[6,6])
df[ A11ey' ] . va1ue_counts( ) . p1ot. p1e(autopct= ' ¥0.1f¥g )
```

‹AxesSubp1ot:y1abe1- A11ey' ›



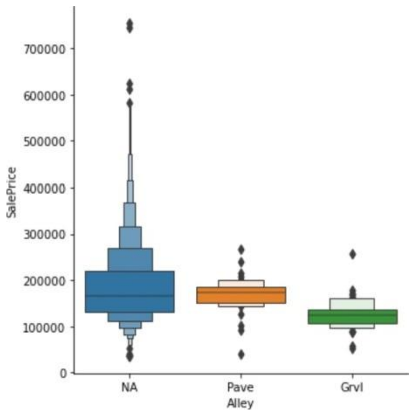
- We can see that approx 94% properly have no alleyaccess.

---

e *I ed 's* c/leek *ttie ef"f-ect* a/ oL key access on the *saLe* price.

```
plt.figure(figsize-a,]
sns.catp1ot(x- 'Al1ey', y- 'Sa1ePrice,'    data-dF. sort_values( ' Sal ePr1ce, '    ascending-False), kind - ' boxen ' )
```

‹Figure size 576x43 2 uito e Axes›



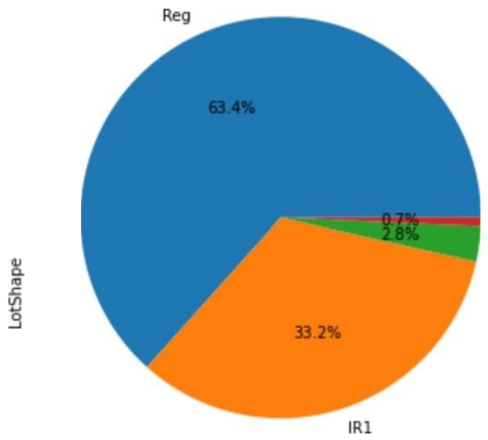
We can observe that the effect or alley access to the properties is very less. So, it is better to remove this column as approx 94% of properties has no aiiey **access.**

---

e Let 's check for the General shape o{ property

```
plt.figure(figsize=(6,6])
df['LotShape'].value_counts().plot.pie(autopct-'%0.1f%%')
```

<AxesSubp1ot:ylabel-'totshape'›



Observations:

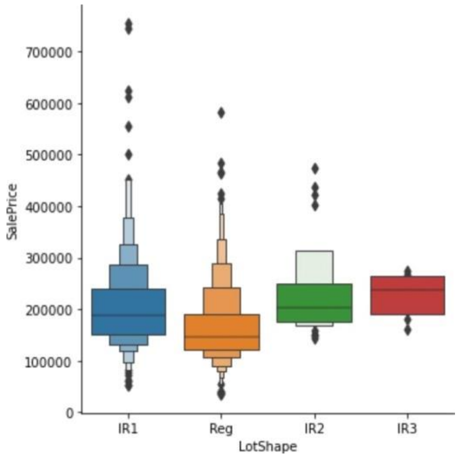
- We can see that most of the properties are of regular shape (approx 63Y«).
- Approx 33°A properties are of slightly irregular shape.

---

e Chech ing the r'eLation of- property shape on the saLe pri ce

```
p1t . f1gure(f1gs1ze• [8, 4] )
sns.catplot(x-'LotShape', y='SalePrice', data-df.sort_values('SalePrlce', ascending=Calse), kind - 'boxen')
plt.show( )
```

‹F1gure s1ze S76x28s u1th e Axes›

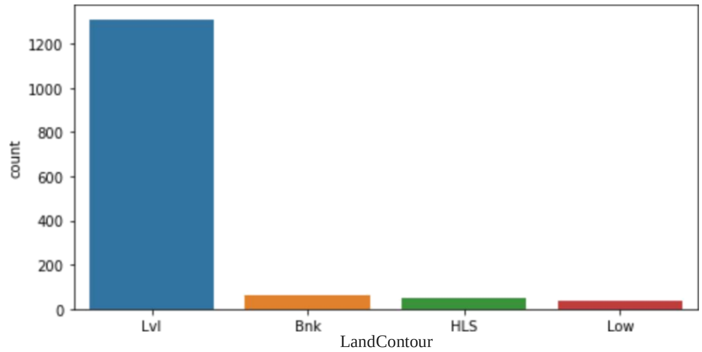


---

e Let's check fi-or the Flatness of- the property

```
pit.figure(figsize-[8,4])
sns.countp1ot(df[ ' LandContour ' ])
```

```
<AxesSubplot:xlabel-'LandContour', ylabel-'count'>
```

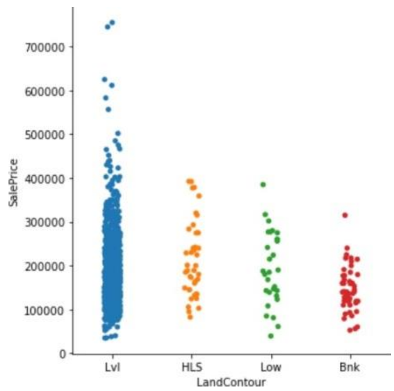


- Most of the properties are of near flat level

£ et 's check /or tfie e-(fee I o/ /Iotners a/ tAe property on the sole pri ce

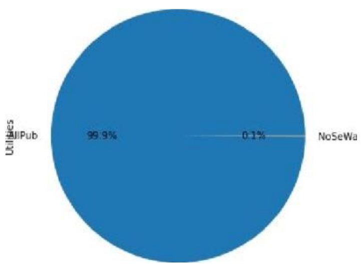
```
plt.flgure(flgslze-[8,4] )
sns.catplot(x- ' £andContour ' , y- 'Sa IePrice ' , data-dT-. sort _values( 'Sa lePrice ', ascending•False) )
```

```
‹ seaborn. axi sgrid. FacetGrid at 6xlB9ce7a873B›
<Figure size 576x288 with 0 Axes>
```



 # Let's check for the type of utilities available in the property

```
df-[ ' Ut i1 its es ' ] . vaLue counts ( ) . pLot . pie (autopct= ' @. yf-@ ' )
```

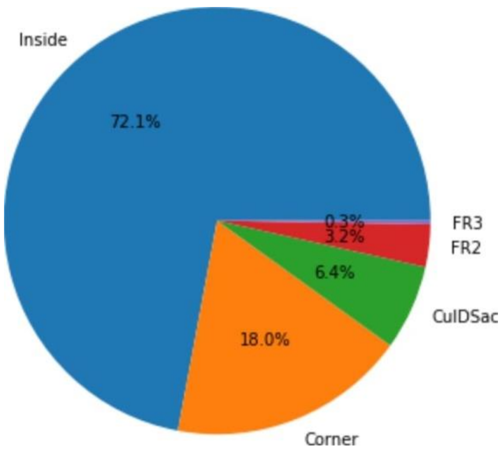


- We can see that approx 100s6 properties have all public utilities (E,G,W,& S) Bo, we can drop this column as this will not contribute to the dataset in the model building.

Let 's checé for the L of conf-i gurat ion

```
plt.figure(figsize=[6,6])
df['LotConfig'].value_counts().plot.pie(autopct='%0.1f%%')
```

```
<AxesSubplot:ylabel='LotConfig' >
```



Approx 72% properties have inside lot configuration
- 18% properties have corner lot.

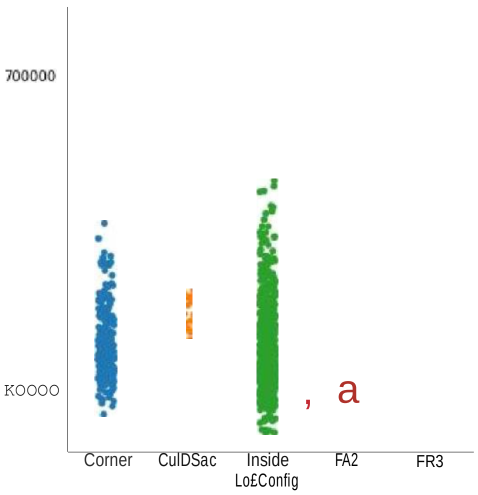
Only 0.3% properties have frontage on 3 sides of property.

Che cb! ng /or Oh e Lot cont £i ur ation and i ts e//ect on the s aL e p r I c tng.

```
pit.figure(figsize=[6,4])
sns.catplot(x='LotConfig', y='SalePrice', data=df.sort_values('SalePrice', ascending=FaIse)}
```

```
<seaborn.axisgrid.Facet£rid at 0x18Pce3c6640>
```

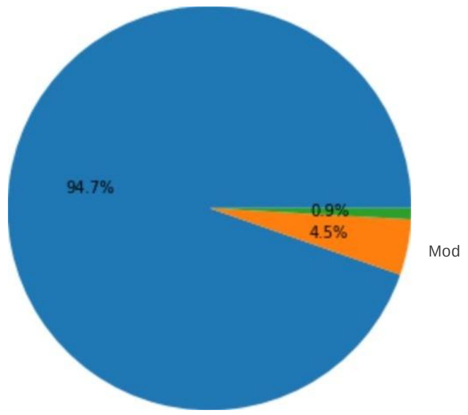
```
<Figure size 43Zx2g8 with B Axes>
```



L e l 's Chec/t /or' the s L ope o{ the pr'oper'ty

```
plt.figure(figsize=[6,6])
df[' LandSlope'].value_counts().plot.pie(autopct='%0.1f%%')
```

```
<AxesSubplot:ylabel='LandSlope'>
```

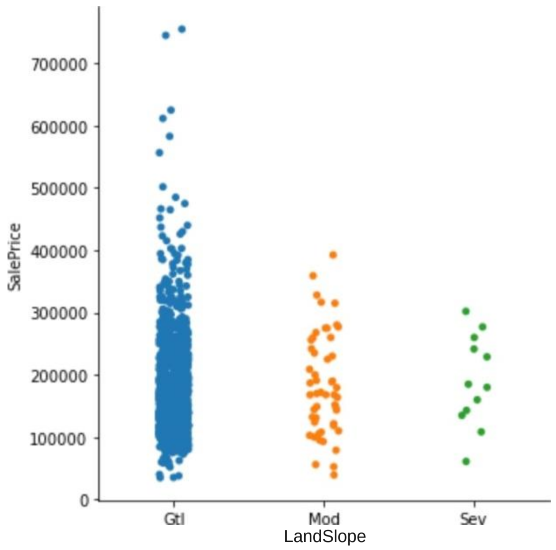


- Approx 95% properties having gentle slope.
- Only approx 1% properties having severe Slope and 4.5% properites having moderate slope.

Cheeck i ng for• the s L ope iii s e s aLe pr•i c i ng of the pnopert i es

```
plt.figure(figsize=[8,61)
sns.catplot(x='LandSlope', y='SalePrice', data=df.sort_values('SalePrice', ascending=False))
```

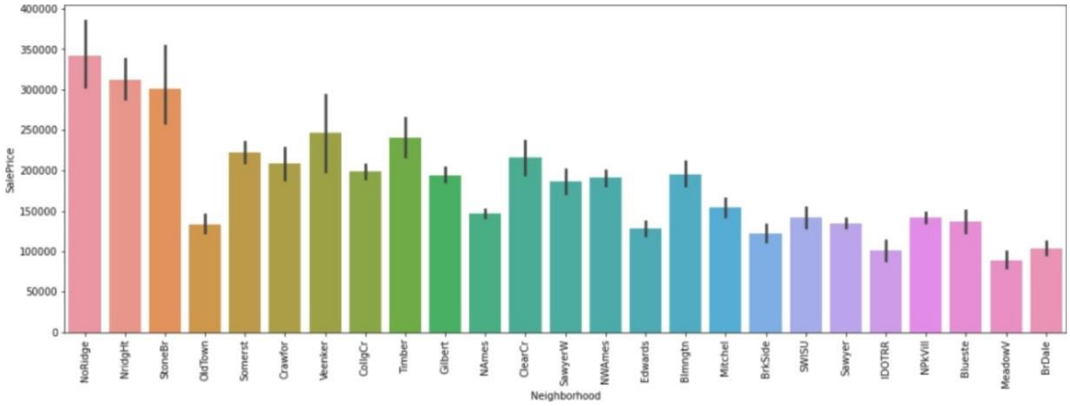
```
<seaborn.axisgrid.FacetGrid at 0x189ce282cl0>
<Figure size S76x432 with 0 Axes>
```



- The maximum gentle slope type properties having the sale price ranging between 100000 to 300000

Cneck ing /or the e f-f-e c L o/ ph'y si co I £ocptions ivi th in Aries ci ty I irii Is on me sa I e pri ce

```
plt . ii gure(I-igsi ze- [18, 6] )
sns . barplot (x- Net ghborhood ' , y- 'Sa\ ePr i ce ' , data• df. sort_va lues( ' Sa IePr i ce ' , ascending-False) )
pLt . xt l cks (rotat1on =9B)
pit.ShOw()
```




```
#Cheele ing /oz' I/ie volue counts a/ I:ype a/ dwef Ling

df[ ' B I dgType ' ] . va1ue_count ( )
```

```
1Fam      *220
TwnhsE     1t4
Dupl ex     52
Twchs       43
2fmCon      31
Name: BldgType, dtype: intd4
```

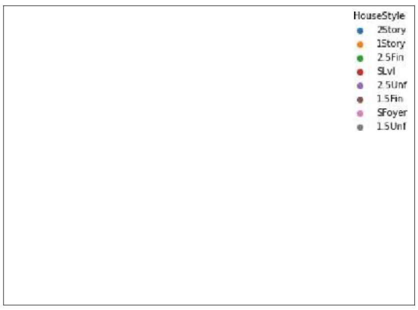
- Single-family Detached dewlling is most popular
- *wo-family Conversion. originally built as one-family dwelling is least popular

```
# Che ed:Eng /or the voLue counts a/ I:the sky Le of dwe l Ling

df['HOuseStyle’].value_counts()
```

```
15tory    72g
2Story    dd5
1.5Fin    154
SLvl       65
5Foyer     37
1. 5Unf    14
2. 5Unf    11

|Name: HouseStyle, dtype: int64
```

- One story style of houses are most popular.
- Two ano one-haF story: 2nd level finished style of house is least popular.

```
sns. scattenp Not (x= B1 dgType ' y= ' a ePrTce ' , hue = Hou e Qtyl e ' , data = df- . sort values( ' la T ePrTc e ' ascendTng=Fa\se$ }
```



Ch ecfi: ng jar th e uo L ue coun ts o/ th e Rot es tn e ave ro L L fno te r i aL an d {in i sh a{ the hous e

```
df['OverallQual'].value_counts()
```

```
S      397
6      374
7      319
8      168
4      116
9       43

i0      i8
2        3
i        Z
Name: OvenallQual, dtype: int64
```

- Most of the houses are rated 5 whicn means the overall material and finish of the houses are average an0 above average.
- Very few houses was rated 1 which says the overall matenal and finisft of very few houses are very poor.

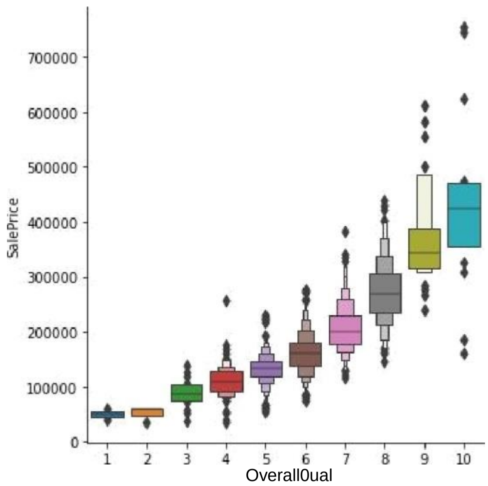
Checfi: I ng f-or the uaL ue counts of the rot es tne o'veraL L c ond! IN on of- the house

```
df['OvenallCond'].value_counts()
```

```
S     821
6     252
7     265
8      72
4      57
3      2S
9      22
2       S
1       1
Name : Over a1tC ond , dtype : int64
```

- fdost of the houses are rated average and above average for the overall condition of the house.
- None of the houses got the ratings of very excellent.

```
<Figure size 576x432 with 0 Axes>
```



Notations:

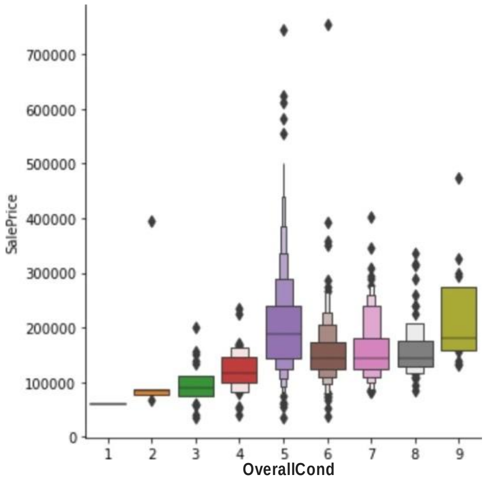
- 1: Very Poor
- 2: Pa0r
- 3: Fair
- 4: Below Average
- 5: Average
- 6: Above Average
- 7: Good
- 8: Very Good
- 9: Excellent
- 10: Very Excellent

- We can see that as the ratings are increasing the price of the property is also increasing.

Let 's check f-or the sat epr i ces based on the ra t i ng s of overa I I condi t i on o f the house

```
p1t. -1gure( f 1gs1ze- [8, 6] )
sns . catp1ot(x= ' over a11Cond ' , y= ' Sa1 ePr1ce ' , data = df. sort_va1ues ( ' Sa1ePr i ce ' , ascend1ng=Fa1se) , k1nd = ' boxen ' )

p1t.show()
```

‹F1gure s1ze 576x43 2 w1th 6 Axes›



- We can see that the price of the house is highest for the house which got 9 ratings (Excellent)

1et 's checfi' for the vat ue counts of type ofi- roof of the losses

```
df['POofStyle'].value_counts()
```

```
Gable      1141
H1p         286
F1at         13
Gambrel      11
Stans and     7
Shed          2
Name: RoofStyle, dtype: int64
```

Maximum houses having Gable ape oT rooT.

Check ! ng f-or the 'va Lue count y the ifat er! aL used yer the roof .

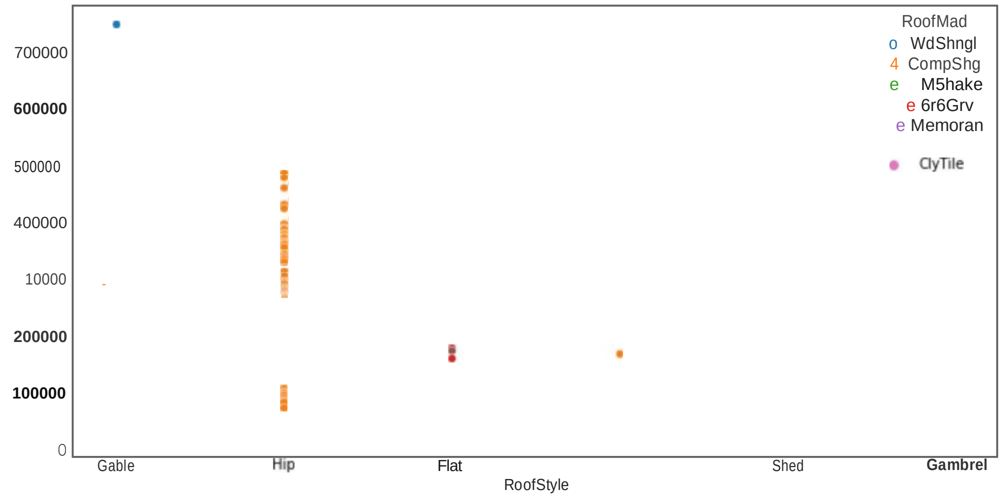
```
df['ROofMatl'].value_coonts()
```

```
CompShg    1434
Tar&Grv      11
WdShngl       6
WdShake       5
ClyTile       I
Roll          I
Metal         I
Membran       1
Name: RoofMatl, dtype: int64
```

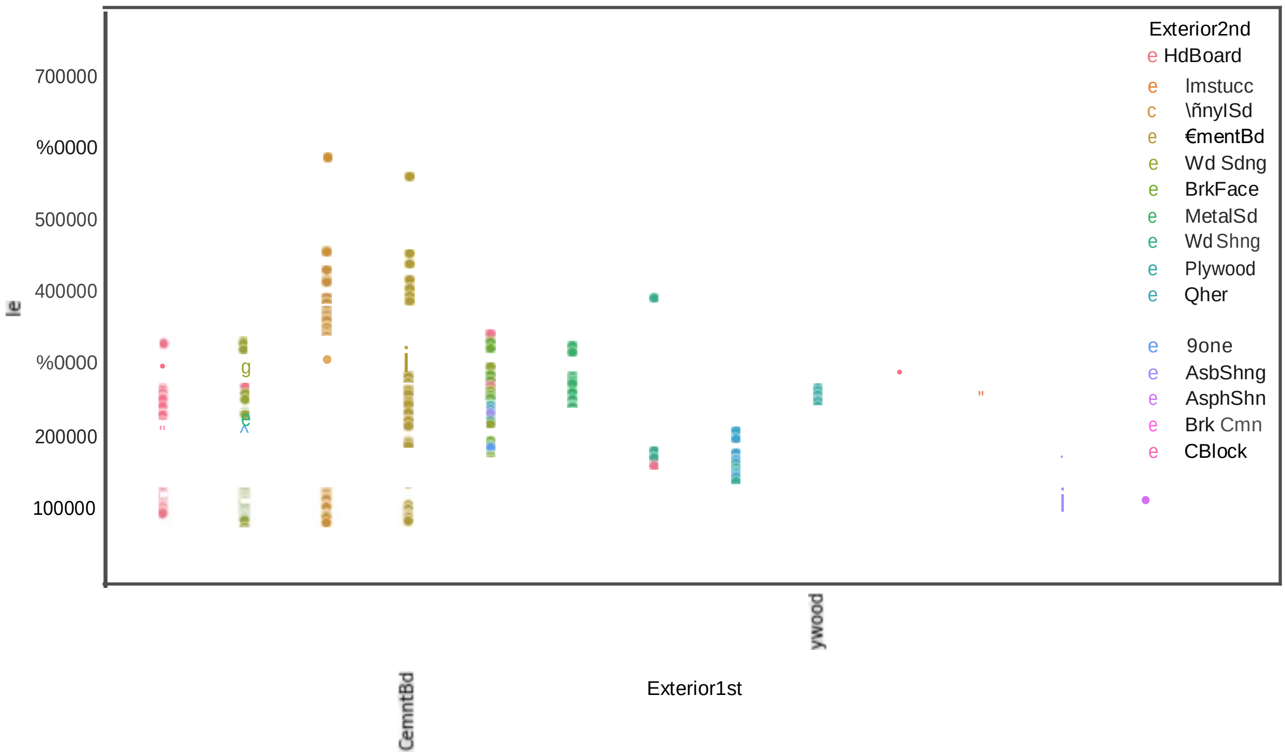
Maximum houses having the roof which is made up of Standard (Composite) Shingle

4 Let 's check for• the ef-fi-ect ofi- roof on the saLe pri ce

```
plt.figure(figsize=[12,61)
sts.scatterplot(x-'RoofStyle', y-'SalePrice', hue - 'RoofMatl', data = df.sort_values('SalePrice', ascending-False))

plt.show()
```



- We can see that the most of the roof are make up of Standard (Composite) Shingle.
- The highest price of the house having Gable roof type and the material of the roof is Wood Shingles



é £ e t 's cliec k f-o n t li.• uc I ue ounts o f- t he .wason ry '.'e neer t y pe

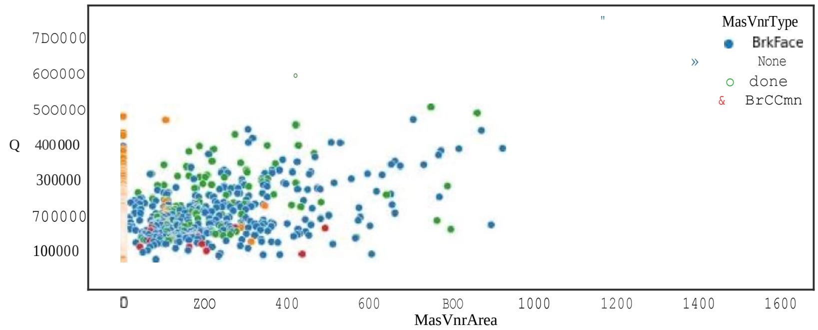
```
df['lesVnrTyp°'I. value_counts()
```

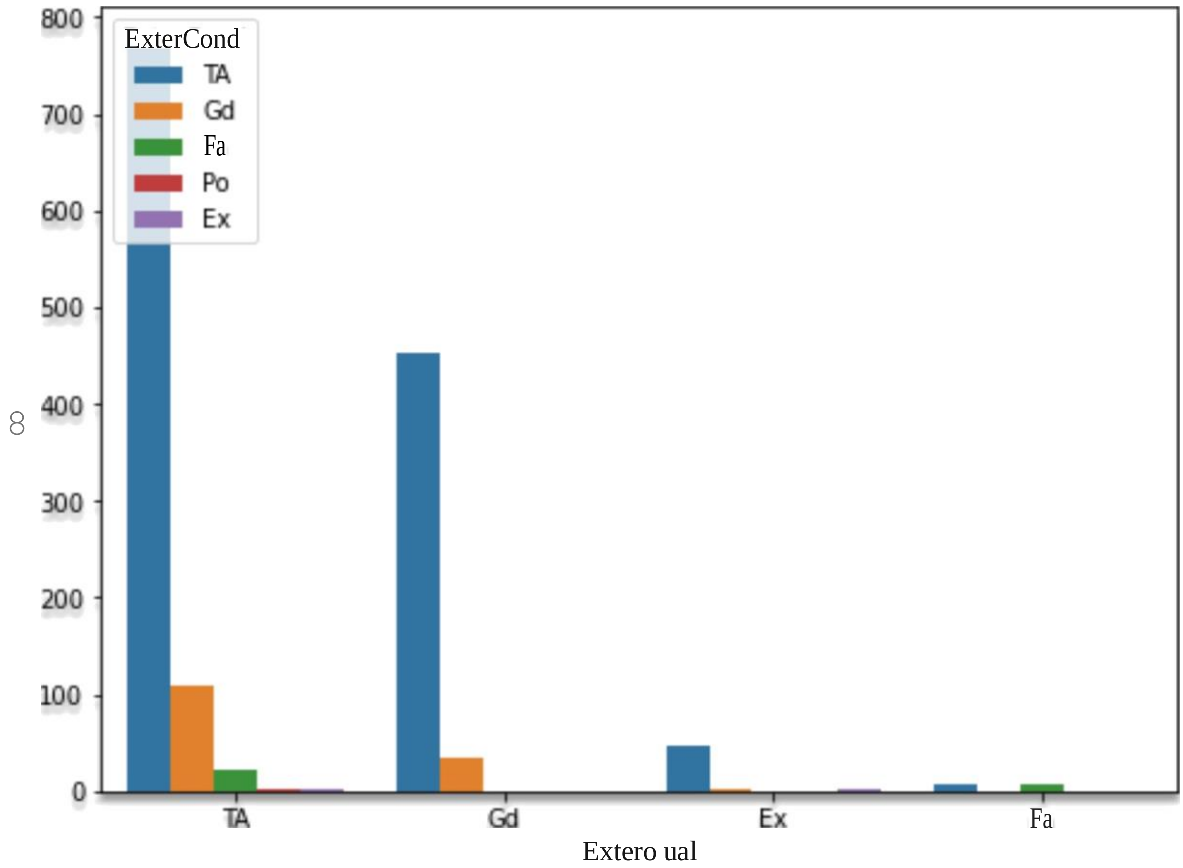
```
None        872
BrkFace     445
Stone       1Z8
BrkCmn       1S
Name: MasVnrType, dtype: int64
```

- fdost of the houses have no masonry veneer.

```
# Let's check for the sale price based on the masonry veneer

plt.figure(figsize=[10,4])
sns.scatterplot(x=':lasVi;n%nea¦   y='SalePnice', hue = '1:asVnrType', data = df.sort_values('SaIePrice', ascending=False))
```

```
<AxesSubplot:xlabel='MasVnnArea', ylabel='SalePrice'>
```

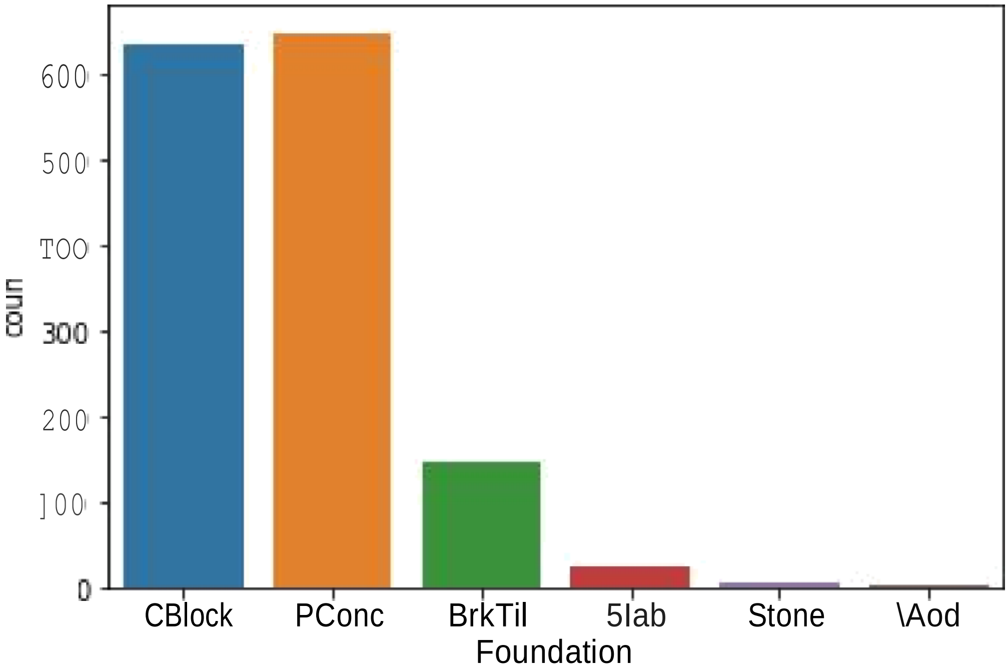


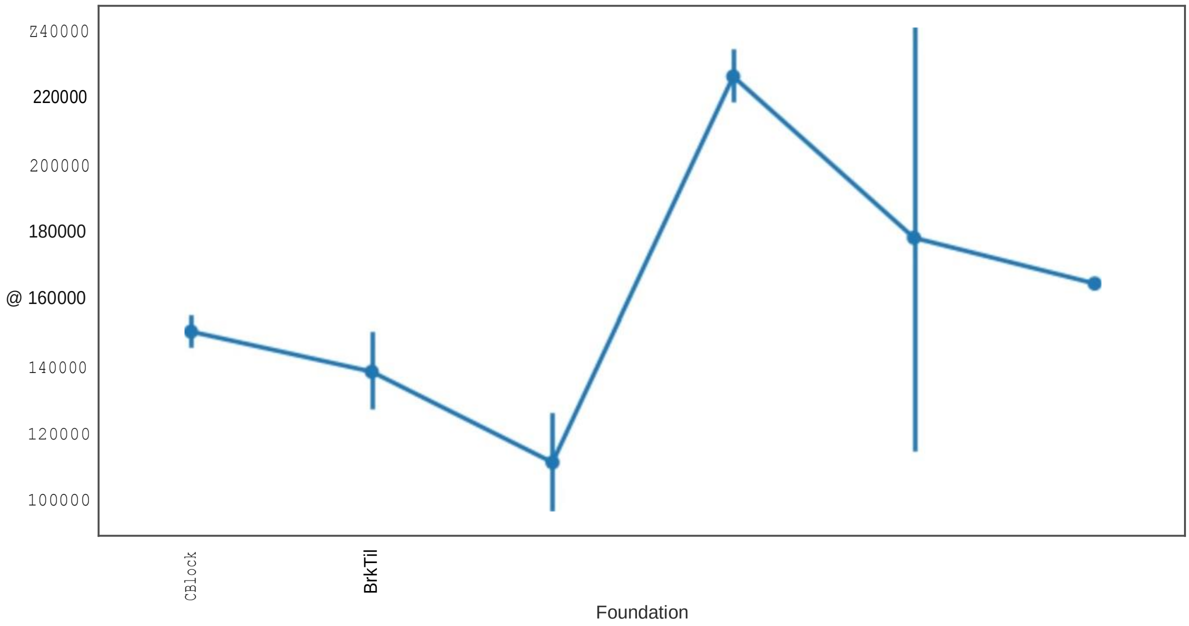


Notations:

- Ex Excellent
- Gd Good
- TAAverage/Tvpical
- la Fair
- Po Poor

I lost of the houses are of average.'typical quality' of the material on the exterior.
None houses have poor quality of material on the exterior.





Let 's check {on the central ai n condi ti on i ng

```
df('CentralAir'l. value_counts()
```

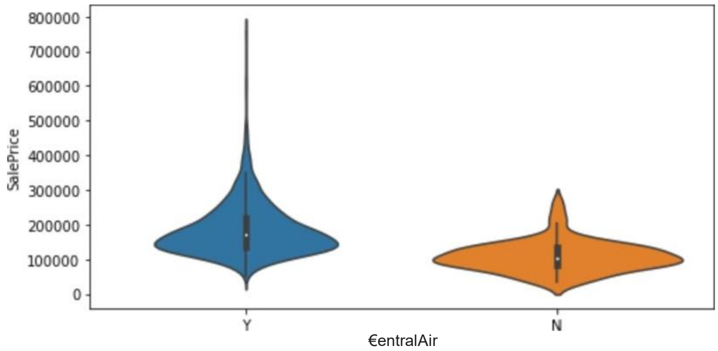
```
Y    1365
N      95
Name: CentralAir, dtype: int64
```

- Most of the nouses having central airconditioning

Check 1 ng f-on the pri ce of- the houses on the bas i s of- a I r' condi t i on1 ng

```
plt.figure(figsize=[8,4])
sns.violinplot(x='CentralAir', y-'SalePrice', data=df.sort_values('SalePrice', ascending-False))
```

p1t. shoal( )



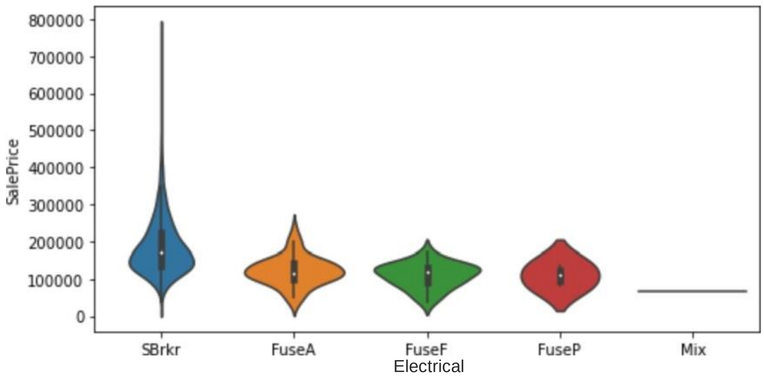
- Houses naving the option of central air conditioning have more price.

Let 's chec k for the eLec tr i caL system of the house

plt . I igure(figsi ze= [8, 4])

```
sns.violinplot(x='Electrical', y='SalePrice', data=df.sort_values('SalePrice', ascending=False))
```

```
plt.show()
```



Notation:

- SBrkr Standard Circuit Breakers & Romex
- FuseA Fuse Box over 60 AMP and all Romex wiring (Average)
- FuseF 60 AMP Fuse Box and mostly Romex wiring (Fair)
- FuseP 60 AMP Fuse Box and mostly knob & tube wiring (poor)
- Mix Mixed
  - Most of the houses are having the electrical system of standard circuit breakers and romex.

P Let 's *check vat ue count jar the home funcTionaL i by (Ass use typecaL un Less deductfi one* ore ivorr•onted}

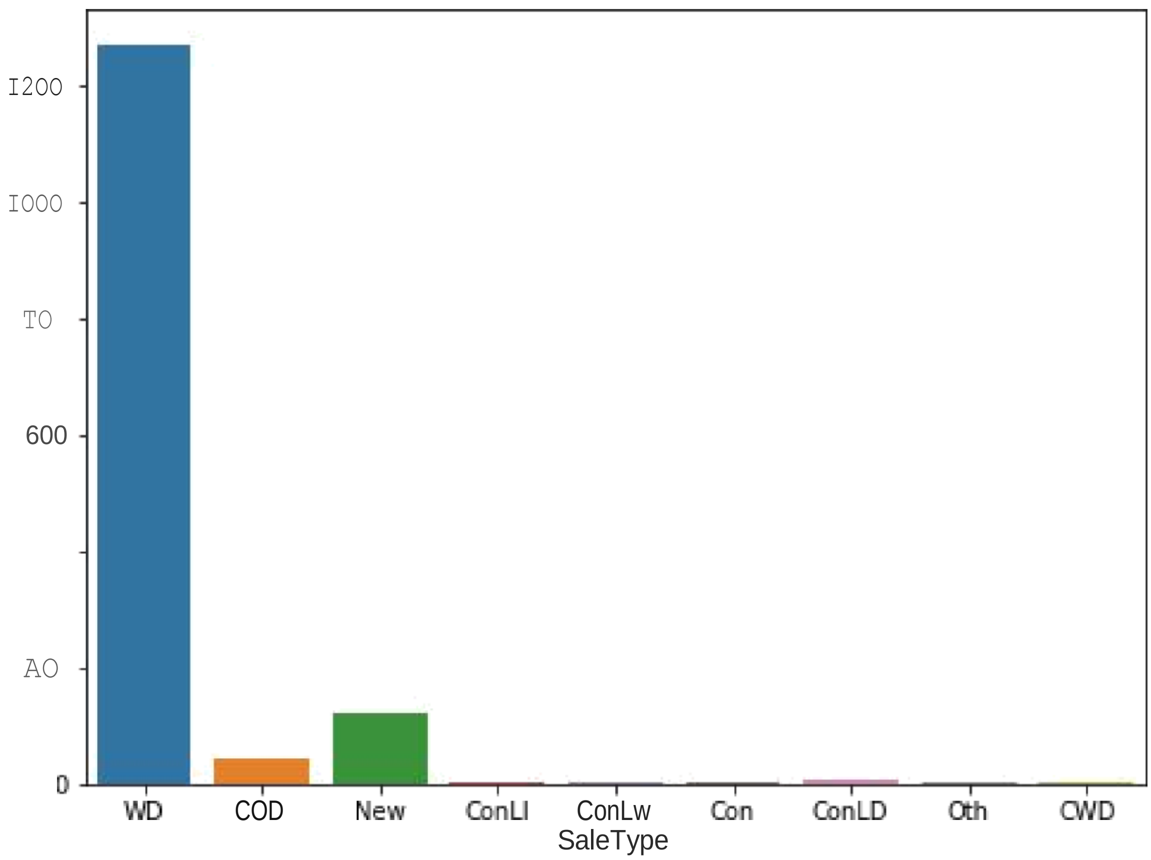
```
df['Functional'].value_counts(j
```

Typ	136B
M1n2	34
M1n1	31
Nod	15
Maj1	14
Maj2	5
Sev	1

Name: Funct1ona1, dtype: 1nt64

Notations:

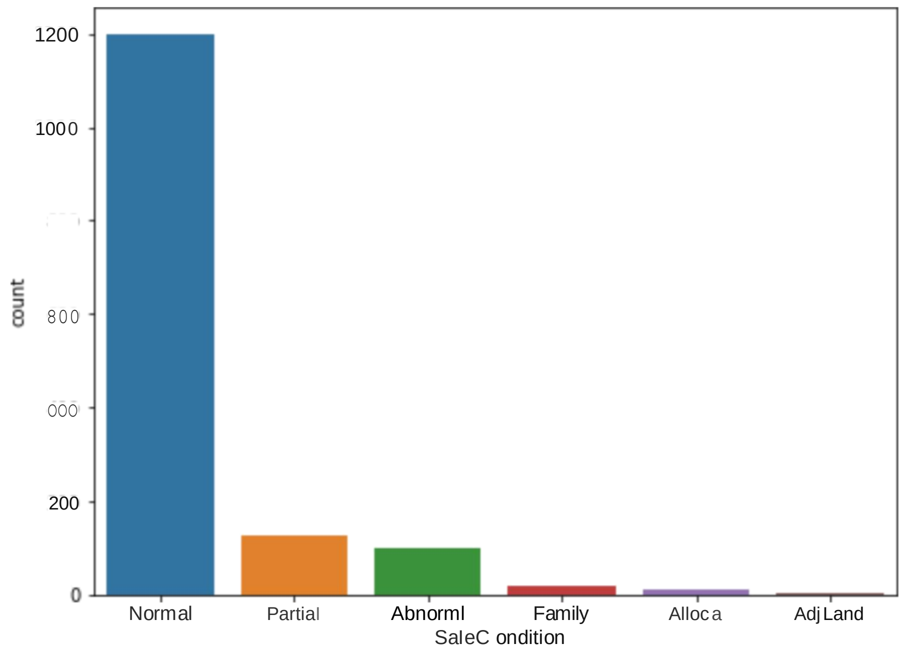
- Typ Typical Functionality
- Mind Minor Deductions 1
- Min2 Minor Deductions 2
- Mad Moderate Deductions
- Maj1 Major Deductions 1
- Maj2 Major Deducaons 2
- Sev Severely Damaged
- Sal Salvage only
    - Maximum horrle have apical knctionality.



Notation:

- .VD Warranty‘ Deed - Conventional
- C'.YD \’farranty Deed - CaSh
- VV7D "Warranty Deed - VA Loan
- Ne 1' Home just constructed and scld
- COD Ccurt Otcer Deed/Estate
- Cen Contract 15% Do an payment regular terms
- ConL'r, Contract Low Do yn payment and Io a interest
- CcnLI Contract Lav.‘ Interest
- ConLD Contract Low Do yn
- Oth Other
  - t›fast of the sale type are V7arranty Deed - Conventional.

```
plt.figure(figsize-[8,‹:])
sns.countplot(df['SaleCcncition'])
pit.show()
```



Notation.

- Normal Normal Sale
  Abnorml Abnormal Sale - trade, foreclosure. shon sale
- AdjLand Adjoining Land Purchase
- Alloca Allocation - Mao linked properties ›.'.ith separate deeds. typicall}’ condo y,ith a garage unit
- Famil}’ Sale bet •.een family’ members
  Partial Home .'. as not completed •. hen last assessed tassociated '‹’ith Ne.'. Homes)
  - I.lost of the sale are normal sale

CONCLUSION

- Key Findings and Conclusions of the Study

    ✓ MS Sub Class seems to have the biggest impact on House Prices, followed by Basement Full Bath and Basement Half Bath.

    ✓ Other than the Basement related features, Condition 2, Exterior Quality and Lot Area are some of the other important features.