HOUSING: PRICE PREDICTION

Prepared by: Maahi Gurnani, Data Science Intern at Flip Robo Technologies

ACKNOWLEDGEMENT

It is my gratification to present this report. I, would like to thanks FlipRobo Technologies and my batch’s SME Khushboo Garg Mam for providing us this dataset and giving us chance to explore such a wide dataset.

Working on this dataset gave many insights and information about the factors that people should or do consider while buying any new property and how the prices are affected with numerous conditions.

Introduction

- **Problem Statement:**

Houses are one of the necessary needs 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.

The company is looking at prospective properties to buy houses to enter the market. We 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?

- **Conceptual Background of the Domain Problem:**

Thousands of houses are sold every day. There are some questions every buyer asks himself like: What is the actual price that this house deserves? Am I paying a fair price? In this report , a machine learning model is proposed to predict a house price based on data related to the house (its size, the year it was built in, etc.).

During the development and evaluation of our model, we will show the code used for each step followed by its output. This will facilitate the reproducibility of our work. In this study, Python programming language with a number of Python packages will be used.

- **Business Goal:**

We are required 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.

- **Motivation for the Problem Undertaken:**

Growing unaffordability of housing has become one of the major challenge for countries around the world. In order to gain a better understanding of the commercialized housing market we are currently facing , we want to figure out what are the top influential factors of the housing price. Apart from the more obvious driving forces such as the inflation and the scarcity of land, there are also a number of variable that are worth looking into. Therefore , we choose to study the house price prediction., which enables us to dig into the variables in depth and to provide a model that could more accurately estimate house prices. In this way, people could make better decision when it comes to home investment.

Our objective is to discuss the major factors that affect housing price and make precise prediction for it. We use 80 explanatory variables including almost every aspect of residential homes in Australia. Methods of both statistical, regression models and machine learning models are applied and further compared according to their performance to better estimate the final price of each house. The model provides price prediction based on similar comparable of people's dream house, which allow both buyers and sellers to better negotiate home prices according to market trend

- **Technical Requirements:**

• Data contains 1460 entries each having 81 variables.
• Data contains Null values. You 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. You need to handle them accordingly.
• You have to build Machine Learning models, apply regularization and determine the optimal values of Hyper Parameters.
• You need to find important features which affect the price positively or negatively.

Analytical Problem Framing

- **Mathematical/ Analytical Modelling of the Problem**

Regression Model

Regression models are often used to determine which independent variables hold the most influence overdependent variables information that can be leveraged to make essential decision.

The given problem is a Regression problem, where our end goal is to predict the Prices of House based on given data. I will be dividing my data 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 the model has performed.

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:

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

The most traditional regression models are:

1)Linear Regression with Lasso and Ridge
2)Decision Tree Regression,
3)Random Forest regression
4)AdaBoost Regression
5)Support Vector Regression
6)K Neighbors Regression
7)Gradient Boosting Regression

<u>The ‘r2’ score will be used to determine the best model amongst the above mentioned</u>

Hardware and Software Requirements and Tools Used

- **Languages Used:** Python
- **Platform Used:** Jupyter Notebook
- **Libraries and Metrics used:**

Following are the libraries and metrics used to start the Regression Model

```
In [1]: #importing the necessary libraries
        import numpy as np
        import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        import warnings
        warnings.filterwarnings('ignore')
        import sklearn
        from sklearn.linear_model import LinearRegression
        from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
        from sklearn.model_selection import train_test_split
```

Data Sources and their formats

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. This dataset is been provided in the CSV file to us.
Dataset contains 1460 entries each having 81 variables.

Including the snapshot of the data set provided and loaded

In [2]:
```
df_housing=pd.read_csv("house_train.csv")
df_housing
```

Out[2]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal
0	127	120	RL	NaN	4928	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0
1	889	20	RL	95.0	15865	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0
2	793	60	RL	92.0	9920	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0
3	110	20	RL	105.0	11751	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0
4	422	20	RL	NaN	16635	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1163	289	20	RL	NaN	9819	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0
1164	554	20	RL	67.0	8777	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0
1165	196	160	RL	24.0	2280	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0
1166	31	70	C (all)	50.0	8500	Pave	Pave	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0
1167	617	60	RL	NaN	7861	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0

1168 rows × 81 columns

Note: We even uploaded test dataset provided and performed all the EDA steps on it too

Data Inputs- Logic- Output Relationships

There are total 81 columns, of which 80 are input variables and 1, the output variable(‘Sale Price’). Below are the details of all the input variables and their entries.

MSSubClass: Identifies the type of dwelling involved in the sale.

20	1-Story 1946 & Newer All Styles
30	1-Story 1945 & Older
40	1-Story W/Finished Attic All Ages
45	1-1/2 Story - Unfinished All Ages
50	1-1/2 Story Finished All Ages
60	2-Story 1946 & Newer
70	2-Story 1945 & Older
75	2-1/2 Story All Ages
80	Split Or Multi-Level
85	Split Foyer
90	Duplex - All Styles And Ages

120	1-Story Pud (Planned Unit Development) - 1946 & Newer
150	1-1/2 Story Pud - All Ages
160	2-Story Pud - 1946 & Newer
180	Pud - Multilevel - Incl Split Lev/Foyer
190	2 Family Conversion - All Styles And Ages

MSZoning: Identifies the general zoning classification of the sale.

A	Agriculture
C	Commercial
FV	Floating Village Residential
I	Industrial
RH	Residential High Density
RL	Residential Low Density
RP	Residential Low Density Park
RM	Residential Medium Density

LotFrontage: Linear feet of street connected to property

LotArea: Lot size in square feet

Street: Type of road access to property

Grvl Gravel , Pave Paved

Alley: Type of alley access to property

Grvl Gravel, Pave Paved, NA No alley access

LotShape: General shape of property

Reg	Regular
IR1	Slightly irregular
IR2	Moderately Irregular
IR3	Irregular

LandContour: Flatness of the property

Lvl	Near Flat/Level
Bnk	Banked - Quick and significant rise from street grade to building
HLS	Hillside - Significant slope from side to side
Low	Depression

Utilities: Type of utilities available

AllPub	All public Utilities (E,G,W,& S)
NoSewr	Electricity, Gas, and Water (Septic Tank)
NoSeWa	Electricity and Gas Only
ELO	Electricity only

LotConfig: Lot configuration

Inside	Inside lot
Corner	Corner lot
CulDSac	Cul-de-sac
FR2	Frontage on 2 sides of property
FR3	Frontage on 3 sides of property

LandSlope: Slope of property

Gtl	Gentle slope,	Mod	Moderate Slope	Sev	Severe Slope

Neighborhood: Physical locations within Ames city limits

Blmngtn	Bloomington Heights
Blueste	Bluestem
BrDale	Briardale
BrkSide	Brookside
ClearCr	Clear Creek
CollgCr	College Creek
Crawfor	Crawford
Edwards	Edwards
Gilbert	Gilbert
IDOTRR	Iowa DOT and Rail Road
MeadowV	Meadow Village
Mitchel	Mitchell
Names	North Ames
NoRidge	Northridge
NPkVill	Northpark Villa
NridgHt	Northridge Heights
NWAmes	Northwest Ames
OldTown	Old Town
SWISU	South & West of Iowa State University
Sawyer	Sawyer
SawyerW	Sawyer West
Somerst	Somerset
StoneBr	Stone Brook
Timber	Timberland
Veenker	Veenker

Condition1: Proximity to various conditions

Artery	Adjacent to arterial street
Feedr	Adjacent to feeder street
Norm	Normal
RRNn	Within 200' of North-South Railroad
RRAn	Adjacent to North-South Railroad
PosN	Near positive off-site feature--park, greenbelt, etc.
PosA	Adjacent to postive off-site feature
RRNe	Within 200' of East-West Railroad
RRAe	Adjacent to East-West Railroad

Condition2: Proximity to various conditions (if more than one is present)

Artery	Adjacent to arterial street
Feedr	Adjacent to feeder street
Norm	Normal
RRNn	Within 200' of North-South Railroad
RRAn	Adjacent to North-South Railroad
PosN	Near positive off-site feature--park, greenbelt, etc.
PosA	Adjacent to postive off-site feature
RRNe	Within 200' of East-West Railroad
RRAe	Adjacent to East-West Railroad

BldgType: Type of dwelling

1Fam	Single-family Detached
2FmCon	Two-family Conversion; originally built as one-family dwelling
Duplx	Duplex
TwnhsE	Townhouse End Unit
TwnhsI	Townhouse Inside Unit

HouseStyle: Style of dwelling

1Story	One story
1.5Fin	One and one-half story: 2nd level finished
1.5Unf	One and one-half story: 2nd level unfinished
2Story	Two story
2.5Fin	Two and one-half story: 2nd level finished
2.5Unf	Two and one-half story: 2nd level unfinished
SFoyer	Split Foyer
SLvl	Split Level

OverallQual: Rates the overall material and finish of the house

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

OverallCond: Rates the overall condition of the house

10	Very Excellent
9	Excellent
8	Very Good
7	Good
6	Above Average
5	Average
4	Below Average

3 Fair
2 Poor
1 Very Poor

YearBuilt: Original construction date

YearRemodAdd: Remodel date (same as construction date if no remodeling or additions)

RoofStyle: Type of roof

Flat	Flat
Gable	Gable
Gambrel	Gabrel (Barn)
Hip	Hip
Mansard	Mansard
Shed	Shed

RoofMatl: Roof material

ClyTile	Clay or Tile
CompShg	Standard (Composite) Shingle
Membran	Membrane
Metal	Metal
Roll	Roll
Tar&Grv	Gravel & Tar
WdShake	Wood Shakes
WdShngl	Wood Shingles

Exterior1st: Exterior covering on house

AsbShng	Asbestos Shingles
AsphShn	Asphalt Shingles
BrkComm	Brick Common
BrkFace	Brick Face
CBlock	Cinder Block
CemntBd	Cement Board
HdBoard	Hard Board
ImStucc	Imitation Stucco
MetalSd	Metal Siding
Other	Other
Plywood	Plywood
PreCast	PreCast
Stone	Stone
Stucco	Stucco
VinylSd	Vinyl Siding
Wd Sdng	Wood Siding
WdShing	Wood Shingles

Exterior2nd: Exterior covering on house (if more than one material)

AsbShng	Asbestos Shingles
AsphShn	Asphalt Shingles

BrkComm	Brick Common
BrkFace	Brick Face
CBlock	Cinder Block
CemntBd	Cement Board
HdBoard	Hard Board
ImStucc	Imitation Stucco
MetalSd	Metal Siding
Other	Other
Plywood	Plywood
PreCast	PreCast
Stone	Stone
Stucco	Stucco
VinylSd	Vinyl Siding
Wd Sdng	Wood Siding
WdShing	Wood Shingles

MasVnrType: Masonry veneer type

BrkCmn	Brick Common
BrkFace	Brick Face
CBlock	Cinder Block
None	None
Stone	Stone

MasVnrArea: Masonry veneer area in square feet

ExterQual: Evaluates the quality of the material on the exterior

Ex	Excellent	Gd	Good	TA	Average/Typical
Fa	Fair	Po	Poor		

ExterCond: Evaluates the present condition of the material on the exterior

Ex	Excellent	Gd	Good	TA	Average/Typical
Fa	Fair	Po	Poor		

Foundation: Type of foundation

BrkTil	Brick & Tile
CBlock	Cinder Block
PConc	Poured Contrete
Slab	Slab
Stone	Stone
Wood	Wood

BsmtQual: Evaluates the height of the basement

Ex Excellent (100+ inches)
GdGood (90-99 inches)
TA Typical (80-89 inches)
Fa Fair (70-79 inches)

Po Poor (<70 inches
NA No Basement

BsmtCond: Evaluates the general condition of the basement

Ex Excellent
Gd Good
TA Typical - slight dampness allowed
Fa Fair - dampness or some cracking or settling
Po Poor - Severe cracking, settling, or wetness
NA No Basement

BsmtExposure: Refers to walkout or garden level walls

Gd Good Exposure
Av Average Exposure (split levels or foyers typically score average or above)
Mn Mimimum Exposure
No No Exposure
NA No Basement

BsmtFinType1: Rating of basement finished area

GLQ Good Living Quarters
ALQ Average Living Quarters
BLQ Below Average Living Quarters
Rec Average Rec Room
LwQ Low Quality
Unf Unfinshed
NA No Basement

BsmtFinSF1: Type 1 finished square feet

BsmtFinType2: Rating of basement finished area (if multiple types)

GLQ Good Living Quarters
ALQ Average Living Quarters
BLQ Below Average Living Quarters
Rec Average Rec Room
LwQ Low Quality
Unf Unfinshed
NA No Basement

BsmtFinSF2: Type 2 finished square feet

BsmtUnfSF: Unfinished square feet of basement area

TotalBsmtSF: Total square feet of basement area

Heating: Type of heating

Floor Floor Furnace
GasA Gas forced warm air furnace

GasW	Gas hot water or steam heat
Grav	Gravity furnace
OthW	Hot water or steam heat other than gas
Wall	Wall furnace

HeatingQC: Heating quality and condition

Ex	Excellent	Gd	Good	TA	Average/Typical
Fa	Fair	Po	Poor		

CentralAir: Central air conditioning

N	No	Y	Yes

Electrical: Electrical system

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

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

Ex	Excellent	Gd	Good	TA	Average/Typical
Fa	Fair	Po	Poor		

TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)

Functional: Home functionality (Assume typical unless deductions are warranted)

Typ	Typical Functionality
Min1	Minor Deductions 1

Min2 Minor Deductions 2
Mod Moderate Deductions
Maj1 Major Deductions 1
Maj2 Major Deductions 2
Sev Severely Damaged
Sal Salvage only

Fireplaces: Number of fireplaces

FireplaceQu: Fireplace quality

Ex Excellent - Exceptional Masonry Fireplace
Gd Good - Masonry Fireplace in main level
TA Average - Prefabricated Fireplace in main living area or Masonry Fireplace in basement
Fa Fair - Prefabricated Fireplace in basement
Po Poor - Ben Franklin Stove
NA No Fireplace

GarageType: Garage location

2Types More than one type of garage
Attchd Attached to home
Basment Basement Garage
BuiltIn Built-In (Garage part of house - typically has room above garage)
CarPort Car Port
Detchd Detached from home
NA No Garage

GarageYrBlt: Year garage was built

GarageFinish: Interior finish of the garage

Fin Finished RFn Rough Finished
Unf Unfinished NA No Garage

GarageCars: Size of garage in car capacity

GarageArea: Size of garage in square feet

GarageQual: Garage quality

Ex Excellent
Gd Good
TA Typical/Average
Fa Fair
Po Poor
NA No Garage

GarageCond: Garage condition

Ex Excellent
Gd Good

TA Typical/Average
Fa Fair
Po Poor
NA No Garage

PavedDrive: Paved driveway

Y Paved
P Partial Pavement
N Dirt/Gravel

WoodDeckSF: Wood deck area in square feet

OpenPorchSF: Open porch area in square feet

EnclosedPorch: Enclosed porch area in square feet

3**SsnPorch**: Three season porch area in square feet

ScreenPorch: Screen porch area in square feet

PoolArea: Pool area in square feet

PoolQC: Pool quality

Ex Excellent
Gd Good
TA Average/Typical
Fa Fair
NA No Pool

Fence: Fence quality

GdPrv Good Privacy
MnPrv Minimum Privacy
GdWo Good Wood
MnWw Minimum Wood/Wire
NA No Fence

MiscFeature: Miscellaneous feature not covered in other categories

Elev Elevator
Gar2 2nd Garage (if not described in garage section)
Othr Other
Shed Shed (over 100 SF)
TenC Tennis Court
NA None

MiscVal: $Value of miscellaneous feature

MoSold: Month Sold (MM)

YrSold: Year Sold (YYYY)

SaleType: Type of sale

WD	Warranty Deed - Conventional
CWD	Warranty Deed - Cash
VWD	Warranty Deed - VA Loan
New	Home just constructed and sold
COD	Court Officer Deed/Estate
Con	Contract 15% Down payment regular terms
ConLw	Contract Low Down payment and low interest
ConLI	Contract Low Interest
ConLD	Contract Low Down
Oth	Other

SaleCondition: Condition of sale

Normal	Normal Sale
Abnorml	Abnormal Sale - trade, foreclosure, short sale
AdjLand	Adjoining Land Purchase
Alloca	Allocation - two linked properties with separate deeds, typically condo with a garage unit
Family	Sale between family members
Partial	Home was not completed when last assessed (associated with New Homes)

Data Pre-processing

1. Checking for null values: We should deal with the problem of missing values

df_housing.isnull().sum()

```
In [7]: #Checking for null values
        df_housing.isnull().sum()

Out[7]: 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
```

Obs- There are many null values in multiple columns, so we will either perform imputation(on columns with few nan entries) or delete (columns with huge nan entries)

2. Deleting the: 1)high missing-data columns, 2) unwanted columns

1)high missing-data columns: We saw missing values were very high in columns: "Alley:1091", "FireplaceQu:551", "PoolQC:1161","Fence:931","MiscFeature:1124" And therefore filling it with using imputer was not efficient and thus we dropped these columns

```
In [10]: drop_columns=['Alley','FireplaceQu','PoolQC','Fence','MiscFeature']
         df_housing.drop(drop_columns,axis = 1,inplace = True)

In [11]: df_housing.shape

Out[11]: (1168, 76)
```

2) unwanted columns: Id wasn't imp column for model performance so deleted it.

```
In [14]: df_housing.drop(columns=['Id'],axis=1,inplace=True)
         df_housing.shape

Out[14]: (1168, 75)

In [15]: df_htest.drop(columns=['Id'],axis = 1,inplace = True)
         df_htest.shape

Out[15]: (292, 74)
```

3. Filling Null Values Columns:

The columns that have acceptable null values are of two types:

a)object: will be filling those columns with **mode** value

"MasVnrType-7", "BsmtQual-30", "BsmtCond-30", "BsmtExposure-31", "BsmtFinType1-30", "BsmtFinType2-31","GarageType- 64","GarageFinish-64","GarageQual-64","GarageCond-64"

b)float: will be filling those columns with **mean** value

"MasVnrArea-7", "GarageYrBlt-64"

In [16]:
```
# a)object type
from sklearn.impute import SimpleImputer
si = SimpleImputer(missing_values = np.nan,strategy = 'most_frequent',verbose = 0 )
si = si.fit(df_housing[['MasVnrType','BsmtQual','BsmtCond','BsmtExposure','BsmtFinType1','BsmtFinType2','GarageType','GarageFini
df_housing[['MasVnrType','BsmtQual','BsmtCond','BsmtExposure','BsmtFinType1','BsmtFinType2','GarageType','GarageFinish','GarageQu
```

In [17]:
```
# b)float values
df_housing["LotFrontage"] = df_housing["LotFrontage"].fillna(df_housing["LotFrontage"].mean())
df_housing["MasVnrArea"] = df_housing["MasVnrArea"].fillna(df_housing["MasVnrArea"].mean())
df_housing["GarageYrBlt"] = df_housing["GarageYrBlt"].fillna(df_housing["GarageYrBlt"].mean())
```

In [18]:
```
df_housing.isnull().sum()
```

Out[18]:
```
MSSubClass       0
MSZoning         0
LotFrontage      0
LotArea          0
Street           0
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
```

4. Separating Continuous(Numeric) columns from the original dataset¶

In [39]:
```
# list of numerical variables
numerical_features = [i for i in df_housing.columns if df_housing[i].dtypes != 'O']

print('Number of numerical variables: ', len(numerical_features))

# visualise the numerical variables
df_housing[numerical_features].head()
```

```
Number of numerical variables:  37
```

Out[39]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...	WoodDeckSF	OpenPorc
0	120	70.98847	4928	6	5	1976	1976	0.0	120	0	...	0	
1	20	95.00000	15865	8	6	1970	1970	0.0	351	823	...	81	
2	60	92.00000	9920	7	5	1996	1997	0.0	862	0	...	180	
3	20	105.00000	11751	6	6	1977	1977	480.0	705	0	...	0	
4	20	70.98847	16635	6	7	1977	2000	126.0	1246	0	...	240	

5 rows × 37 columns

5. Separating categorical columns from the original dataset:

In [46]:
```
categorical_features=[k for k in df_housing.columns if df_housing[k].dtypes=='O']
df_housing[categorical_features].head()
```

Out[46]:

	MSZoning	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	...	Electrical	KitchenQual	Functional	G
0	RL	Pave	IR1	Lvl	AllPub	Inside	Gtl	NPkVill	Norm	Norm	...	SBrkr	TA	Typ	
1	RL	Pave	IR1	Lvl	AllPub	Inside	Mod	NAmes	Norm	Norm	...	SBrkr	Gd	Typ	
2	RL	Pave	IR1	Lvl	AllPub	CulDSac	Gtl	NoRidge	Norm	Norm	...	SBrkr	TA	Typ	
3	RL	Pave	IR1	Lvl	AllPub	Inside	Gtl	NWAmes	Norm	Norm	...	SBrkr	TA	Typ	
4	RL	Pave	IR1	Lvl	AllPub	FR2	Gtl	NWAmes	Norm	Norm	...	SBrkr	Gd	Typ	

5 rows × 38 columns

Data Visualization

For getting the insights of relationship between the various features we started with visualization to discover any hidden patterns.

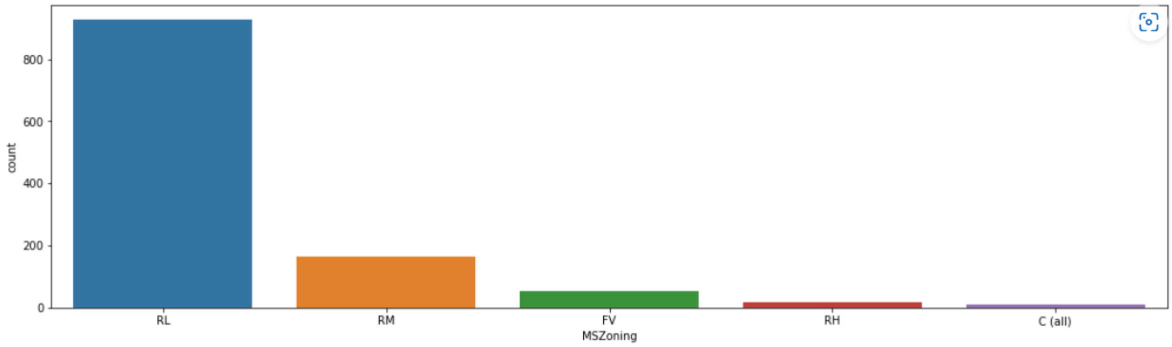
1. Univariate Analysis

a) Count Plots to check the percentage of unique attributes: Attaching few count plots

```
sns.countplot(x="MSZoning", data=df_housing)
```

```
RL         928
RM         163
FV          52
RH          16
C (all)      9
Name: MSZoning, dtype: int64
```

Out[23]: <AxesSubplot:xlabel='MSZoning', ylabel='count'>

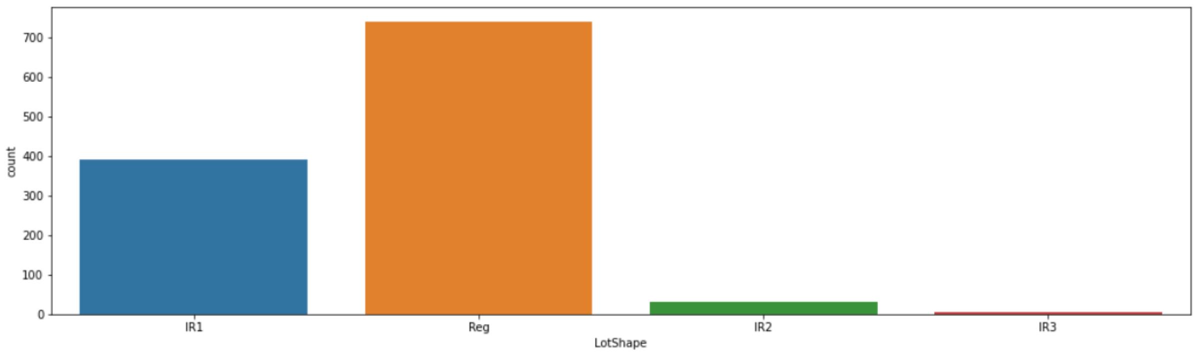


Obs- We can see the maximun properties in their list belong to : "RL: Residential Low Density", followed by "RM:Residential Medium Density","FV:Floating Village Residential","RH : Residential High Density" and least "C:commercial"

In [24]:
```
plt.figure(figsize=(18,5))
print(df_housing["LotShape"].value_counts())
sns.countplot(x="LotShape", data=df_housing)
```

```
Reg    740
IR1    390
IR2     32
IR3      6
Name: LotShape, dtype: int64
```

Out[24]: <AxesSubplot:xlabel='LotShape', ylabel='count'>

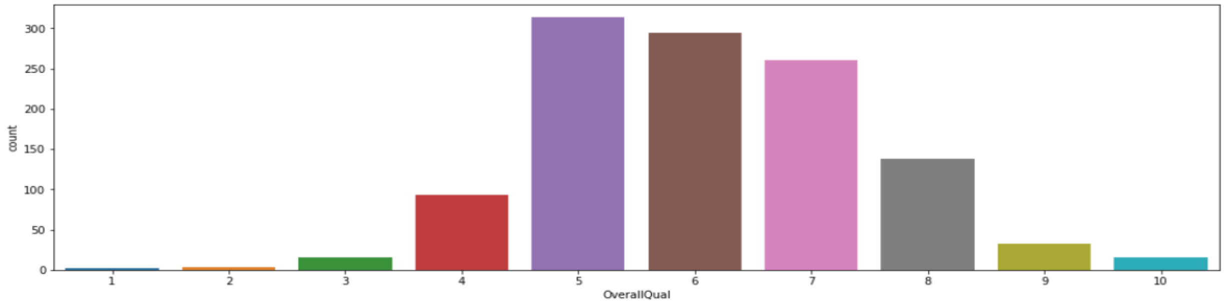


Obs- We can see the maximum properties they have are "Reg:Regular",followed by"IR1:Slightly irregular", very few that are"IR2:Moderately Irregular" and least that are"IR3:Irregular" That means they can earn good amount since they have good properties

```
In [28]: plt.figure(figsize=(18,5))
         print(df_housing["OverallQual"].value_counts())
         sns.countplot(x="OverallQual", data=df_housing)
```

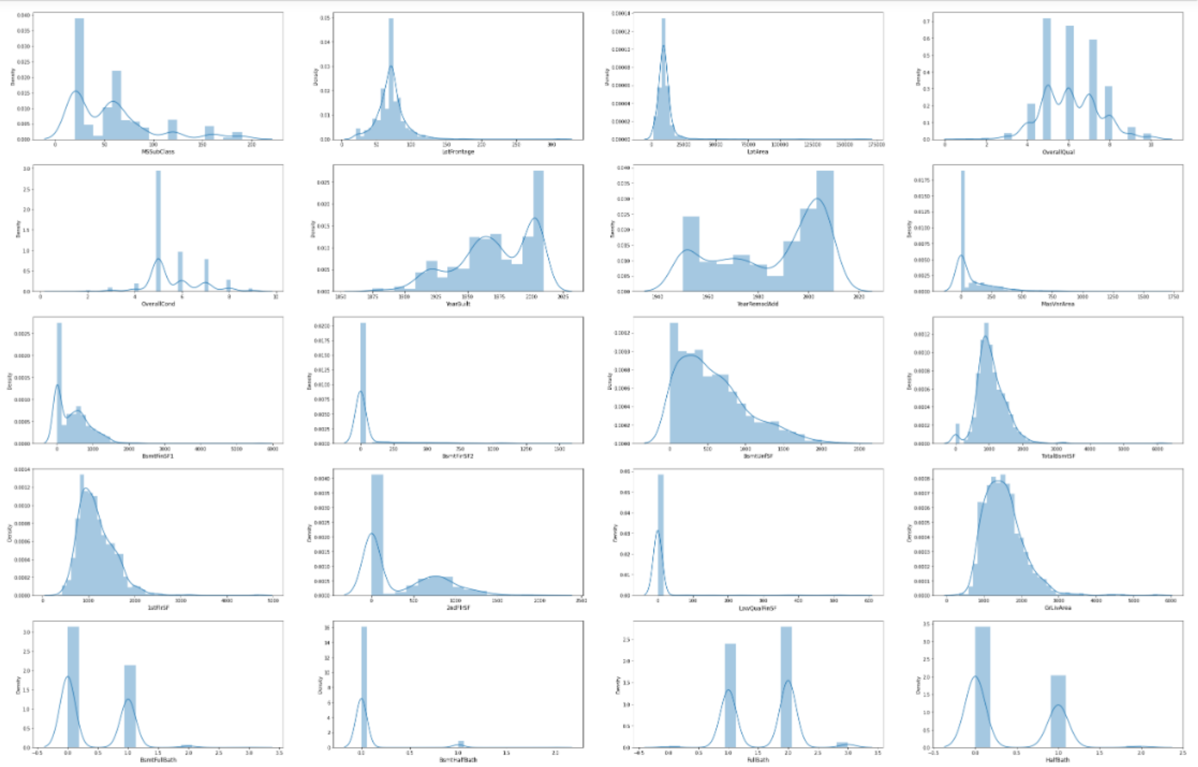
```
5     314
6     295
7     260
8     138
4      93
9      32
3      16
10     15
2       3
1       2
Name: OverallQual, dtype: int64
```

Out[28]: <AxesSubplot:xlabel='OverallQual', ylabel='count'>

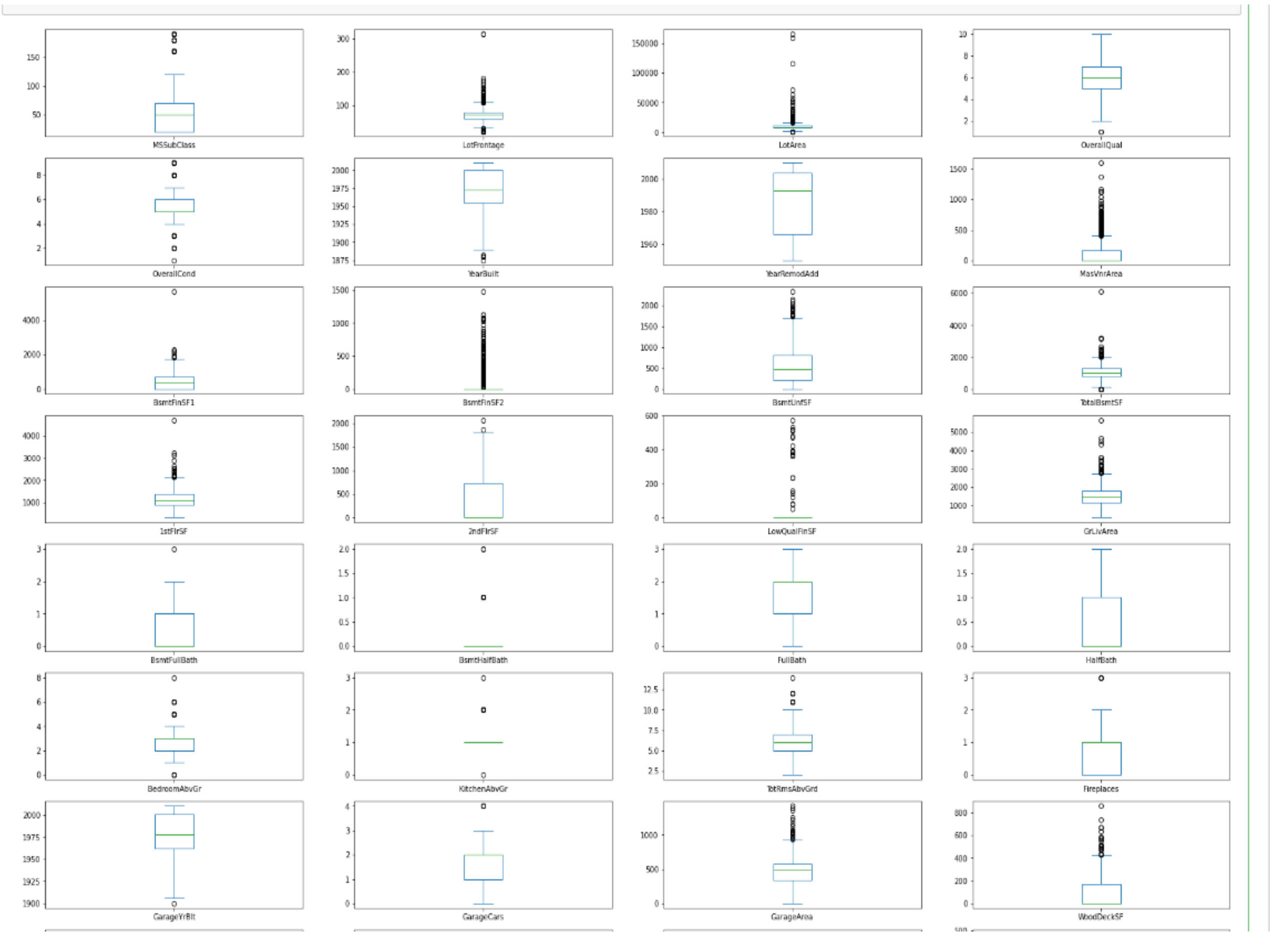


obs:We can see the maximum properties thew owe have the quality of materials used: "5:Average","6:Above Average","7:Good", followed by "8:Very Good", "4:Below Average", "9:Excellent, "3:Fair","10:Very Excellent" and a very few,"2:Poor"and "1:Very Poor"

b) Distribution Plot to check skewness in columns:

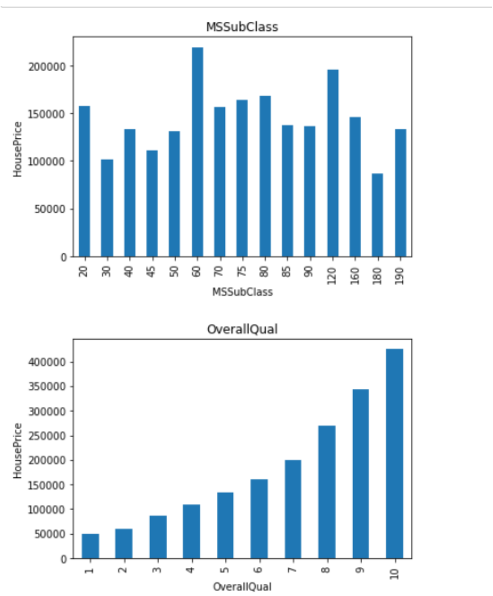


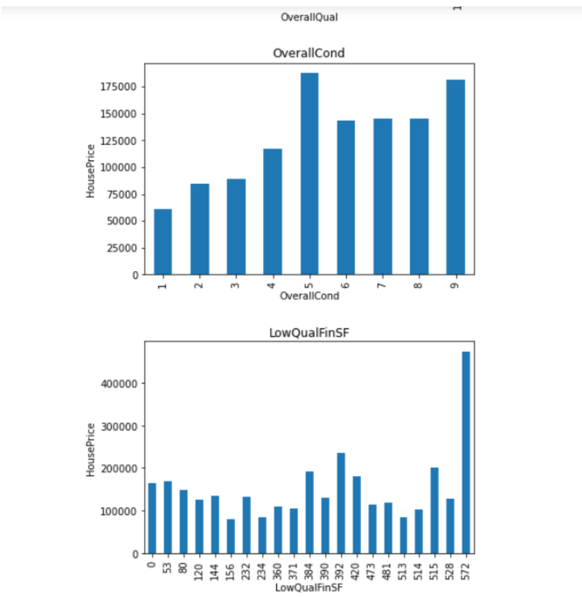
c) Box plot of all columns in same figure(to check outliers):



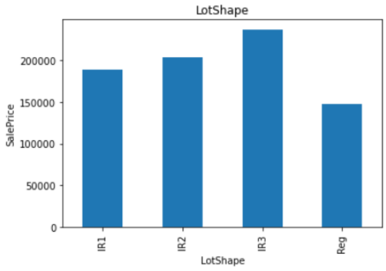
2. Bivariate Analysis:

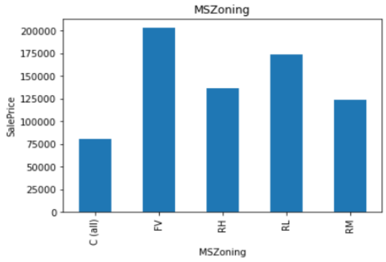
a) All numeric variables with unique count<25 vs House Price

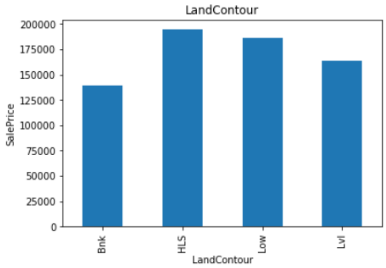


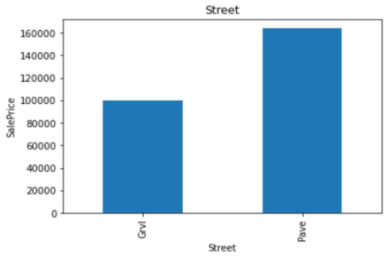


b) Categorical variables vs SalesPrice



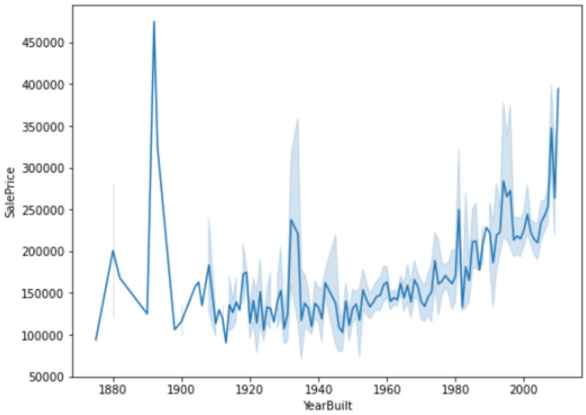






c) Line plot to see the growth of Sale Price

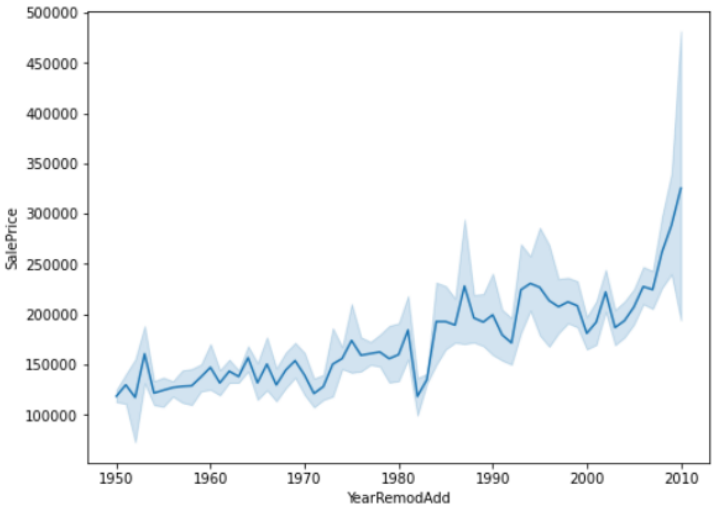
```
In [48]: # Line plot to see the growth of Sale Price
         plt.figure(figsize=(8,6))
         sns.lineplot(x='YearBuilt',y='SalePrice', data=df_housing)

Out[48]: <AxesSubplot:xlabel='YearBuilt', ylabel='SalePrice'>
```



Obs- We can see the prices goes high if the property is new

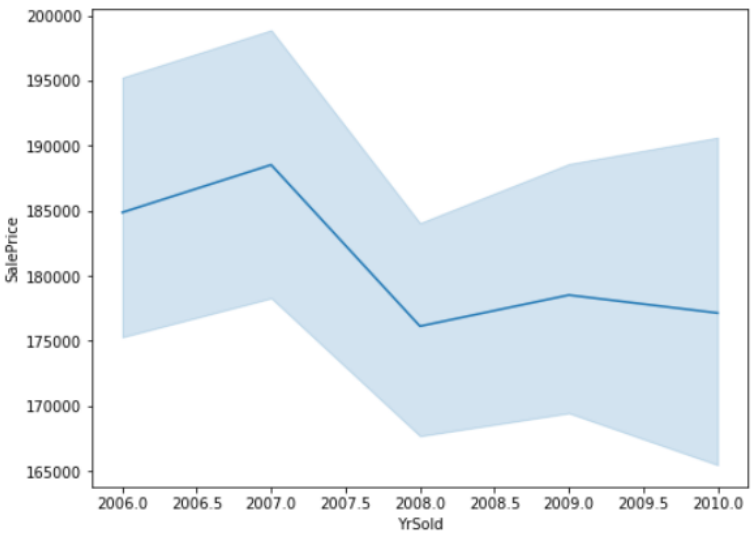
In [49]:
```
plt.figure(figsize=(8,6))
sns.lineplot(x='YearRemodAdd',y='SalePrice', data=df_housing)
```

Out[49]: <AxesSubplot:xlabel='YearRemodAdd', ylabel='SalePrice'>



In [50]:
```
plt.figure(figsize=(8,6))
sns.lineplot(x='YrSold',y='SalePrice', data=df_housing)
```

Out[50]: <AxesSubplot:xlabel='YrSold', ylabel='SalePrice'>



Obs- We can see , the late a property is sold, the lower are the returns

Model/s Development and Evaluation

1. Problem-solving approaches

We have to choose the type of machine learning prediction that is suitable for our problem. We want to determine if that is a regression problem or a classification problem. In this project, we wanted to predict the price of a house with given information about it. The price we want to predict is a continuous value and thus by looking at the target variable 'Sale price', in our dataset Sale Price, We can say our problem is regression problem

We started with pre-processing our data, Now performing the next essential steps:

a) **Encoding Categorical Data**: In this encoding scheme, the categorical feature is first converted into numerical using certain encoders. We have chosen Label Encoder for our problem.

In [51]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for i in categorical_features:
    df_housing[i]=le.fit_transform(df_housing[i])
```

In [52]:
```
df_housing.head()
```

Out[52]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	...	EnclosedPorch	3SsnPorch	ScreenPorch
0	120	3	70.98847	4928	1	0	3	0	4	0	...	0	0	0
1	20	3	95.00000	15865	1	0	3	0	4	1	...	0	0	224
2	60	3	92.00000	9920	1	0	3	0	1	0	...	0	0	0
3	20	3	105.00000	11751	1	0	3	0	4	0	...	0	0	0
4	20	3	70.98847	16635	1	0	3	0	2	0	...	0	0	0

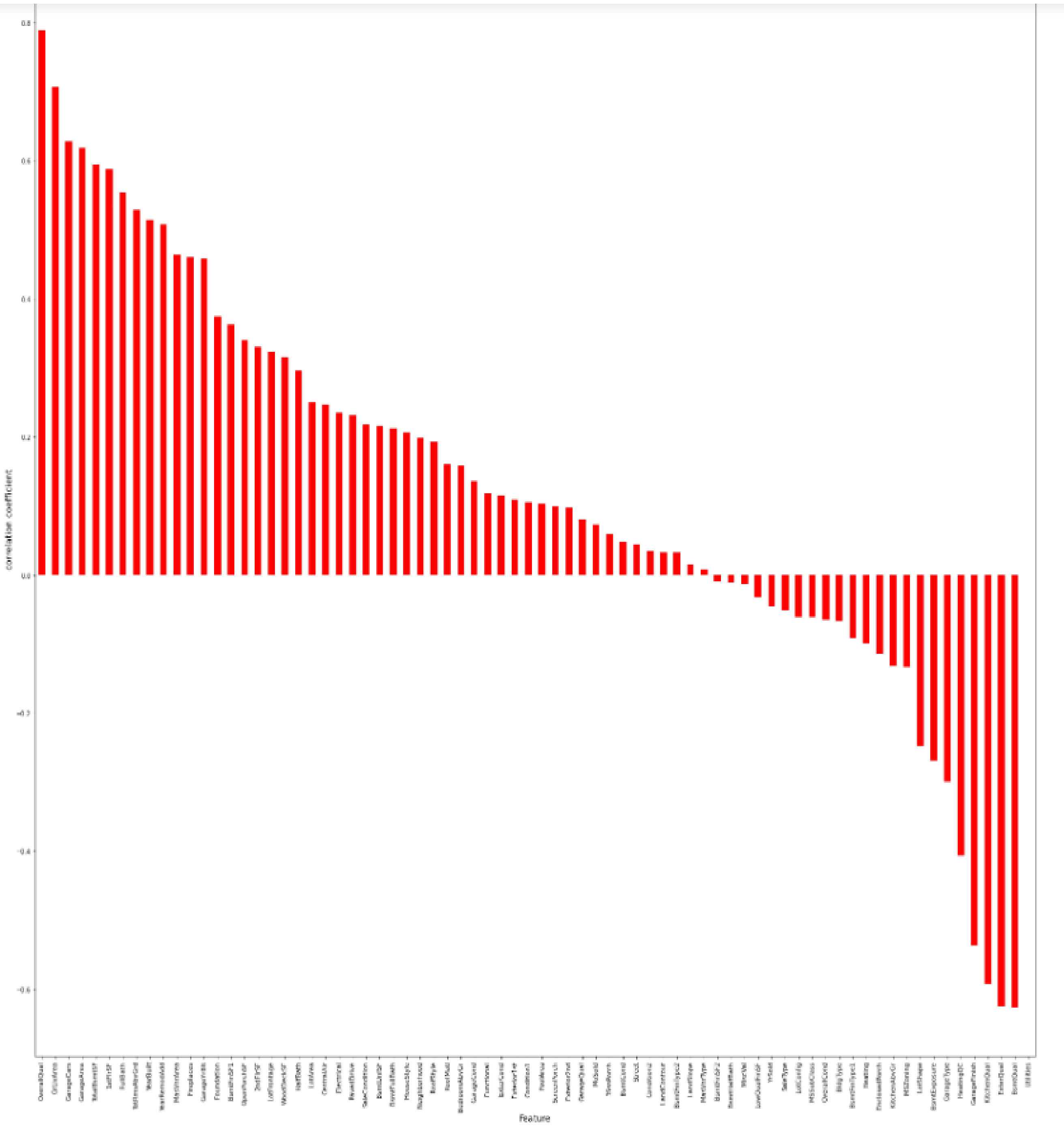
5 rows × 75 columns

b) **Checking Correlation:** It is used to find the pairwise correlation of all columns in the dataframe.

Since there are many columns, it would be hard to judge collinearity between each and every variable, so checking collinearity with target variable only.

In [54]:
```
abs(df_housing.corr()['SalePrice']).sort_values(ascending=False)
```

Out[54]:
```
SalePrice        1.000000
OverallQual      0.789185
GrLivArea        0.707300
GarageCars       0.628329
BsmtQual         0.626850
ExterQual        0.624820
GarageArea       0.619000
TotalBsmtSF      0.595042
KitchenQual      0.592468
1stFlrSF         0.587642
FullBath         0.554988
GarageFinish     0.537121
TotRmsAbvGrd     0.528363
YearBuilt        0.514408
YearRemodAdd     0.507831
MasVnrArea       0.463626
Fireplaces       0.459611
GarageYrBlt      0.458007
HeatingQC        0.406604
Foundation       0.374169
BsmtFinSF1       0.362874
OpenPorchSF      0.339500
2ndFlrSF         0.330386
LotFrontage      0.323779
WoodDeckSF       0.315444
GarageType       0.299470
HalfBath         0.295592
BsmtExposure     0.268559
LotArea          0.249499
LotShape         0.248171
CentralAir       0.246754
Electrical       0.234621
PavedDrive       0.231707
SaleCondition    0.217687
BsmtUnfSF        0.215724
BsmtFullBath     0.212924
HouseStyle       0.205502
Neighborhood     0.198942
RoofStyle        0.192654
RoofMatl         0.159865
BedroomAbvGr     0.158281
GarageCond       0.135071
MSZoning         0.133221
KitchenAbvGr     0.132108
Functional       0.118673
ExterCond        0.115167
EnclosedPorch    0.115004
Exterior1st      0.108451
Condition1       0.105820
PoolArea         0.103280
ScreenPorch      0.100284
Heating          0.100021
Exterior2nd      0.097541
BsmtFinType1     0.092109
GarageQual       0.080795
MoSold           0.072764
BldgType         0.066028
OverallCond      0.065642
MSSubClass       0.060775
LotConfig        0.060452
3SsnPorch        0.060119
SaleType         0.050851
BsmtCond         0.048125
YrSold           0.045508
Street           0.044753
Condition2       0.033956
LandContour      0.032836
LowQualFinSF     0.032381
BsmtFinType2     0.032285
LandSlope        0.015485
MiscVal          0.013071
BsmtHalfBath     0.011109
BsmtFinSF2       0.010151
MasVnrType       0.007732
Utilities             NaN
Name: SalePrice, dtype: float64
```



0.8
0.6
0.4
0.2
0.0
-0.2
-0.4
-0.6
correlation coefficient
Feature

c) Data Transformation

- **Checking for skewness:** Skewness is a measure of symmetry in a distribution. Actually, it's more correct to describe it as a measure of lack of symmetry. A standard normal distribution is perfectly symmetrical and has zero skew. Therefore, we need a way to calculate how much the distribution is skewed

```
In [56]: df_housing.skew()

Out[56]: MSSubClass        1.422019
         MSZoning         -1.796785
         LotFrontage       2.710383
         LotArea          10.659285
         Street          -17.021969
         LotShape         -0.603775
         LandContour      -3.125982
         Utilities         0.000000
         LotConfig        -1.118821
         LandSlope         4.812568
         Neighborhood      0.043735
         Condition1        3.008289
         Condition2       11.514458
         BldgType          2.318657
         HouseStyle        0.285680
         OverallQual       0.175082
         OverallCond       0.580714
         YearBuilt        -0.579204
         YearRemodAdd     -0.495864
         RoofStyle         1.498560
         RoofMatl          7.577352
         Exterior1st      -0.612816
         Exterior2nd      -0.592349
         MasVnrType       -0.104609
         MasVnrArea        2.834658
         ExterQual        -1.810843
         ExterCond        -2.516219
         Foundation       -0.002761
         BsmtQual         -1.343781
         BsmtCond         -3.293554
         BsmtExposure     -1.166987
         BsmtFinType1     -0.068901
         BsmtFinSF1        1.871606
         BsmtFinType2     -3.615783
         BsmtFinSF2        4.365829
         BsmtUnfSF         0.909057
         TotalBsmtSF       1.744591
         Heating          10.103609
         HeatingQC         0.449933
         CentralAir       -3.475188
         Electrical       -3.104209
         1stFlrSF          1.513707
         2ndFlrSF          0.823479
         LowQualFinSF      8.666142
         GrLivArea         1.449952
         BsmtHalfBath      4.264403
         FullBath          0.057809
         HalfBath          0.656492
         BedroomAbvGr      0.243855
         KitchenAbvGr      4.365259
         KitchenQual      -1.408106
         TotRmsAbvGrd      0.644657
         Functional       -3.999663
         Fireplaces        0.671966
         GarageType        0.831142
         GarageYrBlt      -0.662934
         GarageFinish     -0.450190
         GarageCars       -0.358556
         GarageArea        0.189665
         GarageQual       -4.582386
         GarageCond       -5.422472
         PavedDrive       -3.274035
         WoodDeckSF        1.504929
         OpenPorchSF       2.410840
         EnclosedPorch     3.043610
         3SsnPorch         9.770611
         ScreenPorch       4.105741
         PoolArea         13.243711
         MiscVal          23.065943
         MoSold            0.220979
         YrSold            0.115765
         SaleType         -3.660513
         SaleCondition    -2.671829
         SalePrice         1.953878
         dtype: float64
```

Obs- We can see skewness is present in most of the columns, so will be seperating Input and output and then applying Transformation Method

- **Applying Transformation Method/Treating Skewness:** We have used Power transform function to handle skewness in dataset

```
In [60]: from sklearn.preprocessing import PowerTransformer
         pt=PowerTransformer()
         x_scaled=pt.fit_transform(x)
         x=pd.DataFrame(x_scaled,columns=x.columns)

In [61]: x.skew()

Out[61]: MSSubClass        0.064007
         MSZoning          0.233113
         LotFrontage       0.161368
         LotArea           0.032509
         Street          -17.021969
         LotShape         -0.594207
         LandContour      -2.592303
         Utilities         0.000000
         LotConfig        -1.030401
         LandSlope         3.954345
         Neighborhood     -0.146541
         Condition1        0.225468
         Condition2        0.537277
         BldgType          1.857194
         HouseStyle       -0.080331
         OverallQual       0.021658
         OverallCond       0.048063
         YearBuilt        -0.126641
         YearRemodAdd     -0.225131
```

d) **Data Scaling:** In order to make all algorithms work properly with our data, we have to normalize the input features/variables. We have used Standard Scaler, which brings standard deviation to 1.

In [63]:
```
from sklearn.preprocessing import StandardScaler
st=StandardScaler()
x_scale=st.fit_transform(x)
x=pd.DataFrame(data=x_scale)
x
```

Out[63]:

	0	1	2	3	4	5	6	7	8	9	...	64	65	66	67	
0	1.370435	-0.162456	0.093658	-1.213954	0.058621	-1.366794	0.341434	0.0	0.617281	-0.238775	...	1.409990	-0.411301	-0.138554	-0.297551	-(
1	-1.167999	-0.162456	1.117135	1.100521	0.058621	-1.366794	0.341434	0.0	0.617281	4.188040	...	1.414498	-0.411301	-0.138554	3.360787	-(
2	0.490047	-0.162456	0.998803	0.158048	0.058621	-1.366794	0.341434	0.0	-1.482445	-0.238775	...	1.198911	-0.411301	-0.138554	-0.297551	-(
3	-1.167999	-0.162456	1.495566	0.496002	0.058621	-1.366794	0.341434	0.0	0.617281	-0.238775	...	1.169545	-0.411301	-0.138554	-0.297551	-(
4	-1.167999	-0.162456	0.093658	1.196626	0.058621	-1.366794	0.341434	0.0	-1.025661	-0.238775	...	-1.061392	-0.411301	-0.138554	-0.297551	-(
5	0.490047	-0.162456	-0.552490	0.855555	0.058621	-1.366794	0.341434	0.0	0.617281	-0.238775	...	0.275826	-0.411301	-0.138554	-0.297551	-(
6	-1.167999	-0.162456	0.093658	0.424957	0.058621	-1.366794	0.341434	0.0	0.617281	-0.238775	...	-1.061392	-0.411301	-0.138554	3.360729	-(
7	-1.167999	-0.162456	0.837233	0.717859	0.058621	0.753907	0.341434	0.0	-1.725008	-0.238775	...	-1.061392	-0.411301	-0.138554	-0.297551	-(
8	-1.167999	-0.162456	0.047197	0.001967	0.058621	0.753907	0.341434	0.0	-1.725008	-0.238775	...	-1.061392	2.433545	-0.138554	-0.297551	-(
9	0.237618	-0.162456	0.499839	-0.152859	0.058621	0.753907	0.341434	0.0	0.617281	-0.238775	...	-1.061392	2.430968	-0.138554	-0.297551	-(
10	0.237618	2.056505	-0.996296	-0.125083	0.058621	0.753907	-2.967750	0.0	0.617281	-0.238775	...	-1.061392	-0.411301	-0.138554	-0.297551	-(

e) **Using PCA to reduce curse of dimensionality**: PCA is a process to reduce the dimensions of your large data by finding correlation between them and without creating any data loss, here we have reduced the components from 74 to 40

In [66]:
```
from sklearn.decomposition import PCA
pc=PCA(n_components=40)
x_pca=pc.fit_transform(x)
x_pca
```

Out[66]:
```
array([[-7.18324400e-01, -4.42999515e-01, -2.08368759e+00, ...,
        -2.21184226e-01, -4.63551023e-01,  5.01999177e-01],
       [ 2.39747233e+00, -3.13772899e+00,  4.23300981e+00, ...,
         1.16504482e+00,  5.89286844e-02,  2.30828811e-03],
       [ 2.90575578e+00, -1.58244974e-01,  5.16048615e-01, ...,
         7.30960677e-01, -3.36790104e-01, -2.28516362e-01],
       ...,
       [-1.26684273e+00,  8.52432289e-01, -1.99326187e+00, ...,
        -5.71598662e-01,  1.26052532e-01,  1.35649105e-01],
       [-5.96822166e+00,  3.46246274e+00,  1.22425648e+00, ...,
        -1.15162876e+00, -2.13802605e+00,  1.93193470e+00],
       [ 2.82008515e+00,  9.35906391e-01, -1.26583750e+00, ...,
        -2.50902112e-01,  8.08901615e-02,  2.13760887e-02]])
```

2. Modelling Approach:

We have already mentioned our model would be a regression model and for all of the techniques mentioned in the previous section (Linear Regression,Random Forest Regression, Decision Tree Regression, XGBoost, k-nearest neighbors(KNN) etc etc.), we will follow these steps to build a model:

• Choose an algorithm that implements the corresponding technique

• Search for an effective parameter combination for the chosen algorithm

• Create a model using the found parameters

• Train (fit) the model on the training dataset •

Test the model on the test dataset and get the results

a) **Regression Method:** • Using Scikit-Learn, we build a model for Linear Regression Model and used (included) the following regressors

```
In [70]: #importing necessary libraries
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,mean_absolute_error

from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
```

b) **Finding Best Random State, and using it for splitting Train-Test Data**

```
In [71]: maxAccu=0
maxRS=0
for i in range(1,200):
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=.30, random_state =i)
    lr = LinearRegression()
    lr.fit(x_train, y_train)
    pred = lr.predict(x_test)
    acc=r2_score(y_test, pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print("Best accuracy is ",maxAccu," on Random_state ",maxRS)

Best accuracy is  0.8468354906983884  on Random_state  195
```

```
In [72]: # Splitting the data for training and testing
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=195)
```

3. Running and Evaluating selected models

1. Linear Regression

In [77]:
```
lr=LinearRegression()
lr.fit(x_train,y_train)
lr.score(x_train,y_train)
```

Out[77]:
```
0.7996857643777088
```

In [78]:
```
predlr=lr.predict(x_test)
print("r2_score=",r2_score(y_test,predlr),"\n")

print("Mean Absolute Error:",mean_absolute_error(y_test,predlr))
print("Mean Squared Error:",mean_squared_error(y_test,predlr))
print("Root Meand Squared Error:",np.sqrt(mean_squared_error(y_test,predlr)))
```

```
r2_score= 0.8468354906983884

Mean Absolute Error: 22210.742586205022
Mean Squared Error: 860895664.2137587
Root Meand Squared Error: 29341.02357133709
```

02. Support Vector Regressor

In [80]:
```
svr=SVR()
svr.fit(x_train,y_train)
svr.score(x_train,y_train)
```

Out[80]:
```
-0.05313108264573185
```

In [81]:
```
predsvr=svr.predict(x_test)
print("r2_score=",r2_score(y_test,predsvr),"\n")

print("Mean Absolute Error:",mean_absolute_error(y_test,predsvr))
print("Mean Squared Error:",mean_squared_error(y_test,predsvr))
print("Root Meand Squared Error:",np.sqrt(mean_squared_error(y_test,predsvr)))
```

```
r2_score= -0.03728484652218955

Mean Absolute Error: 54805.50017117904
Mean Squared Error: 5830293394.973785
Root Meand Squared Error: 76356.35792109118
```

3. Decision Tree Regressor

In [82]:
```
dtr=DecisionTreeRegressor()
dtr.fit(x_train,y_train)
dtr.score(x_train,y_train)
```

Out[82]:
```
1.0
```

In [83]:
```
preddtr=dtr.predict(x_test)
print("r2_score=",r2_score(y_test,preddtr),"\n")

print("Mean Absolute Error:",mean_absolute_error(y_test,preddtr))
print("Mean Squared Error:",mean_squared_error(y_test,preddtr))
print("Root Meand Squared Error:",np.sqrt(mean_squared_error(y_test,preddtr)))
```

```
r2_score= 0.7408952690537688

Mean Absolute Error: 27147.555555555555
Mean Squared Error: 1456356570.2393162
Root Meand Squared Error: 38162.24011034096
```

4. K Neighbors Regressor

In [84]:
```
knr=KNeighborsRegressor()
knr.fit(x_train,y_train)
knr.score(x_train,y_train)
```

Out[84]:
```
0.8089782517775758
```

In [85]:
```
predknr=knr.predict(x_test)
print("r2_score=",r2_score(y_test,predknr),"\n")

print("Mean Absolute Error:",mean_absolute_error(y_test,predknr))
print("Mean Squared Error:",mean_squared_error(y_test,predknr))
print("Root Meand Squared Error:",np.sqrt(mean_squared_error(y_test,predknr)))
```

```
r2_score= 0.8162237147358579

Mean Absolute Error: 22729.3150997151
Mean Squared Error: 1032956054.1839316
Root Meand Squared Error: 32139.633697102578
```

5.Random Forest Regressor

In [86]:
```
rfr=RandomForestRegressor()
rfr.fit(x_train,y_train)
rfr.score(x_train,y_train)
```

Out[86]:
```
0.9749256940878515
```

In [87]:
```
predrfr=rfr.predict(x_test)
print("r2_score=",r2_score(y_test,predrfr),"\n")

print("Mean Absolute Error:",mean_absolute_error(y_test,predrfr))
print("Mean Squared Error:",mean_squared_error(y_test,predrfr))
print("Root Meand Squared Error:",np.sqrt(mean_squared_error(y_test,predrfr)))
```

```
r2_score= 0.875728249720397

Mean Absolute Error: 18544.416381766383
Mean Squared Error: 698497396.6083114
Root Meand Squared Error: 26429.10132048215
```

6.Ada Boost Regressor

In [88]:
```
adr=AdaBoostRegressor()
adr.fit(x_train,y_train)
adr.score(x_train,y_train)
```

Out[88]:
```
0.8797392262573733
```

In [89]:
```
predadr=adr.predict(x_test)
print("r2_score=",r2_score(y_test,predadr),"\n")

print("Mean Absolute Error:",mean_absolute_error(y_test,predadr))
print("Mean Squared Error:",mean_squared_error(y_test,predadr))
print("Root Meand Squared Error:",np.sqrt(mean_squared_error(y_test,predadr)))
```

```
r2_score= 0.8093723864709754

Mean Absolute Error: 24383.68992021454
Mean Squared Error: 1071465489.7198594
Root Meand Squared Error: 32733.24746675556
```

7.Gradient Boosting Regressor

In [90]:
```
gbr=GradientBoostingRegressor()
gbr.fit(x_train,y_train)
gbr.score(x_train,y_train)
```

Out[90]:
```
0.9753082728429688
```

In [91]:
```
predgbr=gbr.predict(x_test)
print("r2_score=",r2_score(y_test,predgbr),"\n")

print("Mean Absolute Error:",mean_absolute_error(y_test,predgbr))
print("Mean Squared Error:",mean_squared_error(y_test,predgbr))
print("Root Meand Squared Error:",np.sqrt(mean_squared_error(y_test,predgbr)))
```

```
r2_score= 0.885742054875182

Mean Absolute Error: 17286.20805242794
Mean Squared Error: 642212546.5516992
Root Meand Squared Error: 25341.912843187256
```

Regularization to overcome Over-Fitting (Lasso,Ridge)

8. Lasso Regression

In [93]:
```
ls=Lasso(alpha=0.1)
ls.fit(x_train,y_train)
ls.score(x_train,y_train)
```

Out[93]: 0.7996857643240963

In [94]:
```
predls=ls.predict(x_test)
print("r2_score=",r2_score(y_test,predls),"\n")

print("Mean Absolute Error:",mean_absolute_error(y_test,predls))
print("Mean Squared Error:",mean_squared_error(y_test,predls))
print("Root Meand Squared Error:",np.sqrt(mean_squared_error(y_test,predls)))
```

```
r2_score= 0.8468365046488601

Mean Absolute Error: 22210.6304120361
Mean Squared Error: 860889965.0764793
Root Meand Squared Error: 29340.926452252308
```

9. Ridge Regression

In [95]:
```
rg=Ridge(alpha=0.1)
rg.fit(x_train,y_train)
rg.score(x_train,y_train)
```

Out[95]: 0.7996857630783943

In [96]:
```
predrg=rg.predict(x_test)
print("r2_score=",r2_score(y_test,predrg),"\n")

print("Mean Absolute Error:",mean_absolute_error(y_test,predrg))
print("Mean Squared Error:",mean_squared_error(y_test,predrg))
print("Root Meand Squared Error:",np.sqrt(mean_squared_error(y_test,predrg)))
```

```
r2_score= 0.8468406085460842

Mean Absolute Error: 22210.069589080736
Mean Squared Error: 860866898.196673
Root Meand Squared Error: 29340.53336592014
```

4. **Key Metrics for success in solving problem under consideration:** We have considered r2_score as well as Errors as a measure to check model performances.

   Random Forest Regressor and Gradient Boosting Regressor performed well from the other models, so to select the best model, performing Cross Validation.

5. **Cross Validation**: Took k-folds=10

```
In [103]: from sklearn.model_selection import cross_val_score

In [104]: # cv score for Linear Regressor
          print('CV score  for Linear rgerssor',cross_val_score(lr,x,y,cv=10).mean())

          CV score  for Linear rgerssor 0.778786125372394

In [105]: # cv score for Support Vector Regressor
          print('CV score  for Support Vector Regressor',cross_val_score(svr,x,y,cv=10).mean())

          CV score  for Support Vector Regressor -0.0607342112301519

In [106]: # cv score for Decision Tree Regressor
          print('CV score  for Decision Tree Regressor ',cross_val_score(dtr,x,y,cv=10).mean())

          CV score  for Decision Tree Regressor  0.6753154297659809

In [107]: # cv score for K Neighbors Regressor
          print('CV score  for K Neighbors regressor ',cross_val_score(knr,x,y,cv=10).mean())

          CV score  for K Neighbors regressor  0.7533147034124822

In [108]: # cv score for Random Forest Regressor
          print('CV score  for Random Forest Regressor ',cross_val_score(rfr,x,y,cv=10).mean())

          CV score  for Random Forest Regressor  0.8347021035806346

In [109]: # cv score for Ada Boost Regressor
          print('CV score  for Ada Boost Regressor ',cross_val_score(adr,x,y,cv=10).mean())

          CV score  for Ada Boost Regressor  0.7580034373997874

In [110]: # cv score for GradientBoostingRegressor
          print('CV score  for GradientBoostingRegressor',cross_val_score(gbr,x,y,cv=10).mean())

          CV score  for GradientBoostingRegressor 0.8645902633177649

In [111]: # cv score for Lasso Regressor
          print('CV score  for Lasso Regressor ',cross_val_score(ls,x,y,cv=10).mean())

          CV score  for Lasso Regressor  0.7787871295197937

In [112]: # cv score for Ridge Regressor
          print('CV score  for Ridge Regressor ',cross_val_score(rg,x,y,cv=10).mean())

          CV score  for Ridge Regressor  0.7787908082503933
```

Observation- Checking both the scores and errors, we can say Gradient Boosting Regressor has performed well and we selected Gradient Boosting Regressor as our model.

To further improve the score and reduce errors, we have performed hyper parameter tuning to find best parameters for our model.

6. Hyper Parameter Tuning¶

```
In [114]: # Defining parameters for Gradient Boosting Algorithm
parameters = {'n_estimators':[100,150],'min_samples_split':[2,6],'min_samples_leaf':[1,5],'learning_rate': np.arange(0.1,0.5,0.1)
#start the tuning
gbr=GradientBoostingRegressor()
GCV=GridSearchCV(gbr,parameters,cv=10)

GCV.fit(x_train,y_train)
print(GCV.best_params_)        #printing the best parameter found by Gridsearchcv
```

```
{'learning_rate': 0.1, 'min_samples_leaf': 5, 'min_samples_split': 2, 'n_estimators': 100}
```

Applying best parameter values on Gradient Bossting Regressor Algorithm and checking the r2-score and errors

```
In [115]: #Applying best parameter values on Gradient Bossting Regressor Algorithm
gbr1=GradientBoostingRegressor(n_estimators=100 ,min_samples_split=2, min_samples_leaf=5 ,learning_rate= 0.1)
gbr1.fit(x_train,y_train)
gbr1.score(x_train,y_train)
```

```
Out[115]: 0.9703886233500587
```

```
In [116]: predgbr1=gbr1.predict(x_test)
print("r2_score=",r2_score(y_test,predgbr1),"\n")

print("Mean Absolute Error:",mean_absolute_error(y_test,predgbr1))
print("Mean Squared Error:",mean_squared_error(y_test,predgbr1))
print("Root Meand Squared Error:",np.sqrt(mean_squared_error(y_test,predgbr1)))
```

```
r2_score= 0.8853868473648824

Mean Absolute Error: 17373.136460818983
Mean Squared Error: 644209070.465153
Root Meand Squared Error: 25381.274011860653
```

Obs- Our best model after fine tuning has given r2_score=0.8853868473648824 and mean absolute error=17373.136460818983

Feature Importance

From the overall EDA process(visualization, correlation), we can say the "atttributes/features" that effect sale price positively/negatively could be as follows:

1. **Features affecting Positively to Sale Price:**

a) **MSSubClass** (As according to the type of building(story) they have its price increase)
b) **LotFrontage:** Linear feet of street connected to property (The more street, the more price)
c) **LotArea:** Lot size in square feet(The larger the area, higher the price)
d) **TotalBsmtSF:** Total square feet of basement area(The larger the basement area, higher the price)
e) **TotRmsAbvGrd:** Total rooms above grade(More the number of rooms, higher the price, and they do owe properties with max 6-7-8 rooms)
f) **YearBuilt:** Original construction date (As they do owe lots of new properties or old properties that are remodelled,they can get good sale price)
g) **KitchenQual:** Kitchen quality(Most of the properties have good kitchen quality)
h) **FullBath:** Full bathrooms above grade
i) **GarageCars:** Size of garage in car capacity(They do own max properties that have atleast 2-3 cars area for garage and min 1)

2. **Features affecting Negatively to Sale Price**

a) **OverallQual:(**Since the maximum properties they owe have average quality of material used,it is negatively affecting the sale price)
b) **OverallCond**:(Again,the maximum properties they owe have average condition it is negatively affecting the sale price)
c) **LotShape**:(Since they have half of the properties whose shape is irregular, it is negatively affecting the sale price)

Results

Our best model i.e. Gradient Boosting Regressor, after fine tuning has given:

r2_score=0.8853868473648824 and

mean absolute error=17373.136460818983

Saving the best model

```
In [120]: import pickle
          filename='housing_price_report'
          pickle.dump(gbr1,open(filename,'wb'))
```

```
In [121]: housing_best = pickle.load(open(filename, 'rb'))
          housing_best

Out[121]: GradientBoostingRegressor(min_samples_leaf=5)
```

Using Best Model to Predict Provided Test Data

```
In [125]: predict= housing_best.predict(x1)
          pred_price= pd.DataFrame(predict)
          pred_price
```

Out[125]:

	0
0	448688.126419
1	213947.236265
2	252954.108141
3	150121.358193
4	284652.265922
5	60070.575918
6	136634.549913
7	377278.062983
8	194156.003204
9	171368.006642
10	72992.387039
11	145194.863083

Appending this result of predicted price in original test data provided and saving it in csv format

In [128]:
```
df_htest['Predicted Sale Price']=pred_price
```

In [129]:
```
df_htest.head()
```

Out[129]:

ntour	Utilities	LotConfig	LandSlope	...	EnclosedPorch	3SsnPorch	ScreenPorch	PoolArea	MiscVal	MoSold	YrSold	SaleType	SaleCondition	Predicted Sale Price
1	0	0	0	...	0	0	0	0	0	7	2007	5	2	448688.126419
3	0	1	0	...	0	0	0	0	0	8	2009	0	0	213947.236265
3	0	4	0	...	0	0	0	0	0	6	2009	5	2	252954.108141
0	0	4	0	...	0	0	0	0	0	7	2009	5	2	150121.358193
3	0	1	0	...	0	0	0	0	0	1	2008	5	2	284652.265922

Saving the Predicted Test Model in csv format

In [132]:
```
df_htest.to_csv('housing price test data predicted')
```

In [133]:
```
house_pred=pd.read_csv('housing price test data predicted')
```

In [134]:
```
house_pred
```

Out[134]:

	Unnamed: 0	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	...	EnclosedPorch	3SsnPorch	ScreenF
0	0	20	2	86.000000	14157	1	0	1	0	0	...	0	0	
1	1	120	2	66.425101	5814	1	0	3	0	1	...	0	0	
2	2	20	2	66.425101	11838	1	3	3	0	4	...	0	0	
3	3	70	2	75.000000	12000	1	3	0	0	4	...	0	0	
4	4	60	2	86.000000	14598	1	0	3	0	1	...	0	0	
5	5	180	3	21.000000	1936	1	3	3	0	4	...	0	0	
6	6	180	3	35.000000	3675	1	3	3	0	4	...	0	0	
7	7	20	2	107.000000	13891	1	3	3	0	4	...	0	0	
8	8	80	2	66.425101	12800	1	3	2	0	4	...	0	0	
9	9	120	3	32.000000	4500	1	3	3	0	2	...	0	0	
10	10	30	3	60.000000	6324	1	0	3	0	4	...	87	0	

Conclusion

In this project, we built several regression models to predict the price of any house given some of the house features. We evaluated and compared each model to determine the one with highest performance.

In this project, we followed the data science process starting with getting the data, then cleaning and pre-processing the data, followed by visualizations and exploring the data and building models, then evaluating the results through Regularization and Cross-Validation.

As a recommendation, we advise to use this model (or a version of it trained with more recent data) by people who want to buy a house in the area covered by the dataset to have an idea about the actual price. The model can also be used with datasets that cover areas containing the same features. We also suggest that people take into consideration the features that were deemed as most important as provided above; this might help them estimate the house price better.

Limitations of this work and Scope for Future Work

There are many things that can be tried to improve the models' predictions. We can create and add more variables, try different models with different subset of features and/or rows, etc.

The `biggest limitation in the dataset was that not all categories of a particular feature were available in the training data. So, if there is a new category in the test data/new data, the model would not be able to identify the new categories.