Housing Price Prediction Project

Submitted by:

Aishwary Shukla

ACKNOWLEDGMENT

The aim of this project was to Sales Prices of Houses using Machine Learning. A data set was provided. From this dataset models were build in Jupyter notebook using Python and ML libraries. After a preliminary study of the available algorithms and data review, it became apparent that the problem fell under regression category. Thus, we have used regression algorithms; linear regression, decision tree, Random Forest, KNeighbours Regressor, Support Vector Machine, AdaBoostRegressor respectively. The major discovery is that the machine learning approach should be suitable for these types of problems due to many aspects. Python programming language and its libraries namely Pandas, Numpy, Matplotlib, Seaborn etc are also a good choice for a first step, not the least because of the easily grasped user interface, as well as the wide availability of algorithms within machine learning. Advanced methods such as hyper parameter tuning of best models is also available.

References:

- https://www.researchgate.net
- https://www.kaggle.com
- https://www.investopedia.com

Special thanks to:

Mr. Shankar Gaud Tegimanni, DataTrained

Mr. Shwetank Mishra, FlipRobo

Mr. Prateeek Rajvanshi, Clevered Institute

And all colleagues, mentors, trainers from DataTrained and FlipRobo for training me and giving me the opportunity to be an intern and expand my knowledge in the field of Data Science and Artificial Intelligence.

INTRODUCTION

Problem Statement: Real estate represents a significant portion of most people's wealth, and this is especially true for many homeowners in the United States. According to the Survey of Consumer Finances by the Federal Reserve, 64.9% of American families owned their own primary residence in 2019. The size and scale of the real estate market make it an attractive and lucrative sector for many investors.

With Data science and machine learning we will to make a model that predicts sales price of Houses that will 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.

The data is provided in the CSV file below.

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.

We will investigate:

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

Business Goal: We are required to model the price of houses with the available independent variables. This model will then be used to understand how exactly the prices vary with the variables. Housing firms then 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 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.

Review of Literature

Steps followed to conduct the research of finding important features that positively affect the sales price of Houses.

- Exploratory data analysis
- Feature Engineering
- Relation between different features/amenities of house and target column i.e. sales price.
- Correlation and identification of multicollinearity problem.
- Identifying and Selecting best features that affect sales price.
- Data pre-processing.
- Model Initialization and evaluation.
- Model Validation
- Model Testing and Pipelining.
- Prediction of test dataset values.

Motivation for the Problem Undertaken

My objective behind making this project is to implement techniques and methods ive learned and practiced during my PG programme in Data Science. Real estate and house investments interest me thus working on this project will give me domain knowledge and technical expertise in deploying ml models for solving real world problems.

Analytical Problem Framing

- Mathematical/ Analytical Modeling of the Problem

  Describe the mathematical, statistical and analytics modelling done during this project along with the proper justification.

- Data Sources and their formats

  What are the data sources, their origins, their formats and other details that you find necessary? They can be described here. Provide a proper data description. You can also add a snapshot of the data.

- Data Preprocessing Done

  What were the steps followed for the cleaning of the data? What were the assumptions done and what were the next actions steps over that?

- Data Inputs- Logic- Output Relationships

  Describe the relationship behind the data input, its format, the logic in between and the output. Describe how the input affects the output.

- State the set of assumptions (if any) related to the problem under consideration

  Here, you can describe any presumptions taken by you.

- Hardware and Software Requirements and Tools Used

  <u>Hardware and software requirements:</u>

- Python and its libraries relevant to machine learning
- Computer with adequate specification.

Tools, libraries and packages used with describe:

NumPy:

NumPy is a very popular python library for large multi-dimensional array and matrix processing, with the help of a large collection of high-level mathematical functions. It is very useful for fundamental scientific computations in Machine Learning. It is particularly useful for linear algebra, Fourier transform, and random number capabilities. High-end libraries like TensorFlow uses NumPy internally for manipulation of Tensors.

Pandas:

Pandas is a popular Python library for data analysis. It is not directly related to Machine Learning. As we know that the dataset must be prepared before training. In this case, Pandas comes handy as it was developed specifically for data extraction and preparation. It provides high-level data structures and wide variety tools for data analysis. It provides many inbuilt methods for grouping, combining and filtering data.

Matplotlib.pyplot and Seaborn

Matplotlib and Seaborn **act as the backbone of data visualization through Python**. Matplotlib: It is a Python library used for plotting graphs with the help of other libraries like Numpy and Pandas. It is a powerful tool for visualizing data in Python.

%matplotlib inline

Matplotlib Inline command is a command that makes the plots generated by matplotlib show into the IPython shell that we are running and not in a separate output window.

statsmodels.stats.outliers_influence import variance_inflation_factor

The variance inflation factor is a measure for the increase of the variance of the parameter estimates if an additional variable, given by exog_idx is added to the linear regression. It is a measure for multicollinearity of the design matrix, exog.

One recommendation is that if VIF is greater than 5, then the explanatory variable given by exog_idx is highly collinear with the other explanatory variables, and the parameter estimates will have large standard errors because of this.

from sklearn.feature_selection import SelectKBest, f_classif

> Scikit-learn API provides SelectKBest class for extracting best features of given dataset. The SelectKBest method selects the features according to the k highest score. By changing the 'score_func' parameter we can apply the method for both classification and regression data. Selecting best features is important process when we prepare a large dataset for training. It helps us to eliminate less important part of the data and reduce a training time.

from sklearn.preprocessing import PowerTransformer

Apply a power transform feature wise to make data more Gaussian-like.

Power transforms are a family of parametric, monotonic transformations that are applied to make data more Gaussian-like. This is useful for modeling issues related to heteroscedasticity (non-constant variance), or other situations where normality is desired.

Currently, PowerTransformer supports the Box-Cox transform and the Yeo-Johnson transform. The optimal parameter for stabilizing variance and minimizing skewness is estimated through maximum likelihood.

from sklearn.decomposition import PCA

- **PCA Description**: PCA is an unsupervised pre-processing task that is carried out before applying any ML algorithm. PCA is based on “orthogonal linear transformation” which is a mathematical technique to project the attributes of a data set onto a new coordinate system. The attribute which describes the most variance is called the first principal component and is placed at the first coordinate.
- Similarly, the attribute which stands second in describing variance is called a second principal component and so on. In short, the complete dataset can be expressed in terms of principal components. Usually, more than 90% of the variance is explained by two/three principal components.

- Principal component analysis, or PCA, thus converts data from high dimensional space to low dimensional space by selecting the most important attributes that capture maximum information about the dataset.

Importing algorithms

from sklearn.linear_model import LinearRegression

It is one of the most-used regression algorithms in Machine Learning. A significant variable from the data set is chosen to predict the output variables (future values). Linear regression algorithm is used if the labels are continuous, like the number of flights daily from an airport, etc. The representation of linear regression is y = b*x + c.

from sklearn.linear_model import Ridge,Lasso,RidgeCV,LassoCV

Ridge Regression is another popularly used linear regression algorithm in Machine Learning. If only one independent variable is being used to predict the output, it will be termed as a linear regression ML algorithm. ML experts prefer Ridge regression as it minimizes the loss encountered in linear regression (discussed above). In place of OLS (Ordinary Least Squares), the output values are predicted by a ridge estimator in ridge regression. The above-discussed linear regression uses OLS to predict the output values.

Lasso (Least Absolute Shrinkage and Selection Operator) regression is another widely used linear ML regression (one input variable). The sum of coefficient values is penalized in lasso regression to avoid prediction errors. The determination coefficients in lasso regression are reduced towards zero by using the technique 'shrinkage'. The regression coefficients are reduced by lasso regression to make them fit perfectly with various datasets. Besides ML, the lasso algorithm is also used for regression in Data Mining.

from sklearn.tree import DecisionTreeRegressor

Non-linear regression in Machine Learning can be done with the help of decision tree regression. The main function of the decision tree regression algorithm is to split the dataset into smaller sets. The subsets of the dataset are created to plot the value of any data point that connects to the problem statement. The splitting of the data set by this algorithm results in a decision tree that has decision and leaf nodes. ML experts prefer this model in cases where there is not enough change in the data set.

from sklearn.ensemble import RandomForestRegressor

Random forest is also a widely-used algorithm for non-linear regression in Machine Learning. Unlike decision tree regression (single tree), a random forest uses multiple decision trees for predicting the output. Random data points are selected from the given dataset (say k data points are selected), and a decision tree is built with them via this algorithm. Several decision trees are then modeled that predict the value of any new data point.

from sklearn.svm import SVR

SVM can be placed under both linear and non-linear types of regression in ML. The use cases of SVM can range from image processing and segmentation, predicting stock market patterns, text categorization, etc. When you have to identify the output in a multidimensional space, the SVM algorithm is used. In a multidimensional space, the data points are not represented as a point in a 2D plot. The data points are represented as a vector in a multidimensional space.

from sklearn.neighbors import KNeighborsRegressor

KNN model is popularly used for non-linear regression in Machine Learning. KNN (K Nearest Neighbours) follows an easy implementation approach for non-linear regression in Machine Learning. KNN assumes that the new data point is similar to the existing data points. The new data point is compared to the existing categories and is placed under a relatable category. The average value of the k nearest neighbors is taken as the input in this algorithm. The neighbors in KNN models are given a particular weight that defines their contribution to the average value.

from sklearn.ensemble import AdaBoostRegressor

An AdaBoost [1] regressor is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction. As such, subsequent regressors focus more on difficult cases.

Importing metrics

from sklearn import metrics

from sklearn.metrics import r2_score,mean_squared_error

from sklearn.metrics import mean_absolute_error

Mean squared deviation (**MSD**) of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value.

As RMSE is clear by the name itself, that it is a simple square root of mean squared error.

R2 score is a metric that tells the performance of your model, not the loss in an absolute sense that how many wells did your model perform.

In contrast, MAE and MSE depend on the context as we have seen whereas the R2 score is independent of context.

So, with help of R squared we have a baseline model to compare a model which none of the other metrics provides. The same we have in classification problems which we call a threshold which is fixed at 0.5. So basically R2 squared calculates how must regression line is better than a mean line.

Hence, R2 squared is also known as Coefficient of Determination or sometimes also known as Goodness of fit.

Removing warnings

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

Spliting data and hyperparameter tuning

from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

The train_test_split() method is used to split our data into train and test sets.

GridSearchCV is a technique to search through the best parameter values from the given set of the grid of parameters. It is basically a cross-validation method. the model and the parameters are required to be fed in. Best parameter values are extracted and then the predictions are made.

Randomized search on hyper parameters. RandomizedSearchCV implements a "fit" and a "score" method. It also implements "score_samples", "predict", "predict_proba", "decision_function", "transform" and "inverse_transform" if they are implemented in the estimator used.

Pipelining

from sklearn.pipeline import Pipeline

The purpose of the pipeline is to assemble several steps that can be cross-validated together while setting different parameters. For this, it enables setting parameters of the various steps using their names and the parameter name separated by a '__', as in the example below. A step's estimator may be replaced entirely by setting the parameter with its name to another estimator, or a transformer removed by setting it to 'pass-through ' or None.

Model/s Development and Evaluation

- Problem-solving approach taken into consideration:
  1. Exploratory data analysis.
  2. Treatment of null values.
  3. Feature Engineering
  4. Analysis using Plots:
     - Univariate analysis
     - Buvariate analysis
     - Multivariate analysis
  5. Corelation and multicollinearity analysis
  6. Best Feature selection and PCA
  7. Model preprocessing, instantiating, training, testing, Hyperparameter tuning
  8. Best model selection and Pipelining
  9. Deployment

Exploratory Data Analysis:

Functions and methods used:

1. Reading csv files using pd.read_csv function.

2. Inspecting 1st and last rows using .head and .tail method.

```
train_file_loc = r"C:\Users\Lenovo\OneDrive\Desktop\Filp Robo worksheets and projects\Project-Housing_splitted\train.csv"
df = pd.read_csv(train_file_loc)
```

```
print('First 10 rows')
df.head(10)
```

First 10 rows

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	M
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	
5	1197	60	RL	58.0	14054	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
6	561	20	RL	NaN	11341	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
7	1041	20	RL	88.0	13125	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	GdPrv	NaN	0	
8	503	20	RL	70.0	9170	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	GdPrv	Shed	400	
9	576	50	RL	80.0	8480	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	

10 rows × 81 columns

3. Inspecting rows and columns in dataset:

```
df.shape
```

```
(1168, 81)
```

```
df.columns
```

```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

4. Data description: of numerical type columns

```
desc = df.describe().T
desc['range']=desc['max']-desc['min']
desc
```

	count	mean	std	min	25%	50%	75%	max	range
Id	1168.0	724.136130	416.159877	1.0	360.50	714.5	1079.5	1460.0	1459.0
MSSubClass	1168.0	56.767979	41.940650	20.0	20.00	50.0	70.0	190.0	170.0
LotFrontage	954.0	70.988470	24.828750	21.0	60.00	70.0	80.0	313.0	292.0
LotArea	1168.0	10484.749144	8957.442311	1300.0	7621.50	9522.5	11515.5	164660.0	163360.0
OverallQual	1168.0	6.104452	1.390153	1.0	5.00	6.0	7.0	10.0	9.0
OverallCond	1168.0	5.595890	1.124343	1.0	5.00	5.0	6.0	9.0	8.0
YearBuilt	1168.0	1970.930651	30.145255	1875.0	1954.00	1972.0	2000.0	2010.0	135.0
YearRemodAdd	1168.0	1984.758562	20.785185	1950.0	1966.00	1993.0	2004.0	2010.0	60.0
MasVnrArea	1161.0	102.310078	182.595606	0.0	0.00	0.0	160.0	1600.0	1600.0
BsmtFinSF1	1168.0	444.726027	462.664785	0.0	0.00	385.5	714.5	5644.0	5644.0
BsmtFinSF2	1168.0	46.647260	163.520016	0.0	0.00	0.0	0.0	1474.0	1474.0
BsmtUnfSF	1168.0	569.721747	449.375525	0.0	216.00	474.0	816.0	2336.0	2336.0
TotalBsmtSF	1168.0	1061.095034	442.272249	0.0	799.00	1005.5	1291.5	6110.0	6110.0
1stFlrSF	1168.0	1169.860445	391.161983	334.0	892.00	1096.5	1392.0	4692.0	4358.0
2ndFlrSF	1168.0	348.826199	439.696370	0.0	0.00	0.0	729.0	2065.0	2065.0
LowQualFinSF	1168.0	6.380137	50.892844	0.0	0.00	0.0	0.0	572.0	572.0
GrLivArea	1168.0	1525.066781	528.042957	334.0	1143.25	1468.5	1795.0	5642.0	5308.0
BsmtFullBath	1168.0	0.425514	0.521615	0.0	0.00	0.0	1.0	3.0	3.0
BsmtHalfBath	1168.0	0.055651	0.236699	0.0	0.00	0.0	0.0	2.0	2.0
FullBath	1168.0	1.562500	0.551882	0.0	1.00	2.0	2.0	3.0	3.0
HalfBath	1168.0	0.388699	0.504929	0.0	0.00	0.0	1.0	2.0	2.0
BedroomAbvGr	1168.0	2.884418	0.817229	0.0	2.00	3.0	3.0	8.0	8.0

Of categorical columns:

```
df.describe(include='object').T
```

	count	unique	top	freq
MSZoning	1168	5	RL	928
Street	1168	2	Pave	1164
Alley	77	2	Grvl	41
LotShape	1168	4	Reg	740
LandContour	1168	4	Lvl	1046
Utilities	1168	1	AllPub	1168
LotConfig	1168	5	Inside	842
LandSlope	1168	3	Gtl	1105
Neighborhood	1168	25	NAmes	182
Condition1	1168	9	Norm	1005
Condition2	1168	8	Norm	1154
BldgType	1168	5	1Fam	981
HouseStyle	1168	8	1Story	578
RoofStyle	1168	6	Gable	915
RoofMatl	1168	8	CompShg	1144
Exterior1st	1168	14	VinylSd	396
Exterior2nd	1168	15	VinylSd	387
MasVnrType	1161	4	None	696
ExterQual	1168	4	TA	717
ExterCond	1168	5	TA	1022
Foundation	1168	6	CBlock	516
BsmtQual	1138	4	TA	517
BsmtCond	1138	4	TA	1041

5. Feature Engineering,/Selecting categorical and numerical columns:

```
# Feature engineering

avgpr = []
for i in df.SalePrice:
    if i < 181477:
        avgpr.append('Below avg price')
    elif i > 181477 and i < 550000:
        avgpr.append('Avg priced')
    else:
        avgpr.append('High priced')

df['Price Range'] = avgpr
```

```
cont_data = df.select_dtypes(include=['int64','float64'])

cat_data= df.select_dtypes(include=['object'])

cont_columns = cont_data.columns

cat_columns = cat_data.columns
```

```
df[cont_columns].isnull().sum()
```

```
Id               0
MSSubClass       0
LotFrontage    214
LotArea          0
OverallQual      0
OverallCond      0
YearBuilt        0
YearRemodAdd     0
MasVnrArea       7
BsmtFinSF1       0
BsmtFinSF2       0
BsmtUnfSF        0
TotalBsmtSF      0
1stFlrSF         0
2ndFlrSF         0
```

6. Unique values/value counts/Treating null values:

```
from scipy import stats

for i in cat_columns:
    print('For',i,', most frequent value is: ',stats.mode(df[i]),'\n')
```

```
for i in cat_columns:
    print('For column',i,'unique values are: ',df[i].unique())
    print('For column',i,'count of unique values are: ',df[i].nunique(),'\n\n')
```

```
for i in cat_columns:
    print('For column --',i,'-- value counts are: \n',df[i].value_counts(),'\n\n')
```

```
# Treating null values

df['GarageType'].fillna('',axis=0,inplace=True)
df['GarageFinish'].fillna('',axis=0,inplace=True)
df['GarageQual'].fillna('',axis=0,inplace=True)
df['GarageCond'].fillna('',axis=0,inplace=True)
df['BsmtQual'].fillna('Gd',axis=0,inplace=True)
df['BsmtCond'].fillna('TA',axis=0,inplace=True)
df['BsmtExposure'].fillna('No',axis=0,inplace=True)
df['BsmtFinType1'].fillna('Unf',axis=0,inplace=True)
df['BsmtFinType2'].fillna('Unf',axis=0,inplace=True)
df['MasVnrType'].fillna('BrkFace',axis=0,inplace=True)
```

```
for i in df.columns:
    print('Null values in column: ',i,':',df[i].isnull().sum(),'--- of type',df[i].dtype)
```

```
Null values in column:  Id : 0 --- of type int64
Null values in column:  MSSubClass : 0 --- of type int64
Null values in column:  MSZoning : 0 --- of type object
Null values in column:  LotFrontage : 214 --- of type float64
Null values in column:  LotArea : 0 --- of type int64
Null values in column:  Street : 0 --- of type object
Null values in column:  LotShape : 0 --- of type object
Null values in column:  LandContour : 0 --- of type object
Null values in column:  Utilities : 0 --- of type object
```

```
# import the KNNimputer class
from sklearn.impute import KNNImputer


# creating a data frame from the list
Before_imputation = df[['LotFrontage','MasVnrArea','GarageYrBlt']]
#print dataset before imputaion
print("Data Before performing imputation\n",Before_imputation)

# create an object for KNNImputer
imputer = KNNImputer(n_neighbors=10)
After_imputation = imputer.fit_transform(Before_imputation)
# print dataset after performing the operation
print("\n\nAfter performing imputation\n",After_imputation)
```

```
imputed_cont = pd.DataFrame(After_imputation,columns=['LotFrontageIMP','MasVnrAreaIMP','GarageYrBltIMP'])
```

```
imputed_cont
```

	LotFrontageIMP	MasVnrAreaIMP	GarageYrBltIMP
0	58.2	0.0	1977.0
1	95.0	0.0	1970.0
2	92.0	0.0	1997.0
3	105.0	480.0	1977.0
4	59.1	126.0	1977.0
...	...	...	...
1163	75.4	31.0	1970.0
1164	67.0	0.0	2002.0
1165	24.0	0.0	1976.0
1166	50.0	0.0	1920.0
1167	55.8	0.0	2002.0

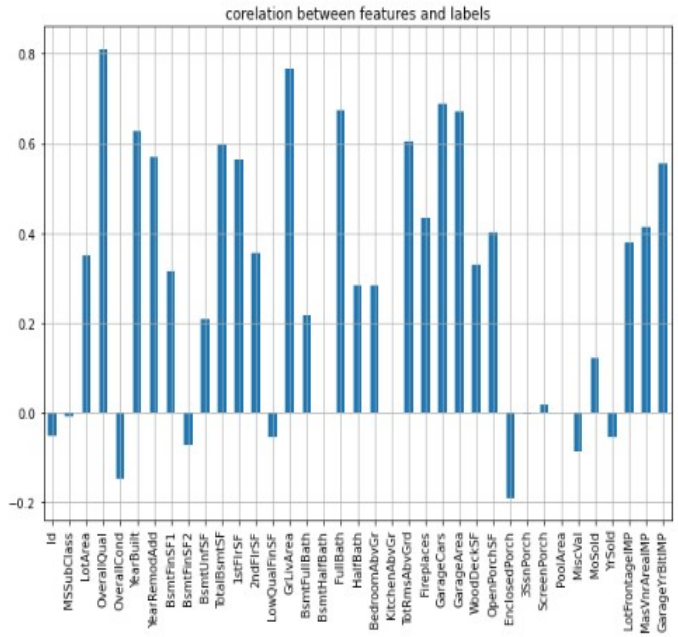
7. Checking skewness:

```
df.skew().sort_values()
```

```
GarageYrBltIMP     -0.693190
YearBuilt          -0.583781
YearRemodAdd       -0.577825
GarageCars         -0.342937
BedroomAbvGr       -0.159870
GarageArea         -0.034515
Id                 -0.004853
PoolArea            0.000000
BsmtHalfBath        0.000000
KitchenAbvGr        0.000000
OverallQual         0.014959
FullBath            0.032689
YrSold              0.112178
TotalBsmtSF         0.121094
LotFrontageIMP      0.140987
MoSold              0.277080
TotRmsAbvGrd        0.378751
BsmtFullBath        0.404771
HalfBath            0.496141
GrLivArea           0.551514
BsmtFinSF1          0.574263
LotArea             0.590032
Fireplaces          0.598419
1stFlrSF            0.645319
OverallCond         0.728678
2ndFlrSF            0.737746
BsmtUnfSF           0.742732
SalePrice           0.830901
WoodDeckSF          0.945520
MSSubClass          1.394020
OpenPorchSF         1.444519
MasVnrAreaIMP       1.686176
EnclosedPorch       2.864272
BsmtFinSF2          4.753041
ScreenPorch         4.758951
MiscVal             8.100346
LowQualFinSF       15.502663
3SsnPorch          28.337255
dtype: float64
```

8. Corelation between features and Sales Price:

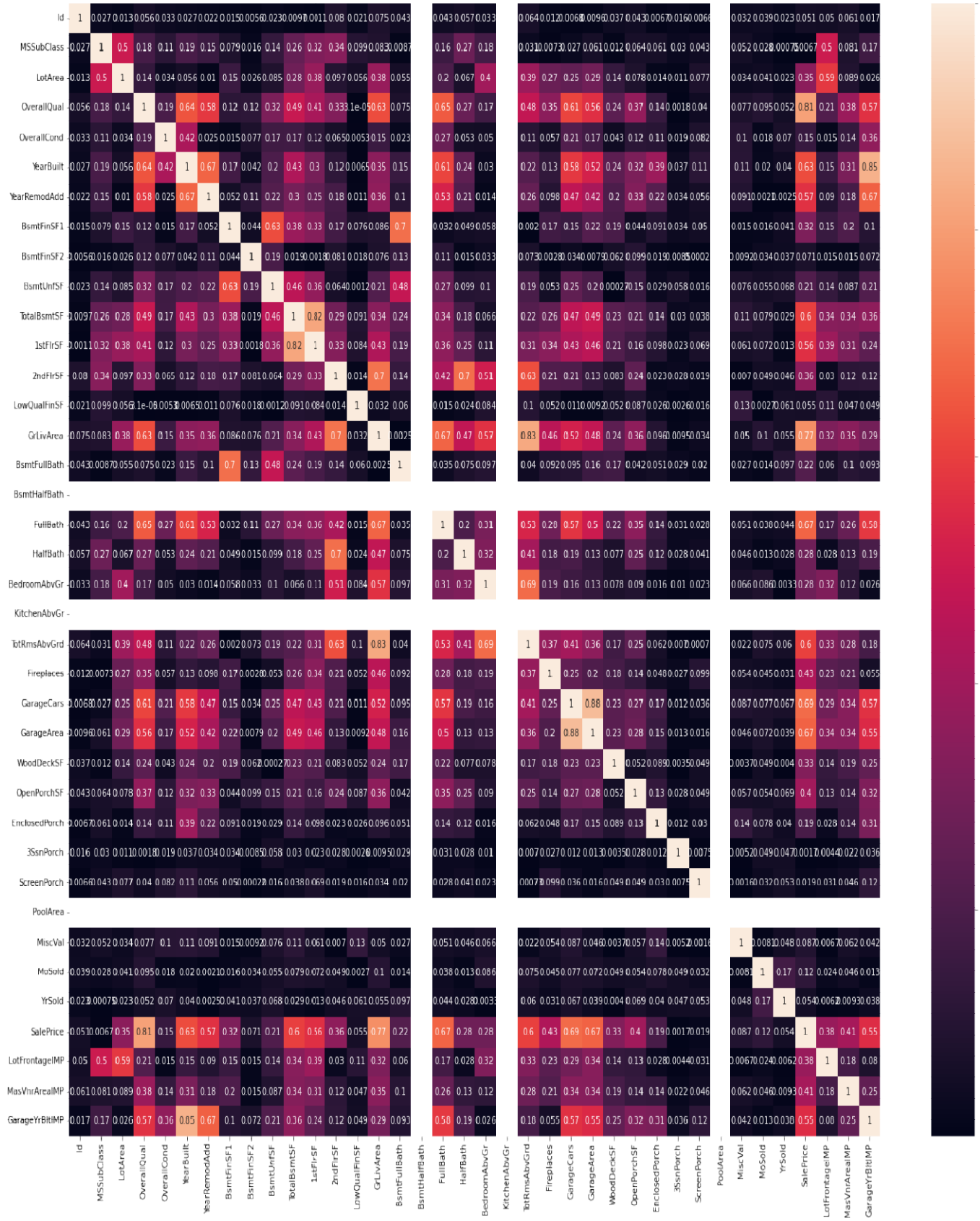
```
# Correlation between features and label:

# Replacing attrition column values:

df.drop(columns = 'SalePrice',axis = 1).corrwith(df.SalePrice).plot(kind='bar',grid=True,figsize=(10,7),title='corelation betwee
plt.show()
```



9. <u>Heatmap</u>:

SYNTAX

```
df_corr = df.corr().abs()

plt.figure(figsize=(25,22))

sns.heatmap(df_corr,annot=True,annot_kws={'size':10})

plt.show()
```


Id
MSSubClass
LotArea
OverallQual
OverallCond
YearBuilt
YearRemodAdd
BsmtFinSF1
BsmtFinSF2
BsmtUnfSF
TotalBsmtSF
1stFlrSF
2ndFlrSF
LowQualFinSF
GrLivArea
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageCars
GarageArea
WoodDeckSF
OpenPorchSF
EnclosedPorch
3SsnPorch
ScreenPorch
PoolArea
MiscVal
MoSold
YrSold
SalePrice
LotFrontageIMP
MasVnrArealMP
GarageYrBltIMP

10. Variance inflation factor:

SYNTAX

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()

vif['vif'] = [variance_inflation_factor(df[cont_columns],i) for i in
range(df[cont_columns].shape[1])]

vif['features'] = df[cont_columns].columns

vif
```

	vif	features
0	1.035992e+00	Id
1	2.358587e+00	MSSubClass
2	2.078867e+00	LotArea
3	4.046995e+00	OverallQual
4	1.658520e+00	OverallCond
5	6.857407e+00	YearBuilt
6	2.789055e+00	YearRemodAdd
7	inf	BsmtFinSF1
8	inf	BsmtFinSF2
9	inf	BsmtUnfSF
10	inf	TotalBsmtSF
11	inf	1stFlrSF
12	inf	2ndFlrSF
13	inf	LowQualFinSF
14	inf	GrLivArea
15	2.189198e+00	BsmtFullBath
16	NaN	BsmtHalfBath
17	3.411463e+00	FullBath
18	2.523953e+00	HalfBath
19	2.555100e+00	BedroomAbvGr
20	2.393743e+06	KitchenAbvGr
21	4.495846e+00	TotRmsAbvGrd
22	1.486247e+00	Fireplaces
23	5.439587e+00	GarageCars

11. Feature and Label Selection/Finding best features:

```
X = df.drop('SalePrice',axis = 1)
y = df.SalePrice
```

```
Xcont_data = X.select_dtypes(include=['int64','float64'])

Xcat_data= X.select_dtypes(include=['object'])

Xcont_columns = Xcont_data.columns

Xcat_columns = Xcat_data.columns
```

```
# Using SelectKBest feature selection method: # It is one of the feature selection method:
# when there are lot of features and you cant graphically analyse , short way , selectKbest can be
# used:

from sklearn.feature_selection import SelectKBest, f_classif

# SelectKBest uses f_classif function to select best features where f_classif uses ANOVA test

best_features = SelectKBest(score_func = f_classif,k = 15)
fit = best_features.fit(Xcont_data,y)

data_scores = pd.DataFrame(fit.scores_)
data_columns = pd.DataFrame(Xcont_data.columns)

features_score = pd.concat([data_columns,data_scores],axis=1)

features_score.columns = ['Features','Scores']
print(features_score.nlargest(15,'Scores'),'\n') # print 15 best features

# Here we are getting top 15 features we got based on f_classify that uses ANOVA test of statistics.
```

	Features	Scores
28	3SsnPorch	inf
13	LowQualFinSF	4.621270
3	OverallQual	4.478420
14	GrLivArea	3.671677
17	FullBath	3.491706
5	YearBuilt	2.474115
24	GarageArea	2.395244
23	GarageCars	2.376853
11	1stFlrSF	2.237095
10	TotalBsmtSF	2.182005
21	TotRmsAbvGrd	2.138774
36	GarageYrBltIMP	1.902150
35	MasVnrAreaIMP	1.816586
6	YearRemodAdd	1.669807
12	2ndFlrSF	1.510321

12. Power Transform and PCA:

SYNTAX

from sklearn.preprocessing import PowerTransformer

pt = PowerTransformer()

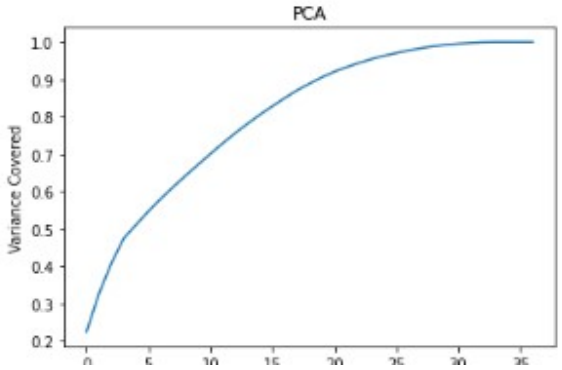
X_cont_trans = pt.fit_transform(Xcont_data)

PCA: For numerical columns in features:

```
# Splitting our data to training data and testing data
# x_train,x_test,y_train,y_test

x_train1,x_test1,y_train1,y_test1 = train_test_split(princi_compf,y,test_size=0.20,random_state=1)

# Here we are keeping training data as our scalled data and testing data as our label or target.
```

```
from sklearn.decomposition import PCA
# Using PCA i.e. Principal Component Analysis that is a diamensionallity reduction technique:

pca = PCA()
pca.fit_transform(X_cont_trans)
```

```
# Using Scree Plot to identify best components:

plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Principal Components')
plt.ylabel('Variance Covered')
plt.title('PCA')
plt.show()
```

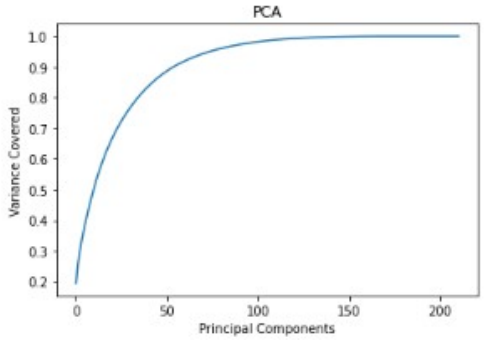


For categorical columns in features:

```
Xcat_dums = pd.get_dummies(Xcat_data)
```

```
from sklearn.decomposition import PCA
# Using PCA i.e. Principal Component Analysis that is a diamensionallity reduction technique:

pca1 = PCA()
pca1.fit_transform(Xcat_dums)
```

```
# Using Scree Plot to identify best components:

plt.figure()
plt.plot(np.cumsum(pca1.explained_variance_ratio_))
plt.xlabel('Principal Components')
plt.ylabel('Variance Covered')
plt.title('PCA')
plt.show()
```




```
pca1 = PCA(n_components=100)
new_pcomp1 = pca1.fit_transform(Xcat_dums)
princi_comp1 = pd.DataFrame(new_pcomp1)
princi_comp1
```

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	
0	-0.819565	1.262585	0.313982	0.061500	-1.327876	0.455167	1.038028	-0.135120	-0.215133	-0.831659	...	0.025788	0.002919	-0.047128	-0.041303	-(
1	1.016840	0.056432	1.062288	0.878379	-0.300218	-1.008067	0.482156	0.921480	0.238571	0.112348	...	0.045933	-0.027670	0.110380	0.056815	(
2	0.144027	1.873207	0.065780	-0.316458	0.658555	0.207576	0.337532	-0.038442	0.125672	-0.759699	...	0.145404	0.115313	0.077044	-0.122778	-(
3	2.345178	-0.947921	1.058606	-0.570890	0.329790	0.554575	0.940963	0.809971	-0.758276	0.621009	...	-0.004429	-0.051107	0.067191	-0.010599	-(
4	-1.471372	1.043251	0.356622	0.024229	0.424961	-0.318671	-0.156278	0.502175	-0.326980	-0.228684	...	0.080706	-0.155728	0.143868	0.110934	-(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
798	-1.890720	0.788870	0.100360	0.225768	0.479863	-1.009006	0.676650	0.330604	-0.108328	-0.607278	...	-0.030304	-0.000011	0.085129	0.117191	-(
799	-0.101285	-0.875808	0.005162	-0.602363	-0.370586	-0.001375	-0.423079	-0.273788	-0.582278	-0.730439	...	-0.018332	0.018431	-0.063336	-0.007008	(
800	-1.182026	0.297514	0.227738	0.930553	-0.707682	0.505115	0.474875	-0.416700	-0.726905	0.331744	...	-0.033879	-0.069712	0.089969	-0.174674	-(
801	-2.100508	-1.554105	0.289553	0.212612	0.657225	0.469583	-0.561867	0.092889	0.077714	0.132980	...	-0.033537	0.105642	-0.405957	0.028887	-(
802	2.202082	-0.252655	1.038456	0.533872	-0.104461	-0.160912	-0.120657	0.242624	-0.800961	-0.722941	...	0.059481	0.026686	0.004059	0.002724	-(

803 rows × 100 columns

```
X_F = pd.concat([princi_comp,princi_comp1],axis=1)
```

ALGORITHMS USED TO PREDICT PRICE OF HOUSES:

- Linear Regression
- Ridge and Lasso regression
- Decission Tree Regressor
- Random Forest Regressor (with Hyperparameter tuning)
- K-Neighbours Regressor (with Hyperparameter tuning)
- Support Vector Regressor
- AdaBoost Regressor (with Hyperparameter tuning)

Run and Evaluating selected models

1. Linear Regression:

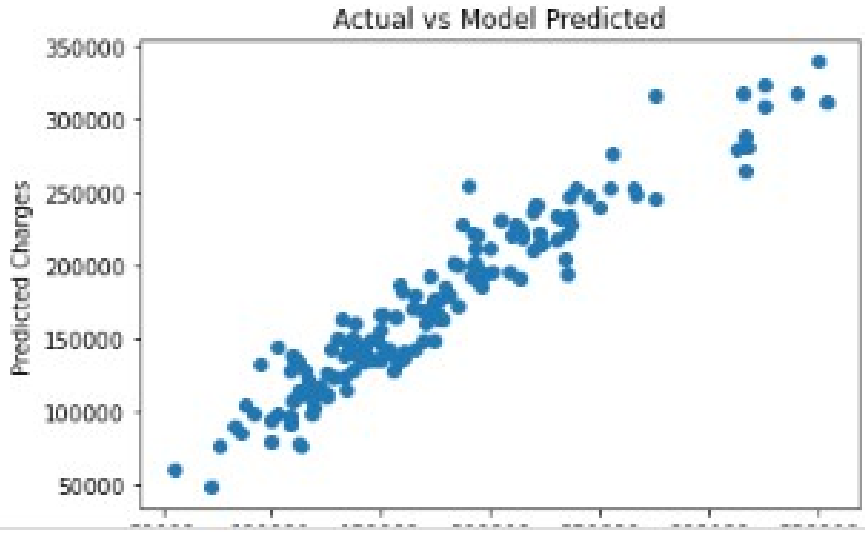
```
# MOdel instantiating and training

rm = LinearRegression()
rm.fit(x_train,y_train)
# here we will pass training data
```

```
# Testing our model with Adjusted R2 Square:

# on training data

rm.score(x_train,y_train)
```

```
0.9431110385753498
```

```
# PLotting and visualizing

y_pred = rm.predict(x_test)
```

```
plt.scatter(y_test,y_pred)
plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.title('Actual vs Model Predicted')
plt.show()
```



2. Ridge and lasso regression:

```
from sklearn.linear_model import Ridge,Lasso,RidgeCV,LassoCV

lassocv = LassoCV(alphas = None , max_iter = 100, normalize = True)
lassocv.fit(x_train,y_train)
```

```
LassoCV(max_iter=100, normalize=True)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
# Best alpha parameter
alpha = lassocv.alpha_ # Best alpha rate
alpha
```

```
16.69869451758854
```

```
# Now since we have the best parameter, Lasso regression will be used:
lasso_reg = Lasso(alpha)
lasso_reg.fit(x_train,y_train)
# i.e. when model is training it will learn at this speed 6....
```

```
Lasso(alpha=16.69869451758854)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
lasso_reg.score(x_test,y_test)
```

```
0.9208166318434565
```

```
# Ridge Method:

ridgecv = RidgeCV(alphas = np.arange(0.001,0.1,0.01),normalize = True)
ridgecv.fit(x_train,y_train)
```

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

```
: ridge_model.score(x_test,y_test)
```

```
: 0.9152887199012137
```

```
: #model performance

  print("MAE: ", metrics.mean_absolute_error(y_test, y_pred))
  print("MSE: ", metrics.mean_squared_error(y_test, y_pred))
  print("RMSE: ", metrics.mean_squared_error(y_test, y_pred, squared=False))
  print("R2: ", metrics.r2_score(y_test, y_pred), "\n")
  print("Score: ", rm.score(x_test, y_test))
```

```
MAE:  13717.077869572711
MSE:  324910213.48318124
RMSE:  18025.265975379705
R2:  0.9147844190196017

Score:  0.9147844190196017
```

```
: print(rm.coef_)
  print(f"{rm.intercept_:.2f}")
```

```
[-1.20834954e+04 -2.48932029e+03 -3.59251465e+03 -9.16063428e+03
 -1.40050097e+03  1.38499418e+03  7.76704763e+03  3.41566024e+03
  4.82336239e+03  5.84046858e+03 -1.99487065e+02 -7.53222410e+03
  6.21969247e+03 -1.71531277e+03 -4.19446263e+03 -1.27062381e+04
 -6.16627977e+03  5.98660225e+03 -5.58923387e+03  3.51062909e+02
 -5.93816992e+03 -3.50239431e+03 -7.77066831e+03 -4.72913445e+03
  5.16892378e+02  4.26910871e+03 -3.73324538e+03 -1.30692176e+03
 -1.39152411e+03  2.55001282e+03  4.20772951e+03 -8.56683164e+03
 -4.43904191e+03  6.37515959e+03 -2.37867674e+16  1.27634125e+04
  8.58231106e+02 -5.92251894e+03 -2.69005342e+03  2.78735780e+04
  1.44340559e+03 -6.85422299e+03  1.00368537e+04  1.03926662e+04
  2.45693896e+03 -4.87153891e+03  2.07730665e+03 -5.87231220e+03
  5.38101135e+03 -3.29883381e+03  3.38614913e+03 -2.90421620e+03
 -6.62451586e+03  4.27941642e+03  3.02264681e+03 -9.21062785e+03
 -1.21506666e+04 -8.01360898e+03 -8.39704888e+03  1.31067735e+03
  4.77378056e+03 -8.64313312e+03 -4.44877821e+03 -8.28936669e+03
 -1.10663159e+04 -2.06213917e+03  4.90730534e+03  6.60274630e+03
```

3. Decision Tree and Random Forest Models:

```
# Using decision tree:

from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor()

model.fit(x_train, y_train)

y_preddt = model.predict(x_test)

r2_score(y_test,y_preddt)
```

```
0.585855955277989
```

```
# Random forest:

from sklearn.ensemble import RandomForestRegressor
regressor_rf = RandomForestRegressor(n_estimators = 100)

regressor_rf.fit(x_train, y_train)

lr_normal_rf = regressor_rf.score(x_train, y_train)

lr_normal_rf
```

```
0.9780425851263904
```

```
y_predrf = regressor_rf.predict(x_test)

lr_normal_rf_test = regressor_rf.score(x_test, y_test)

lr_normal_rf_test

mse_lr_normal_rf  = mean_absolute_error(y_test, y_predrf)

mse_lr_normal_rf
```

```
14742.218074534161
```

```
# Model Evaluation: MAE , MSE , RMSE

from sklearn.metrics import mean_absolute_error,mean_squared_error

print("MAE: ", metrics.mean_absolute_error(y_test, y_predrf))
print("MSE: ", metrics.mean_squared_error(y_test, y_predrf))
print("RMSE: ", metrics.mean_squared_error(y_test, y_predrf, squared=False))
print("R2: ", metrics.r2_score(y_test, y_predrf), "\n")
print("Score: ", regressor_rf.score(x_test, y_predrf))
```

```
MAE:  14742.218074534161
MSE:  437828759.062105
RMSE:  20924.358032257645
R2:  0.8851687927152976

Score:  1.0
```

4. Support Vector machine and K-Nearest Neighbour:

```
# Using Support vector regressor:

# Fit the model over the training data
from sklearn.svm import SVR
svr = SVR()
svr.fit(x_train, y_train)

y_predsvr = svr.predict(x_test)

r2_score(y_test,y_predsvr)
```

```
-0.06275680040757048
```

```
# Innitiate k neighbour Regressor:

from sklearn.neighbors import KNeighborsRegressor

params = {'n_neighbors':[2,3,4,5,6,7,8,9]}

knn = KNeighborsRegressor()

knnmodel = GridSearchCV(knn, params, cv=5)
knnmodel.fit(x_train,y_train)
knnmodel.best_params_
```

```
{'n_neighbors': 7}
```

```
knn_best = KNeighborsRegressor(n_neighbors = 7)
knn_best.fit(x_train,y_train)
```

```
KNeighborsRegressor(n_neighbors=7)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
y_predknn = knn_best.predict(x_test)

r2_score(y_test,y_predknn)
```

```
0.8767186408745636
```

5. AdaBoost Regressor:

```
# Adaboost regressor

from sklearn.ensemble import AdaBoostRegressor
regr = AdaBoostRegressor(random_state=0, n_estimators=100)
regr.fit(x_train,y_train)
y_predada = regr.predict(x_test)
regr.score(x_test, y_predada)
```

```
1.0
```

```
print("MAE: ", metrics.mean_absolute_error(y_test, y_predada))
print("MSE: ", metrics.mean_squared_error(y_test, y_predada))
print("RMSE: ", metrics.mean_squared_error(y_test, y_predada, squared=False))
print("R2: ", metrics.r2_score(y_test, y_predada), "\n")
print("Score: ", regr.score(x_test, y_predada))
```

```
MAE:  17571.788782517036
MSE:  555857380.9305034
RMSE:  23576.627853247024
R2:  0.8542129250095508

Score:  1.0
```

```
ada=AdaBoostRegressor()
search_grid={'n_estimators':[500,1000,2000],'learning_rate':[.001,0.01,.1],'random_state':[1]}
search=GridSearchCV(estimator=ada,param_grid=search_grid,scoring='neg_mean_squared_error',n_jobs=1)
```

```
search.fit(x_train,y_train)
search.best_params_
```

```
{'learning_rate': 0.1, 'n_estimators': 1000, 'random_state': 1}
```

```
regr_f = AdaBoostRegressor(learning_rate= 0.1, n_estimators=1000, random_state=1)
regr_f.fit(x_train,y_train)
y_predada_f = regr.predict(x_test)
regr_f.score(x_test, y_predada_f)
```

```
0.9951554293305714
```

```
print("MAE: ", metrics.mean_absolute_error(y_test, y_predada_f))
print("MSE: ", metrics.mean_squared_error(y_test, y_predada_f))
print("RMSE: ", metrics.mean_squared_error(y_test, y_predada_f, squared=False))
print("R2: ", metrics.r2_score(y_test, y_predada_f), "\n")
print("Score: ", regr_f.score(x_test, y_predada_f))
```

```
MAE:  17571.788782517036
MSE:  555857380.9305034
RMSE:  23576.627853247024
R2:  0.8542129250095508

Score:  0.9951554293305714
```

```
regr.predict(x_test)
```

```
array([217744.11057692, 144607.70833333, 108518.12820513, 219082.45505618,
       175383.85314685, 208035.09126984, 122333.33333333, 161244.6875    ,
       152489.34246575, 226786.29559748, 187251.77572559, 171669.78571429,
       220251.94516971, 200384.61538462, 286530.60606061, 114666.86394558,
       153357.71212121, 124135.65625   , 134668.12903226, 121392.7037037 ,
       140414.11320755, 213410.39568345, 210413.48895899, 133060.11764706,
       130453.55072464, 165277.87878788, 121392.7037037 , 288931.62325581,
       187251.77572559, 268485.38709677, 259887.19081272, 303938.61842105,
       121392.7037037 , 115011.23076923, 130595.        , 161011.76811594,
        92726.5308642 , 273749.04      , 208035.09126984, 154106.34567901,
       228449.64739884, 288907.08571429, 223242.16842105, 294518.6223176 ,
       105490.        , 140414.11320755, 189188.65745856, 171141.9379845 ,
       177982.69662921, 106992.30769231, 147769.44827586, 111031.5       ,
       177982.69662921, 175090.11538462, 139080.        , 158818.42777778,
       222573.64179104, 123016.42424242,  81406.54166667, 121554.97297297,
       175383.85314685, 144832.54651163, 120191.36363636, 137128.74418605,
       220851.84984985, 174360.97222222, 234362.49602122, 191525.72625698,
       174862.        , 215291.72321429, 172025.9504644 , 121465.3030303 ,
       104564.125     , 167666.60674157, 121618.98550725, 130595.        ,
       220251.94516971, 213410.39568345, 119385.01388889, 220851.84984985,
       147769.44827586, 119385.01388889, 102402.5       , 232177.59433962,
       171141.9379845 , 131703.67692308, 116313.64545455, 178769.57142857,
       225690.21656051, 156060.52631579, 228572.77726218, 111266.08571429,
       228572.77726218, 169012.07692308, 275767.7765043 , 221585.91798942,
       152959.375     , 158127.48888889, 228446.09446254, 162183.21985816,
       133637.7253886 , 228449.64739884, 223242.16842105, 191746.2173913 ,
       227536.48958333, 140414.11320755, 215412.79607843, 121554.97297297,
       198892.74914089, 232177.59433962, 219225.24315068, 162183.21985816,
```

Evaluation Metrics Used

(MSE MAE RMSE R2_SCORE and SCORE for each regression algorithm is used as metrics and indicated in above screenshots)

Results/Observations over different Evaluation Metrics

Key Metrics for success

Metrics Used:

- Mean absolute error
- Mean Squared Error
- Root Mean Squared Error
- R-squared

Justification: The essential step in any machine learning model is to evaluate the accuracy of the model. The Mean Squared Error, Mean absolute error, Root Mean Squared Error, and R-Squared or Coefficient of determination metrics are used to evaluate the performance of the model in regression analysis.

Lets understand what these metrics do:

- The Mean absolute error represents the average of the absolute difference between the actual and predicted values in the dataset. It measures the average of the residuals in the dataset.

- Mean Squared Error represents the average of the squared difference between the original and predicted values in the data set. It measures the variance of the residuals.
- Root Mean Squared Error is the square root of Mean Squared error. It measures the standard deviation of residuals.
- The coefficient of determination or R-squared represents the proportion of the variance in the dependent variable which is explained by the linear regression

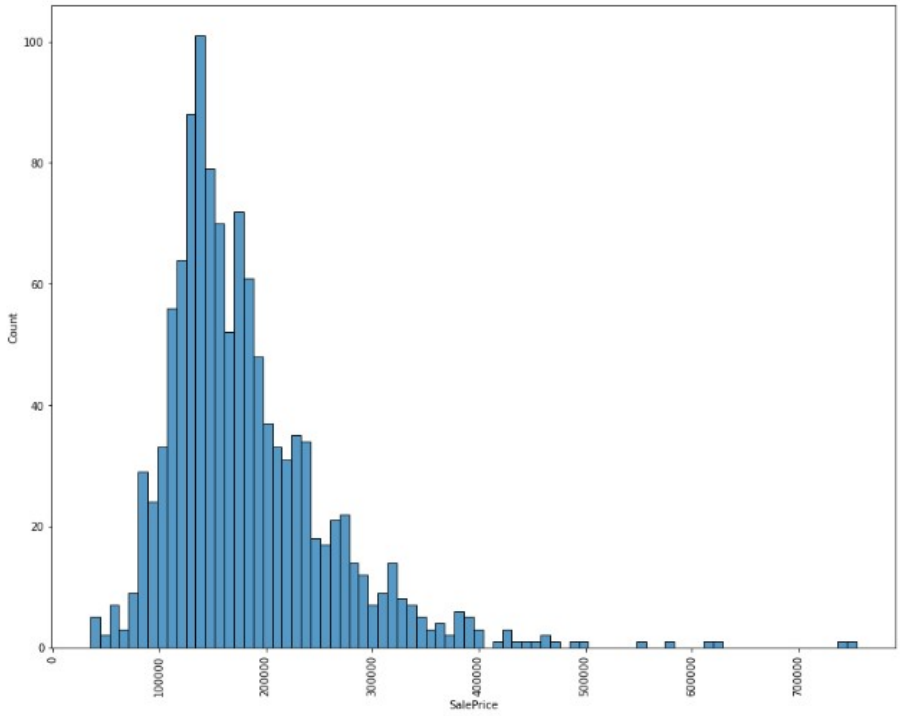
model. It is a scale-free score i.e. irrespective of the values being small or large, the value of R square will be less than one.

- RMSE and R- Squared quantifies how well a linear regression model fits a dataset. The RMSE tells how well a regression model can predict the value of a response variable in absolute terms while R- Squared tells how well the predictor variables can explain the variation in the response variable.

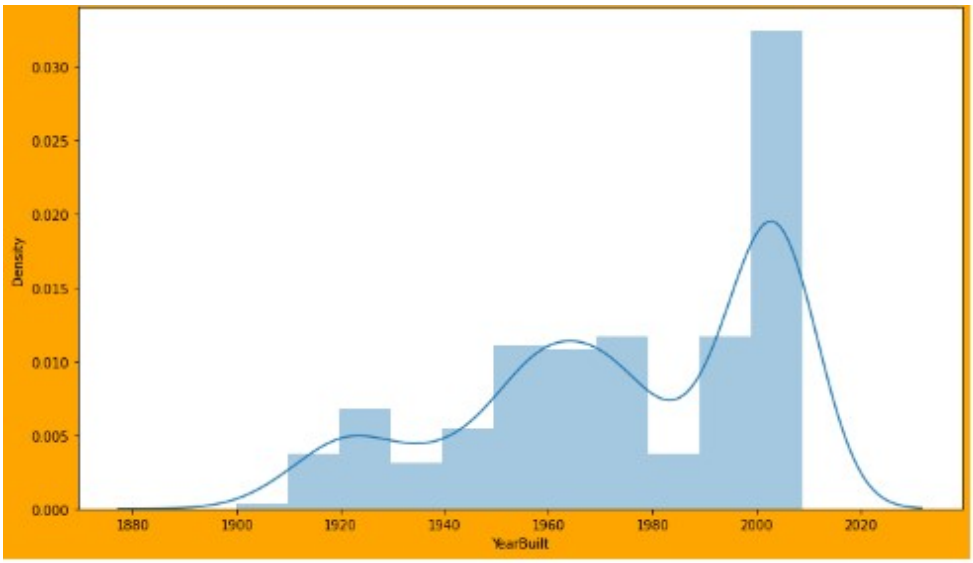
Visualizations

Plots used are:

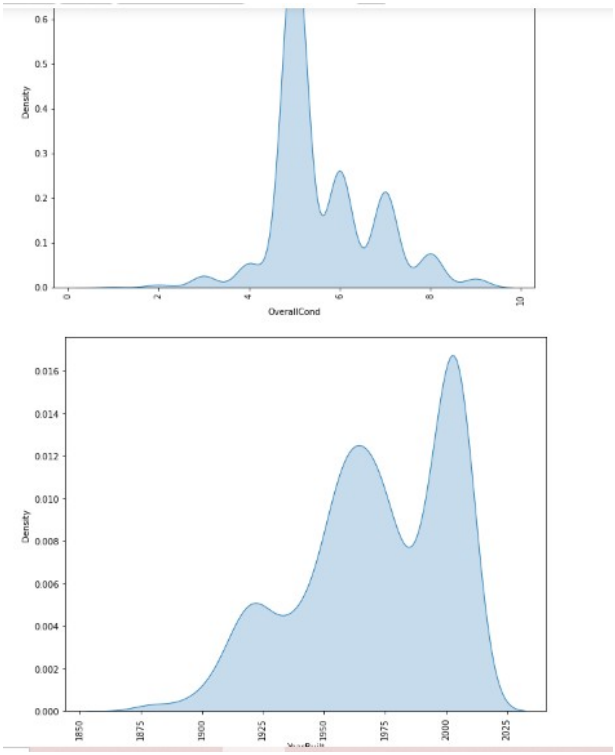
- Frequency analysis plots:
  1. Histogram plot.



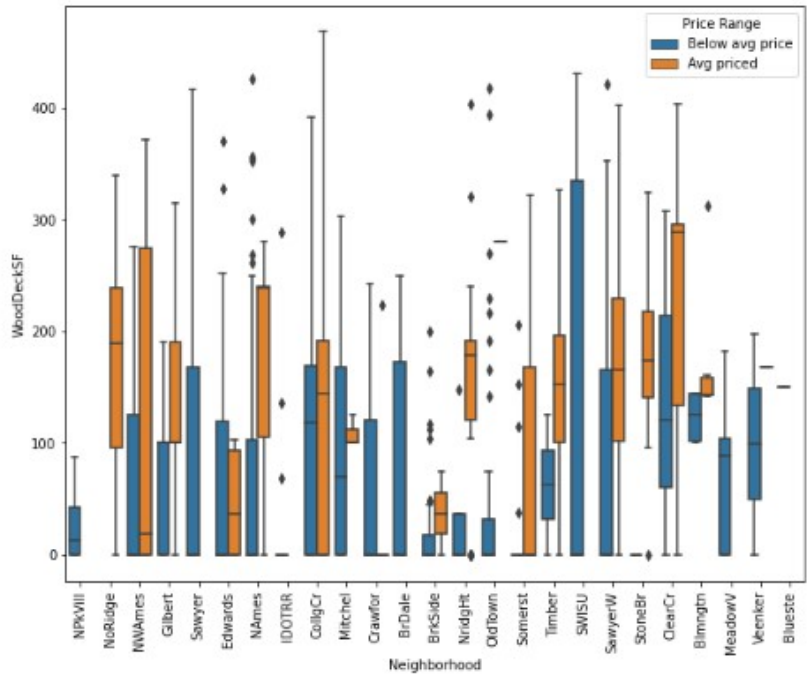
2. Distribution plot



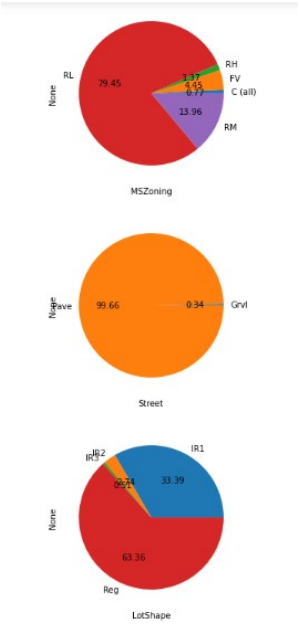
3. KDE plot



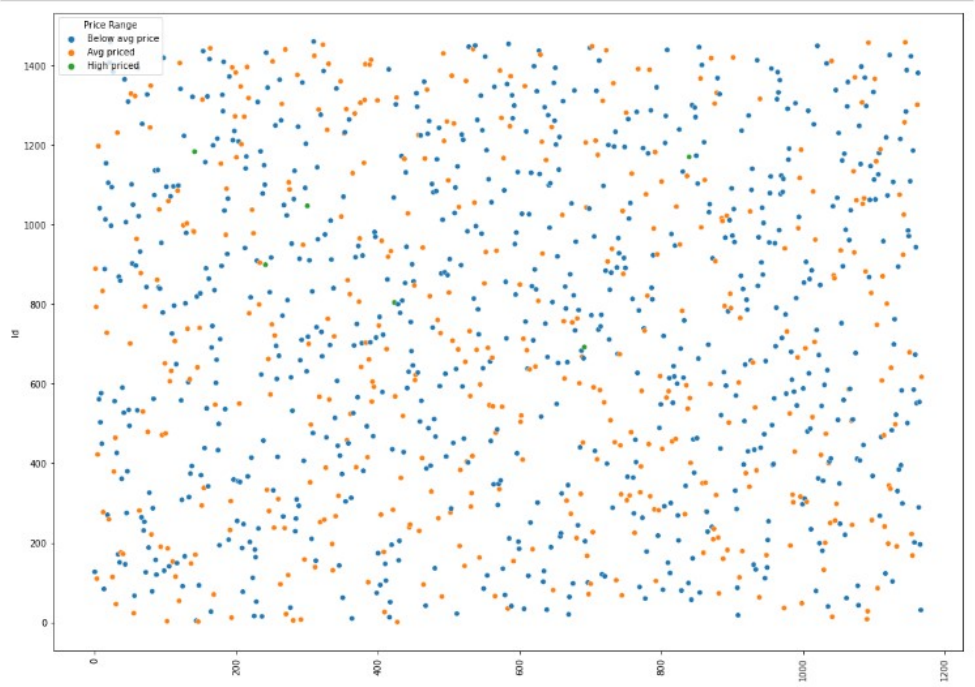
- Outlier presence analysis:
  1. Boxplots



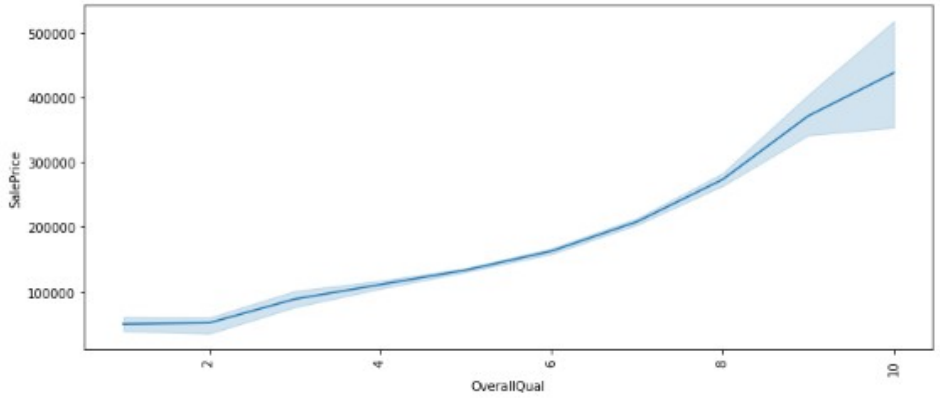
- Univariate Analysis:
  1. Pieplot

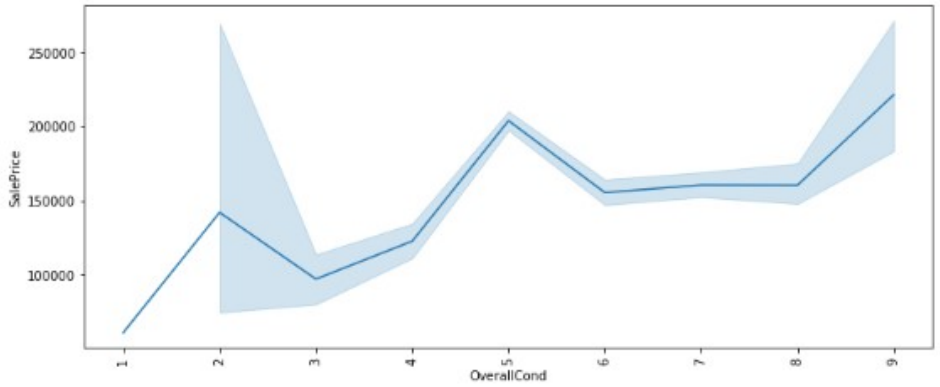


2. Scatterplot

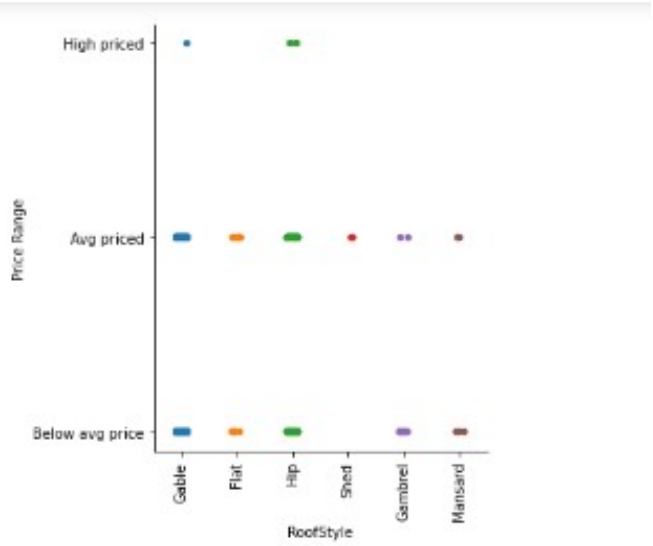


3. Lineplot

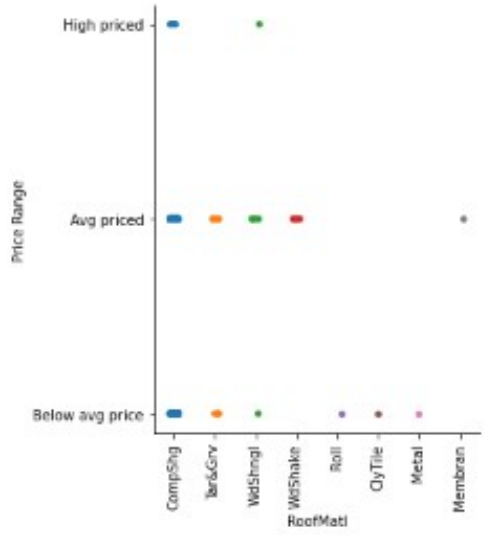




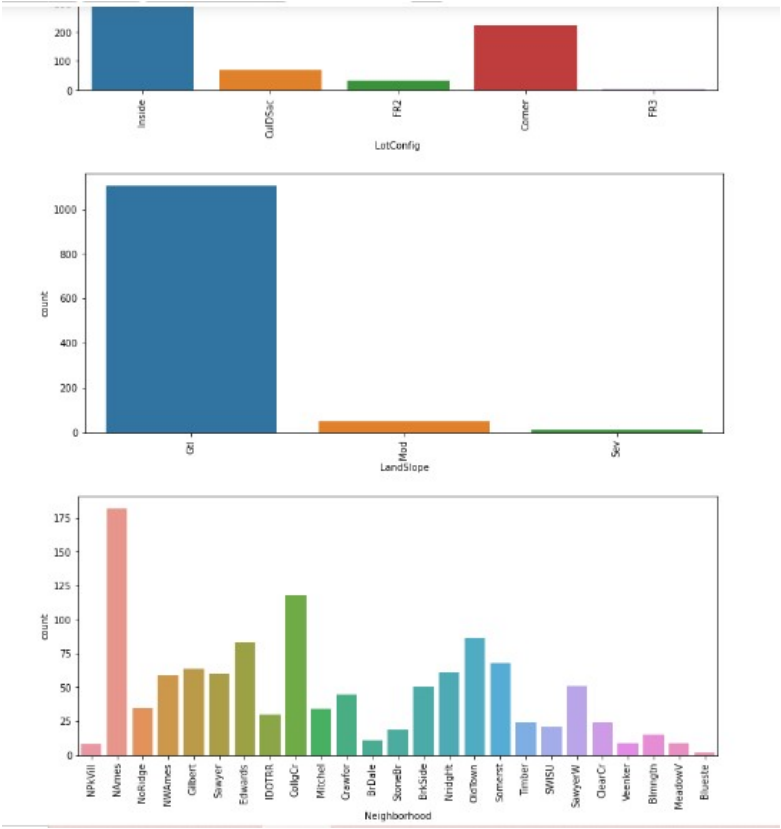
- Bivariate Analysis
  1. Relplot



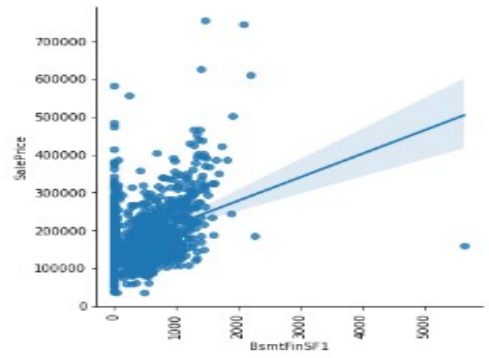

```
<Figure size 864x360 with 0 Axes>
```



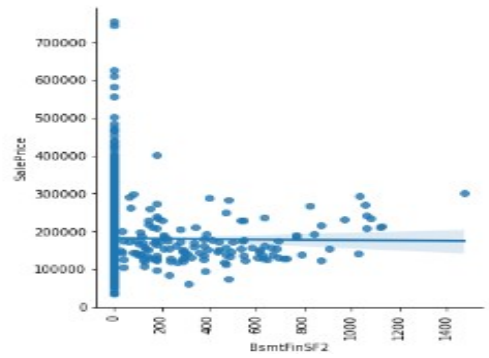
2. Catplot



3. LMplot:

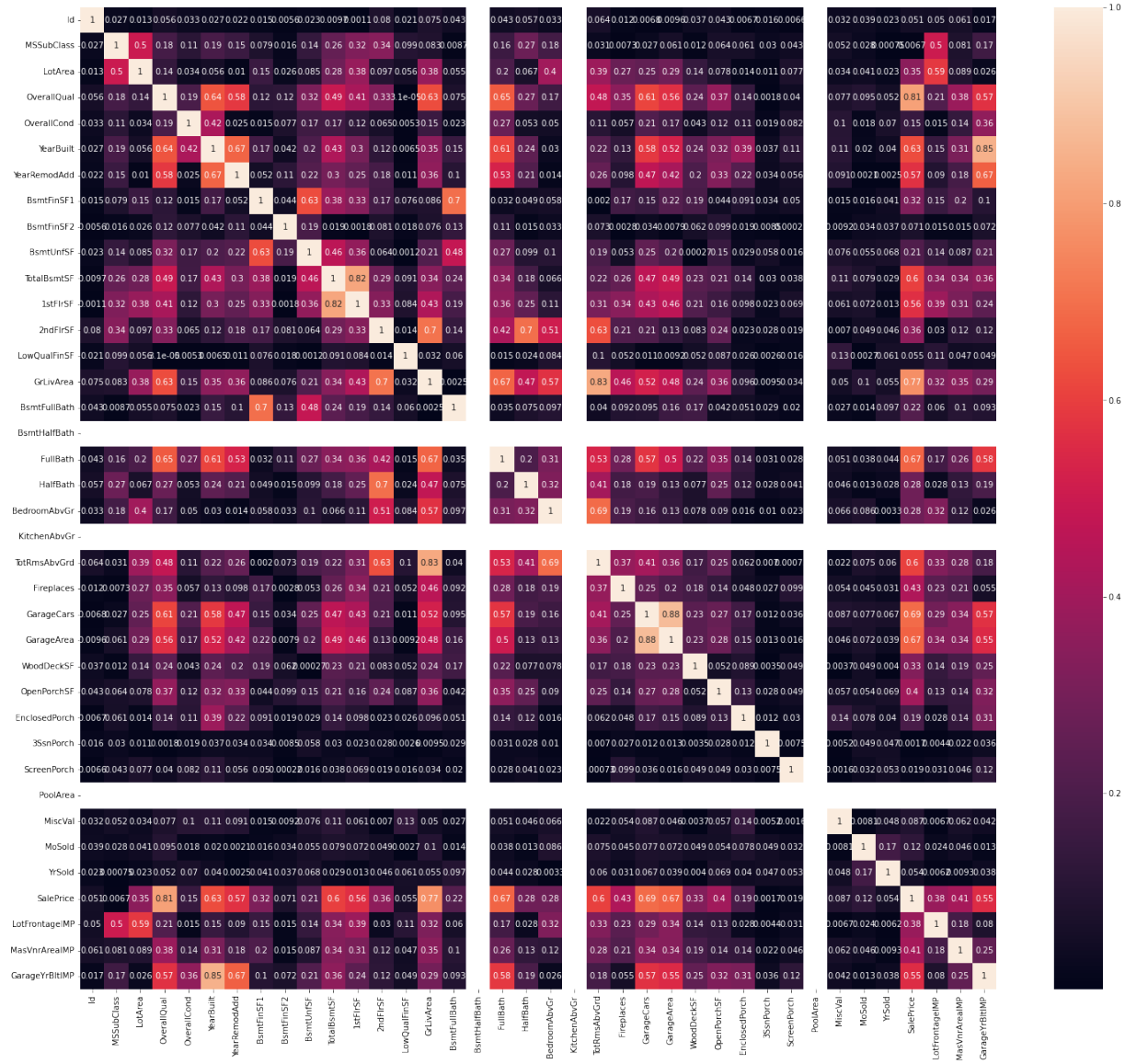


<Figure size 1296x936 with 0 Axes>



- Multivariate Analysis:
  1. Pairplot

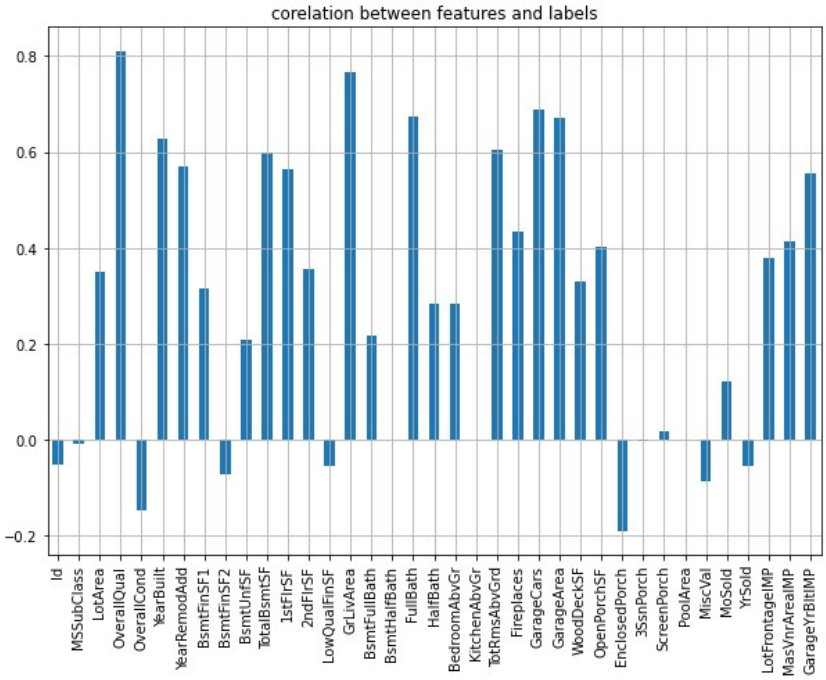
- HeatMap:



Inferences drawn from plots:

Results

Most important features affecting Sales Price:



Best features by there Select K best values that affect Sales price of a house are:

	Features	Scores
28	3SsnPorch	inf
13	LowQualFinSF	4.621270
3	OverallQual	4.478420
14	GrLivArea	3.671677
17	FullBath	3.491706
5	YearBuilt	2.474115
24	GarageArea	2.395244
23	GarageCars	2.376853
11	1stFlrSF	2.237095
10	TotalBsmtSF	2.182005
21	TotRmsAbvGrd	2.138774
36	GarageYrBltIMP	1.902150
35	MasVnrAreaIMP	1.816586
6	YearRemodAdd	1.669807
12	2ndFlrSF	1.510321

Inferences Drawn by plotting

1. Kde plot Summary

For different columns we can see that, for MS subclass major density lies between 0 to 100

For lot area measure density lies in between 0 to 25000

Overall quality of most houses ranges between 4 to 8 where as overall condition ranges between 4 to 8 as well,

Most houses were built in year between 1952 2025

For year remote ad column maximum density is at 2000.

For bsmt f i n sf1 Max density lies between 0 to 2000,

For bsmt firsf 2 density peak is at 0 and max between 0 to 250,

Max density for bsmt unfsf ranges between 0 to 1500,

Max frequency density for total bsmtsf and first floor SF lies at thousand,

Maths density for 2nd floor SF lies between 0 to 500.

GR live area Max density lies between 1000 to 3000,

For bsmt full bath peak values are at 0 and 1,

For sale price most house price range between 0 to 400000.

2. Distribution plot summary

ID not area overall quality overall condition bsmt unf SF total bsmtsf first floor SF gr l i v area, garage area m o sold years old sale price lot front page imp columns are partially skewed but our within permissible limits.

Still data needs to be treated for skewness.

3. Heat map analysis Summary

Result of heat map and by computing vif we can conclude that many columns in the data set are highly correlated and the smellic collinearity problem exist that needs to be treated for best result.

4. **Box plot summary**

Using box plot whether outliers are present in the data we hereby confirm that most of the outlets and the data set as a whole contains lot of outliers which need to be treated before final modelling.

5. **Line plot summary**

We can see that as overall quality overall condition year built year remod add, bsmt fin sf2 2nd floor SF full bath fire places and garage built MP values are increasing sales price is also increasing.

6. **Using rel plot**

We can conclude that most of the houses belong to below average and average price range and very less houses belong to high priced category.

Key pointers and Learning from Outcomes

Power of visualization, data cleaning and various algorithms used.

Before the hype of machine learning, artificial intelligence, and big data analysis, there was statistics. Statistics is the study of patterns using mathematics. Math is the foundation of algorithms. Statistics is one way to apply math to understand problems in the real world. In statistics, exploratory data analysis is one of the first ways to zoom in on the data that matters. In a sense, exploratory data analysis is a way to cut out the noise. It's also a way to forge a path into the forest of data that your algorithm will comb through. Data visualization has become popular in recent years due to its power to display the results at the end of the machine learning process, but it is also increasingly being used as a tool for **exploratory data analysis before applying machine learning models**.

At the beginning of the machine learning process, data visualization is a powerful tool.

Machine learning is inherently an iterative process. Modeling can be cumbersome when you are performing the process over and over again to ensure your model is optimized and can generalize well. Add on the time you spend on model selection and model tuning, the process can easily become a frustrating one. Good exploratory data analysis combined with relevant data visualization is essential for pinpointing the right direction to take. It both shortens the machine learning process and provides more accuracy for its outcome. Data visualization tools enable data scientists to quickly identify and focus on the most important data and the most important paths to take. Even during the modeling process, model graphs can help to speed up the model-creation process by displaying the model maps conceptually. When evaluating the models, visualizing the results of hyper parameter tuning can help data scientists narrow down the groupings of hyper parameters that are most important.

Best algorithms are:

Ridge regularization for Regression algorithm with regularization techniques.

MAE: 13968.36639955988
MSE: 334826336.8224348
RMSE: 18298.260486243897
R2: 0.9121836752560588
Score: 0.997042314942911

AdaBoost Regressor algorithm with Hyperparameter tuning.
MAE: 17571.788782517036
MSE: 555857380.9305034
RMSE: 23576.627853247024
R2: 0.8542129250095508
Score: 0.9951554293305714

FINAL RESULT

Hereby, we can conclude that Ridge regularization technique is the best technique for predicting sales price of houses with 91% testing accuracy and least error.

Limitations in other algorithms

Regression Model	Advantages	Disadvantages
Linear Regression	1. Works well irrespective of the dataset size. 2. Gives information about the relevance of features.	1. The assumptions of Linear Regression.
Polynomial Regression	1. Works on any size of the dataset. 2. Works very well on non-linear problems.	1. We need to choose the right polynomial degree for good bias/ variance tradeoff.
Support Vector Regression	1. Easily adaptable. 2. Works very well on non-linear problems. 3. Not biased by outliers (object that deviates significantly from the rest).	1. Compulsory to apply feature scaling. 2. Not well known. 3. Difficult to understand.
Decision Tree Regression	1. Interpretability. 2. Works well on both linear and non-linear problems. 3. No need to apply feature scaling.	1. Poor results on small datasets. 2. Overfitting can easily occur.
Random Forest Regression	1. Powerful. 2. Accurate. 3. Good performance on many problems including non-linear.	1. No interpretability. 2. Overfitting can easily occur. 3. We need to choose the number of trees.

One of the limitations of decision trees is that **they are largely unstable compared to other decision predictors**. A small change in the data can result in a major change in the structure of the decision tree, which can convey a different result from what users will get in a normal event.

The main limitation of random forest is that **a large number of trees can make the algorithm too slow and ineffective for real-time predictions**. In general, these algorithms are fast to train, but quite slow to create predictions once they are trained.

The k-nearest neighbors algorithm, also known as KNN or k-NN, is **a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point**.

SvM model doesn't perform well when we have large data set because the required training time is higher.

Limitations of this work and Scope for Future Work

1. Other modelling approaches could have been utilized.
2. Intense hyperparameter tuning could have been performed.

With improvement this model can be deployed to predict prices of Houses for different companies and locations and further can be used to elevate marketing strategies of real estate firms.