HOUSING: PRICE PREDICTION

Submitted by:

KIRAN KUMAR T

Acknowledgment

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. 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.

Introduction

- **Business Problem Framing:**

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 paper, 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.

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

The main objectives of this study are as follows:

- To apply data pre-processing and preparation techniques in order to obtain clean data
- To build machine learning models able to predict house price based on house features
- To analyse and compare model's performance in order to choose the best model

- # Literature Review

Machine learning is a form of artificial intelligence which compose available computers with the efficiency to be trained without being veraciously programmed. Machine learning interest on the extensions of computer programs which is capable enough to modify when unprotected to new-fangled data. Machine learning algorithms are broadly classified into three divisions, namely; Supervised learning, Unsupervised learning and Reinforcement learning. Supervised learning is a learning in which we teach or train the machine using data which is well labelled that means some data is already tagged with correct answer. After that, machine is provided with new set of examples so that

supervised learning algorithm analyses the training data and produces a correct outcome from labelled data. Unsupervised learning is the training of machine using information that is neither classified nor labelled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data. Unlike, supervised learning, no teacher is provided that means no training will be given to the machine. Therefore, machine is restricted to find the hidden structure in unlabelled data by our-self.

Reinforcement learning is an area of Machine Learning. Reinforcement. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behaviour or path it should take in a specific situation. Reinforcement learning differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of training dataset, it is bound to learn from its experience. Machine learning has many applications out of which one of the applications is prediction of real estate. The real estate market is one of the most competitive in terms of pricing and same tends to be vary significantly based on lots of factor, forecasting property price is an important modules in decision making for both the buyers and investors in supporting budget allocation, finding property finding stratagems and determining suitable policies hence it becomes one of the prime fields to apply the concepts of machine learning to optimize and predict the prices with high accuracy. The study on land price trend is felt important to support the decisions in urban planning. The real estate system is an unstable stochastic process. Investor's decisions are based on the market trends to reap maximum returns. Developers are interested to know the future trends for their decision making. To accurately estimate property prices and future trends, large amount of data that influences land price is required for analysis, modelling and forecasting. The factors that affect the land price have to be studied and their impact on price has also to be modelled. An analysis of the past data is to be considered. It is inferred that establishing a simple linear mathematical relationship for these time-series data is found not viable for forecasting. Hence it became imperative to establish a non-linear model which can well fit the data characteristic to analyse and forecast future trends. As the real estate is fast developing sector, the analysis and forecast of land prices using mathematical modelling and other scientific techniques is an immediate urgent need for decision making by all those concerned. The increase in population as well as the industrial activity is attributed to various factors, the most prominent being the recent spurt in the knowledge sector viz. Information Technology (IT) and Information technology enabled services. Demand for land started of showing an upward trend and housing and the real estate activity started booming. All barren lands and paddy fields ceased their existence to pave way for multistore and high-rise buildings. Investments started pouring in Real estate Industry and there was no uniform pattern in the land price over the years. The need for predicting the trend in land prices was felt by all in the industry viz. the Government, the regulating bodies, lending institutions, the developers and the investors. Therefore, in this paper, we present various important features to use while predicting housing prices with good accuracy. We can use regression models, using various features to have lower Root mean Squared error. While using features in a regression model some feature engineering is required for better prediction. Often a set of features linear regression, random forest regression and decision tree regression is used for making better model fit. For these models are expected to be susceptible towards over fitting random forest regression is

used to reduce it. So, it directs to the best application of regression models in addition to other techniques to optimize the result.

Linear Regression:

To establish baseline performance with a linear classifier, we used Linear Regression to model the price targets, Y, as a linear function of the data, X

$$f(X) = w_0 + w_1 x_1 + \ \ldots + w_m x_m + x_m$$

$$= \sum_{j=l:m}^{\infty} (w_j x_j)$$

Advantage: A linear model can include more than one predictor as long as the predictors are additive. the best fit line is the line with minimum error from all the points, it has high efficiency but sometimes this high efficiency created.

Disadvantage: Linear Regression Is Limited to Linear Relationships. Linear Regression Only Looks at the Mean of the Dependent Variable. Linear Regression Is Sensitive to Outliers. Data Must Be Independent

Random Forest Regression:

The Random Forest Regression (RFR) is an ensemble algorithm that combines multiple Regression Trees (RTs). Each RT is trained using a random subset of the features, and the output is the average of the individual RTs. The sum of squared errors for a tree T is:

Advantages: There is no need for feature normalization. Individual decision trees can be trained in parallel. Random forests are widely used. They reduce overfitting.

Disadvantages: They're not easily interpretable. They're not a state-of-the-art

$$S = \sum_{c \in leaves(T) i \in C} \sum (y_i - m_c)^2$$

Where $m_c = \frac{1}{n_c} + \sum_{i \in C} y_i$

- # Motivation of the problem undertaken:

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

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

• Which variables are important to predict the price of variable?

• How do these variables describe the price of the house?

Analytical Problem Framing

- ## Mathematical/ Analytical Modelling of the Problem Statistical Analysis

  Once it comes time to analyze the data, there are an array of statistical model's analysts may choose to utilize. The most common techniques will fall into the following two groups:

  - Supervised learning, including regression and classification models.
  - Unsupervised learning, including clustering algorithms and association rules

Regression Model:

The regression models are used to examine relationships between variables. Regression models are often used to determine which independent variables hold the most influence over dependent variables information that can be leveraged to make essential decision.
The most traditional regression model is linear regression, decision tree regression, random forest regression, xgboost regression and knn-neighbours.

There are 4 main components of an analytics model, namely: 1) Data Component, 2) Algorithm Component, 3) Real World Component, and 4) Ethical Component.

- # Data Preparation

In this study, we will use a housing dataset presented by A US-based housing company named Surprise Housing has decided to enter the Australian market. The company uses data analytics to purchase houses at a price below their actual values and flip them at a higher price. For the same purpose, the company has collected a data set from the sale of houses in Australia. The data is provided in the csv file below housing_train.csv and housing_test.csv

- # Data Description

The dataset contains 1460 records (rows) and 81 features (columns).

Here, we will provide a brief description of dataset features. Since the number of features is large (81), we will attach the original data description file to this study for more information about the dataset. Now, we will mention the feature name with a short description of its meaning.

Dataframe Description:

The dataset contains the data of the house. On the basis of the data we have to predict the sale price of the house, the datset contains the data like, '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'.

In the above dataset the target is to predict the Sale price of the house:

- # Data Pre-processing

Then we move to see statistical information about the non-numerical columns in our dataset:

Preprocessing

```
In [57]: df.describe() #statistics summary for numerical columns
```

Out[57]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2
count	1168.000000	1168.000000	954.00000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1161.000000	1168.000000	1168.000000
mean	724.136130	56.767979	70.96647	10484.749144	6.104452	5.595890	1970.930651	1984.756562	102.310078	444.726027	46.647260
std	416.159877	41.940650	24.82875	8957.442311	1.390153	1.124343	30.145255	20.785185	182.595606	462.664785	163.520016
min	1.000000	20.000000	21.00000	1300.000000	1.000000	1.000000	1875.000000	1950.000000	0.000000	0.000000	0.000000
25%	360.500000	20.000000	60.00000	7621.500000	5.000000	5.000000	1954.000000	1966.000000	0.000000	0.000000	0.000000
50%	714.500000	50.000000	70.00000	9522.500000	6.000000	5.000000	1972.000000	1993.000000	0.000000	385.500000	0.000000
75%	1079.500000	70.000000	80.00000	11515.500000	7.000000	6.000000	2000.000000	2004.000000	160.000000	714.500000	0.000000
max	1460.000000	190.000000	313.00000	164660.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	1474.000000

From the table above, we can see, for example, that the average lot area of the houses in our dataset is 10,1484.74 ft^2 with a standard deviation of 8957.44 ft^2. We can see also that the minimum lot area is 1300 ft^2 and the maximum lot area is 164660 ft^2 with a median of 9522 ft^2. Similarly, we can get a lot of information about our dataset variables from the table.

- # Data Cleaning

Dealing with Missing Values:

We should deal with the problem of missing values because some machine learning models don't accept data with missing values. Firstly, let's see the number of missing values in our dataset. We want to see the number and the percentage of missing values for each column that actually contains missing values.

Columns	non-null	Columns	non-null	Columns	non-null
Id	0	ExterCond	0	Functional	0
MSSubClass	0	Foundation	0	Fireplaces	0
MSZoning	0	BsmtQual	30	FireplaceQu	551
LotFrontage	214	BsmtCond	30	GarageType	64
LotArea	0	BsmtExposure	31	GarageYrBlt	64
Street	0	BsmtFinType1	30	GarageFinish	64
Alley	1091	BsmtFinSF1	0	GarageCars	0
LotShape	0	BsmtFinType2	31	GarageArea	0
LandContour	0	BsmtFinSF2	0	GarageQual	64
LotConfig	0	BsmtUnfSF	0	GarageCond	64
LandSlope	0	TotalBsmtSF	0	PavedDrive	0

Neighborhood	0	Heating	0	WoodDeckSF	0
Condition1	0	HeatingQC	0	OpenPorchSF	0
Condition2	0	CentralAir	0	EnclosedPorch	0
BldgType	0	Electrical	0	3SsnPorch	0
HouseStyle	0	1stFlrSF	0	ScreenPorch	0
OverallQual	0	2ndFlrSF	0	PoolArea	0
OverallCond	0	LowQualFinSF	0	PoolQC	1161
YearBuilt	0	GrLivArea	0	Fence	931
YearRemodAdd	0	BsmtFullBath	0	MiscFeature	1124
RoofStyle	0	BsmtHalfBath	0	MiscVal	0
RoofMatl	0	FullBath	0	MoSold	0
Exterior1st	0	HalfBath	0	YrSold	0
Exterior2nd	0	BedroomAbvGr	0	SaleType	0
MasVnrType	7	KitchenAbvGr	0	SaleCondition	0
MasVnrArea	7	KitchenQual	0	SalePrice	0
ExterQual	0	TotRmsAbvGrd	0		

In the above table we got, count represents the number of non-null values in each column.

Filling the missing values using fillna method.

```
#filling the missing values for numerical terms by mean
df['LotFrontage']=df['LotFrontage'].fillna(df['LotFrontage'].mean())
df['GarageYrBlt']=df['GarageYrBlt'].fillna(df['GarageYrBlt'].mean())
df['MasVnrArea']=df['MasVnrArea'].fillna(df['MasVnrArea'].mean())
df['GarageYrBlt']=df['GarageYrBlt'].fillna(df['GarageYrBlt'].mean())
```

```
#filling the missing values for categorical terms by mode
df['Alley']=df['Alley'].fillna(df['Alley'].mode()[0])
df['BsmtQual']=df['BsmtQual'].fillna(df['BsmtQual'].mode()[0])
df['BsmtCond']=df['BsmtCond'].fillna(df['BsmtCond'].mode()[0])
df['BsmtExposure']=df['BsmtExposure'].fillna(df['BsmtExposure'].mode()[0])
df['BsmtFinType1']=df['BsmtFinType1'].fillna(df['BsmtFinType1'].mode()[0])
df['BsmtFinType2']=df['BsmtFinType2'].fillna(df['BsmtFinType2'].mode()[0])
df['FireplaceQu']=df['FireplaceQu'].fillna(df['FireplaceQu'].mode()[0])
df['GarageType']=df['GarageType'].fillna(df['GarageType'].mode()[0])
df['GarageFinish']=df['GarageFinish'].fillna(df['GarageFinish'].mode()[0])
df['MasVnrType']=df['MasVnrType'].fillna(df['MasVnrType'].mode()[0])
df['GarageFinish']=df['GarageFinish'].fillna(df['GarageFinish'].mode()[0])
df['GarageQual']=df['GarageQual'].fillna(df['GarageQual'].mode()[0])
df['GarageCond']=df['GarageCond'].fillna(df['GarageCond'].mode()[0])
df['PoolQC']=df['PoolQC'].fillna(df['PoolQC'].mode()[0])
df['Fence']=df['Fence'].fillna(df['Fence'].mode()[0])
df['MiscFeature']=df['MiscFeature'].fillna(df['MiscFeature'].mode()[0])
```

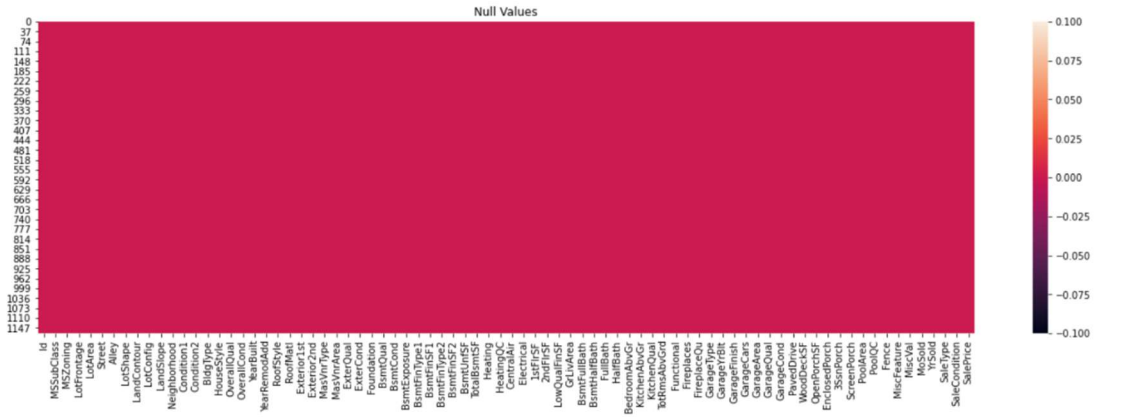
Now let's check if there is any remaining missing value in our dataset:

Columns	**non-null**	**Columns**	**non-null**	**Columns**	**non-null**
Id	0	ExterCond	0	Functional	0
MSSubClass	0	Foundation	0	Fireplaces	0
MSZoning	0	BsmtQual	0	FireplaceQu	0
LotFrontage	0	BsmtCond	0	GarageType	0
LotArea	0	BsmtExposure	0	GarageYrBlt	0
Street	0	BsmtFinType1	0	GarageFinish	0
Alley	0	BsmtFinSF1	0	GarageCars	0

LotShape	0	BsmtFinType2	0	GarageArea	0
LandContour	0	BsmtFinSF2	0	GarageQual	0
LotConfig	0	BsmtUnfSF	0	GarageCond	0
LandSlope	0	TotalBsmtSF	0	PavedDrive	0
Neighborhood	0	Heating	0	WoodDeckSF	0
Condition1	0	HeatingQC	0	OpenPorchSF	0
Condition2	0	CentralAir	0	EnclosedPorch	0
BldgType	0	Electrical	0	3SsnPorch	0
HouseStyle	0	1stFlrSF	0	ScreenPorch	0
OverallQual	0	2ndFlrSF	0	PoolArea	0
OverallCond	0	LowQualFinSF	0	PoolQC	0
YearBuilt	0	GrLivArea	0	Fence	0
YearRemodAdd	0	BsmtFullBath	0	MiscFeature	0
RoofStyle	0	BsmtHalfBath	0	MiscVal	0
RoofMatl	0	FullBath	0	MoSold	0
Exterior1st	0	HalfBath	0	YrSold	0
Exterior2nd	0	BedroomAbvGr	0	SaleType	0
MasVnrType	0	KitchenAbvGr	0	SaleCondition	0
MasVnrArea	0	KitchenQual	0	SalePrice	0
ExterQual	0	TotRmsAbvGrd	0		

This means that our dataset is now complete; it doesn't contain any missing value anymore. To show graphical representation of null using heatmap for entire dataset.

```
In [62]: # Heatmap for Null value for data.
         plt.figure(figsize=[22,6])
         sns.heatmap(df.isnull())
         plt.title('Null Values')
         plt.show()
```



Column	Data types	Column	Data types
Id	int64	CentralAir	object
MSSubClass	int64	Electrical	object
MSZoning	object	1stFlrSF	int64
LotFrontage	float64	2ndFlrSF	int64
LotArea	int64	LowQualFinSF	int64
Street	object	GrLivArea	int64
Alley	object	BsmtFullBath	int64
LotShape	object	BsmtHalfBath	int64
LandContour	object	FullBath	int64
LotConfig	object	HalfBath	int64
LandSlope	object	BedroomAbvGr	int64
Neighborhood	object	KitchenAbvGr	int64

Condition1	object	KitchenQual	object
Condition2	object	TotRmsAbvGrd	int64
BldgType	object	Functional	object
HouseStyle	object	Fireplaces	int64
OverallQual	int64	FireplaceQu	object
OverallCond	int64	GarageType	object
YearBuilt	int64	GarageYrBlt	float64
YearRemodAdd	int64	GarageFinish	object
RoofStyle	object	GarageCars	int64
RoofMatl	object	GarageArea	int64
Exterior1st	object	GarageQual	object
Exterior2nd	object	GarageCond	object
MasVnrType	object	PavedDrive	object
MasVnrArea	float64	WoodDeckSF	int64
ExterQual	object	OpenPorchSF	int64
ExterCond	object	EnclosedPorch	int64
Foundation	object	3SsnPorch	int64
BsmtQual	object	ScreenPorch	int64
BsmtCond	object	PoolArea	int64
BsmtExposure	object	PoolQC	object
BsmtFinType1	object	Fence	object
BsmtFinSF1	int64	MiscFeature	object
BsmtFinType2	object	MiscVal	int64
BsmtFinSF2	int64	MoSold	int64
BsmtUnfSF	int64	YrSold	int64
TotalBsmtSF	int64	SaleType	object
Heating	object	SaleCondition	object
HeatingQC	object	SalePrice	int64

Since the dataset has a lot string values. We will use the encoding techniques to convert the string data to numerical one.

- ## Encoding of Data Frame:

In ordinal encoding, each unique category value is assigned an integer value. This ordinal encoding transform is available in the scikit-learn Python machine learning library via the Ordinal Encoder class. By default, it will assign integers to labels in the order that is observed in the data.

In this encoding scheme, the categorical feature is first converted into numerical using an ordinal encoder. Then the numbers are transformed in the binary number. After that binary value is split into different columns. Binary encoding works really well when there are a high number of categories.

Encoding of DataFrame:

```
In [63]: from sklearn.preprocessing import OrdinalEncoder
         enc=OrdinalEncoder()

In [64]: for i in df.columns:
             if df[i].dtypes=="object":
                 df[i]=enc.fit_transform(df[i].values.reshape(-1,1))

In [65]: df.head()    # informtion about top of the data after Ordinal encoder
```

Out[65]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	Bl
0	127	120	3.0	70.98847	4928	1.0	0.0	0.0	3.0	4.0	0.0	13.0	2.0	2.0	
1	889	20	3.0	95.00000	15865	1.0	0.0	0.0	3.0	4.0	1.0	12.0	2.0	2.0	
2	793	60	3.0	92.00000	9920	1.0	0.0	0.0	3.0	1.0	0.0	15.0	2.0	2.0	
3	110	20	3.0	105.00000	11751	1.0	0.0	0.0	3.0	4.0	0.0	14.0	2.0	2.0	
4	422	20	3.0	70.98847	16635	1.0	0.0	0.0	3.0	2.0	0.0	14.0	2.0	2.0	

- **Correlation matrix:**

A correlation matrix is simply a table which displays the correlation. The measure is best used in variables that demonstrate a linear relationship between each other. The fit of the data can be visually represented in a heatmap.

Pandas dataframe. corr() method is used for creating the correlation matrix. It is used to find the pairwise correlation of all columns in the dataframe.

To create correlation matrix using pandas, these steps should be taken:

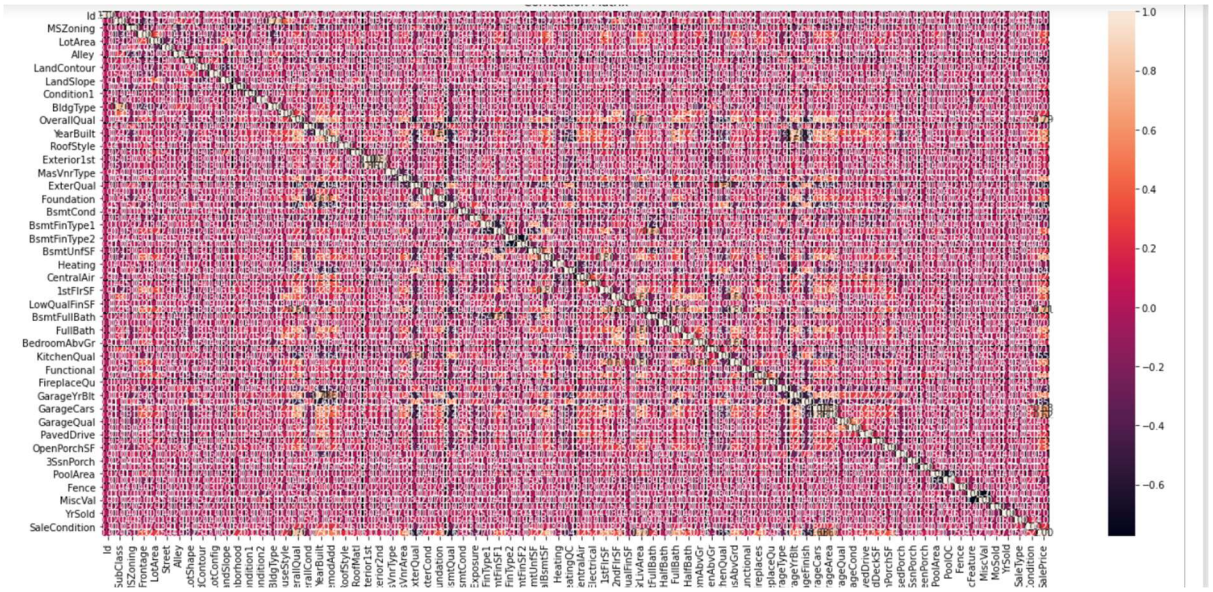
1. Obtain the data.
2. Create the DataFrame using Pandas.
3. Create correaltion matrix using Pandas.

Checking Correlation ¶

```
In [67]: corr_mat=df.corr()
         corr_mat
```

Out[67]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	LotConfig	LandSlope	Neighborho
Id	1.000000	0.004259	0.009307	-0.005969	-0.029212	0.003613	-0.009049	0.022978	-0.020245	0.053927	0.007152	-0.0149
MSSubClass	0.004259	1.000000	0.007478	-0.336681	-0.124151	-0.035981	0.216396	0.104485	-0.021387	0.076880	-0.014930	0.0139
MSZoning	0.009307	0.007478	1.000000	-0.069661	-0.023328	0.140215	-0.371755	0.053655	0.001175	-0.027246	-0.023952	-0.2518
LotFrontage	-0.005969	-0.336681	-0.069661	1.000000	0.299452	-0.035309	-0.187657	-0.144523	-0.073451	-0.192468	0.046051	0.0658
LotArea	-0.029212	-0.124151	-0.023328	0.299452	1.000000	-0.263973	-0.093239	-0.189201	-0.159038	-0.152063	0.395410	0.0107
Street	0.003613	-0.035981	0.140215	-0.035309	-0.263973	1.000000	0.010454	-0.012941	0.105226	0.000153	-0.141572	0.0014
Alley	-0.009049	0.216396	-0.371755	-0.187657	-0.093239	0.010454	1.000000	0.046387	-0.040922	0.047806	-0.005436	0.1597
LotShape	0.022978	0.104485	0.053655	-0.144523	-0.189201	-0.012941	0.046387	1.000000	0.081803	0.211395	-0.101187	-0.0318
LandContour	-0.020245	-0.021387	0.001175	-0.073451	-0.159038	0.105226	-0.040922	0.081803	1.000000	-0.031496	-0.386787	0.0372
LotConfig	0.053927	0.076880	-0.027246	-0.192468	-0.152063	0.000153	0.047806	0.211395	-0.031496	1.000000	-0.007934	-0.0304
LandSlope	0.007152	-0.014930	-0.023952	0.046051	0.395410	-0.141572	-0.005436	-0.101187	-0.386787	-0.007934	1.000000	-0.1108



Observations: We are unable to identify the correlation in above heatmap due to huge number of columns.

How correlation matrix is calculated?
A correlation matrix is a table showing correlation coefficients between sets of variables. Each random variable (x) in the table is correlated with each of the other values in the table x.The diagonal of the table is always a set of ones, because the correlation between a variable and itself is always 1.

```
corr_matrix=df.corr()
corr_matrix["SalePrice"].sort_values(ascending=False)
```

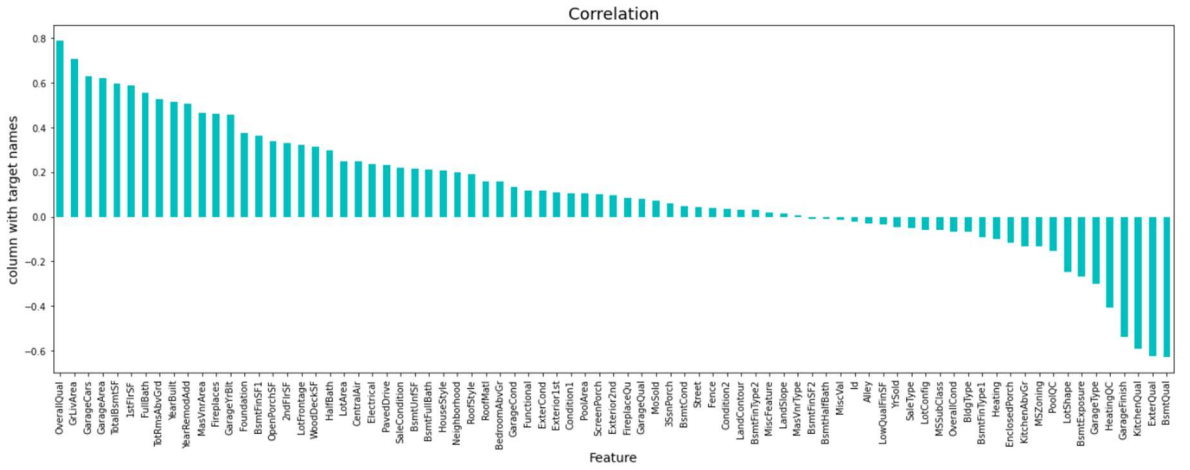
Columns	Correlation matrix	Columns	Correlation matrix
SalePrice	1.000000	ExterCond	0.115167
OverallQual	0.789185	Exterior1st	0.108451
GrLivArea	0.707300	Condition1	0.105820
GarageCars	0.628329	PoolArea	0.103280
GarageArea	0.619000	ScreenPorch	0.100284
TotalBsmtSF	0.595042	Exterior2nd	0.097541
1stFlrSF	0.587642	LandSlope	0.015485
FullBath	0.554988	MasVnrType	0.007732
TotRmsAbvGrd	0.528363	BsmtFinSF2	-0.010151
YearBuilt	0.514408	BsmtHalfBath	-0.011109
YearRemodAdd	0.507831	MiscVal	-0.013071
MasVnrArea	0.463626	Id	-0.023897
Fireplaces	0.459611	Alley	-0.029798
GarageYrBlt	0.458007	LowQualFinSF	-0.032381
Foundation	0.374169	YrSold	-0.045508
BsmtFinSF1	0.362874	SaleType	-0.050851
OpenPorchSF	0.339500	LotConfig	-0.060452
2ndFlrSF	0.330386	MSSubClass	-0.060775
LotFrontage	0.323779	OverallCond	-0.065642
WoodDeckSF	0.315444	BldgType	-0.066028
HalfBath	0.295592	BsmtFinType1	-0.092109
LotArea	0.249499	Heating	-0.100021
CentralAir	0.246754	EnclosedPorch	-0.115004
Electrical	0.234621	KitchenAbvGr	-0.132108

PavedDrive	0.231707	MSZoning	-0.133221
SaleCondition	0.217687	PoolQC	-0.152611
BsmtUnfSF	0.215724	LotShape	-0.248171
BsmtFullBath	0.212924	BsmtExposure	-0.268559
HouseStyle	0.205502	GarageType	-0.299470
Neighborhood	0.198942	HeatingQC	-0.406604
RoofStyle	0.192654	GarageFinish	-0.537121
RoofMatl	0.159865	KitchenQual	-0.592468
BedroomAbvGr	0.158281	ExterQual	-0.624820
GarageCond	0.135071	BsmtQual	-0.626850
Functional	0.118673		

Now we can clearly identify the correlation of independent variables with the target variables "Sale Price". There are around 25 variables who has less than 0.01 correlation value (very week relationship.)

Checking the columns which are positively and negative correlated with the target columns:

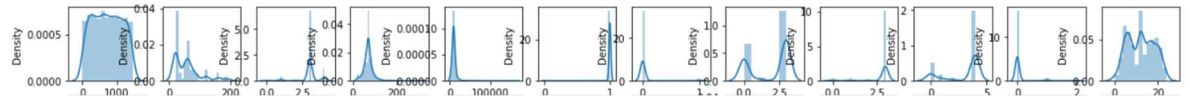
```
plt.figure(figsize=(22,7))
df.corr()['SalePrice'].sort_values(ascending=False).drop(['SalePrice']).plot(kind='bar',color='c')
plt.xlabel('Feature',fontsize=14)
plt.ylabel('column with target names',fontsize=14)
plt.title('Correlation',fontsize=18)
plt.show()
```

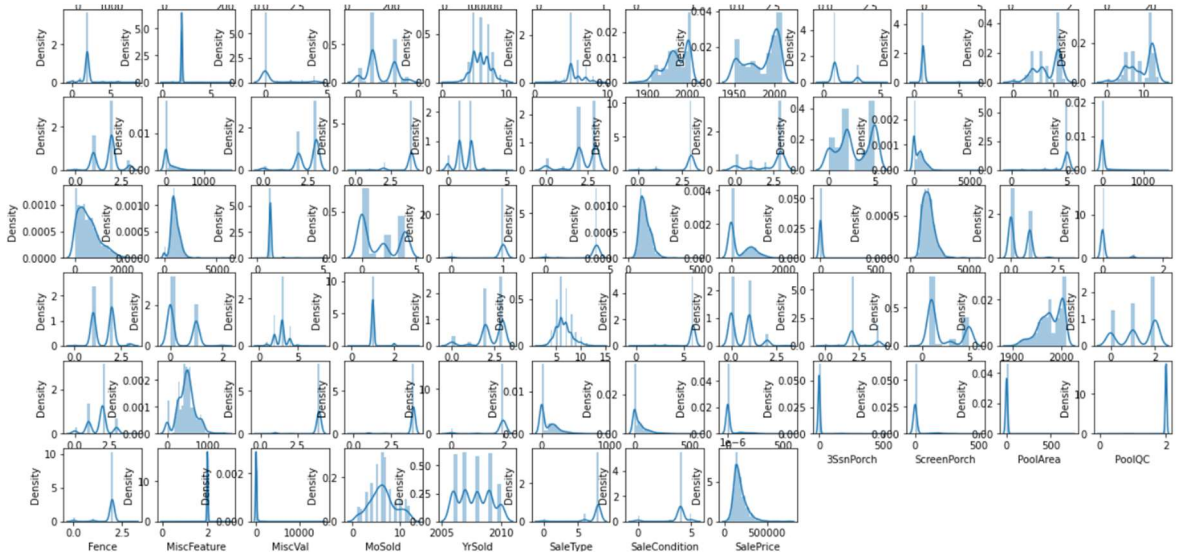


Let’s check the data distribution among all the columns.

```
collist=df.columns.values
ncol=12
nrow=10
```

```
#plt.figure(figsize=(5*totalcol,5*totalcol))
plt.figure(figsize=(20,16))
for i in range(0, len(collist)):
    plt.subplot(nrow,ncol,i+1)
    sns.distplot(df[collist[i]])
```



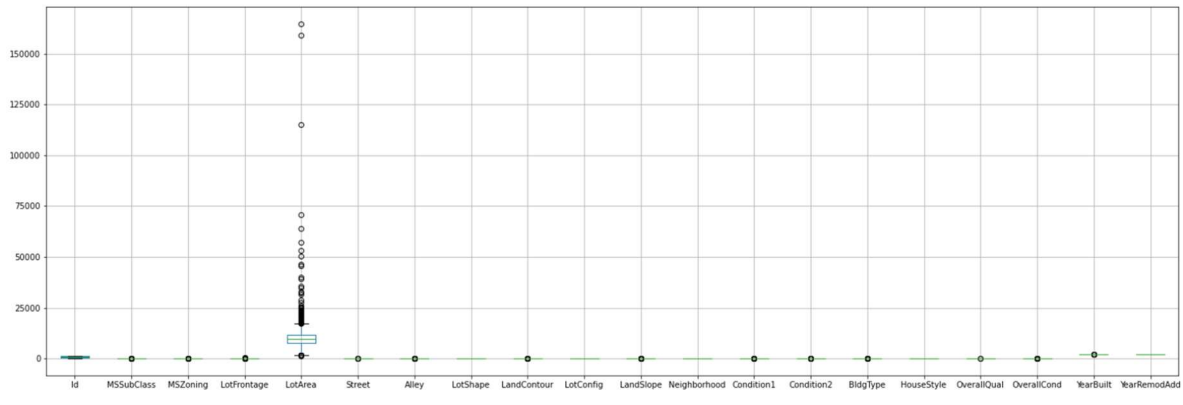


We can see skewness in data for the multiple columns, will handle the skewness in further steps.

- **Outliers Check:**

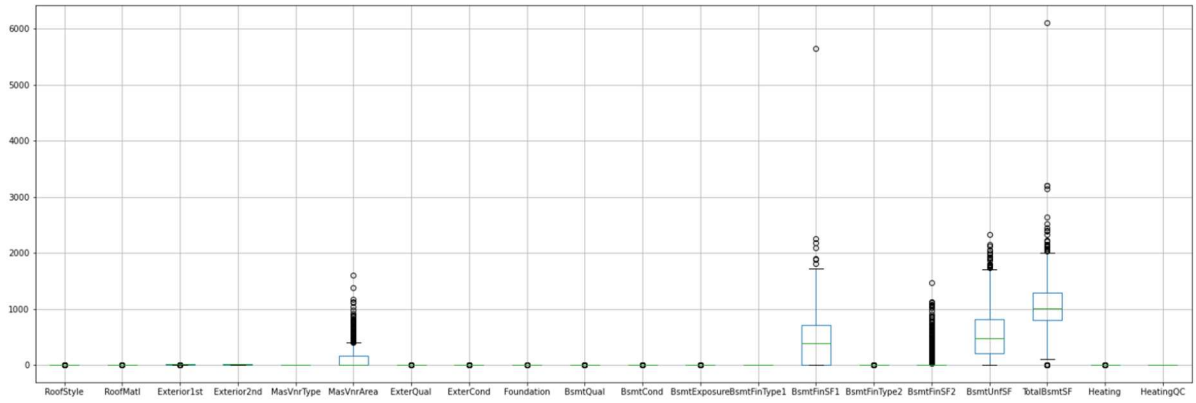
There are 80 columns in dataset so it's not possible to plot each and every column separately or plot all together. so, we will print in 4 steps:

```
#Plotting Boxplots for first 20 columns
df.iloc[:,0:20].boxplot(figsize=[25,10])
plt.subplots_adjust(bottom=0.25)
plt.show()
```



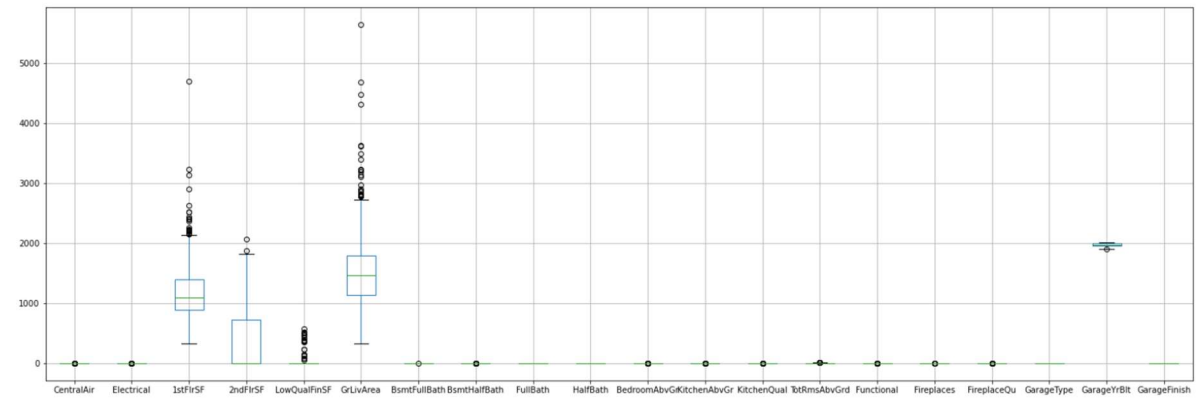
There are outliers in lot area column.

```
#Plotting boxplot for next 20 columns
df.iloc[:,20:40].boxplot(figsize=[25,10])
plt.subplots_adjust(bottom=0.25)
plt.show()
```



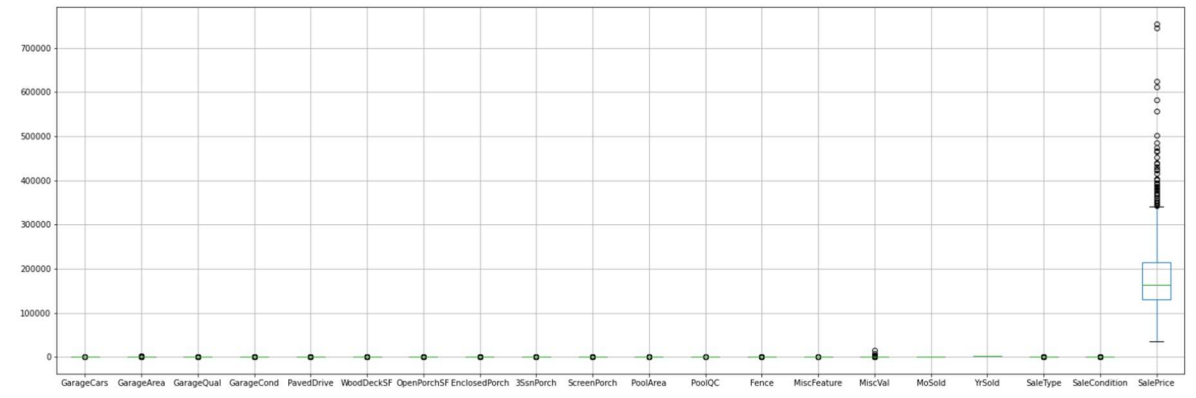
We can see outliers in MasVnrArea, BsmtFinSF1,BsmtFinSF2,BsmtUnfSF,TotalBsmtSF.

```
#Plotting boxplot for next 20 columns
df.iloc[:,40:60].boxplot(figsize=[25,10])
plt.subplots_adjust(bottom=0.25)
plt.show()
```



We can see outliers in 1stFlrSF, LowQualFinSF,GrLivArea,GarageYrBuilt.

```
#Plotting boxplot for rest all the columns
df.iloc[:,60:80].boxplot(figsize=[25,10])
plt.subplots_adjust(bottom=0.25)
plt.show()
```



We can see one or two values outer of whiskers but those are near to whiskers so these are not outliers.The outliers are not consider for target variables.

- ## 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.

- ## Checking Skewness:

Columns	Skewness	Columns	Skewness
Id	0.026526	CentralAir	-3.475188
MSSubClass	1.422019	Electrical	-3.104209
MSZoning	-1.796785	1stFlrSF	1.513707
LotFrontage	2.710383	2ndFlrSF	0.823479
LotArea	10.659285	LowQualFinSF	8.666142
Street	-17.021969	GrLivArea	1.449952
Alley	5.436187	BsmtFullBath	0.627106
LotShape	-0.603775	BsmtHalfBath	4.264403
LandContour	-3.125982	FullBath	0.057809
LotConfig	-1.118821	HalfBath	0.656492
LandSlope	4.812568	BedroomAbvGr	0.243855
Neighborhood	0.043735	KitchenAbvGr	4.365259
Condition1	3.008289	KitchenQual	-1.408106
Condition2	11.514458	TotRmsAbvGrd	0.644657
BldgType	2.318657	Functional	-3.999663
HouseStyle	0.285680	Fireplaces	0.671966
OverallQual	0.175082	FireplaceQu	0.753507
OverallCond	0.580714	GarageType	0.831142
YearBuilt	-0.579204	GarageYrBlt	-0.662934
YearRemodAdd	-0.495864	GarageFinish	-0.450190
RoofStyle	1.498560	GarageCars	-0.358556
RoofMatl	7.577352	GarageArea	0.189665
Exterior1st	-0.612816	GarageQual	-4.582386
Exterior2nd	-0.592349	GarageCond	-5.422472
MasVnrType	-0.104609	PavedDrive	-3.274035
MasVnrArea	2.834658	WoodDeckSF	1.504929
ExterQual	-1.810843	OpenPorchSF	2.410840
ExterCond	-2.516219	EnclosedPorch	3.043610
Foundation	-0.002761	3SsnPorch	9.770611
BsmtQual	-1.343781	ScreenPorch	4.105741
BsmtCond	-3.293554	PoolArea	13.243711
BsmtExposure	-1.166987	PoolQC	-19.401558
BsmtFinType1	-0.068901	Fence	-3.185107
BsmtFinSF1	1.871606	MiscFeature	-17.238424
BsmtFinType2	-3.615783	MiscVal	23.065943
BsmtFinSF2	4.365829	MoSold	0.220979
BsmtUnfSF	0.909057	YrSold	0.115765
TotalBsmtSF	1.744591	SaleType	-3.660513
Heating	10.103609	SaleCondition	-2.671829
HeatingQC	0.449933	SalePrice	1.953878

To handle skewness of the data using different types of functions:

1. Log Transform

2. Square Root Transform
3. Box-Cox Transform
4. Power transform

Now here, we are going to use Power transform function to handle skewness in dataset. Then, splitting the independent and target variable in x and y.

```
# Splitting the indepedent and target variable in x and y
x= df.drop('SalePrice',axis=1)
y= df['SalePrice']
```

In statistics, a power transform is a family of functions applied to create a monotonic transformation of data using power functions. It is a data transformation technique used to stabilize variance, make the data more normal distribution-like, improve the validity of measures of association (such as the Pearson correlation between variables), and for other data stabilization procedures.

```
from sklearn.preprocessing import power_transform
df_new=power_transform(x)
df_new=pd.DataFrame(df_new,columns=x.columns)
```

After performing such statistics, the skewness is removed in dataset as shown below:

Columns	Skewness	Columns	Skewness
Id	-0.268486	CentralAir	-3.475188
MSSubClass	0.064007	Electrical	-3.006845
MSZoning	0.233113	1stFlrSF	-0.002391
LotFrontage	0.161368	2ndFlrSF	0.280208
LotArea	0.032509	LowQualFinSF	6.922843
Street	-17.021969	GrLivArea	-0.000054
Alley	5.436187	BsmtFullBath	0.365488
LotShape	-0.594207	BsmtHalfBath	3.954345
LandContour	-2.592303	FullBath	-0.045944
LotConfig	-1.030401	HalfBath	0.498003
LandSlope	3.954345	BedroomAbvGr	0.116498
Neighborhood	-0.146541	KitchenAbvGr	-2.370593
Condition1	0.225468	KitchenQual	-0.435558
Condition2	0.537277	TotRmsAbvGrd	0.002332
BldgType	1.857194	Functional	-3.343664
HouseStyle	-0.080331	Fireplaces	0.084950
OverallQual	0.021658	FireplaceQu	0.082653
OverallCond	0.048063	GarageType	0.222501
YearBuilt	-0.126641	GarageYrBlt	-0.132523
YearRemodAdd	-0.225131	GarageFinish	-0.335248
RoofStyle	-0.292233	GarageCars	-0.022970
RoofMatl	-6.314987	GarageArea	-0.320370
Exterior1st	-0.338023	GarageQual	-4.327379
Exterior2nd	-0.352793	GarageCond	-4.925781
MasVnrType	-0.016203	PavedDrive	-3.025809
MasVnrArea	0.416370	WoodDeckSF	0.113026

ExterQual	-0.605112	OpenPorchSF	-0.002749
ExterCond	-2.270791	EnclosedPorch	2.022616
Foundation	0.004296	3SsnPorch	7.087955
BsmtQual	-0.413999	ScreenPorch	3.067153
BsmtCond	-3.025865	PoolArea	12.817372
BsmtExposure	-0.914214	PoolQC	-17.021969
BsmtFinType1	-0.206639	Fence	1.116688
BsmtFinSF1	-0.404528	MiscFeature	9.291637
BsmtFinType2	-2.420885	MiscVal	4.991071
BsmtFinSF2	2.394737	MoSold	-0.035838
BsmtUnfSF	-0.284390	YrSold	0.112893
TotalBsmtSF	0.286779	SaleType	-2.067563
Heating	-4.541694	SaleCondition	-0.353292
HeatingQC	0.156511		

- # Hardware and Software Requirements and Tools Used

PYTHON Jupyter Notebook:

Key Features:

An open-source solution that has simple coding processes and syntax so it's fairly easy to learn Integration with other languages such as C/C++, Java, PHP, C#, etc.
Advanced analysis processes through machine learning and text mining.

Python is extremely accessible to code in comparison to other popular languages such as Java, and its syntax is relatively easy to learn making this tool popular among users that look for an open-source solution and simple coding processes. In data analysis, Python is used for data crawling, cleaning, modelling, and constructing analysis algorithms based on business scenarios. One of the best features is actually its user-friendliness: programmers don't need to remember the architecture of the system nor handle the memory – Python is considered a high-level language that is not subject to the computer's local processor.

Libraries and Packages used:

Matplotlib:

Matplotlib is a Python library that uses Python Script to write 2-dimensional graphs and plots. Often mathematical or scientific applications require more than single axes in a representation. This library helps us to build multiple plots at a time. You can, however, use Matplotlib to manipulate different characteristics of figures as well.

The task carried out is visualization of dataset i.e., nominal data, ordinal data, continuous data, heatmap display distribution for correlation matrix and null values, boxplot distribution for checking outliers, scatter plot distribution for modelling approach, subplot distribution for analysis and comparison, feature importance and common importance features, line plot for prediction values vs actual values.

Numpy:

Numpy is a popular array – processing package of Python. It provides good support for different dimensional array objects as well as for matrices. Numpy is not only confined to providing arrays only, but it also provides a variety of tools to manage these arrays. It is fast, efficient, and really good for managing matrices and arrays.

The Numpy is used to managing matrices i.e., MAE, MSE and RMSE and arrays i.e., described the values of train test dataset.

Pandas:

Pandas is a python software package. It is a must to learn for data-science and dedicatedly written for Python language. It is a fast, demonstrative, and adjustable platform that offers intuitive data-structures. You can easily manipulate any type of data such as – structured or time-series data with this amazing package.

The Pandas is used to execute a Data frame i.e., test set.csv, train set.csv, skewness, co-efficient, predicted values of model approach, conclusion.

Scikit Learn:

Scikit learn is a simple and useful python machine learning library. It is written in python, cython, C, and C++. However, most of it is written in the Python programming language. It is a free machine learning library. It is a flexible python package that can work in complete harmony with other python libraries and packages such as Numpy and Scipy.

Scikit learn library is used to import a pre-processing function i.e., power transform, ordinal encoder, minimax scaler, linear, random forest, decision tree, xgboost, k-nearest neighbours, r2 score, mean absolute error, mean squared error, train test split, grid search cv and ensemble technique.

Models Development and Evaluation

In this section, we choose the type of machine learning prediction that is suitable to our problem. We want to determine if this is a regression problem or a classification problem. In this project, we want to predict the price of a house given information about it. The price we want to predict is a continuous value; it can be any real number. This can be seen by looking at the target variable in our dataset Sale Price:

```
df['SalePrice'].value_counts() # checking the value counts of Target variable
```

```
140000    18
135000    16
155000    12
139000    11
160000    11
145000    10
110000     9
```

That means that the prediction type that is appropriate to our problem is regression. Now, we move to choose the modelling techniques we want to use. There are a lot of techniques

available for regression problems like Linear Regression, Decision Trees, Random Forest, XGBoost, k-nearest neighbors (KNN) etc. In this project, we will test many modelling techniques, and then choose the technique(s) that yield the best results. The techniques that we will try are:

1. Linear Regression

This technique models the relationship between the target variable and the independent variables (predictors). It fits a linear model with coefficients to the data in order to minimize the residual sum of squares between the target variable in the dataset, and the predicted values by the linear approximation.

2. Random Forest

Bagging is an ensemble method where many base models are used with a randomized subset of data to reduce the variance of the base model.

3. Decision Trees

For this technique, the goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

Each one of these techniques has many algorithmic implementations. We will choose algorithm(s) for each of these techniques in the next section.

4. XGBoost

It's a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks.

5. k-nearest neighbors (KNN)

The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems.

Model Building and Evaluation

In this part, we will build our prediction model: we will choose algorithms for each of the techniques we mentioned in the previous section. After we build the model, we will evaluate its performance and results.

Feature Scaling:

In order to make all algorithms work properly with our data, A way to normalize the input features/variables is the Min-Max scaler. By doing so, all features will be transformed into the range [0,1] meaning that the minimum and maximum value of a feature/variable is going to be 0 and 1, respectively.

Importing libraries for metrics and model building:

```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
```

Import Minimax Scaler for scaling of dataset:

```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
```

Modelling Approach:

For each one 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

Regression Method:

- Using Scikit-Learn, we can build a model for Linear Regression Model

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
```

Splitting the Dataset:

As usual for supervised machine learning problems, we need a training dataset to train our model and a test dataset to evaluate the model. So, we will split our dataset randomly into two parts, one for training and the other for testing. For that, we will use another function from Scikit-Learn called train_test_split():

```
for i in range(0,100):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=i)
    lr.fit(x_train,y_train)
    pred_train=lr.predict(x_train)
    pred_test=lr.predict(x_test)
    print(f'At random state {i},the training accuracy is:{r2_score(y_train,pred_train)}')
    print(f'At random state {i},the testing accuracy is:{r2_score(y_test,pred_test)}')
    print('\n')
```

```
At random state 0,the training accuracy is:0.8611922773551914
At random state 0,the testing accuracy is:0.6281062711267046
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=99)
lr.fit(x_train,y_train)
```

```
LinearRegression()
```

Performance Metric:

For evaluating the performance of our models, we will use mean absolute error (MAE) and mean squared error (MSE). If the predicted value of the element, and the corresponding true value, then for all the elements, RMSE is calculated as:

```
pred=lr.predict(x_test)
```

```
df=pd.DataFrame({'Actual':y_test,'Predicted':pred})
df.head()
```

	Actual	Predicted
1166	40000	57240.896385
610	285000	292995.009450
304	190000	184681.765573
266	175500	177710.196520
434	159950	120132.127974

```
print("error:")
print("Mean absolute error:",mean_absolute_error(y_test,pred))
print("Mean squared error:",mean_squared_error(y_test,pred))
print("Root mean squared error:",np.sqrt(mean_squared_error(y_test,pred)))
```

```
error:
Mean absolute error: 23292.08090260509
Mean squared error: 1092376863.4258146
Root mean squared error: 33051.124994859325
```

In Linear Regressor model, The Root mean squared error value (RMSE) is high so we should compare with more model.

The Predict test and train values are calculated for Linear Regression model:

```
predict_test=lr.predict(x_test)
print(r2_score(y_test,predict_test)*100)
```

```
82.14120374676183
```

```
predict_train=lr.predict(x_train)
print(r2_score(y_train,predict_train)*100)
```

```
83.17074217579034
```

The Cross-validation score:

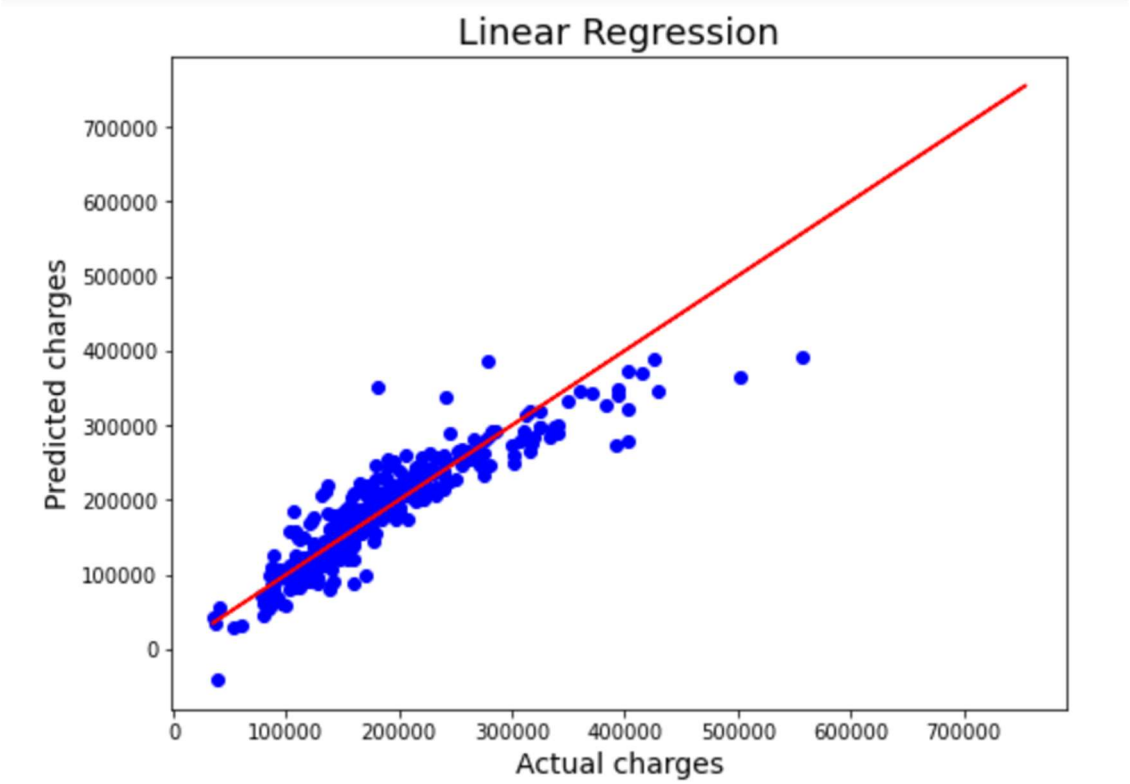
```
Train_accuracy=r2_score(y_train,predict_train)
Test_accuracy=r2_score(y_test,predict_test)

from sklearn.model_selection import cross_val_score
for j in range(2,10):
    cv_score=cross_val_score(lr,x,y,cv=j)
    cv_mean=cv_score.mean()
    print(f'At cross fold(j) the cv score is {cv_mean} and accuracy score for training is {Train_accuracy} and accuracy score fo
    print('\n')
```

```
At cross fold(j) the cv score is 0.6840977991336635 and accuracy score for training is 0.8317074217579034 and accuracy score fo
r tesing is 0.8214120374676184
```

Since the number don't have such impact on the accuracy and cv score.
Here we have handled the problem of the overfitting and the underfitting by checking the training and testing score.

Visualization:

```
plt.figure(figsize=(8,6))
plt.scatter(x=y_test,y=predict_test, color='b')
plt.plot(y,y, color='r')
plt.xlabel('Actual charges',fontsize=14)
plt.ylabel('Predicted charges',fontsize=14)
plt.title('Linear Regression',fontsize=18)
plt.show()
```



- Using Scikit-Learn, we can build a model for Random Forest Regression Model

```
from sklearn.ensemble import RandomForestRegressor
rdr=RandomForestRegressor()
```

Splitting the Dataset:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=99)
rdr.fit(x_train,y_train)
```

```
RandomForestRegressor()
```

```
for i in range(0,100):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=i)
    rdr.fit(x_train,y_train)
    pred_train=rdr.predict(x_train)
    pred_test=rdr.predict(x_test)
    print(f'At random state {i},the training accuracy is:{r2_score(y_train,pred_train)}')
    print(f'At random state {i},the testing accuracy is:{r2_score(y_test,pred_test)}')
    print('\n')
```

```
At random state 0,the training accuracy is:0.9822211262650222
At random state 0,the testing accuracy is:0.7158165016806713
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=99)
rdr.fit(x_train,y_train)
```

```
RandomForestRegressor()
```

Performance Metric:

```
pred=rdr.predict(x_test)
```

```
df=pd.DataFrame({'Actual':y_test,'Predicted':pred})
df.head()
```

	Actual	Predicted
1166	40000	103501.31
610	285000	306372.82
304	190000	214912.54
266	175500	183282.00
434	159950	152483.60

```
print("error:")
print("Mean absolute error:",mean_absolute_error(y_test,pred))
print("Mean squared error:",mean_squared_error(y_test,pred))
print("Root mean squared error:",np.sqrt(mean_squared_error(y_test,pred)))
```

```
error:
Mean absolute error: 17220.003276353276
Mean squared error: 663925650.8746455
Root mean squared error: 25766.75476024572
```

In Random Forest Regressor model, the root mean squared error value is low when compared to linear regressor model but still we proceed with decision tree regressor model.

The Predicted test and train values are calculated for Random Forest Regressor model:

```
predict_test=rdr.predict(x_test)
print(r2_score(y_test,predict_test)*100)
```

```
88.85930988270783
```

```
predict_train=rdr.predict(x_train)
print(r2_score(y_train,predict_train)*100)
```

```
97.38620959439758
```

The Cross-validation score:

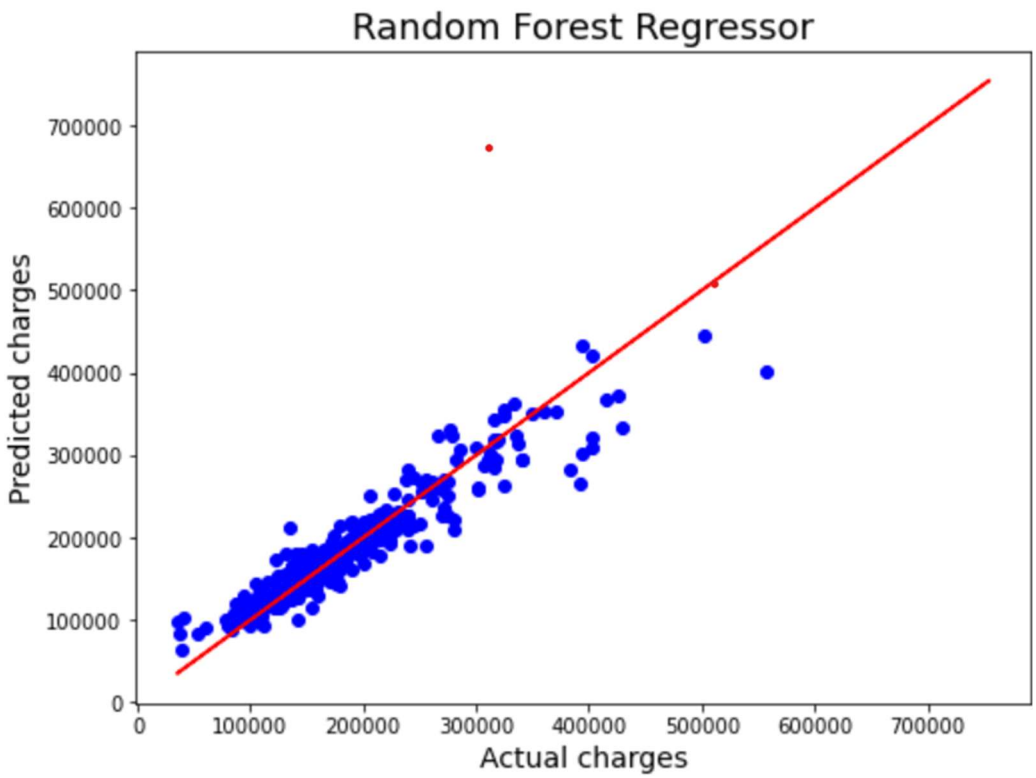
```
Train_accuracy=r2_score(y_train,predict_train)
Test_accuracy=r2_score(y_test,predict_test)

from sklearn.model_selection import cross_val_score
for j in range(2,10):
    cv_score=cross_val_score(rdr,x,y,cv=j)
    cv_mean=cv_score.mean()
    print(f'At cross fold(j) the cv score is {cv_mean} and accuracy score for training is {Train_accuracy} and accuracy score  f
    print('\n')
```

```
At cross fold(j) the cv score is 0.813182818886898 and accuracy score for training is 0.9738620959439759 and accuracy score  fo
r testing is 0.8885930988270783
```

Since the number don’t have such impact on the accuracy and cv_score.

Here we have handled the problem of the overfitting and the underfitting by checking the training and testing score.

Visualization:



- Using Scikit-Learn, we can build a model for Decision Tree Regressor Model:

```
from sklearn.tree import DecisionTreeRegressor
dtr=DecisionTreeRegressor()
```

```
for i in range(0,100):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=i)
    dtr.fit(x_train,y_train)
    pred_train=dtr.predict(x_train)
    pred_test=dtr.predict(x_test)
    print(f'At random state {i},the training accuracy is:{r2_score(y_train,pred_train)}')
    print(f'At random state {i},the testing accuracy is:{r2_score(y_test,pred_test)}')
    print('\n')
```

```
At random state 0,the training accuracy is:1.0
At random state 0,the testing accuracy is:0.4229707223464202


At random state 1,the training accuracy is:1.0
At random state 1,the testing accuracy is:0.7577159767119074
```

Splitting the Dataset:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=6)
dtr.fit(x_train,y_train)
```

```
DecisionTreeRegressor()
```

Performance Metric:

```
df=pd.DataFrame({'Actual':y_test,'Predicted':pred})
df.head()
```

	Actual	Predicted
601	289000	324000.0
137	106500	113000.0
33	128500	134900.0
315	155000	127000.0
199	237500	272000.0

```
print("error:")
print("Mean absolute error:",mean_absolute_error(y_test,pred))
print("Mean squared error:",mean_squared_error(y_test,pred))
print("Root mean squared error:",np.sqrt(mean_squared_error(y_test,pred)))
```

```
error:
Mean absolute error: 25741.56695156695
Mean squared error: 1271277570.7635329
Root mean squared error: 35654.97960683098
```

In Decision Tree Regressor model, The Root mean squared error value is high when compare with other models.

The Predicted test and train values are calculated for Decision Tree Regressor model:

```
predict_test=dtr.predict(x_test)
print(r2_score(y_test,predict_test)*100)
```

```
77.5188152496359
```

```
predict_train=dtr.predict(x_train)
print(r2_score(y_train,predict_train)*100)
```

```
100.0
```

The Cross-validation score:

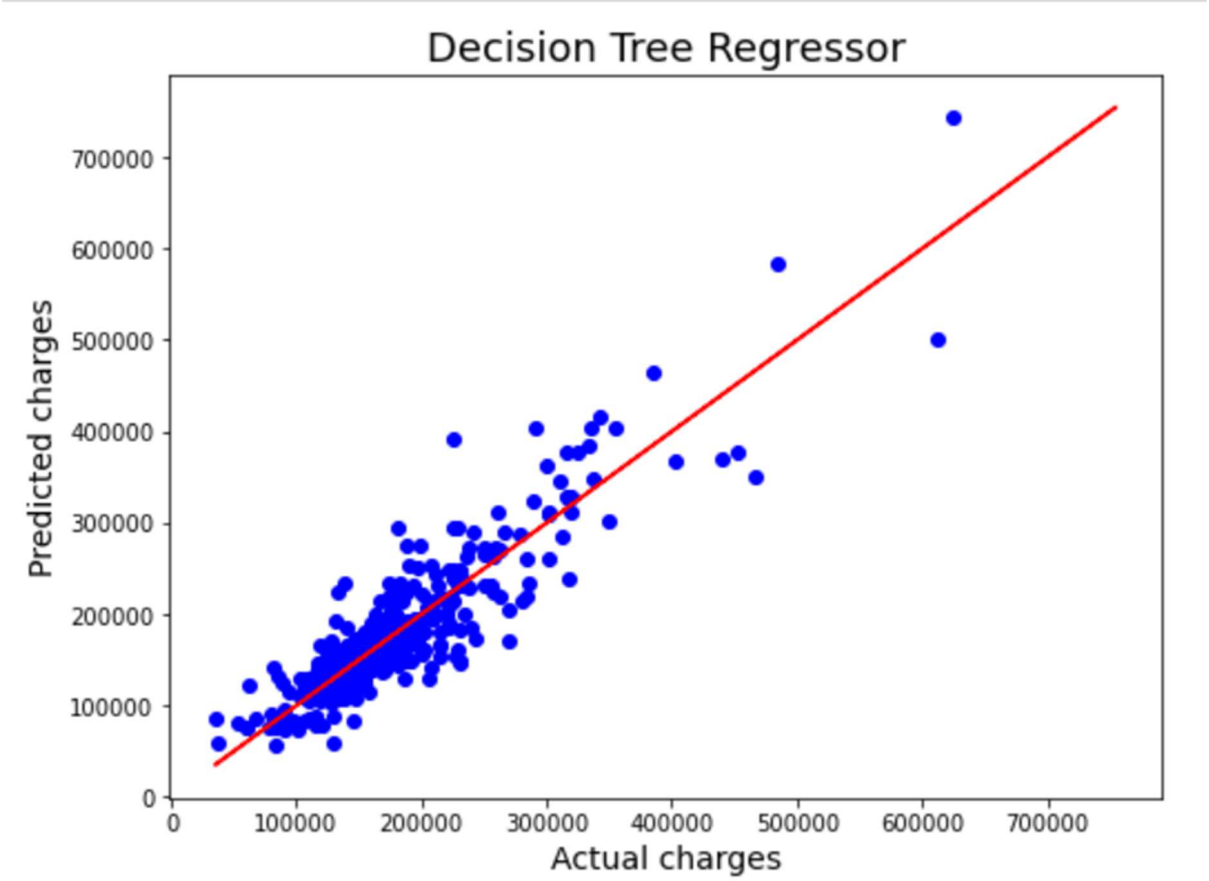
```
Train_accuracy=r2_score(y_train,predict_train)
Test_accuracy=r2_score(y_test,predict_test)

from sklearn.model_selection import cross_val_score
for j in range(2,10):
    cv_score=cross_val_score(dtr,x,y,cv=j)
    cv_mean=cv_score.mean()
    print(f'At cross fold(j) the cv score is {cv_mean} and accuracy score for training is {Train_accuracy} and accuracy score  fo
    print('\n')
```

```
At cross fold(j) the cv score is 0.5684118128193696 and accuracy score for training is 1.0 and accuracy score  for testing is
0.7751881524963591
```

Since the number don’t have such impact on the accuracy and cv_score,

Here we have handled the problem of the overfitting and the underfitting by checking the training and testing score.

Visualization:



- Using Scikit-Learn, we can build a model for XGBoost Regressor Model:

```
from xgboost import XGBRegressor
xgb=XGBRegressor()
```

Splitting the Dataset:

```
for i in range(0,100):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=i)
    xgb.fit(x_train,y_train)
    pred_train=xgb.predict(x_train)
    pred_test=xgb.predict(x_test)
    print(f'At random state {i},the training accuracy is:{r2_score(y_train,pred_train)}')
    print(f'At random state {i},the testing accuracy is:{r2_score(y_test,pred_test)}')
    print('\n')
```

```
At random state 0,the training accuracy is:0.9999708192693281
At random state 0,the testing accuracy is:0.6903268904314472


At random state 1,the training accuracy is:0.9999805300614729
At random state 1,the testing accuracy is:0.8369764399296244
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=91)
xgb.fit(x_train,y_train)
```

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=8, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

Performance Metric:

```
df=pd.DataFrame({'Actual':y_test,'Predicted':pred})
df.head()
```

	Actual	Predicted
541	139000	129710.453125
298	117000	132248.406250
715	107000	147713.718750
144	155000	150678.093750
942	173000	153960.187500

```
print("error:")
print("Mean absolute error:",mean_absolute_error(y_test,pred))
print("Mean squared error:",mean_squared_error(y_test,pred))
print("Root mean squared error:",np.sqrt(mean_squared_error(y_test,pred)))
```

```
error:
Mean absolute error: 17275.816228187323
Mean squared error: 683514616.461609
Root mean squared error: 26144.11246268668
```

In XGBoost Regressor, The Root mean squared error value is slightly high when compare with Random Forest.

The Predicted test and train values are calculated for XGBoost Regressor model:

```
predict_test=xgb.predict(x_test)
print(r2_score(y_test,predict_test)*100)
```

```
89.23427072753314
```

```
predict_train=xgb.predict(x_train)
print(r2_score(y_train,predict_train)*100)
```

```
99.9952165169556
```

The Cross-validation score:

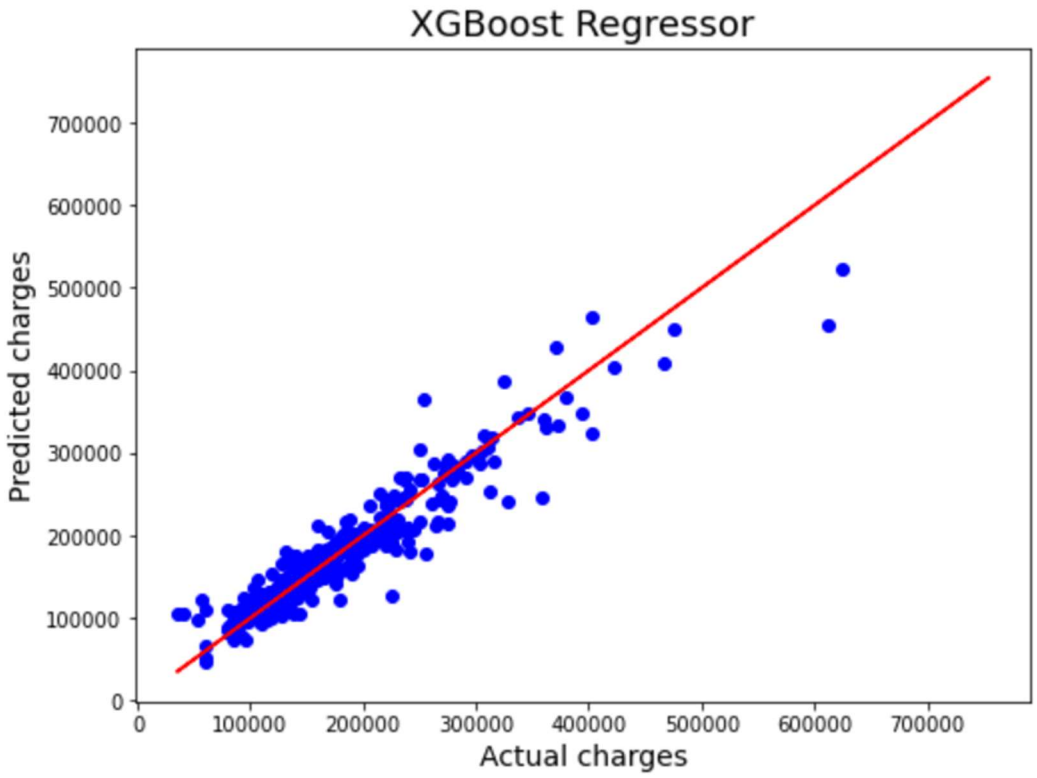
```
Train_accuracy=r2_score(y_train,predict_train)
Test_accuracy=r2_score(y_test,predict_test)

from sklearn.model_selection import cross_val_score
for j in range(2,10):
    cv_score=cross_val_score(xgb,x,y,cv=j)
    cv_mean=cv_score.mean()
    print(f'At cross fold(j) the cv score is {cv_mean} and accuracy score for training is {Train_accuracy} and accuracy score  f
    print('\n')
```

```
At cross fold(j) the cv score is 0.8310814932956666 and accuracy score for training is 0.999952165169556 and accuracy score  fo
r testing is 0.8923427072753314
```

Since the number don't have such impact on the accuracy and cv_score.

Here we have handled the problem of the overfitting and the underfitting by checking the training and testing score.

Visualization:



- Using Scikit-Learn, we can build a model for KNeighbors Regressor Model:

```
from sklearn.neighbors import KNeighborsRegressor
knn=KNeighborsRegressor()
```

Splitting the Dataset:

```
for i in range(0,100):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=i)
    knn.fit(x_train,y_train)
    pred_train=knn.predict(x_train)
    pred_test=knn.predict(x_test)
    print(f'At random state {i},the training accuracy is:{r2_score(y_train,pred_train)}')
    print(f'At random state {i},the testing accuracy is:{r2_score(y_test,pred_test)}')
    print('\n')
```

```
At random state 0,the training accuracy is:0.8534033664108981
At random state 0,the testing accuracy is:0.6612470529095025


At random state 1,the training accuracy is:0.8184293174912819
At random state 1,the testing accuracy is:0.7571249028987074
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=74)
knn.fit(x_train,y_train)
```

```
KNeighborsRegressor()
```

Performance Metric:

```
df=pd.DataFrame({'Actual':y_test,'Predicted':pred})
df.head()
```

	Actual	Predicted
823	120500	120380.0
902	97000	99700.0
903	145000	131551.6
795	107000	127900.0
535	157900	141390.0

```
print("error:")
print("Mean absolute error:",mean_absolute_error(y_test,pred))
print("Mean squared error:",mean_squared_error(y_test,pred))
print("Root mean squared error:",np.sqrt(mean_squared_error(y_test,pred)))
```

```
error:
Mean absolute error: 21576.032478632478
Mean squared error: 853157344.5457551
Root mean squared error: 29208.857296131173
```

In KNeighbors Regressor, The Root mean squared error value is high when compare with another model.

The Predicted test and train values are calculated for KNeighbors Regressor model:

```
predict_test=knn.predict(x_test)
print(r2_score(y_test,predict_test)*100)
```

```
81.81449423787076
```

```
predict_train=knn.predict(x_train)
print(r2_score(y_train,predict_train)*100)
```

```
81.60799145180421
```

The Cross-validation score:

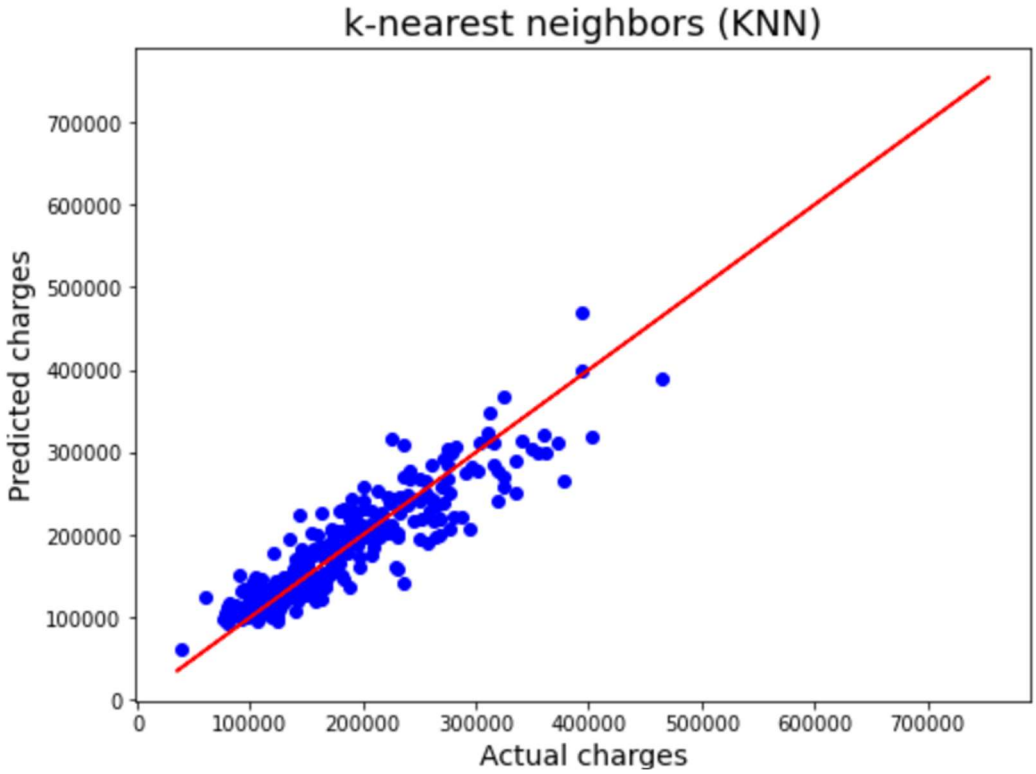
```
Train_accuracy=r2_score(y_train,predict_train)
Test_accuracy=r2_score(y_test,predict_test)

from sklearn.model_selection import cross_val_score
for j in range(2,10):
    cv_score=cross_val_score(knn,x,y,cv=j)
    cv_mean=cv_score.mean()
    print(f'At cross fold(j) the cv score is {cv_mean} and accuracy score for training is {Train_accuracy} and accuracy score  fo
    print('\n')
```

```
At cross fold(j) the cv score is 0.7348250914784015 and accuracy score for training is 0.8160799145180421 and accuracy score  f
or testing is 0.8181449423787076
```

Since the number don’t have such impact on the accuracy and cv_score.

Here we have handled the problem of the overfitting and the underfitting by checking the training and testing score.

Visualization:



Hyper Parameter Tuning:

Hyperparameters are crucial as they control the overall behaviour of a machine learning model. The ultimate goal is to find an optimal combination of hyperparameters that minimizes a predefined loss function to give better results.

The predict test value for both Random Forest regressor and XGBoost Regressor are same values.so, we need to perform hyperparameter tuning for both of them.

According to the performance of modelling approach, the RMSE value is low for Random Forest Regressor model but R2 score is similar for both Random Forest and XGBoost. Apply ensemble and regularization technique to determine the optimal values of Hyper Parameters.

Ensemble techniques:

Ensemble methods is a machine learning **technique** that combines several base models in order to produce one optimal predictive model. To better understand this definition let's take a step back into ultimate goal of machine learning and model building.

Importing GridSearchCV

```
from sklearn.model_selection import GridSearchCV
```

Hyperparameter Tuning for Random Forest Regressor:

Firstly, we will use GridSearchCV() to search for the best model parameters in a parameter space provided by us. criterion, max features and random state.

```
from sklearn.ensemble import RandomForestRegressor
parameters={'criterion':['mse','mae'],'max_features':['auto','sqrt','log2'],'random_state':list(range(0,10))}
rdr=RandomForestRegressor()
clf=GridSearchCV(rdr,parameters)
clf.fit(x_train,y_train)
print(clf.best_params_)
```

```
{'criterion': 'mae', 'max_features': 'auto', 'random_state': 8}
```

We defined the parameter space above using reasonable values for chosen parameters.

```
rdr=RandomForestRegressor(criterion='mae',max_features='auto',random_state=8)
rdr.fit(x_train,y_train)
rdr.score(x_train,y_train)
pred_decision=rdr.predict(x_test)
rdrs=r2_score(y_test,pred_decision)
print('R2 Score:',rdrs*100)
rdrscore=cross_val_score(rdr,x,y,cv=5)
rdrc=rdrscore.mean()
print('Cross Val Score:',rdrc*100)
```

```
R2 Score: 86.79417521156296
Cross Val Score: 84.6832604168003
```

We defined the R2 score and Cross validation score of ensemble technique using chosen parameters. We are getting model accuracy and cross validation has 86.8% & 84.7% respectively.

Hyperparameter Tuning for XGBoost Regressor:

Secondly, we will use GridSearchCV() to search for the best model parameters in a parameter space provided by us. max_depth, learning_rate and n_estimators.

```
from sklearn.ensemble import GradientBoostingRegressor
parameters = {"max_depth": [4, 5, 6],"learning_rate": [0.005, 0.009, 0.01],
                          "n_estimators": [700, 1000, 2500]}
xgb=XGBRegressor()
clf=GridSearchCV(xgb,parameters)
clf.fit(x_train,y_train)
print(clf.best_params_)
```

```
{'learning_rate': 0.009, 'max_depth': 4, 'n_estimators': 2500}
```

We defined the parameter space above using reasonable values for chosen parameters.

```
xgb=GradientBoostingRegressor(max_depth=4,learning_rate=0.009,n_estimators=2500)
xgb.fit(x_train,y_train)
xgb.score(x_train,y_train)
pred_decision=xgb.predict(x_test)
xgbs=r2_score(y_test,pred_decision)
print('R2 Score:',xgbs*100)
xgbscore=cross_val_score(xgb,x,y,cv=5)
xgbc=xgbscore.mean()
print('Cross Val Score:',xgbc*100)
```

```
R2 Score: 89.15088827778462
Cross Val Score: 86.577520878503
```

We defined the R2 score and Cross validation score of ensemble technique using chosen parameters. We are getting model accuracy and cross validation has 89.2% & 86.6% respectively.

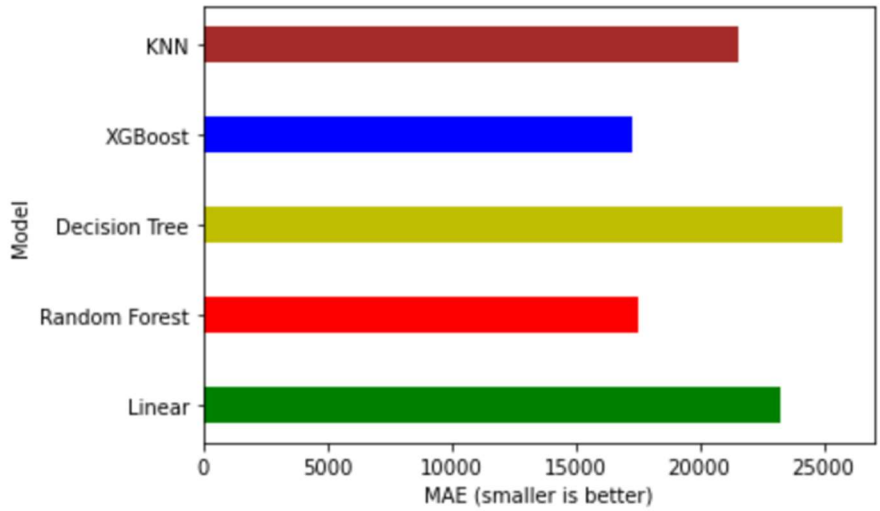
We are getting model accuracy and cross validation has 89.2% & 86.6% respectively.

Finally, Hyperparameter Tuning is compared for both the model. In XGBoost model accuracy and cross validation is better when compared to Random Forest. We consider XGboost regressor is our best model for these datasets

Performance Interpretation:

MAE (Mean Absolute Error)

```
x = ['Linear','Random Forest', 'Decision Tree', 'XGBoost', 'KNN']
y = [23292.08, 17556.97 , 25741.56, 17275.81, 21576.03]
colors = ["g", "r", "y", "b", "brown"]
fig, js = plt.subplots()
plt.barh(y=range(len(x)), tick_label=x, width=y, height=0.4, color=colors);
js.set(xlabel="MAE (smaller is better)", ylabel="Model");
```

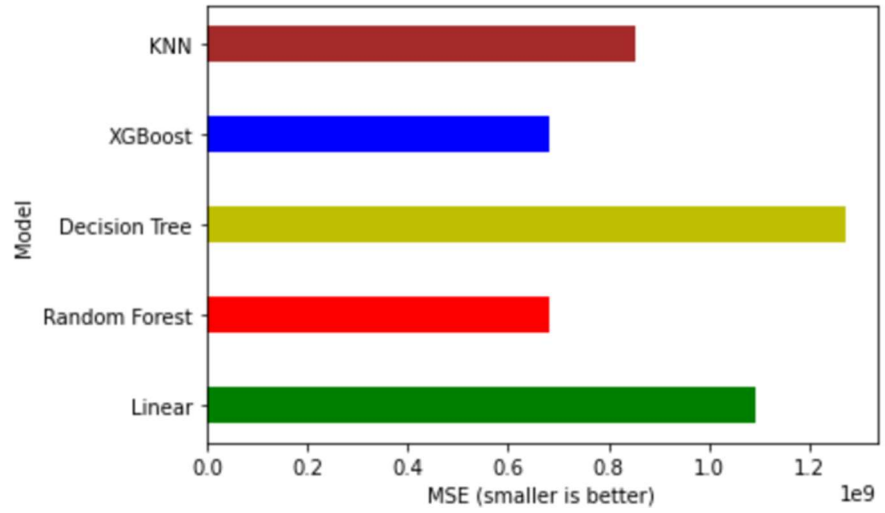


By looking at the table and the graph, we can see that XGBoost model has the smallest MAE, 17,275.81 followed by Random Forest model with a little larger error of 17556.97. After that, K-Nearest Neighbors come with error of 21,576.03. at last, the linear and Decision Tree model comes with an similar errors: 23,292.08 and 25,741.56 respectively.

So, in our experiment, the best model is XGBoost model and the worst model is Decision Tree model. We can see that the difference in MAE between the best model and the worst model is significant; the best model has almost 30 % of the error of the worst model.

MSE (Mean Squared Error)

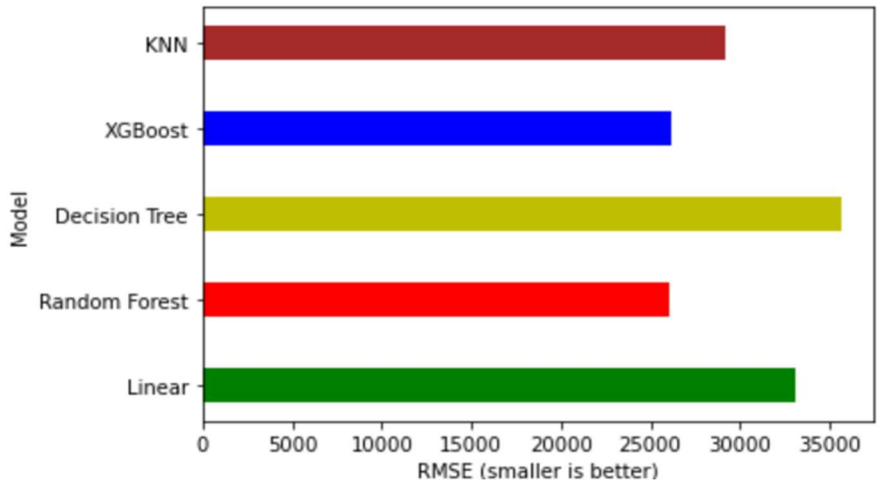
```
x = ['Linear','Random Forest', 'Decision Tree', 'XGBoost', 'KNN']
y = [1092376863.42, 681447503.74, 1271277570.76, 683514616.46, 853157344.54]
colors = ["g", "r", "y", "b", "brown"]
fig, js = plt.subplots()
plt.barh(y=range(len(x)), tick_label=x, width=y, height=0.4, color=colors);
js.set(xlabel="MSE (smaller is better)", ylabel="Model");
```



By looking at the table and the graph, we can see that Random Forest model has the smallest MSE, 68,14,47,503.74 followed by XGBoost with a little larger error of 68,35,14,616.46. After that, K-Nearest Neighbors come with errors of 85,31,57,344.54. At last, the Linear and Decision tree model comes with an similar errors: 109,23,76,863.42 and 127,12,77,570.76 respectively.
So, in our experiment, the best model is Random Forest model and the worst model is Decision Tree model. We can see that the difference in MSE between the best model and the worst model is significant; the best model has almost 40 % of the error of the worst model.

RMSE (Root Mean Squared Error)

```
x = ['Linear','Random Forest', 'Decision Tree','XGBoost', 'KNN']
y = [33051.12, 26104.54, 35654.97, 26144.11, 29208.85]
colors = ["g", "r", "y", "b", "brown"]
fig, js = plt.subplots()
plt.barh(y=range(len(x)), tick_label=x, width=y, height=0.4, color=colors);
js.set(xlabel="RMSE (smaller is better)", ylabel="Model");
```

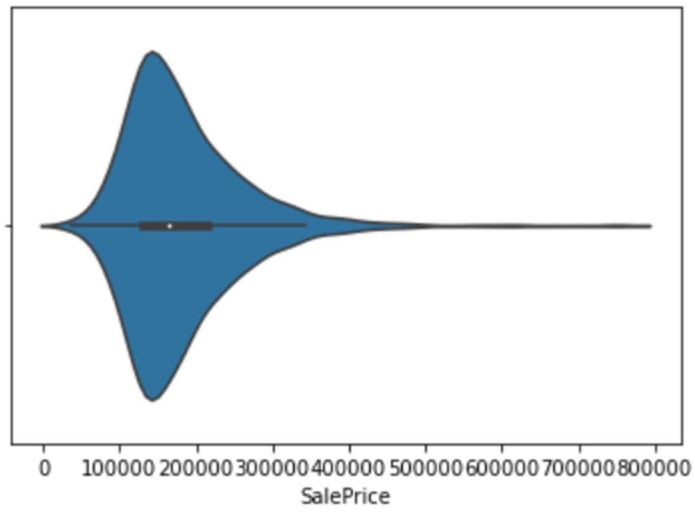


By looking at the table and the graph, we can see that Random Forest model has the smallest RMSE, 26,104.54 followed by XGBoost with a little larger error of 26,144.11. After that, K-Nearest Neighbors come with error of 29,208.85. At last, the linear and Decision tree model comes with an similar errors: 33,051.12 and 35,654.97 respectively.

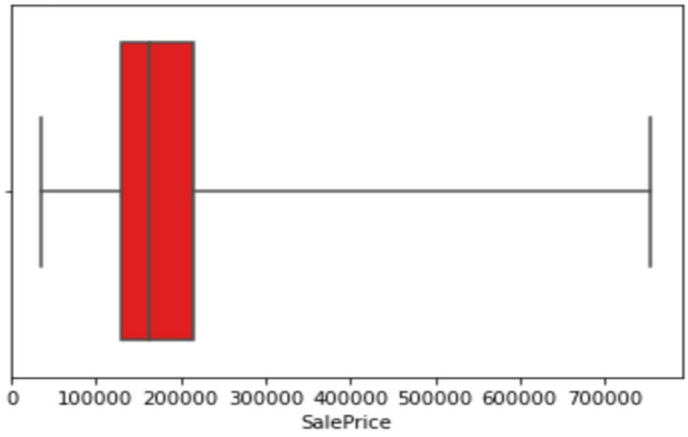
So, in our experiment, the best model is Random Forest and the worst model is Decision Tree model. We can see that the difference in MSE between the best model and the worst model is significant; the best model has almost 25 % of the error of the worst model.

We chose the root mean squared error (RMSE) as our performance metric to evaluate and co mpare models. RMSE presents a value that is easy to understand; it shows the average value of model error. For example, for our Random Forest model, its RMSE is 26104.54, which me ans that on average Random Forest will predict a value that is bigger or smaller than the true value by 26104.54. Now to understand how good this RMSE is, we need to know the range a nd distribution of the data. In our case, we need to see the values of the target variable Sale Pr ice which contains the actual house prices. Let's see the violin plot, box plot, dist plot of Sale Price in our dataset:

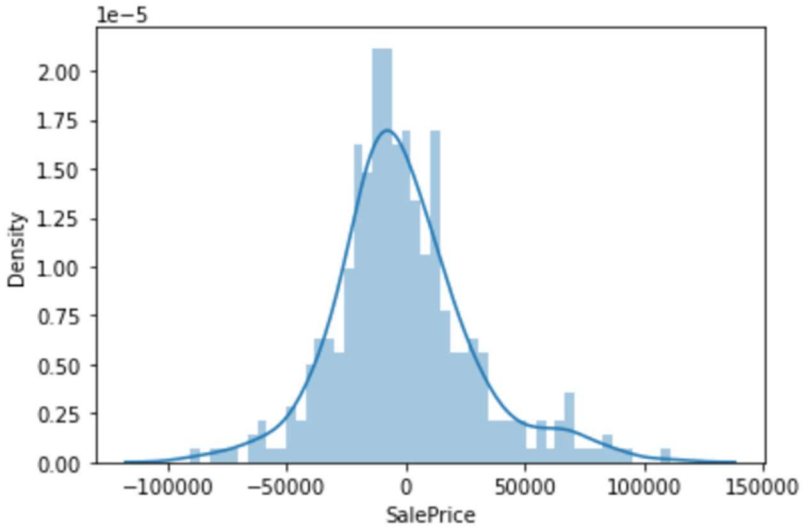
- **Violin plot:**



- **Box plot:**



- **Dist plot:**



Notice here that our residuals looked to be normally distributed and that's really a good sign which means that our model was a correct choice for the data.

From these plots above, we can understand the distribution of Sale Price.

Finally, we came to know that our best model is both XGBoost and the worst model is Decision Tree.

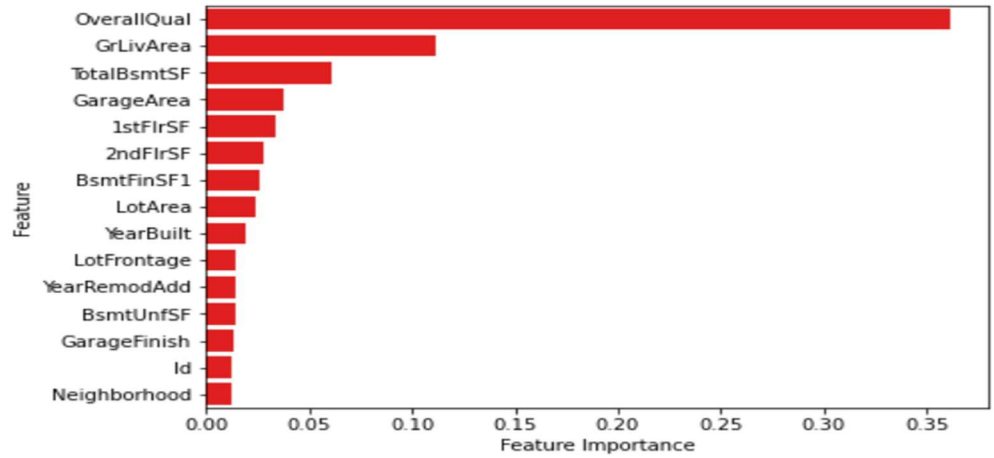
Feature Importance's:

Some of the models we used provide the ability to see the importance of each feature in the dataset after fitting the model. We will look at the feature importance's provided by both Random Forest and XGBoost models. We have 81 features in our data which is a big number, so we will take a look at the 15 most important features.

Random Forest

Now, let's see the most important features as for Random Forest model:

```
rdr_feature_importances = rdr.feature_importances_
rdr_feature_importances = pd.Series(rdr_feature_importances,
                                    index=x_train.columns.values).sort_values(ascending=False).head(15)

fig, ax = plt.subplots(figsize=(7,5))
sns.barplot(x=rdr_feature_importances, y=rdr_feature_importances.index, color="r");
plt.xlabel('Feature Importance');
plt.ylabel('Feature');
```

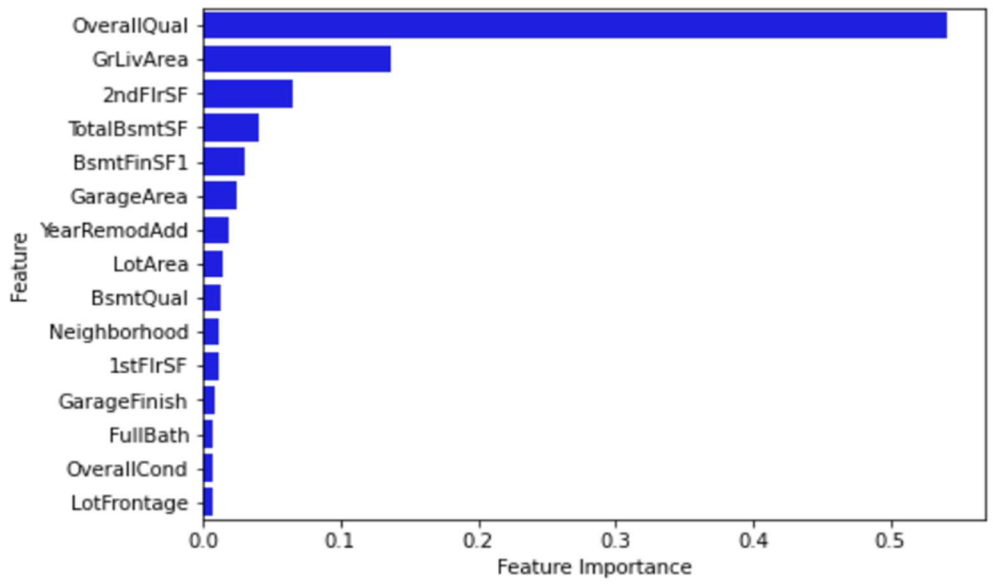


Notice here in feature importance of Random Forest, the overall qualification feature plays a prominent role for target variable.

XGBoost

Let's discover the most important features as determined by XGBoost model:

```
xgb_feature_importances = xgb.feature_importances_
xgb_feature_importances = pd.Series(xgb_feature_importances,
                                    index=x_train.columns.values).sort_values(ascending=False).head(15)

fig, ax = plt.subplots(figsize=(7,5))
sns.barplot(x=xgb_feature_importances, y=xgb_feature_importances.index, color="b");
plt.xlabel('Feature Importance');
plt.ylabel('Feature');
```

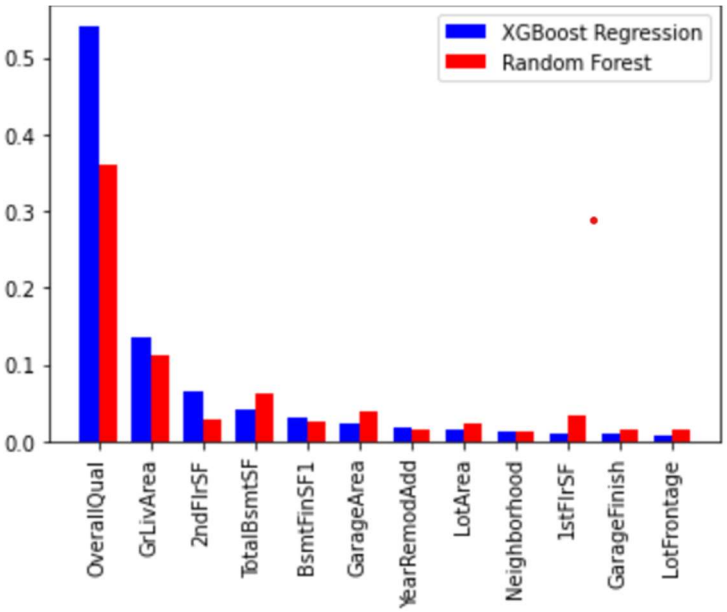


Notice here in feature importance of XGboost, the overall qualification feature plays a prominent role for target variable.

Common Important Features:

Now, let us see which features are among the most important features for both XGBoost and Random Forest models, and let's find out the difference in their importance regarding the two models:

```
common_imp_feat = [x for x in xgb_feature_importances.index
                   if x in rdr_feature_importances.index]
commImpFeat_xgb = [xgb_feature_importances[x]
                          for x in common_imp_feat]
commImpFeat_rdr = [rdr_feature_importances[x]
                         for x in common_imp_feat]

ind = np.arange(len(commImpFeat_xgb))
width = 0.35

fig, ax = plt.subplots()
ax.bar(ind - width/2, commImpFeat_xgb, width,
       color='b', label='XGBoost Regression');
ax.bar(ind + width/2, commImpFeat_rdr, width,
       color='r', label='Random Forest')
ax.set_xticks(ind);
ax.set_xticklabels(common_imp_feat);
ax.legend();
plt.xticks(rotation=90);
```



Finally, we noticed that overall qualification feature plays a prominent role for deciding the Sale price of the house.

Now, we compare with actual final vs sample prediction:

```
data=pd.DataFrame({'Y Test':y_test , 'Prediction':predict_test},columns=['Y Test','Prediction'])
sns.lmplot(x='Y Test',y='Prediction',data=data,palette='rainbow')
data.head()
```

	Y Test	Prediction
823	120500	120380.0
902	97000	99700.0
903	145000	131551.6
795	107000	127900.0
535	157900	141390.0

Conclusion

In this paper, we built several regression models to predict the price of some house given some of the house features. We evaluated and compared each model to determine the one with highest performance. We also looked at how some models rank the features according to their importance. In this paper, we followed the data science process starting with getting the data, then cleaning and pre-processing the data, followed by exploring the data and building models, then evaluating the results and communicating them with visualizations.

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 be used also with datasets that covered areas provided that they contain the same features. We also suggest that people take into consideration the features that were deemed as most important as seen in the previous section; this might help them estimate the house price better.

Learning Outcomes of the Study in respect of Data Science:

- Obtain, clean/process, and transform data.
- Analyze and interpret data using an ethically responsible approach.
- Use appropriate models of analysis, assess the quality of input, derive insight from results, and investigate potential issues.
- Apply computing theory, languages, and algorithms, as well as mathematical and statistical models, and the principles of optimization to appropriately formulate and use data analyses
- Formulate and use appropriate models of data analysis to solve hidden solutions to business-related challenges

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. Some of the ideas are listed below:

- Combine the applicants with 1,2,3 or more dependents and make a new feature as discussed in the EDA part.

- Make independent vs independent variable visualizations to discover some more patterns.
- Arrive at the EMI using a better formula which may include interest rates as well.
- Try neural network using TensorFlow or PyTorch.

****