FLIP ROBO

HOUSING PRICE PREDICTION PROJECT

Submitted by:-

PANKAJ SURYAWANSHI

Internship-30

ACKNOWLEDGMENT

I would like to convey my heartfelt gratitude to Flip Robo Technologies for providing me with this wonderful opportunity to work on a Machine Learning project “Housing Price Prediction” and also want to thank my SME “Shwetank Mishra“ for providing the dataset and directions to complete this project.

I would also like to thank my academic “Data Trained Education” and their team who has helped me to learn Machine Learning and how to work on it. This project would not have been accomplished without their help and insights.

Working on this project was an incredible experience as I learnt more from this Project during completion as I have to do some research also.

INTRODUCTION

Houses are one of the necessary need of each and every person around the globe and therefore housing and real estate market is one of the markets which is one of the major contributors in the world's economy. It is a very large market and there are various companies working in the domain. Data science comes as a very important tool to solve problems in the domain to help the companies increase their overall revenue, profits, improving their marketing strategies and focusing on changing trends in house sales and purchases. Predictive modelling, Market mix modelling, recommendation systems are some of the machine learning techniques used for achieving the business goals for housing companies. Our problem is related to one such housing company. A US-based housing company named Surprise Housing has decided to enter the Australian market. The company uses data analytics to purchase houses at a price below their actual values and flip them at a higher price. For the same purpose, the company has collected a data set from the sale of houses in Australia.

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 project, a machine learning model is proposed to predict a house price based on data related to the house. We will show code and output of our model step by step with its output. 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 techniques in order to obtain clean data
- To visualize data with matplot lib.
- To build machine learning models to predict house sale price
- To analyse and compare model's performance in order to choose the best model

Review of Literature

Machine learning (ML) is a type of artificial intelligence (AI) that allows software applications to become more accurate at predicting outcomes without being explicitly programmed to do so. Machine learning algorithms use historical data as input to predict new output values.
The type of algorithm data scientists choose to use depends on what type of data they want to predict.

Machine learning algorithms are classified into three divisions: Supervised learning, Unsupervised learning and Reinforcement learning.

Supervised learning: In this type of machine learning, data scientists supply algorithms with labeled training data and define the variables they want the algorithm to assess for correlations. Both the input and the output of the algorithm is specified.

Unsupervised learning: This type of machine learning involves algorithms that train on unlabeled data. The algorithm scans through data sets looking for any meaningful connection. The data that algorithms train on as well as the predictions or recommendations they output are predetermined.

Reinforcement learning: Data scientists typically use reinforcement learning to teach a machine to complete a multi-step process for which there are clearly defined rules. Data scientists program an algorithm to complete a task and give it positive or negative cues as it works out how to complete a task. But for the most part, the algorithm decides on its own what steps to take along the way.

We used regression models for predicting Sale price of houses by 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.

Linear Regression:

Linear Regression is **a machine learning algorithm based on supervised learning**. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting.

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:

Random Forest Regression is **a supervised learning algorithm that uses ensemble learning method for regression**. Ensemble learning method is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model.

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 .

Motivation for the Problem Undertaken

Houses are one of the necessary need of each and every person around the globe and therefore housing and real estate market is one of the markets which is one of the major contributors in the world's economy.

Analytical Problem Framing

Mathematical/ Analytical Modeling of the Problem

To analyze the data, there there are many techniques but the most common are these two techniques:

- 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 overdependent variables information that can be leveraged to make essential decision.

The most traditional regression model is linear regression, decision tree regression, randomforest regression, gradient boosting regression and knn-neighbours.

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

Data Sources and their formats

In this project, 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: housing_train.csv and housing_test.csv

Checking Top 5 rows Data

```
#To print all columns
pd.set_option('display.max_columns',None)

housing_train.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType
0	127	120	RL	NaN	4928	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	NPkVill	Norm	Norm	TwnhsE
1	889	20	RL	95.0	15865	Pave	NaN	IR1	Lvl	AllPub	Inside	Mod	NAmes	Norm	Norm	1Fam
2	793	60	RL	92.0	9920	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl	NoRidge	Norm	Norm	1Fam
3	110	20	RL	105.0	11751	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	NWAmes	Norm	Norm	1Fam
4	422	20	RL	NaN	16635	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	NWAmes	Norm	Norm	1Fam

```
#To print all columns
pd.set_option('display.max_columns',None)
housing_test.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType
0	337	20	RL	86.0	14157	Pave	NaN	IR1	HLS	AllPub	Corner	Gtl	StoneBr	Norm	Norm	1Fam
1	1018	120	RL	NaN	5814	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl	StoneBr	Norm	Norm	TwnhsE
2	929	20	RL	NaN	11838	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	1Fam
3	1148	70	RL	75.0	12000	Pave	NaN	Reg	Bnk	AllPub	Inside	Gtl	Crawfor	Norm	Norm	1Fam
4	1227	60	RL	86.0	14598	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl	Somerst	Feedr	Norm	1Fam

Data Preprocessing Done by:

- ✓ Checking Total Numbers of Rows and Column
- ✓ Checking All Column Name
- ✓ Checking Data Type of All Data
- ✓ Checking for Null Values
- ✓ Information about Data
- ✓ Checking total number of unique value
- ✓ Checking all value of each columns
- ✓ Handling Null Values

Handling Null Values

```
#these columns consist mostly Null Values so it will not help in prediction
housing.drop(columns=['Alley', 'PoolQC', 'MiscFeature'],inplace=True)
```

```
#Column ID have unique value so we will drop this column
housing.drop(columns=['Id'],inplace=True)
```

SalePrice is our Target Column

Filling continuous column with mean

```
housing["LotFrontage"].fillna(housing["LotFrontage"].mean(), inplace=True)
```

```
housing["MasVnrArea"].fillna(housing["MasVnrArea"].mean(), inplace=True)
```

```
housing["GarageYrBlt"].fillna(housing["GarageYrBlt"].mean(), inplace=True)
```

```
categorical_feature= ['MasVnrType', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Electrical',
                      'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond', 'Fence']
```

```
housing['Fence'].value_counts()
```

```
MnPrv    157
GdPrv     59
GdWo      54
MnWw      11
Name: Fence, dtype: int64
```

```
for feature in categorical_feature:
    if(housing[feature].isnull().sum()*100/len(housing))>0:
        housing[feature] = housing[feature].fillna(housing[feature].mode()[0])
```

```
housing.isnull().sum()
```

```
MSSubClass       0
MSZoning         0
LotFrontage      0
LotArea          0
Street           0
LotShape         0
LandContour      0
Utilities        0
LotConfig        0
LandSlope        0
```

✓ 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), So, we will mention the feature name with a short description of its meaning

Features with Description:

i. **MSSubClass:** The type of the house involved in the sale
ii. **MSZoning:** The general zoning classification of the sale
iii. **LotFrontage:** Linear feet of street connected to the house
iv. **LotArea:** Lot size in square feet
v. **Street:** Type of road access to the house
vi. **Alley:** Type of alley access to the house
vii. **LotShape:** General shape of the house
viii. **LandContour:** House flatness
ix. **Utilities:** Type of utilities available
x. **LotConfig:** Lot configuration
xi. **LandSlope:** House Slope
xii. **Neighborhood:** Locations within Ames city limits
xiii. **Condition1:** Proximity to various conditions
xiv. **Condition2:** Proximity to various conditions (if more than one is present)
xv. **BldgType:** House type
xvi. **HouseStyle:** House style
xvii. **OverallQual:** Overall quality of material and finish of the house
xviii. **OverallCond:** Overall condition of the house
xix. **YearBuilt:** Construction year
xx. **YearRemodAdd:** Remodel year (if no remodeling nor addition, same as YearBuilt)
xxi. **RoofStyle:** Roof type
xxii. **RoofMatl:** Roof material
xxiii. **Exterior1st:** Exterior covering on house
xxiv. **Exterior2nd:** Exterior covering on house (if more than one material)
xxv. **MasVnrType:** Type of masonry veneer
xxvi. **MasVnrArea:** Masonry veneer area in square feet
xxvii. **ExterQual:** Quality of the material on the exterior
xxviii. **ExterCond:** Condition of the material on the exterior
xxix. **Foundation:** Foundation type
xxx. **BsmtQual:** Basement height
xxxi. **BsmtCond:** Basement Condition
xxxii. **BsmtExposure:** Refers to walkout or garden level walls
xxxiii. **BsmtFinType1:** Rating of basement finished area

xxxiv. **BsmtFinSF1:** Type 1 finished square feet
xxxv. **BsmtFinType2:** Rating of basement finished area (if multiple types)
xxxvi. **BsmtFinSF2:** Type 2 finished square feet
xxxvii. **BsmtUnfSF:** Unfinished basement area in square feet
xxxviii. **TotalBsmtSF:** Total basement area in square feet
xxxix. **Heating:** Heating type
xl. **HeatingQC:** Heating quality and condition
xli. **CentralAir:** Central air conditioning
xlii. **Electrical:** Electrical system type
xliii. **1stFlrSF:** First floor area in square feet
xliv. **2ndFlrSF:** Second floor area in square feet
xlv. **LowQualFinSF:** Low quality finished square feet in all floors
xlvi. **GrLivArea:** Above-ground living area in square feet
xlvii. **BsmtFullBath:** Basement full bathrooms
xlviii. **BsmtHalfBath:** Basement half bathrooms
xlix. **FullBath:** Full bathrooms above ground
l. **HalfBath:** Half bathrooms above ground
li. **Bedroom:** Bedrooms above ground
lii. **Kitchen:** Kitchens above ground
liii. **KitchenQual:** Kitchen quality
liv. **TotRmsAbvGrd:** Total rooms above ground (excluding bathrooms)
lv. **Functional:** Home functionality
lvi. **Fireplaces:** Number of fireplaces
lvii. **FireplaceQu:** Fireplace quality
lviii. **GarageType:** Garage location
lix. **GarageYrBlt:** Year garage was built in
lx. **GarageFinish:** Interior finish of the garage
lxi. **GarageCars:** Size of garage (in car capacity)
lxii. **GarageArea:** Garage size in square feet
lxiii. **GarageQual:** Garage quality
lxiv. **GarageCond:** Garage condition
lxv. **PavedDrive:** How driveway is paved
lxvi. **WoodDeckSF:** Wood deck area in square feet
lxvii. **OpenPorchSF:** Open porch area in square feet
lxviii. **EnclosedPorch:** Enclosed porch area in square feet
lxix. **3SsnPorch:** Three season porch area in square feet
lxx. **ScreenPorch:** Screen porch area in square feet
lxxi. **PoolArea:** Pool area in square feet
lxxii. **PoolQC:** Pool quality
lxxiii. **Fence:** Fence quality
lxxiv. **MiscFeature:** Miscellaneous feature
lxxv. **MiscVal:** Value of miscellaneous feature
lxxvi. **MoSold:** Sale month
lxxvii. **YrSold:** Sale year
lxxviii. **SaleType:** Sale type
lxxix. **SaleCondition:** Sale condition

✓ Descriptive Statistics

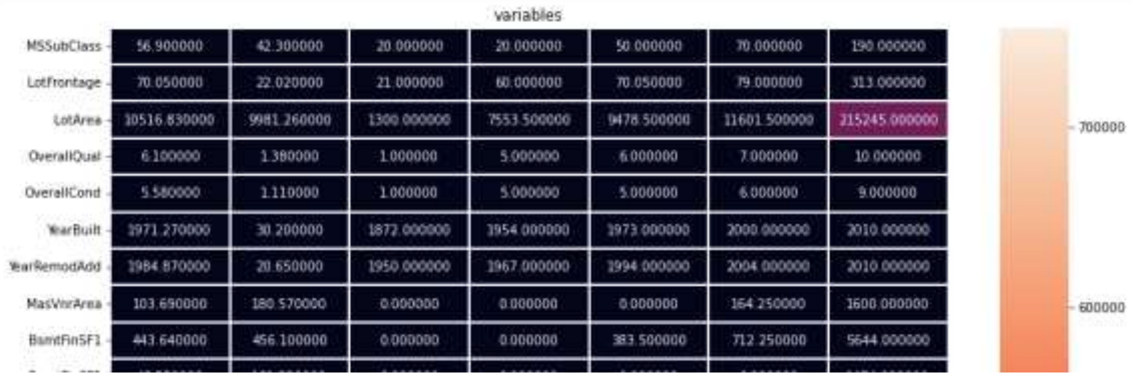
In [37]:
```python
#To print all columns
pd.set_option('display.max_columns',None)
# Description of Dataset : works only on continuous column
housing.describe()
```

Out[37]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF
count	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000
mean	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262	443.639726	46.549315	567.240411
std	42.300571	22.024023	9981.264932	1.382997	1.112799	30.202904	20.645407	180.569112	456.098091	161.319273	441.866955
min	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.000000	0.000000	0.000000
25%	20.000000	60.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000	0.000000	0.000000	223.000000
50%	50.000000	70.049958	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000	383.500000	0.000000	477.500000
75%	70.000000	79.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	164.250000	712.250000	0.000000	808.000000
max	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	1474.000000	2336.000000

Checking Description through heatmap also.

```python
plt.figure(figsize=(15,20))
sns.heatmap(round(housing.describe()[1:].transpose(),2),linewidth=2,annot=True,fmt='f')
plt.xticks(fontsize=18)
plt.xticks(fontsize=12)
plt.title('variables')
plt.show()
```



Outcome of Describe of Datasets:

- We are determining Mean, Standard Deviation, Minimum and Maximum Values of each column. The summary of this dataset looks good as there are no negative/ invalid value present.
- Total No of Rows: 1460 Total No. of Columns: 78
- Describe Method works only on continuous column
- We observe that the dataset seems to be having more outliers as well as skewness in the data.

Making DataFrame of Nominal Data

```python
housing_categorical=housing[['MSZoning', 'Street', 'LotShape', 'LandContour', 'Utilities', 'LotConfig', 'LandSlope',
                             'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl',
                             'Exterior1st', 'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
                             'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Heating', 'HeatingQC', 'CentralAir',
                             'Electrical', 'KitchenQual', 'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish',
                             'GarageQual', 'GarageCond', 'PavedDrive', 'Fence', 'SaleType', 'SaleCondition', 'dataset']].copy()
```

Making DataFrame of Continuous Data

```python
housing_continuous=housing[['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
                            'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF',
                            'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr',
                            'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea',
                            'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
                            'MoSold', 'YrSold', 'SalePrice']].copy()
```

✓ Data Visualization

1. Univariate Analysis
✓ Using Countplot (for categorical data)
✓ Using Histplot (for continuous data)

2. Bivariate Analysis (for comparision between features and target)
✓ Using Countplot (for comparision between categorical data and target)
✓ Using Scatterplot (for comparision between continuous data and target)

3. Multivariate Analysis
✓ Using Pairplot (comparision between all continuous features and target)

✓ Label Encoding

ENCODING

- **Using Label Encoder:**

  Transformation of all string data from object datatype to Integer datatype.

```
enc = LabelEncoder()
for i in housing.columns.drop(['dataset']):
    if housing[i].dtypes=="object":
        housing[i]=enc.fit_transform(housing[i].values.reshape(-1,1))
```

Checking dataset after transformation

```
housing.head()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgT
0	120	3	70.049958	4928	1	0	3	0	4	0	13	2	2	
1	20	3	95.000000	15865	1	0	3	0	4	1	12	2	2	
2	60	3	92.000000	9920	1	0	3	0	1	0	15	2	2	
3	20	3	105.000000	11751	1	0	3	0	4	0	14	2	2	
4	20	3	70.049958	16635	1	0	3	0	2	0	14	2	2	

Data Inputs- Logic- Output Relationships

✓ Checking Correlation

```
housing.corr()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2
MSSubClass	1.000000	0.035900	-3.570559e-01	-0.139781	-0.024969	0.119289	-0.002940	-2.284384e-02	0.075910	-0.025672	-0.005985	-0.024762	-0.042395
MSZoning	0.035900	1.000000	-1.063635e-01	-0.034452	0.087654	0.061887	-0.017854	-1.192034e-03	-0.009895	-0.022055	-0.249679	-0.027874	0.044606
LotFrontage	-0.357056	-0.106363	1.000000e+00	0.306795	-0.037323	-0.144931	-0.075647	-8.360070e-17	-0.181253	0.067608	0.084545	-0.008483	0.003214
LotArea	-0.139781	-0.034452	3.067946e-01	1.000000	-0.197131	-0.165315	-0.149083	1.012318e-02	-0.121161	0.436868	0.044569	0.023846	0.022164
Street	-0.024969	0.087654	-3.732277e-02	-0.197131	1.000000	-0.010224	0.115995	1.681767e-03	0.013960	-0.179360	-0.011561	-0.071657	0.002039
LotShape	0.119289	0.061887	-1.449209e-01	-0.165315	-0.010224	1.000000	0.085434	-3.610068e-02	0.221102	-0.099951	-0.038894	-0.115003	-0.043768
LandContour	-0.002940	-0.017854	-7.564653e-02	-0.149083	0.115995	0.085434	1.000000	8.238030e-03	-0.025527	-0.374267	0.019116	0.024801	-0.016185
Utilities	-0.022844	-0.001192	-8.360070e-17	0.010123	0.001682	-0.036101	0.008238	1.000000e+00	-0.032589	-0.005909	0.046806	-0.000950	-0.000831
LotConfig	0.075910	-0.009895	-1.812535e-01	-0.121161	0.013960	0.221102	-0.025527	-3.258930e-02	1.000000	-0.007258	-0.036597	0.021457	0.033868
LandSlope	-0.025672	-0.022055	6.760810e-02	0.436868	-0.179360	-0.099951	-0.374267	-5.909285e-03	-0.007258	1.000000	-0.080405	-0.016762	-0.026322
Neighborhood	-0.005985	-0.249679	8.454536e-02	0.044569	-0.011561	-0.038894	0.019116	4.680907e-02	-0.036597	-0.080405	1.000000	-0.025401	0.022432
Condition1	-0.024762	-0.027874	-8.483196e-03	0.023846	-0.071657	-0.115003	0.024801	-9.500550e-04	0.021457	-0.016762	-0.025401	1.000000	-0.074268
Condition2	-0.042395	0.044606	3.213723e-03	0.022164	0.002039	-0.043768	-0.016185	-8.309651e-04	0.033868	-0.026322	0.022432	-0.074268	1.000000

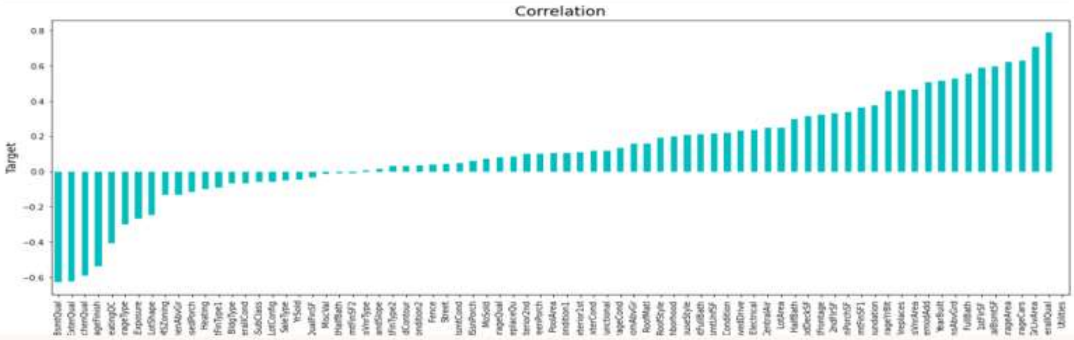
```
pd.set_option('display.max_rows',None)
housing.corr()["SalePrice"].sort_values()
```

```
BsmtQual        -0.626850
ExterQual       -0.624820
KitchenQual     -0.592468
GarageFinish    -0.537121
HeatingQC       -0.406604
GarageType      -0.299470
BsmtExposure    -0.268559
LotShape        -0.248171
MSZoning        -0.133221
KitchenAbvGr    -0.132108
EnclosedPorch   -0.115004
Heating         -0.100021
BsmtFinType1    -0.092109
BldgType        -0.066028
OverallCond     -0.065642
MSSubClass      -0.060775
LotConfig       -0.060452
SaleType        -0.050851
YrSold          -0.045508
LowQualFinSF    -0.032381
MiscVal         -0.013071
BsmtHalfBath    -0.011109
```

- Correlation is checked for relation between the dependent and independent variables.
- Also Checked through heatmap and BarPlot (Visualization)

Checking correlation with barplot

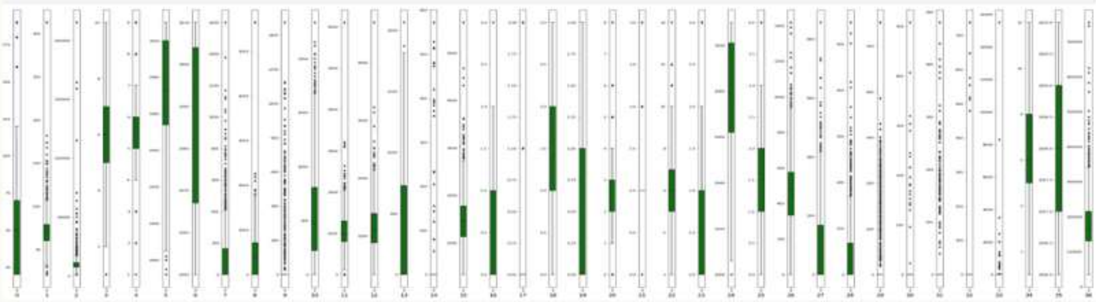
```
plt.figure(figsize=(20,7))
housing.corr()['SalePrice'].sort_values(ascending=True).drop(['SalePrice']).plot(kind='bar',color='c')
plt.xlabel('Feature',fontsize=18)
plt.ylabel('Target',fontsize=14)
plt.title('Correlation',fontsize=18)
plt.show()
```



CHECKING OUTLIERS

- ***Outliers are removed only from continuous features and not from target and categorical features.***

```
collist=['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea',
         'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea',
         'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr','KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces',
         'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch',
         'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'SalePrice']
ncol=37
nrows=8
plt.figure(figsize=(ncol,3*ncol))
for column in range(0,len(collist)):
    plt.subplot(nrows,ncol,column+1)
    sns.boxplot(data=housing[collist[column]],color='green',orient='v')
    plt.xlabel(column,fontsize = 15)
    plt.tight_layout()
```



REMOVING OUTLIERS

- Checking two methods and compare between them which is give less percentage loss and then using that method for further process.

1. Zscore method using Scipy
2. IQR (Inter Quantile Range) method

1.1 Zscore method using Scipy

```
#  Outliers will be removed only from column i.e;  'MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond','YearBuilt', 'MasVnrArea', 'B:
# We will not remove outliers from Target column i.e; 'SalePrice'.

variable = housing[['BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF','GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'BedroomAbvGr',
'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal']]

z=np.abs(zscore(variable))

# Creating new dataframe
housing_price= housing[(z<3).all(axis=1)]
housing_price.head()
```

2. IQR (Inter Quantile Range) method

```
#1st quantile
Q1=variable.quantile(0.25)

# 3rd quantile
Q3=variable.quantile(0.75)

#IQR
IQR=Q3 - Q1
housing_price_pred=housing[~((housing < (Q1 - 1.5 * IQR)) |(housing > (Q3 + 1.5 * IQR))).any(axis=1)]
```

Checking Skewness

```
pd.set_option('display.max_rows',None)
housing_price.skew()
```

```
MSSubClass       1.440879
MSZoning        -1.684441
LotFrontage      1.866804
LotArea          7.791523
Street         -19.131048
LotShape        -0.643270
LandContour     -3.373814
LotConfig       -1.225288
LandSlope        5.053912
Neighborhood     0.106654
Condition1       3.329420
Condition2      21.238748
BldgType         2.283887
HouseStyle       0.318479
OverallQual      0.030738
OverallCond      0.707544
YearBuilt       -0.654379
YearRemodAdd    -0.597253
RoofStyle        1.642685
RoofMatl        13.020468
Exterior1st     -0.755625
Exterior2nd     -0.724403
MasVnrType      -0.034395
MasVnrArea       2.533248
ExterQual       -1.579094
ExterCond       -2.692044
Foundation      -0.150653
BsmtQual        -1.314776
```

REMOVING SKEWNESS

Using yeo-johnson method

```
from sklearn.preprocessing import PowerTransformer
```

```
collist=['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea',
        'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea',
        'BsmtFullBath', 'HalfBath', 'TotRmsAbvGrd', 'Fireplaces', 'GarageYrBlt', 'GarageCars', 'WoodDeckSF', 'OpenPorchSF',
        'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'MiscVal']
housing_price[collist]=power_transform(housing_price[collist],method='yeo-johnson')
housing_price[collist]
```

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFl
0	1.408088	0.123016	-1.110893	-0.064347	-0.469229	-0.134534	-0.706783	-0.829290	-0.075035	-0.327600	0.937383	0.168921	-0.350817	-0.855
2	0.595115	1.101521	0.248881	0.675242	-0.469229	0.710775	0.408361	-0.829290	0.962234	-0.327600	-0.578777	0.271990	0.182795	1.202
3	-1.116500	1.632317	0.612800	-0.064347	0.495693	-0.097238	-0.665052	1.393440	0.834609	-0.327600	1.218880	2.119053	1.832857	-0.855
5	0.595115	-0.472329	1.013240	0.675242	-0.469229	1.221942	1.077895	-0.829290	-1.329447	-0.327600	0.806058	-0.365464	-0.631330	1.221
6	-1.116500	0.123016	0.535305	-0.807736	0.495693	-0.758593	0.342144	1.217855	1.242348	-0.327600	-1.240434	0.985788	0.884419	-0.855
7	-1.116500	0.931617	0.858248	-0.807736	-1.559547	-0.758593	0.616383	0.996529	0.071425	3.052800	-0.488886	0.316762	1.756415	-0.855
8	-1.116500	0.120645	0.084796	-0.807736	1.364833	-0.514456	-1.107150	-0.829290	0.828421	3.051947	-0.119118	0.526248	0.428814	-0.855
9	0.349385	0.581220	-0.075519	-0.807736	-0.469229	-1.030012	-1.511127	-0.829290	0.559156	-0.327600	-0.194726	-0.493986	-0.809665	1.044
10	0.349385	-0.898151	-0.046937	-0.064347	0.495693	-1.268072	-1.511127	-0.829290	-1.329447	-0.327600	0.632693	-0.637347	-1.018338	1.130

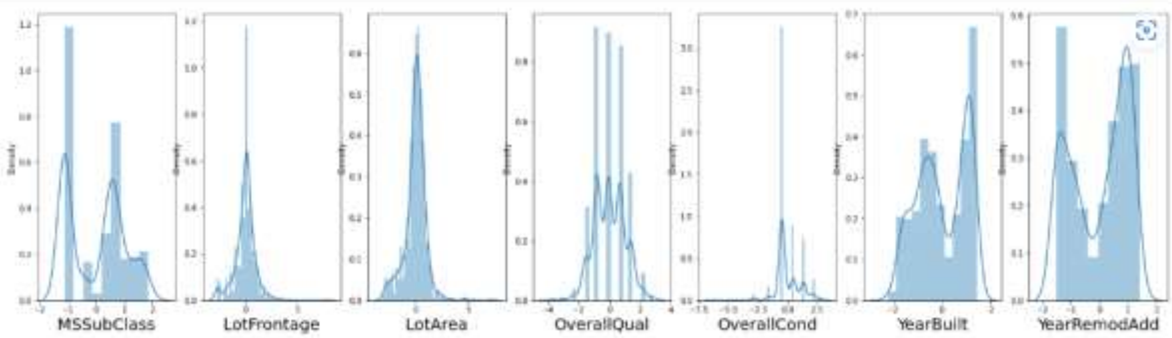
CHECKING SKEWNESS AFTER REMOVAL

```
housing_price.skew()
```

```
MSSubClass        0.108407
MSZoning         -1.684441
LotFrontage       0.214713
LotArea           0.161216
Street          -19.131048
LotShape         -0.643270
LandContour      -3.373814
LotConfig        -1.225288
LandSlope         5.053912
Neighborhood      0.106654
Condition1        3.329420
Condition2       21.238748
BldgType          2.283887
HouseStyle        0.318479
OverallQual       0.008719
OverallCond       0.043442
YearBuilt        -0.176807
YearRemodAdd     -0.295674
RoofStyle         1.642685
RoofMatl         13.020468
Exterior1st      -0.755625
Exterior2nd      -0.724403
MasVnrType       -0.034395
MasVnrArea        0.401460
ExterQual        -1.579094
ExterCond        -2.692044
Foundation       -0.150653
BsmtQual         -1.314776
BsmtCond         -3.651064
BsmtExposure     -1.220853
```

checking skewness after removal through data visualization using distplot

```
collist=[ 'MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea',
        'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea',
        'BsmtFullBath', 'HalfBath', 'TotRmsAbvGrd', 'Fireplaces', 'GarageYrBlt', 'GarageCars', 'WoodDeckSF', 'OpenPorchSF',
        'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'MiscVal' ]
plt.figure(figsize=(25,30), facecolor='white')
plotnumber = 1

for column in housing_price[collist]:
    if plotnumber<=28:
        ax = plt.subplot(4,7,plotnumber)
        sns.distplot(housing_price[column])
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.show()
```



Data preprocessing

```
#Lets seprate the train and test from df_flight_final
housing_train_pred=housing_price.loc[housing_price["dataset"]=="train"]
housing_test1_pred=housing_price.loc[housing_price["dataset"]=="test"]
```

```
housing_train_pred.head()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	H
0	1.408088	3	0.123016	-1.110893	1	0	3	4	0	13	2	2	4	
2	0.595115	3	1.101521	0.248881	1	0	3	1	0	15	2	2	0	
3	-1.116500	3	1.632317	0.612800	1	0	3	4	0	14	2	2	0	
5	0.595115	3	-0.472329	1.013240	1	0	3	4	0	8	2	2	0	
6	-1.116500	3	0.123016	0.535305	1	0	3	4	0	19	2	2	0	

```
#re-indexing the test dataset
housing_test1_pred.reset_index(drop=True,inplace=True)
```

```
#Droping "SalePrice" and "dataset" columns from the test dataset and also droping "dataset" columns from the train dataset
housing_test1_pred.drop(columns=["SalePrice","dataset"],inplace=True)
housing_train_pred.drop(columns=["dataset"],inplace=True)
```

```
housing_train_pred.head()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	H
0	1.408088	3	0.123016	-1.110893	1	0	3	4	0	13	2	2	4	
2	0.595115	3	1.101521	0.248881	1	0	3	1	0	15	2	2	0	
3	-1.116500	3	1.632317	0.612800	1	0	3	4	0	14	2	2	0	
5	0.595115	3	-0.472329	1.013240	1	0	3	4	0	8	2	2	0	
6	-1.116500	3	0.123016	0.535305	1	0	3	4	0	19	2	2	0	

```
housing_test1_pred.head()
```

Spliting data into Target and Features:

```
x=housing_train_pred.drop("SalePrice",axis=1)
y=housing_train_pred["SalePrice"]
```

Scaling data using Standard Scaler

```
scaler = StandardScaler()
x = pd.DataFrame(scaler.fit_transform(x), columns = x.columns)
```

```
x.head()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	HouseStyle	Overa
0	1.417463	0.006988	0.087606	-1.129390	0.047836	-1.387786	0.295663	0.582831	-0.215146	0.174154	-0.04133	-0.012771	2.889951	-0.565752	-0.(
1	0.602497	0.006988	1.061094	0.235031	0.047836	-1.387786	0.295663	-1.280526	-0.215146	0.499441	-0.04133	-0.012771	-0.378735	1.032954	0.(
2	-1.113317	0.006988	1.589169	0.600194	0.047836	-1.387786	0.295663	0.582831	-0.215146	0.336797	-0.04133	-0.012771	-0.378735	-0.565752	-0.(
3	0.602497	0.006988	-0.504686	1.002003	0.047836	-1.387786	0.295663	0.582831	-0.215146	-0.639061	-0.04133	-0.012771	-0.378735	1.032954	0.(
4	-1.113317	0.006988	0.087606	0.522434	0.047836	-1.387786	0.295663	0.582831	-0.215146	1.150013	-0.04133	-0.012771	-0.378735	-0.565752	-0.(

Scaling data using Standard Scaler

```
scaler = StandardScaler()
x = pd.DataFrame(scaler.fit_transform(x), columns = x.columns)
```

```
x.head()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType
0	1.417463	0.006988	0.087606	-1.129390	0.047836	-1.387786	0.295663	0.582831	-0.215146	0.174154	-0.04133	-0.012771	2.889951
1	0.602497	0.006988	1.061094	0.235031	0.047836	-1.387786	0.295663	-1.280526	-0.215146	0.499441	-0.04133	-0.012771	-0.378735
2	-1.113317	0.006988	1.589169	0.600194	0.047836	-1.387786	0.295663	0.582831	-0.215146	0.336797	-0.04133	-0.012771	-0.378735
3	0.602497	0.006988	-0.504686	1.002003	0.047836	-1.387786	0.295663	0.582831	-0.215146	-0.639061	-0.04133	-0.012771	-0.378735
4	-1.113317	0.006988	0.087606	0.522434	0.047836	-1.387786	0.295663	0.582831	-0.215146	1.150013	-0.04133	-0.012771	-0.378735

Checking for Multicolinearity

VIF (Variance Inflation factor)

```
vif = pd.DataFrame()
vif['VIF values']= [variance_inflation_factor(x.values,i) for i in range(len(x.columns))]
vif['Features'] = x.columns
vif
```

	VIF values	Features
0	7.209135	MSSubClass
1	1.516502	MSZoning
2	2.163897	LotFrontage
3	2.930407	LotArea
4	1.151918	Street
5	1.337667	LotShape
6	1.346047	LandContour
7	1.178040	LotConfig
8	1.495065	LandSlope
9	1.359852	Neighborhood
10	1.218499	Condition1

- The VIF value is more than 10 in the columns YearBuilt, 1stFlrSF, 2ndFlrSF, GrLivArea,. But column 'GrLivArea' is having highest VIF value. So, we will drop column 'GrLivArea'.
- columns: BsmtHalfBath, KitchenAbvGr and PoolArea have no relation with target Column, so we will drop these columns.

1]:
```
#droping not important features
x = x.drop(['GrLivArea'],axis=1)
```

2]:
```
x = x.drop(['BsmtHalfBath', 'KitchenAbvGr', 'PoolArea'],axis=1)
```

Variance Threshold Method

It removes all features which variance doesn't meet some threshold. By default, it removes all zero-variance features.

```
var_threshold = VarianceThreshold(threshold=0)
var_threshold.fit(x)
```

```
VarianceThreshold(threshold=0)
```

```
var_threshold.get_support()
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True])
```

```
# taking  out  all the  constant  columns
cons_columns = [column for column in x.columns
                if column not in x.columns[var_threshold.get_support()]]
print(len(cons_columns))
```

```
0
```

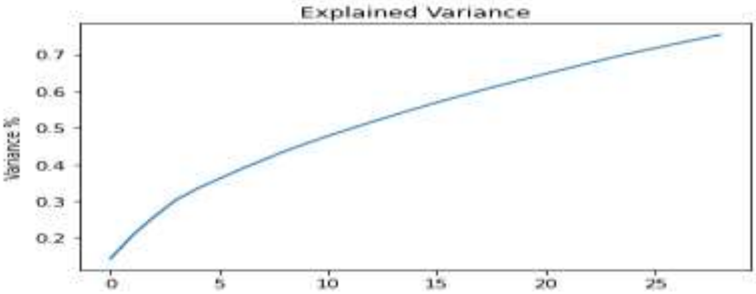
So we can see that, with the help of variance threshold method, we got to know all the features here are important.

Principle Component Analysis

```
from sklearn.decomposition import PCA
```

```
#Lets use PCA for dimensionality reduction
pca = PCA(n_components=29)
x_pca=pca.fit_transform(x)
print("vraiance :{}".format(np.sum(pca.explained_variance_ratio_)))
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Variance %')
plt.title('Explained Variance')
plt.show()
```

```
vraiance :0.753674621766593
```



Creating Model

Finding the best random state among all the models

As target column contains continuous data , so we have to understand this by Regression Algorithm

```
maxAcc = 0
maxRS=0
for i in range(1,100):
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = .20, random_state = i)
    modDTR =  DecisionTreeRegressor()
    modDTR.fit(x_train,y_train)
    pred = modDTR.predict(x_test)
    acc  = r2_score(y_test,pred)
    if acc>maxAcc:
        maxAcc=acc
        maxRS=i
print(f"Best Accuracy is: {maxAcc} on random_state: {maxRS}")
```

```
Best Accuracy is: 0.8067070031721606 on random_state: 5
```

Creating train-test-split

```
# creating new train test split using the random state.
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = .20, random_state = 28)
```

- Hardware and Software Requirements and Tools Used

  ✓ **Used PYTHON Jupyter Notebook:**

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 toremember 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:**

1. **Numpy:**

   It 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.

2. **Pandas:**

   It 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 ortime-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.

3. **Scikit Learn:**

   It 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 isa 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, label encoder, standard scaler, linear, random forest, decision tree, Gradient boosting Regressor, k-nearest neighbours, r2 score, mean absolute error, mean squared error, train test split, grid search cv and ensemble technique.

4. **Matplotlib:**
   It 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. We can 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.

Model/s Development and Evaluation

Identification of possible problem-solving approaches (methods)

In this project, we want to predict the sale price of a houses. The sale price we want to predict is a continuous data, so need to understand it with regression problem.

Testing of Identified Approaches (Algorithms)

1. Linear Regression
2. Random Forest Regressor
3. KNN Regressor
4. Support Vector Regressor
5. Gradient Boosting Regressor

6. Decision Tree Regressor

Run and Evaluate selected models

Creating Model

Finding the best random state among all the models

As target column contains continuous data , so we have to understand this by Regression Algorithm

```
maxAcc = 0
maxRS=0
for i in range(1,100):
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = .20, random_state = i)
    modDTR =  DecisionTreeRegressor()
    modDTR.fit(x_train,y_train)
    pred = modDTR.predict(x_test)
    acc  = r2_score(y_test,pred)
    if acc>maxAcc:
        maxAcc=acc
        maxRS=i
print(f"Best Accuracy is: {maxAcc} on random_state: {maxRS}")
```

```
Best Accuracy is: 0.8067070031721606 on random_state: 5
```

Creating train-test-split

```
# creating new train test split using the random state.
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = .20, random_state = 28)
```

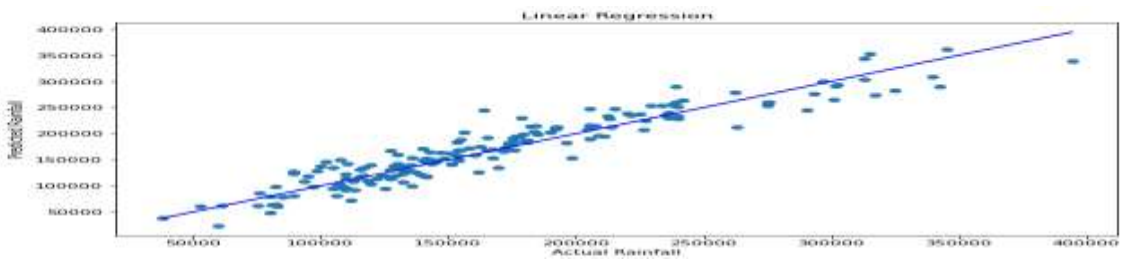
Linear Regression

```
# Checking r2score for Linear Regression
LR = LinearRegression()
LR.fit(x_train,y_train)

# prediction
predLR=LR.predict(x_test)
print('R2_score:',r2_score(y_test,predLR))
print('Mean abs error:',mean_absolute_error(y_test, predLR))
print('Mean squared error:',mean_squared_error(y_test, predLR))
print("Root Mean Squared Error: ", np.sqrt(mean_squared_error(y_test,predLR)))
```

```
R2_score: 0.8902107705211676
Mean abs error: 17471.11287971289
Mean squared error: 499031138.7062404
Root Mean Squared Error:  22339.0048727834
```

Checking the performance of the model by graph

```
plt.figure(figsize=(10,6))
plt.scatter(x=y_test,y=predLR,cmap='set1')
plt.plot(y_test,y_test,color='b')
plt.xlabel("Actual Rainfall")
plt.ylabel("Predicted Rainfall")
plt.title("Linear Regression")
plt.show()
```



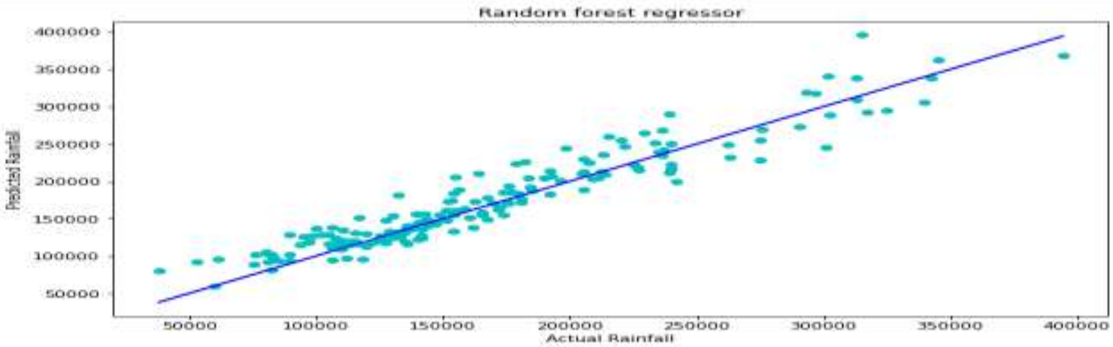
Random forest Regression Model

```
# Checking R2 score for Random Forest Regressor
RFR=RandomForestRegressor(n_estimators=600, random_state=28)
RFR.fit(x_train,y_train)

# prediction
predRFR=RFR.predict(x_test)
print('R2_Score:',r2_score(y_test,predRFR))
print('Mean abs error:',mean_absolute_error(y_test, predRFR))
print('Mean squared error:',mean_squared_error(y_test, predRFR))
print("Root Mean Squared Error: ", np.sqrt(mean_squared_error(y_test,predRFR)))
```

```
R2_Score: 0.90138755786864
Mean abs error: 15961.309952651516
Mean squared error: 448228660.68937784
Root Mean Squared Error:  21171.411400503694
```

Checking the performance of the model by graph

```
#Verifying the performance of the model by graph
plt.figure(figsize=(10,6))
plt.scatter(x=y_test,y=predRFR,color='c')
plt.plot(y_test,y_test,color='b')
plt.xlabel("Actual Rainfall")
plt.ylabel("Predicted Rainfall")
plt.title("Random forest regressor")
plt.show()
```



KNN Regressor

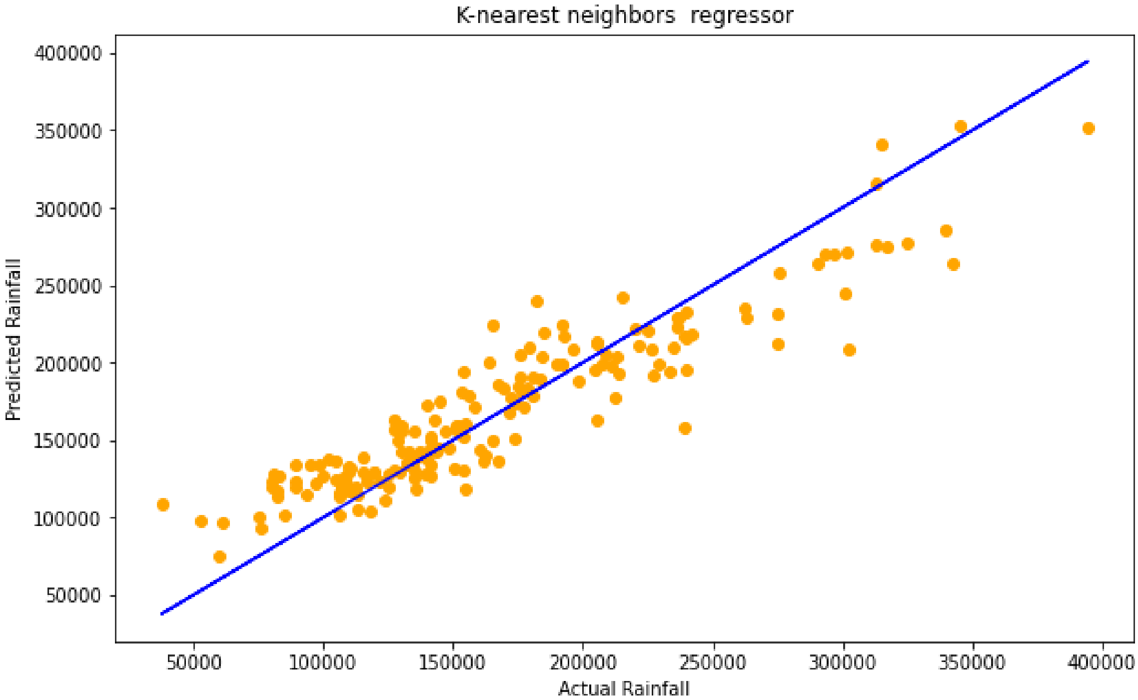
```
# Checking R2 score for KNN regressor
knn=KNeighborsRegressor(n_neighbors=9 )
knn.fit(x_train,y_train)

#prediction
predknn=knn.predict(x_test)
print('R2_Score:',r2_score(y_test,predknn))
print('Mean abs error:',mean_absolute_error(y_test, predknn))
print('Mean squared error:',mean_squared_error(y_test, predknn))
print("Root Mean Squared Error: ", np.sqrt(mean_squared_error(y_test,predknn)))
```

```
R2_Score: 0.8417360162194223
Mean abs error: 20780.01452020202
Mean squared error: 719366156.5630612
Root Mean Squared Error:  26821.002154339072
```

Checking the performance of the model by graph

```
plt.figure(figsize=(10,6))
plt.scatter(x=y_test,y=predknn,color='orange')
plt.plot(y_test,y_test,color='b')
plt.xlabel("Actual Rainfall")
plt.ylabel("Predicted Rainfall")
plt.title("K-nearest neighbors  regressor")
```

```
Text(0.5, 1.0, 'K-nearest neighbors  regressor')
```



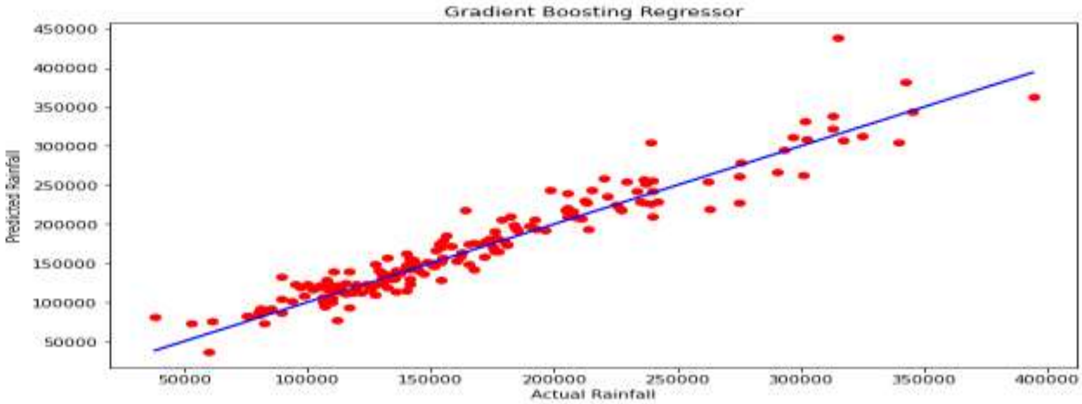
Grdient boosting Regressor

```
# Checking R2 score for GBR
Gb= GradientBoostingRegressor(n_estimators=400,  random_state=29, learning_rate=0.1, max_depth=3)
Gb.fit(x_train,y_train)

#prediction
predGb=Gb.predict(x_test)
print('R2_Score:',r2_score(y_test,predGb))
print('Mean abs error:',mean_absolute_error(y_test, predGb))
print('Mean squared error:',mean_squared_error(y_test, predGb))
print("Root Mean Squared Error: ", np.sqrt(mean_squared_error(y_test,predGb)))
```

```
R2_Score: 0.9122354161264967
Mean abs error: 13928.84888571393
Mean squared error: 398921282.500829
Root Mean Squared Error:  19973.013856221824
```

Checking the performance of the model by graph

```
plt.figure(figsize=(10,6))
plt.scatter(x=y_test,y=predGb,color='r')
plt.plot(y_test,y_test,color='b')
plt.xlabel("Actual Rainfall")
plt.ylabel("Predicted Rainfall")
plt.title("Gradient Boosting Regressor")
plt.show()
```



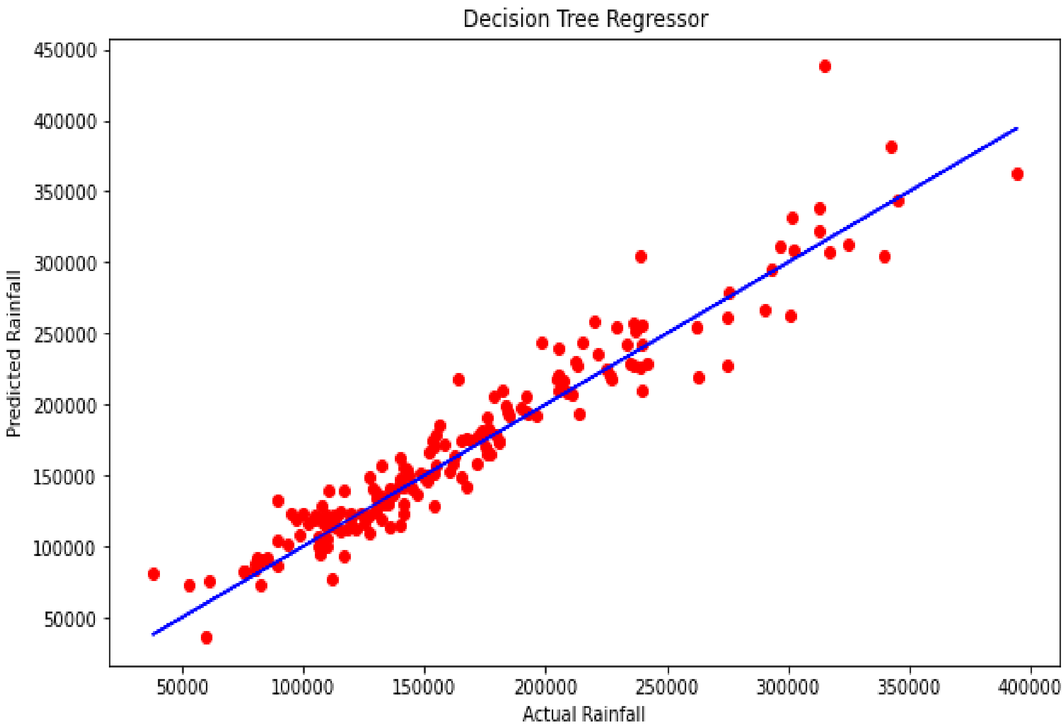
Decision Tree Regressor

```
# Checking R2 score for GBR
DTR= DecisionTreeRegressor()
DTR.fit(x_train,y_train)

#prediction
predDTR=DTR.predict(x_test)
print('R2_Score:',r2_score(y_test,predDTR))
print('Mean abs error:',mean_absolute_error(y_test, predDTR))
print('Mean squared error:',mean_squared_error(y_test, predDTR))
print("Root Mean Squared Error: ", np.sqrt(mean_squared_error(y_test,predDTR)))
```

```
R2_Score: 0.7223103986516524
Mean abs error: 24115.704545454544
Mean squared error: 1262198110.1931818
Root Mean Squared Error:  35527.427576355454
```

Checking the performance of the model by graph

```
plt.figure(figsize=(10,6))
plt.scatter(x=y_test,y=predGb,color='r')
plt.plot(y_test,y_test,color='b')
plt.xlabel("Actual Rainfall")
plt.ylabel("Predicted Rainfall")
plt.title("Decision Tree Regressor")
plt.show()
```



Cross Validation Score for all the model

```
#CV Score for Linear Regression
print('CV score for Linear Regression: ',cross_val_score(LR,x,y,cv=5).mean())

#CV Score for Random Forest Regression
print('CV score for Random forest Regression: ',cross_val_score(RFR,x,y,cv=5).mean())

#CV Score for KNN Regression
print('CV score for KNN Regression: ',cross_val_score(knn,x,y,cv=5).mean())

#CV Score for Support Vector Regression
print('CV score for Support Vector  Regression: ',cross_val_score(sv,x,y,cv=5).mean())

#CV Score for Gradient Boosting Regression
print('CV score for Gradient Boosting Regression: ',cross_val_score(Gb,x,y,cv=5).mean())

#CV Score for Decision Tree Regression
print('CV score for Decision Tree Regression: ',cross_val_score(DTR,x,y,cv=5).mean())
```

```
CV score for Linear Regression:  0.8591273605000366
CV score for Random forest Regression:  0.8501230863363538
CV score for KNN Regression:  0.790314360680247З
CV score for Support Vector  Regression:  0.08900628853391453
CV score for Gradient Boosting Regression:  0.8833735741416137
CV score for Decision Tree Regression:  0.6032693535076248
```

Hyper Parameter Tuning

The Gradient boosting regressor with GridsearchCV

```
parameter = {'n_estimators':[100,200,300,400],
             'learning_rate':[0.1,0.01,0.001,1],
             'subsample': [0.1,0.2,0.3,0.5,1],
             'max_depth':[1,2,3,4],
             'alpha':[0.1,0.01,0.001,1]}
```

```
CV_GBR = GridSearchCV(GradientBoostingRegressor(),parameter,cv=6,n_jobs = 3,verbose = 2)
```

```
CV_GBR.fit(x_train,y_train)
```

```
Fitting 6 folds for each of 1280 candidates, totalling 7680 fits
GridSearchCV(cv=6, estimator=GradientBoostingRegressor(), n_jobs=3,
             param_grid={'alpha': [0.1, 0.01, 0.001, 1],
                         'learning_rate': [0.1, 0.01, 0.001, 1],
                         'max_depth': [1, 2, 3, 4],
                         'n_estimators': [100, 200, 300, 400],
                         'subsample': [0.1, 0.2, 0.3, 0.5, 1]},
             verbose=2)
```

```
CV_GBR.best_params_
```

```
{'alpha': 0.001,
 'learning_rate': 0.1,
 'max_depth': 2,
 'n_estimators': 300,
 'subsample': 1}
```

Creating Regressor Model with Gradient Boosting Regressor

```
GBR = GradientBoostingRegressor(n_estimators=300,alpha=0.001,learning_rate= 0.1, max_depth= 2, subsample = 1)
GBR.fit(x_train, y_train)
```

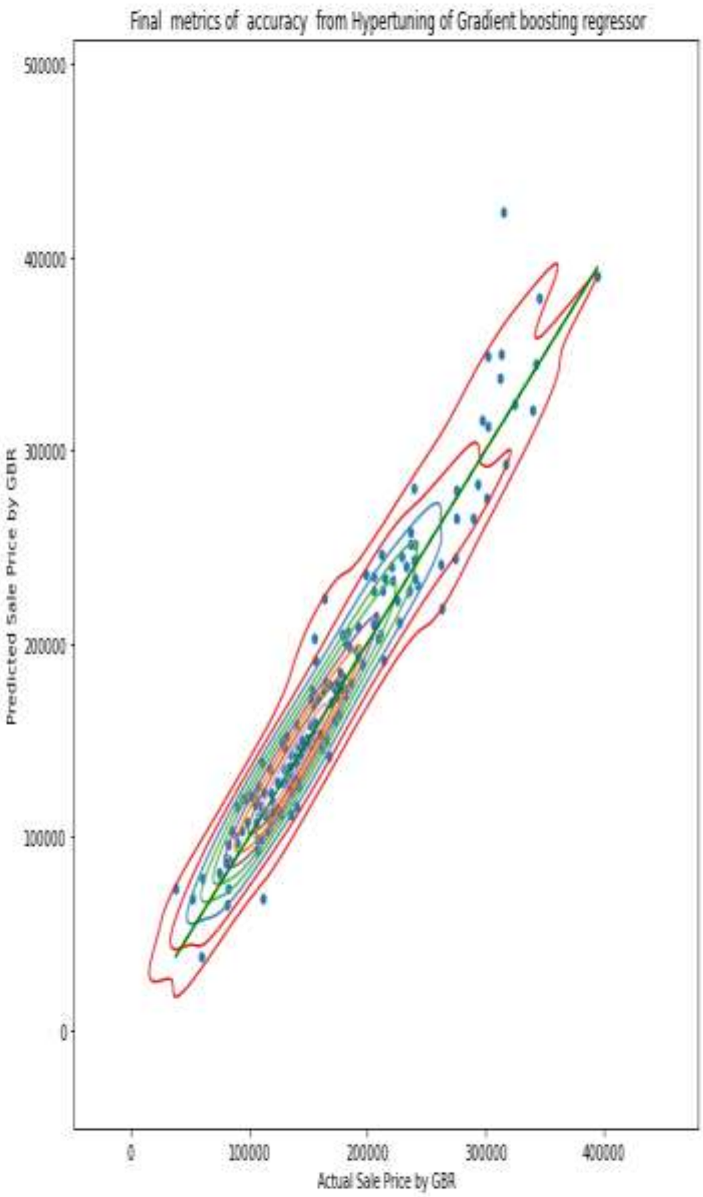
```
GradientBoostingRegressor(alpha=0.001, max_depth=2, n_estimators=300,
                          subsample=1)
```

```
#prediction
GBRpred = GBR.predict(x_test)
#R2 score
acc = r2_score(y_test,GBRpred)
print(acc*100)
```

```
92.32309796065434
```

So after the Hypertuning now we have got a descent accuracy score of 92% on Gradient boosting

```
#Verifying the final  performance of the model by graph
plt.figure(figsize=(10,10))
sns.scatterplot(x=y_test,y=GBRpred,palette='Set2')
sns.kdeplot(x=y_test,y=GBRpred, cmap='Set1')
plt.plot(y_test,y_test,color='g')
#Verifying the performance of the model by graph
plt.xlabel("Actual Sale Price by GBR")
plt.ylabel("Predicted Sale Price by GBR")
plt.title(" Final  metrics of  accuracy  from Hypertuning of Gradient boosting regressor")
plt.show()
```



Saving The Predictive Model ¶

```
#saving the model at local file system
filename='Housing_Price_Prediction.pickle'
pickle.dump(CV_GBR,open(filename,'wb'))
#prediction using the saved model
loaded_model = pickle.load(open(filename, 'rb'))
loaded_model.predict(x_test)
```

```
array([121901.7074758 ,  38259.45210333,  92660.35565497, 100153.17514876,
       177711.34392777, 350055.45072388, 142813.84415788, 158454.39494496,
       264801.25144214,  73647.89634064, 173697.7781192 , 114596.88846379,
       257243.89686981, 107146.51175728, 251129.60717669, 284279.78882806,
       168865.18337245, 171818.09508897, 337373.92691082, 233591.72476918,
       146476.88712049, 151014.98799697, 127981.66421615, 323811.03449172,
       147012.73565717, 113626.42443213, 111969.84786898, 134742.38906856,
```

Checking predicted and original values ¶

```
import numpy as np
a = np.array(y_test)
predict = np.array(loaded_model.predict(x_test))
Housing_Price_Prediction = pd.DataFrame({"Original":a,"Predicted":predict},index= range(len(a)))
Housing_Price_Prediction
```

	Original	Predicted
0	105000.0	121901.707476
1	60000.0	38259.452103
2	107000.0	92660.355655
3	89000.0	100153.175149
4	176000.0	177711.343928
5	313000.0	350055.450724
6	135000.0	142813.844158
7	140000.0	158454.394945
8	275000.0	264801.251442
9	82000.0	73647.896341
10	181000.0	173697.778119

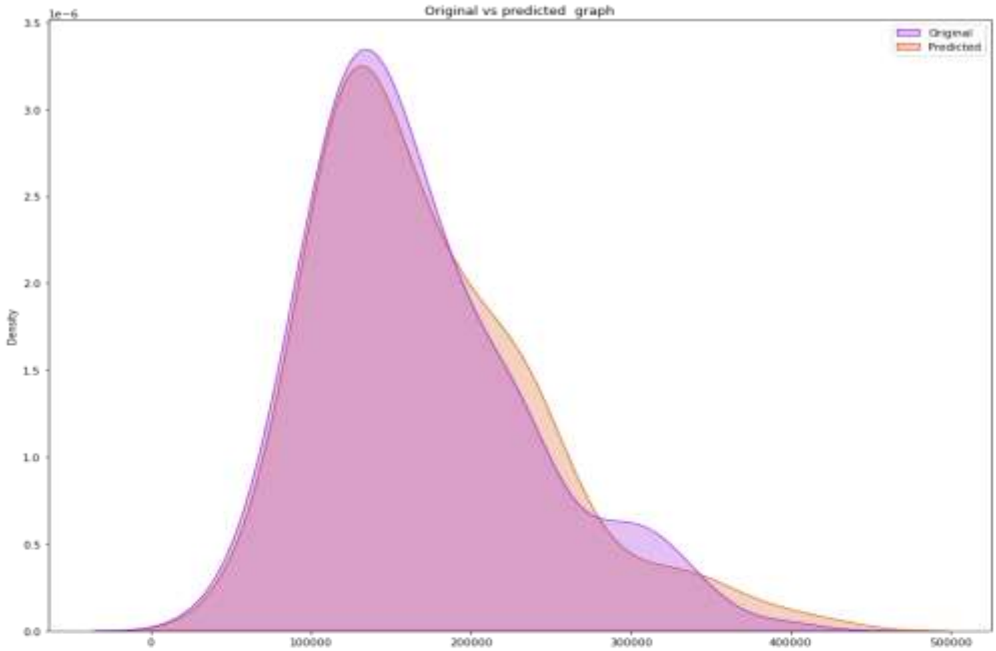
Let's plot and visualize

```
plt.figure(figsize=(15,12))
sns.kdeplot(data=Housing_Price_Prediction, palette='gnuplot',gridsize=900, shade=True)
plt.title('Original vs predicted  graph')
```

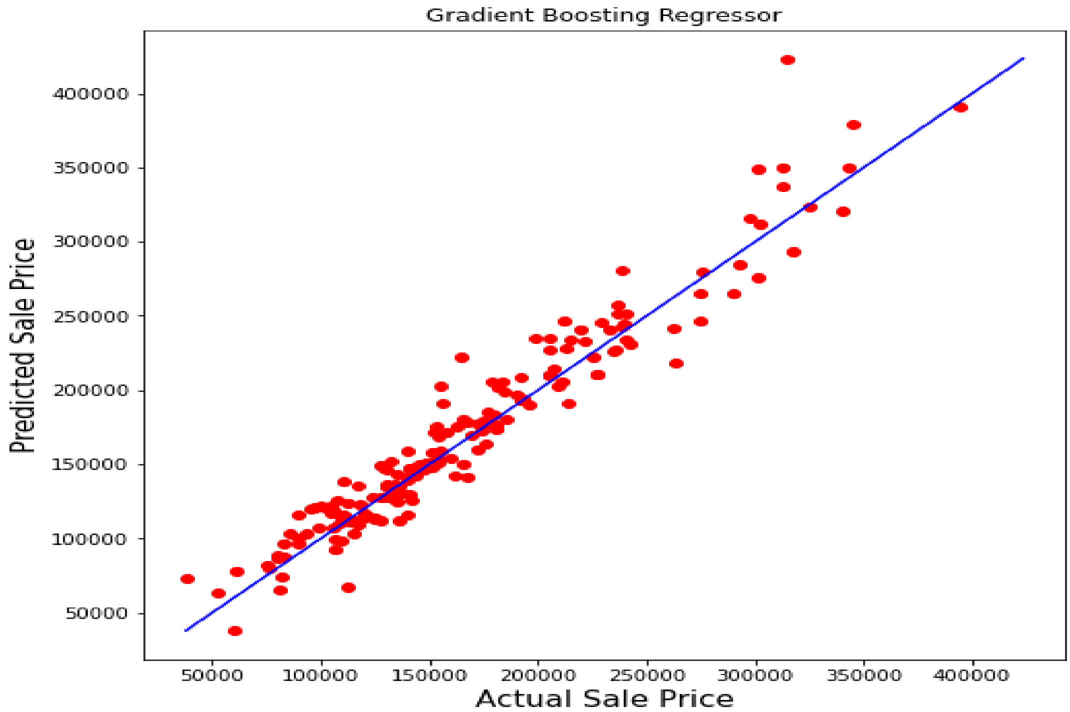
Visualization

Let's plot and visualize

```
plt.figure(figsize=(15,12))
sns.kdeplot(data=Housing_Price_Prediction, palette='gnuplot',gridsize=900, shade=True)
plt.title('Original vs predicted  graph')
```




```
plt.figure(figsize=(8,8))
plt.scatter(y_test,predict,c='r')
plt1 = max(max(predict),max(y_test))
plt2 = min(min(predict),min(y_test))
plt.plot([plt1,plt2],[plt1,plt2],'b-')
plt.xlabel('Actual Sale Price',fontsize=15)
plt.ylabel('Predicted Sale Price',fontsize=15)
plt.title("Gradient Boosting Regressor")
plt.show()
```



CONCLUSION

In this Project we have predicted Sale Price of Houses, We have done prediction of Selling price of houses on basis of Data using EDA, Data Visualization, Data Pre-processing, Checking Correlation, Outliers, Skewness and removed irrelevant features for prediction and at last train our data by splitting our data through train-test split process. Created our model using multiple model and finally selected best model which was giving best accuracy. And at last compared our predicted and Actual Sale Price of Houses. Thus our project is completed.

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

- We can create and add more variables, try different models with different subset of features and/or rows.
- Some of the ideas are listed below:
  • 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