FLIP ROBO

# HOUSING PRICE PREDICTION

Submitted by:

DINESH MUTHA

## ACKNOWLEDGMENT

I wish to thank my Mentor Mr. Shankargouda Tegginmani, my wife, my Friends who in one or other way helped me understand, workout, continue my project and the help they have provided me throughout the project duration.

I also wish to thank FLIP ROBO TECHNOLOGIES for providing me a chance to prove my know-how.

# INTRODUCTION

## Business Problem Framing

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

## Conceptual Background of the Domain Problem

- Houses are one of the necessary need of each and every person around the globe and therefore housing and real estate market is one of the markets which is one of the major contributors in the world's economy. It is a very large market and there are various companies working in the domain.
- Data science comes as a very important tool to solve problems in the domain to help the companies increase their overall revenue, profits, improving their marketing strategies and focusing on changing trends in house sales and purchases. Predictive modelling, Market mix modelling, recommendation systems are some of the machine learning techniques used for achieving the business goals for housing companies. Our problem is related to one such housing company.
- 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.

## Technical Requirements

- Data contains 1460 entries each having 81 variables.
- Data contains Null values. We need to treat them using the
- domain knowledge and your own understanding.
- Extensive EDA has to be performed to gain relationships of important variable and price.
- Data contains numerical as well as categorical variable. We need to handle them accordingly.
- We have to build Machine Learning models, apply regularization and determine the optimal values of Hyper Parameters.
- We need to find important features which affect the price positively or negatively.
- Two datasets were provided to us (test.csv, train.csv).

## Motivation for the Problem Undertaken

- The objective behind this project is to harness & improvise the required data science skills.
- Improve Analytical thinking.
- Get into the real world problem solving processes.

## Analytical Problem Framing

- Mathematical/ Analytical Modeling of the Problem

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

- Data Sources and their formats

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

- Data Preprocessing Done

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

- Hardware and Software Requirements and Tools Used

  Listing down the hardware and software requirements along with the tools, libraries and packages used. Describe all the software tools used along with a detailed description of tasks done with those tools.

## Model Development and Evaluation

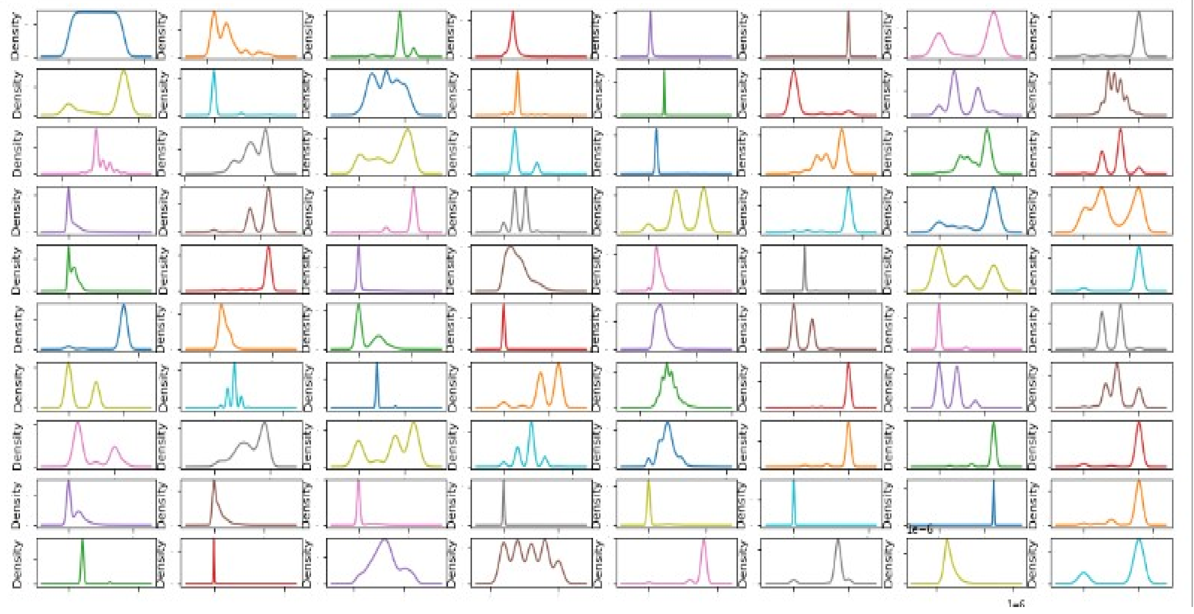
This is a Regression problem, where our end goal is to predict the Prices of House based on given data provided in the dataset. We have divided the provided dataset into Training and Testing parts. A Regression Model will be built and trained using the Training data and the Test data will be used to predict the outcomes. This will be compared with available test results to find how well our model has performed.

We have used Mean Absolute Error, Root Mean Square Error, and ‘R2_Score’ to determine the best model among,

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

## Data Distribution of different columns

```
df.plot(kind = 'density', subplots = True, layout = (12, 8), sharex = False, legend = False, fontsize = 1, figsize = [20, 10

plt.show()
```



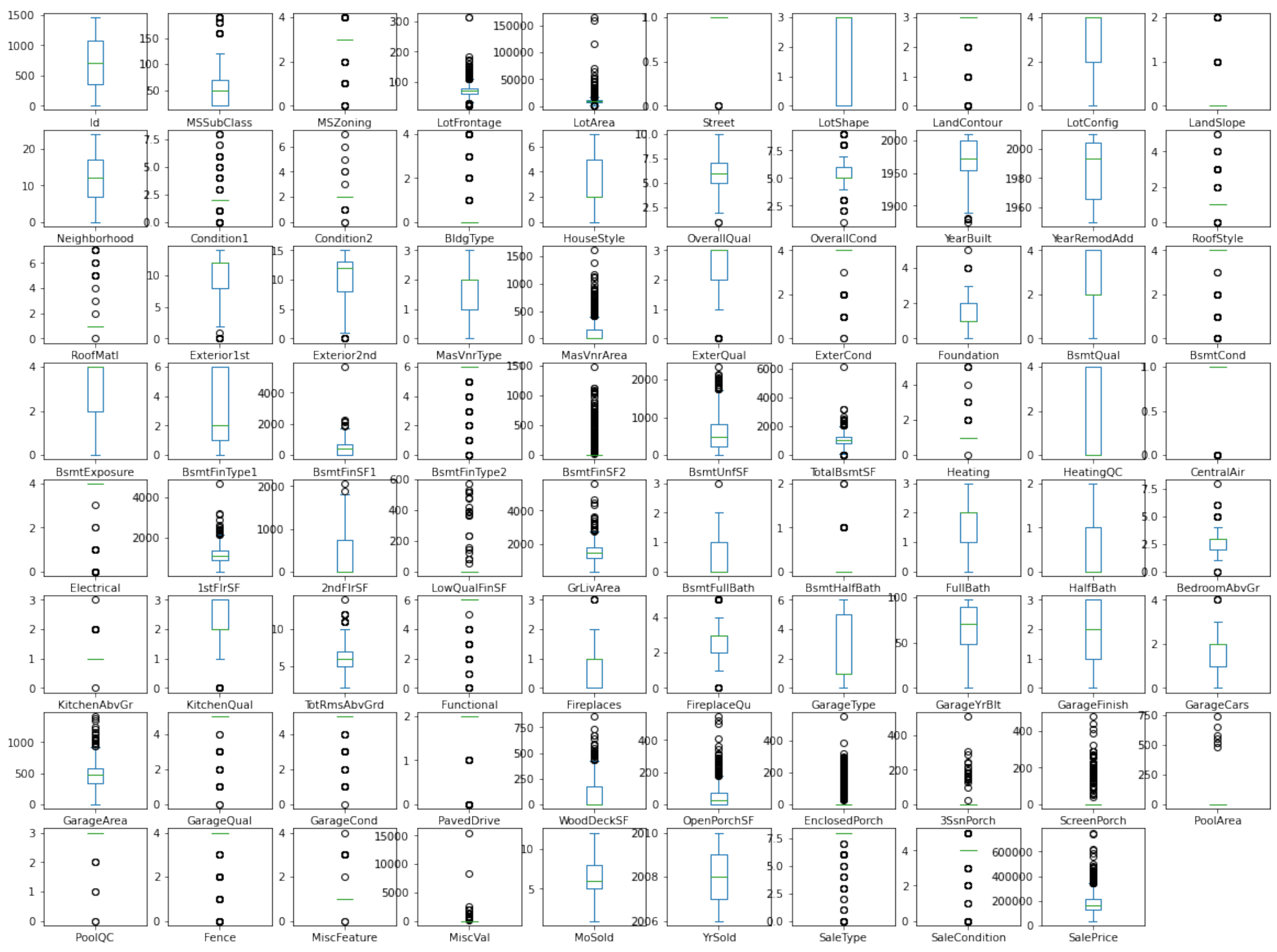
Statistical distribution of Dataset

```
# Statistical description of dataset

df.describe()
```

```
               Id   MSSubClass   MSZoning  LotFrontage        LotArea       Street     LotShape  LandContour    LotConfig    LandSlope  Neighborhood
count 1460.000000  1460.000000 1460.000000 1460.000000    1460.000000  1460.000000  1460.000000  1460.000000  1460.000000  1460.000000   1460.000000
mean   730.500000    56.897260    3.028767   70.049958   10516.828082     0.995890     1.942466     2.777397     3.019178     0.062329     12.251370
std    421.610009    42.300571    0.632017   22.024023    9981.264932     0.063996     1.409156     0.707666     1.622634     0.276232      6.013735
min      1.000000    20.000000    0.000000   21.000000    1300.000000     0.000000     0.000000     0.000000     0.000000     0.000000      0.000000
25%    365.750000    20.000000    3.000000   60.000000    7553.500000     1.000000     0.000000     3.000000     2.000000     0.000000      7.000000
50%    730.500000    50.000000    3.000000   70.049958    9478.500000     1.000000     3.000000     3.000000     4.000000     0.000000     12.000000
75%   1095.250000    70.000000    3.000000   79.000000   11601.500000     1.000000     3.000000     3.000000     4.000000     0.000000     17.000000
max   1460.000000   190.000000    4.000000  313.000000  215245.000000     1.000000     3.000000     3.000000     4.000000     2.000000     24.000000
```

## Check skewness and outliers in training dataset

```
# Check for the outliers

train_data.plot(kind = 'box', subplots = True, layout = (20, 10), sharex = False, legend = True, figsize = (20, 40))

plt.show()
```



## Assumptions:

I. <u>Linearity</u> : The relationship between X & mean of Y is linear.

II. <u>Same variance</u> : The variance of residual is the same for any value of X.

III. <u>Independence</u> : Observations are independent of each other.

IV. <u>Normality :</u> For any fixed value of X, Y is normally distributed.

## Data Sources and their formats

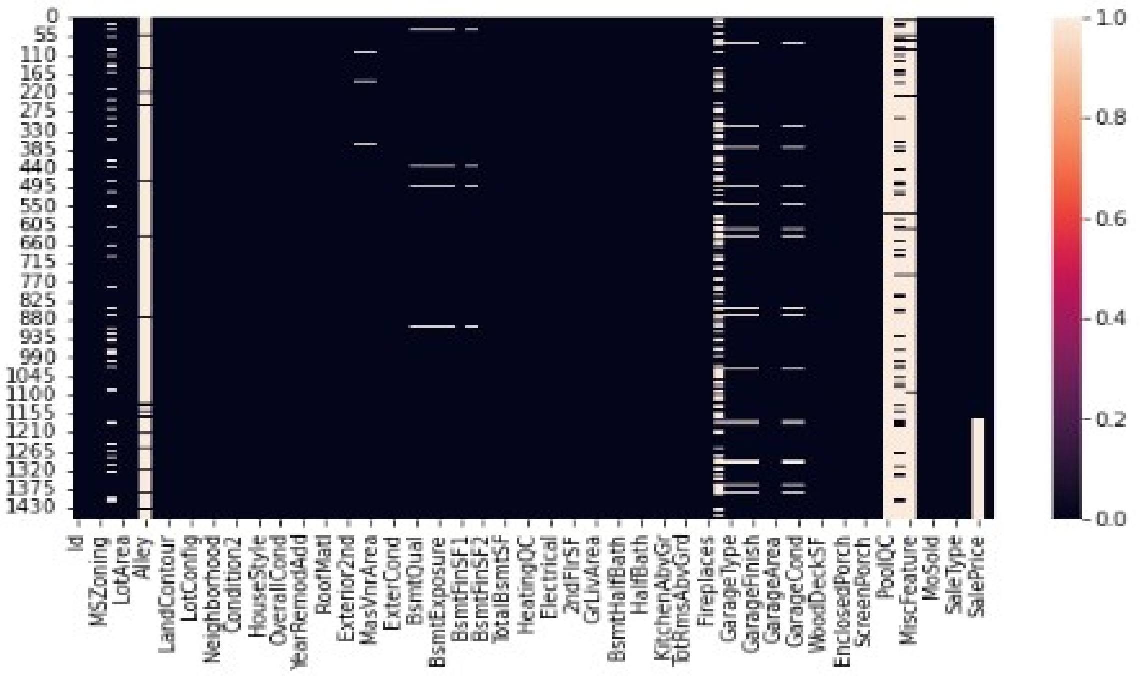
A US-based housing company named Surprise Housing has decided to enter the Australian market. The company uses data analytic to purchase houses at a price below their actual values and flip them at a higher price. For the same purpose, the company has collected a data set from the sale of houses in Australia.
The dataset contains 1460 rows and 81 columns (including the train dataset and test dataset)
The column 'SalePrice' is the target column. We need to predict the sale price of the houses.

## Data Preprocessing Done

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

```
# Plot a heat map for visualizing null values

plt.figure(figsize = [10, 5])

sns.heatmap(df.isnull())
```

```
<AxesSubplot:>
```



From above we can observe that PoolQC has NULL values, hence we consider it as NO PULL AVAILABLE. We fill the values with ‘NA’.

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

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

```
# We can see most of the values are empty, fill the empty values with NA.

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

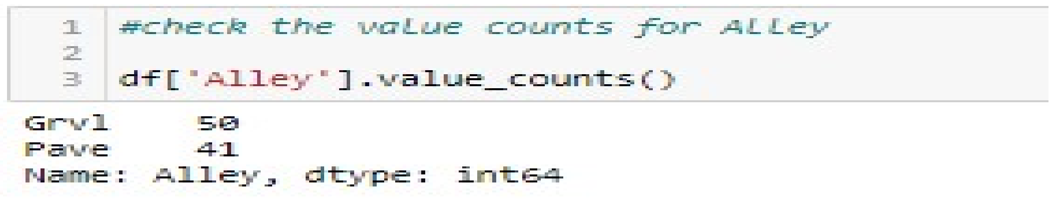
Similarly MiscFeature also has NULL values hence we replace those with ‘NA’.

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

```
Shed    49
Gar2     2
Othr     2
TenC     1
Name: MiscFeature, dtype: int64
```

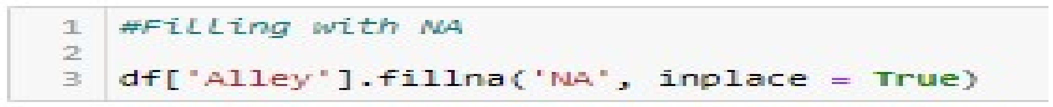
```
# Similiarly in this column most values in rows are empty, fill it with NA

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

Alley, Fence, FireplaceQu also have NULL values present, We replace those with ‘NA‘ too

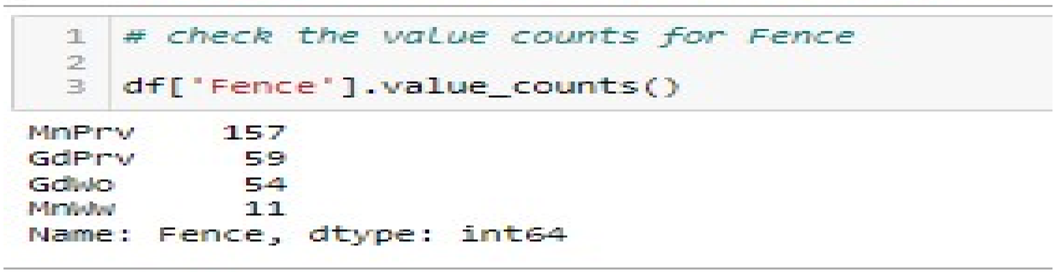


```
#check the value counts for Alley

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

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

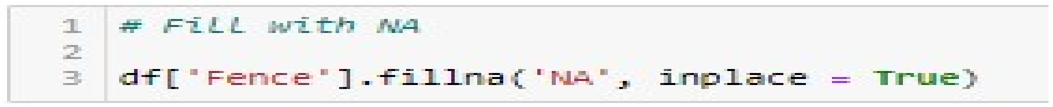


```
#Filling with NA

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

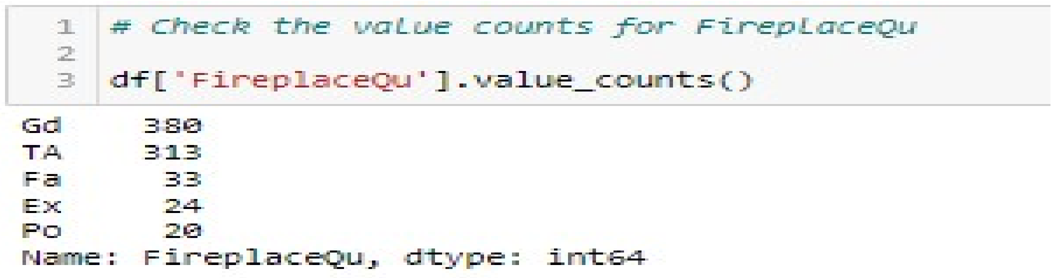


```
# check the value counts for Fence

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

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

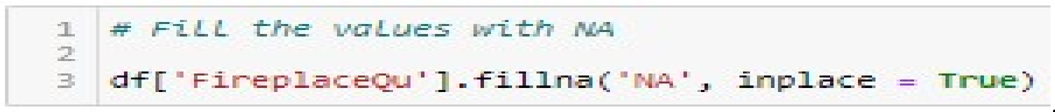


```
# Fill with NA

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



```
# Check the value counts for FireplaceQu

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

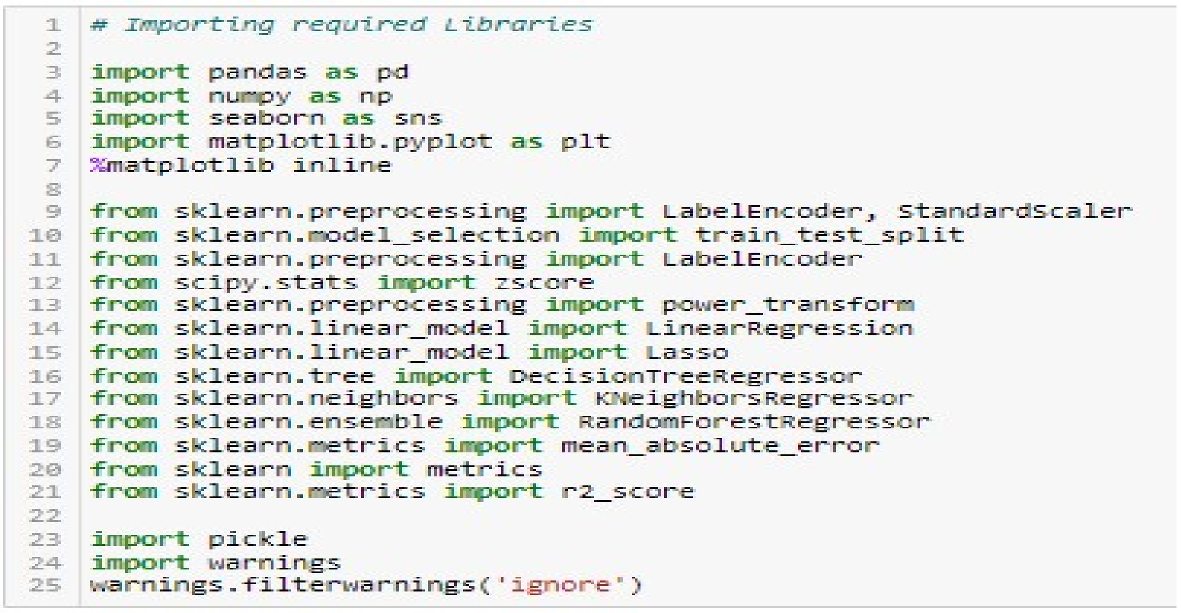
```
Gd    380
TA    313
Fa     33
Ex     24
Po     20
Name: FireplaceQu, dtype: int64
```



```
# Fill the values with NA

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

## Data Inputs- Logic- Output Relationships

EDA was performed by creating valuable insights using various visualization libraries



```
# Importing required Libraries

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

from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from scipy.stats import zscore
from sklearn.preprocessing import power_transform
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn import metrics
from sklearn.metrics import r2_score

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

## Hardware and Software Requirements and Tools Used

### Hardware Configuration:

**Operating System** Windows 11 Home
**Processor** Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
**Installed RAM** 12.0 GB
**System type** 64-bit operating system, x64-based processor

### Software & Tools:

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

b) Python (used for scientific computation)

c) Pandas (used for scientific computation)

d) Numpy (used for scientific computation)

e) Matplotlib (used for visualization)

f) Seaborn (used for visualization)

g) Scikit-learn (used as algorithmic libraries)

## Model Development and Evaluation

### Identification of possible problem - solving approaches

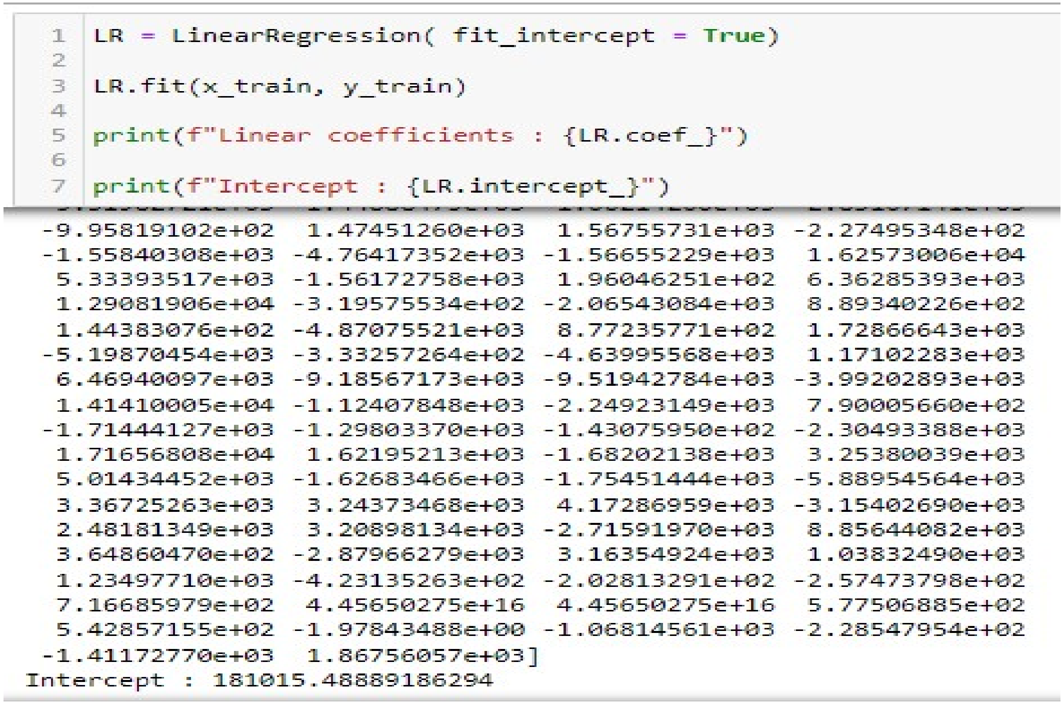
1. Performed Exploratory Data Analysis.
2. Data Cleaning and drop out the columns which were not contributing to the dataset.
3. Handling the missing values.
4. Checking for the outliers and tried to remove it from the dataset.
5. Checked for the skewness in the dataset and removed it for better model building.
6. Split the dataset into independent and dependent variables.
7. Model Building.

## Testing of Identified Approaches (Algorithms)

Below are the algorithms used for the training and testing:

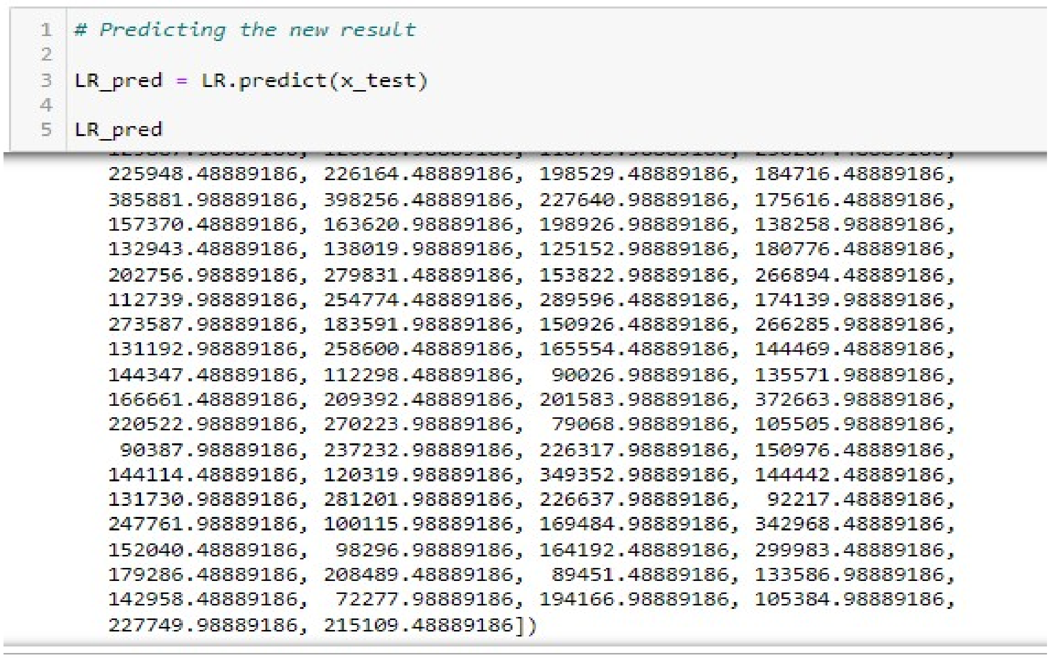
- **Linear Regression**
- **Lasso**
- **Decision Tree Regression**
- **K Neighbors Regression**
- **Random Forest Regression**

## Linear Regeression



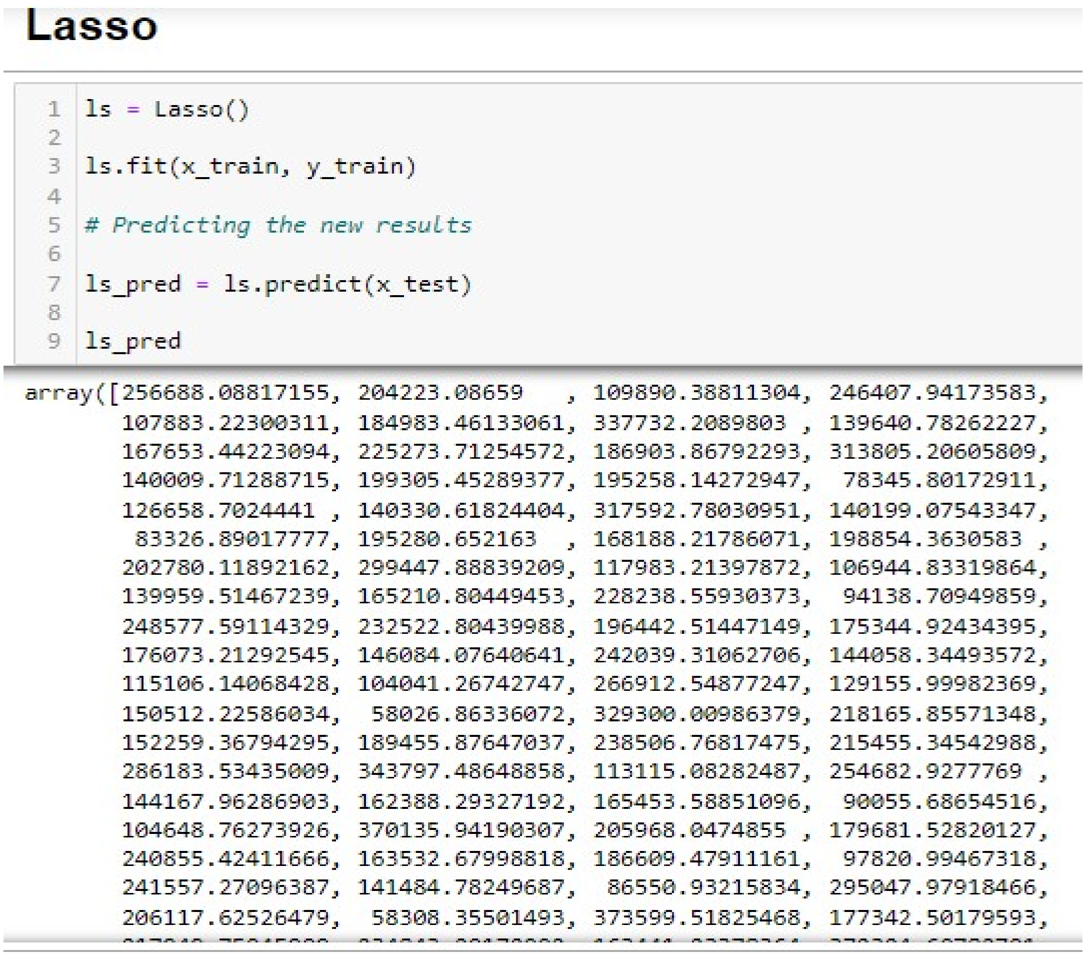
```
LR = LinearRegression( fit_intercept = True)

LR.fit(x_train, y_train)

print(f"Linear coefficients : {LR.coef_}")

print(f"Intercept : {LR.intercept_}")
```

```
 -9.95819102e+02  1.47451260e+03  1.56755731e+03 -2.27495348e+02
 -1.55840308e+03 -4.76417352e+03 -1.56655229e+03  1.62573006e+04
  5.33393517e+03 -1.56172758e+03  1.96046251e+02  6.36285393e+03
  1.29081906e+04 -3.19575534e+02 -2.06543084e+03  8.89340226e+02
  1.44383076e+02 -4.87075521e+03  8.77235771e+02  1.72866643e+03
 -5.19870454e+03 -3.33257264e+02 -4.63995568e+03  1.17102283e+03
  6.46940097e+03 -9.18567173e+03 -9.51942784e+03 -3.99202893e+03
  1.41410005e+04 -1.12407848e+03 -2.24923149e+03  7.90005660e+02
 -1.71444127e+03 -1.29803370e+03 -1.43075950e+02 -2.30493388e+03
  1.71656808e+04  1.62195213e+03 -1.68202138e+03  3.25380039e+03
  5.01434452e+03 -1.62683466e+03 -1.75451444e+03 -5.88954564e+03
  3.36725263e+03  3.24373468e+03  4.17286959e+03 -3.15402690e+03
  2.48181349e+03  3.20898134e+03 -2.71591970e+03  8.85644082e+03
  3.64860470e+02 -2.87966279e+03  3.16354924e+03  1.03832490e+03
  1.23497710e+03 -4.23135263e+02 -2.02813291e+02 -2.57473798e+02
  7.16685979e+02  4.45650275e+16  4.45650275e+16  5.77506885e+02
  5.42857155e+02 -1.97843488e+00 -1.06814561e+03 -2.28547954e+02
 -1.41172770e+03  1.86756057e+03]
Intercept : 181015.48889186294
```



```
# Predicting the new result

LR_pred = LR.predict(x_test)

LR_pred
```

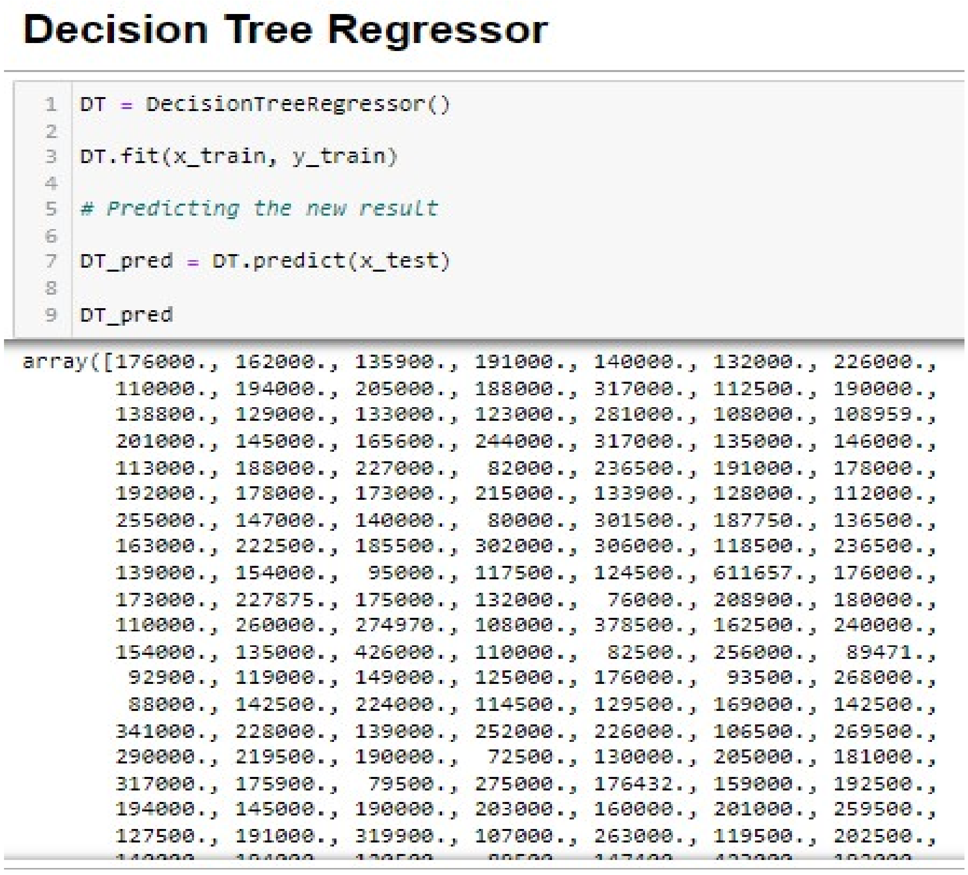
```
       225948.48889186, 226164.48889186, 198529.48889186, 184716.48889186,
       385881.98889186, 398256.48889186, 227640.98889186, 175616.48889186,
       157370.48889186, 163620.98889186, 198926.98889186, 138258.98889186,
       132943.48889186, 138019.98889186, 125152.98889186, 180776.48889186,
       202756.98889186, 279831.48889186, 153822.98889186, 266894.48889186,
       112739.98889186, 254774.48889186, 289596.48889186, 174139.98889186,
       273587.98889186, 183591.98889186, 150926.48889186, 266285.98889186,
       131192.98889186, 258600.48889186, 165554.48889186, 144469.48889186,
       144347.48889186, 112298.48889186,  90026.98889186, 135571.98889186,
       166661.48889186, 209392.48889186, 201583.98889186, 372663.98889186,
       220522.98889186, 270223.98889186,  79068.98889186, 105505.98889186,
        90387.98889186, 237232.98889186, 226317.98889186, 150976.48889186,
       144114.48889186, 120319.98889186, 349352.98889186, 144442.48889186,
       131730.98889186, 281201.98889186, 226637.98889186,  92217.48889186,
       247761.98889186, 100115.98889186, 169484.98889186, 342968.48889186,
       152040.48889186,  98296.98889186, 164192.48889186, 299983.48889186,
       179286.48889186, 208489.48889186,  89451.48889186, 133586.98889186,
       142958.48889186,  72277.98889186, 194166.98889186, 105384.98889186,
       227749.98889186, 215109.48889186])
```



## Lasso

```python
ls = Lasso()

ls.fit(x_train, y_train)

# Predicting the new results

ls_pred = ls.predict(x_test)

ls_pred
```

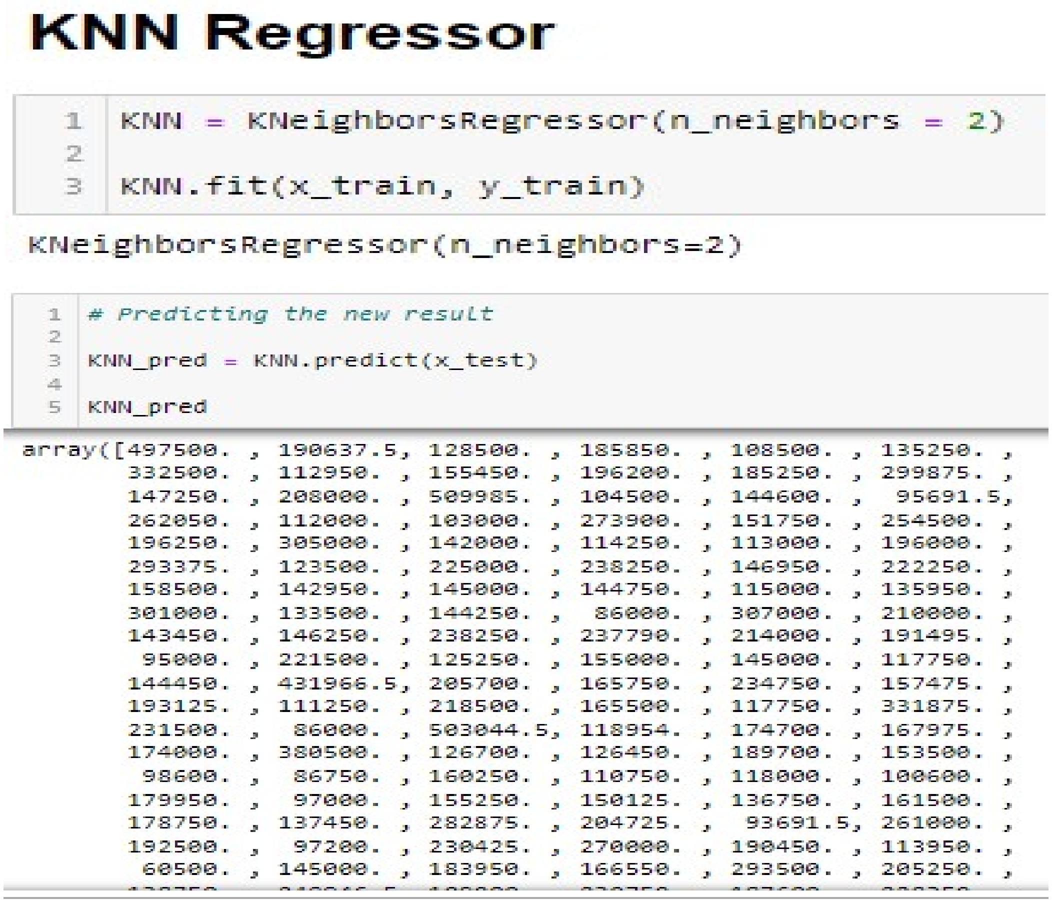
```
array([256688.08817155, 204223.08659   , 109890.38811304, 246407.94173583,
       107883.22300311, 184983.46133061, 337732.2089803 , 139640.78262227,
       167653.44223094, 225273.71254572, 186903.86792293, 313805.20605809,
       140009.71288715, 199305.45289377, 195258.14272947,  78345.80172911,
       126658.7024441 , 140330.61824404, 317592.78030951, 140199.07543347,
        83326.89017777, 195280.652163  , 168188.21786071, 198854.3630583 ,
       202780.11892162, 299447.88839209, 117983.21397872, 106944.83319864,
       139959.51467239, 165210.80449453, 228238.55930373,  94138.70949859,
       248577.59114329, 232522.80439988, 196442.51447149, 175344.92434395,
       176073.21292545, 146084.07640641, 242039.31062706, 144058.34493572,
       115106.14068428, 104041.26742747, 266912.54877247, 129155.99982369,
       150512.22586034,  58026.86336072, 329300.00986379, 218165.85571348,
       152259.36794295, 189455.87647037, 238506.76817475, 215455.34542988,
       286183.53435009, 343797.48648858, 113115.08282487, 254682.9277769 ,
       144167.96286903, 162388.29327192, 165453.58851096,  90055.68654516,
       104648.76273926, 370135.94190307, 205968.0474855 , 179681.52820127,
       240855.42411666, 163532.67998818, 186609.47911161,  97820.99467318,
       241557.27096387, 141484.78249687,  86550.93215834, 295047.97918466,
       206117.62526479,  58308.35501493, 373599.51825468, 177342.50179593,
```



## Decision Tree Regressor

```python
DT = DecisionTreeRegressor()

DT.fit(x_train, y_train)

# Predicting the new result

DT_pred = DT.predict(x_test)

DT_pred
```

```
array([176000., 162000., 135900., 191000., 140000., 132000., 226000.,
       110000., 194000., 205000., 188000., 317000., 112500., 190000.,
       138800., 129000., 133000., 123000., 281000., 108000., 108959.,
       201000., 145000., 165600., 244000., 317000., 135000., 146000.,
       113000., 188000., 227000.,  82000., 236500., 191000., 178000.,
       192000., 178000., 173000., 215000., 133900., 128000., 112000.,
       255000., 147000., 140000.,  80000., 301500., 187750., 136500.,
       163000., 222500., 185500., 302000., 306000., 118500., 236500.,
       139000., 154000.,  95000., 117500., 124500., 611657., 176000.,
       173000., 227875., 175000., 132000.,  76000., 208900., 180000.,
       110000., 260000., 274970., 108000., 378500., 162500., 240000.,
       154000., 135000., 426000., 110000.,  82500., 256000.,  89471.,
        92900., 119000., 149000., 125000., 176000.,  93500., 268000.,
        88000., 142500., 224000., 114500., 129500., 169000., 142500.,
       341000., 228000., 139000., 252000., 226000., 106500., 269500.,
       290000., 219500., 190000.,  72500., 130000., 205000., 181000.,
       317000., 175900.,  79500., 275000., 176432., 159000., 192500.,
       194000., 145000., 190000., 203000., 160000., 201000., 259500.,
       127500., 191000., 319900., 107000., 263000., 119500., 202500.,
```



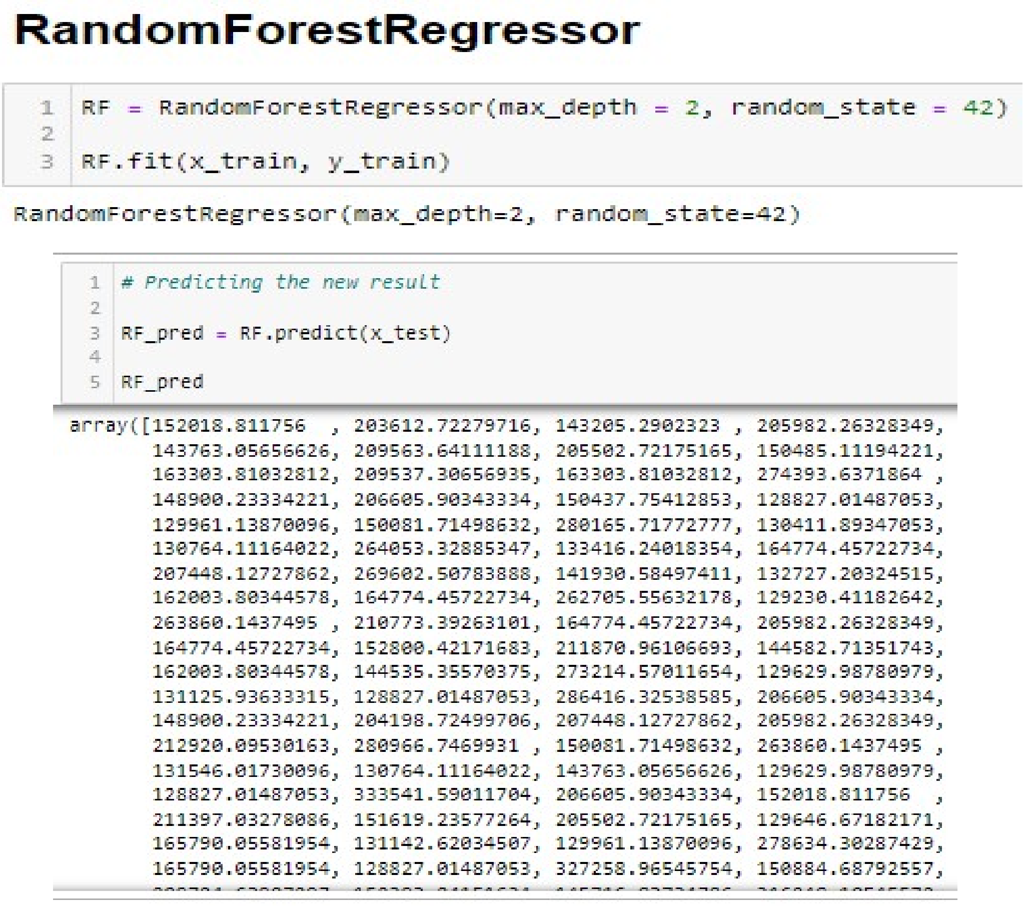
## KNN Regressor

```python
KNN = KNeighborsRegressor(n_neighbors = 2)

KNN.fit(x_train, y_train)
```

```
KNeighborsRegressor(n_neighbors=2)
```

```python
# Predicting the new result

KNN_pred = KNN.predict(x_test)

KNN_pred
```

```
array([497500. , 190637.5, 128500. , 185850. , 108500. , 135250. ,
       332500. , 112950. , 155450. , 196200. , 185250. , 299875. ,
       147250. , 208000. , 509985. , 104500. , 144600. ,  95691.5,
       262050. , 112000. , 103000. , 273900. , 151750. , 254500. ,
       196250. , 305000. , 142000. , 114250. , 113000. , 196000. ,
       293375. , 123500. , 225000. , 238250. , 146950. , 222250. ,
       158500. , 142950. , 145000. , 144750. , 115000. , 135950. ,
       301000. , 133500. , 144250. ,  86000. , 307000. , 210000. ,
       143450. , 146250. , 238250. , 237790. , 214000. , 191495. ,
        95000. , 221500. , 125250. , 155000. , 145000. , 117750. ,
       144450. , 431966.5, 205700. , 165750. , 234750. , 157475. ,
       193125. , 111250. , 218500. , 165500. , 117750. , 331875. ,
       231500. ,  86000. , 503044.5, 118954. , 174700. , 167975. ,
       174000. , 380500. , 126700. , 126450. , 189700. , 153500. ,
        98600. ,  86750. , 160250. , 110750. , 118000. , 100600. ,
       179950. ,  97000. , 155250. , 150125. , 136750. , 161500. ,
       178750. , 137450. , 282875. , 204725. ,  93691.5, 261000. ,
       192500. ,  97200. , 230425. , 270000. , 190450. , 113950. ,
        60500. , 145000. , 183950. , 166550. , 293500. , 205250. ,
```



## RandomForestRegressor

```python
RF = RandomForestRegressor(max_depth = 2, random_state = 42)

RF.fit(x_train, y_train)
```

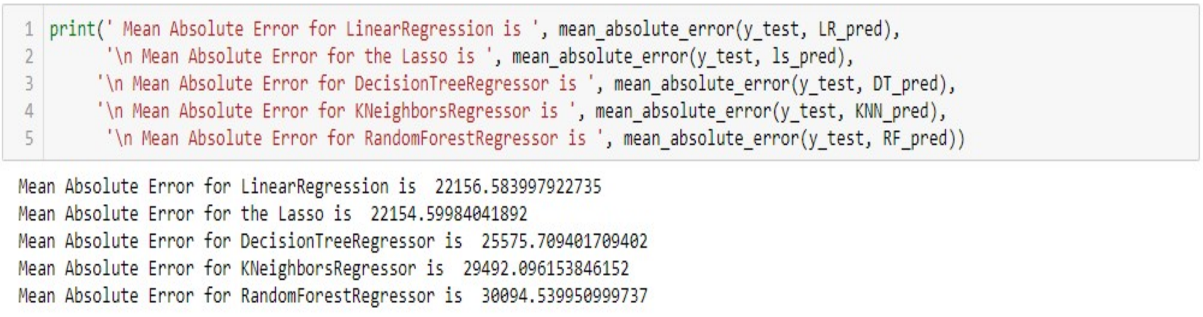
```
RandomForestRegressor(max_depth=2, random_state=42)
```

```python
# Predicting the new result

RF_pred = RF.predict(x_test)

RF_pred
```

```
array([152018.811756  , 203612.72279716, 143205.2902323 , 205982.26328349,
       143763.05656626, 209563.64111188, 205502.72175165, 150485.11194221,
       163303.81032812, 209537.30656935, 163303.81032812, 274393.6371864 ,
       148900.23334221, 206605.90343334, 150437.75412853, 128827.01487053,
       129961.13870096, 150081.71498632, 280165.71772777, 130411.89347053,
       130764.11164022, 264053.32885347, 133416.24018354, 164774.45722734,
       207448.12727862, 269602.50783888, 141930.58497411, 132727.20324515,
       162003.80344578, 164774.45722734, 262705.55632178, 129230.41182642,
       263860.1437495 , 210773.39263101, 164774.45722734, 205982.26328349,
       164774.45722734, 152800.42171683, 211870.96106693, 144582.71351743,
       162003.80344578, 144535.35570375, 273214.57011654, 129629.98780979,
       131125.93633315, 128827.01487053, 286416.32538585, 206605.90343334,
       148900.23334221, 204198.72499706, 207448.12727862, 205982.26328349,
       212920.09530163, 280966.7469931 , 150081.71498632, 263860.1437495 ,
       131546.01730096, 130764.11164022, 143763.05656626, 129629.98780979,
       128827.01487053, 333541.59011704, 206605.90343334, 152018.811756  ,
       211397.03278086, 151619.23577264, 205502.72175165, 129646.67182171,
       165790.05581954, 131142.62034507, 129961.13870096, 278634.30287429,
       165790.05581954, 128827.01487053, 327258.96545754, 150884.68792557,
```

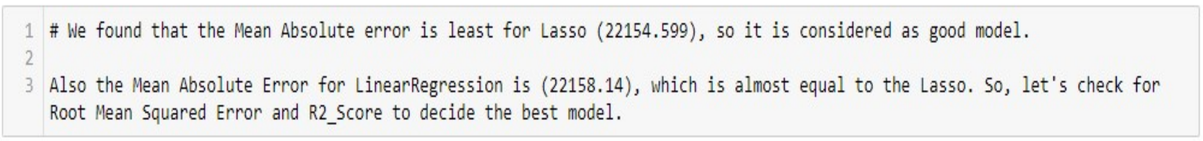
## Key Metrics for success in solving problem under consideration

### MEAN ABSOLUTE ERROR



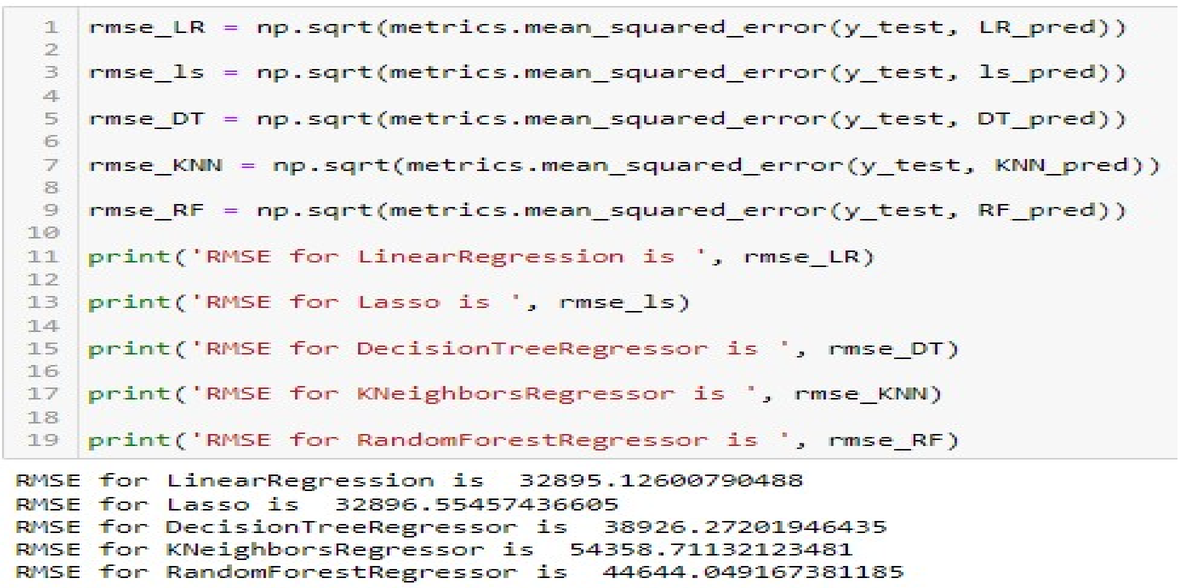
```
print(' Mean Absolute Error for LinearRegression is ', mean_absolute_error(y_test, LR_pred),
      '\n Mean Absolute Error for the Lasso is ', mean_absolute_error(y_test, ls_pred),
     '\n Mean Absolute Error for DecisionTreeRegressor is ', mean_absolute_error(y_test, DT_pred),
     '\n Mean Absolute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

```
Mean Absolute Error for LinearRegression is  22156.583997922735
Mean Absolute Error for the Lasso is  22154.59984041892
Mean Absolute Error for DecisionTreeRegressor is  25575.709401709402
Mean Absolute Error for KNeighborsRegressor is  29492.096153846152
Mean Absolute Error for RandomForestRegressor is  30094.539950999737
```



```
# We found that the Mean Absolute error is least for Lasso (22154.599), so it is considered as good model.

Also the Mean Absolute Error for LinearRegression is (22158.14), which is almost equal to the Lasso. So, let's check for
Root Mean Squared Error and R2_Score to decide the best model.
```

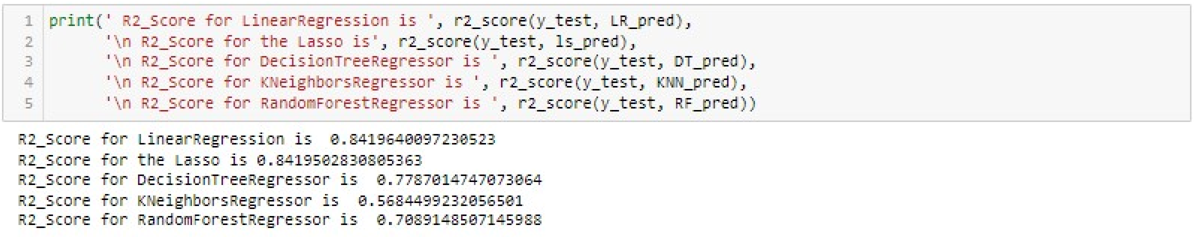
### ROOT MEAN SQUARE ERROR



```
rmse_LR = np.sqrt(metrics.mean_squared_error(y_test, LR_pred))

rmse_ls = np.sqrt(metrics.mean_squared_error(y_test, ls_pred))

rmse_DT = np.sqrt(metrics.mean_squared_error(y_test, DT_pred))

rmse_KNN = np.sqrt(metrics.mean_squared_error(y_test, KNN_pred))

rmse_RF = np.sqrt(metrics.mean_squared_error(y_test, RF_pred))

print('RMSE for LinearRegression is ', rmse_LR)

print('RMSE for Lasso is ', rmse_ls)

print('RMSE for DecisionTreeRegressor is ', rmse_DT)

print('RMSE for KNeighborsRegressor is ', rmse_KNN)

print('RMSE for RandomForestRegressor is ', rmse_RF)
```

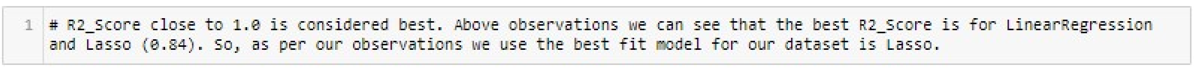
```
RMSE for LinearRegression is  32895.12600790488
RMSE for Lasso is  32896.55457436605
RMSE for DecisionTreeRegressor is  38926.27201946435
RMSE for KNeighborsRegressor is  54358.71132123481
RMSE for RandomForestRegressor is  44644.049167381185
```

### We found least RMSE with Linear Regression hence we can consider it as best model, we will cross verify it with R2 method.



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

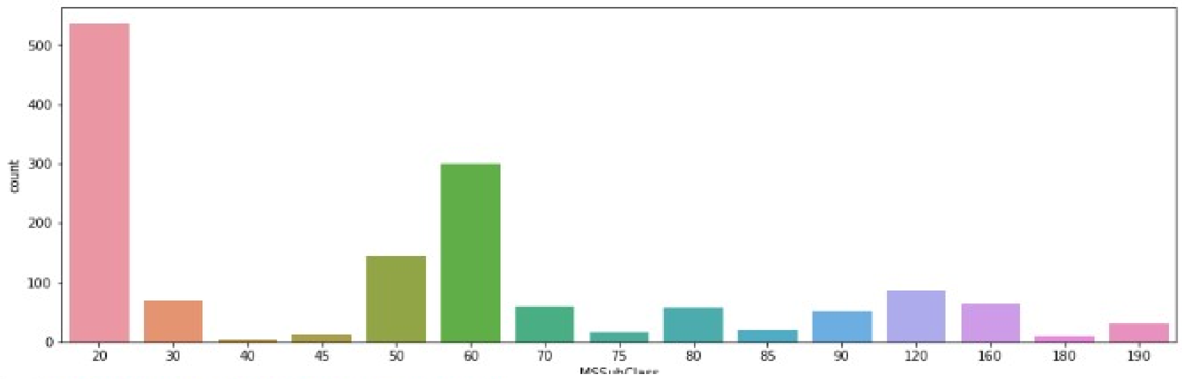
```
R2_Score for LinearRegression is  0.8419640097230523
R2_Score for the Lasso is 0.8419502830805363
R2_Score for DecisionTreeRegressor is  0.7787014747073064
R2_Score for KNeighborsRegressor is  0.5684499232056501
R2_Score for RandomForestRegressor is  0.7089148507145988
```



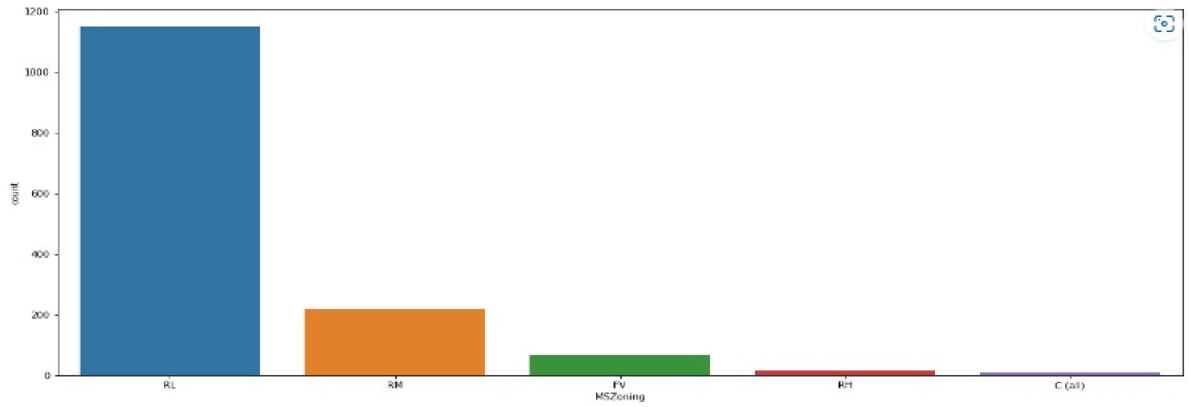
```
# R2_Score close to 1.0 is considered best. Above observations we can see that the best R2_Score is for LinearRegression
and Lasso (0.84). So, as per our observations we use the best fit model for our dataset is Lasso.
```

## Data Visualization

```python
#Visualize the value counts of the columns MSSubClass

plt.figure(figsize = (15, 5))

sns.countplot(df.MSSubClass)

plt.show()
```

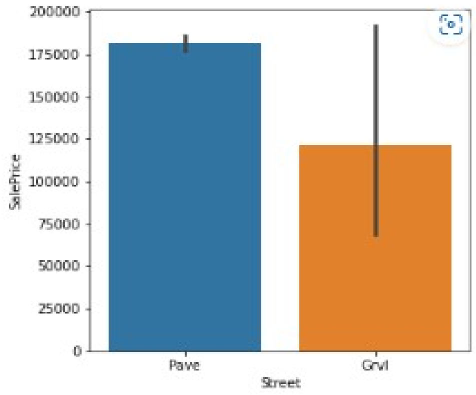


```python
# Check the value counts of the column 'MSZoning'

plt.figure(figsize = [20, 8])

sns.countplot(df.MSZoning)

plt.show()
```



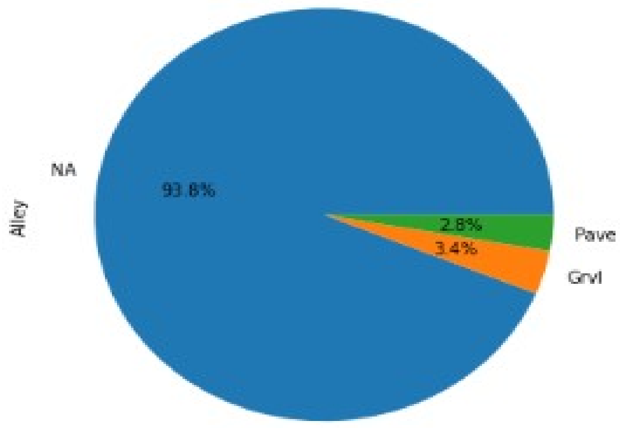
```python
# We can see maximum number of sale is Residential Low Density (RL) and the minimum is for the commercial.
```

```python
# Checking for the sale price on the basis of road access to the property

plt.figure(figsize = [5, 5])

sns.barplot(x = 'Street', y = 'SalePrice', data = df.sort_values('SalePrice', ascending = False))

plt.show()
```



```python
# we can observe that property with the road access of Paver are more demand and so its price is also high.
# Let's check the alley access to property

plt.figure(figsize = [6, 6])

df['Alley'].value_counts().plot.pie(autopct = '%0.1f%%')
```

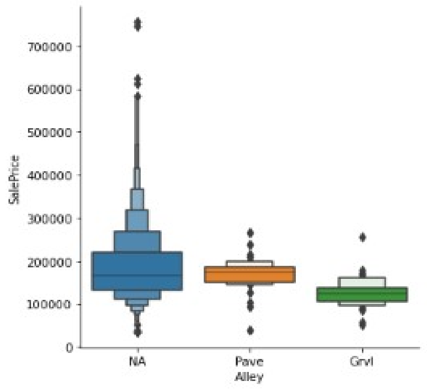
```
<AxesSubplot:ylabel='Alley'>
```



```python
# Approx 94% properties have no alley access.
```

```python
# Check the effect of alley access on sale price.

plt.figure(figsize = [10, 6])

sns.catplot(x = 'Alley', y = 'SalePrice', data = df.sort_values('SalePrice', ascending = False), kind = 'boxen')

plt.show()
```

```
<Figure size 720x432 with 0 Axes>
```



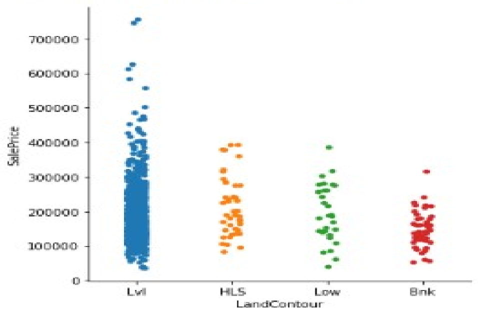
```python
# As we can see max property has no alley access we can drop this column
```

```python
# Drop the alley column

df = df.drop(['Alley'], axis = 1)

df.head()
```

| | Id | MSSubClass | MSZoning | LotFrontage | LotArea | Street | LotShape | LandContour | Utilities | LotConfig | LandSlope | Neighborhood | Condition1 | Condition2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 127 | 120 | RL | 70.049958 | 4928 | Pave | IR1 | Lvl | AllPub | Inside | Gtl | NPkVill | Norm | Norm |
| 1 | 889 | 20 | RL | 95.000000 | 15865 | Pave | IR1 | Lvl | AllPub | Inside | Mod | NAmes | Norm | Norm |
| 2 | 793 | 60 | RL | 92.000000 | 9920 | Pave | IR1 | Lvl | AllPub | CulDSac | Gtl | NoRidge | Norm | Norm |
| 3 | 110 | 20 | RL | 105.000000 | 11751 | Pave | IR1 | Lvl | AllPub | Inside | Gtl | NWAmes | Norm | Norm |
| 4 | 422 | 20 | RL | 70.049958 | 16635 | Pave | IR1 | Lvl | AllPub | FR2 | Gtl | NWAmes | Norm | Norm |

```
# Check the effect of LandContour of property on sale price

plt.figure(figsize = [20, 8])

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

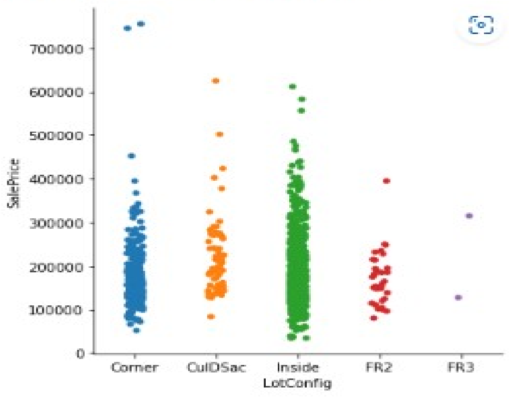
```
<seaborn.axisgrid.FacetGrid at 0x22ab0f283d0>
<Figure size 1440x576 with 0 Axes>
```



```
# plot shows that levelled properties have higher sale price.
```

```
# Check for Lot configuration and its effect on sale price.

plt.figure(figsize = [20, 8])

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

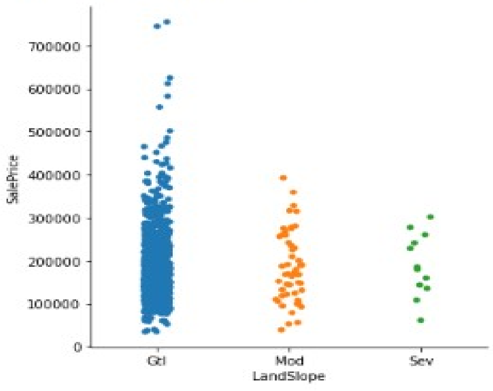
```
<seaborn.axisgrid.FacetGrid at 0x22ab0fad1f0>

<Figure size 1440x576 with 0 Axes>
```



```
# From above we can observe that for INSIDE LOT Config Sale price increases.
```

```
# Check LandSlope wise sale pricing of the properties

plt.figure(figsize = [20, 8])

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

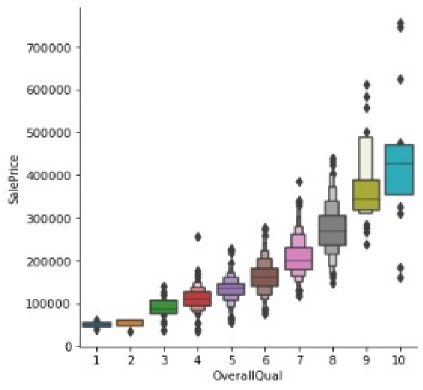
```
<seaborn.axisgrid.FacetGrid at 0x22ab150e2b0>

<Figure size 1440x576 with 0 Axes>
```



```
# Properties with Gentle slope have higher sale price compared to moderate and severe slope.
```

```python
# Check the sale prices based on the ratings of overall material and finish of the house

plt.figure(figsize = [15, 8])

sns.catplot(x = 'OverallQual', y = 'SalePrice', data = df.sort_values('SalePrice', ascending = False), kind = 'boxen')

plt.show()
```

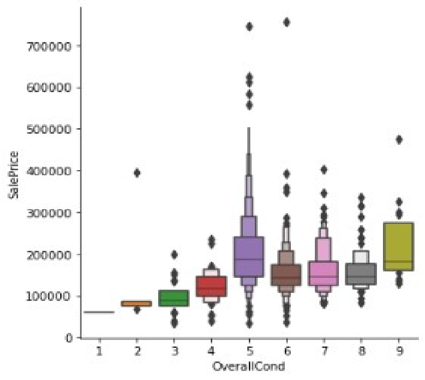
```
<Figure size 1080x576 with 0 Axes>
```



```python
# Sale price goes on increasing according to increase in overall quality of the house.
```

```python
# Check the sale prices based on the ratings of overall condition of the house

plt.figure(figsize = [15, 8])

sns.catplot(x = 'OverallCond', y = 'SalePrice', data = df.sort_values('SalePrice', ascending = False), kind = 'boxen')

plt.show()
```

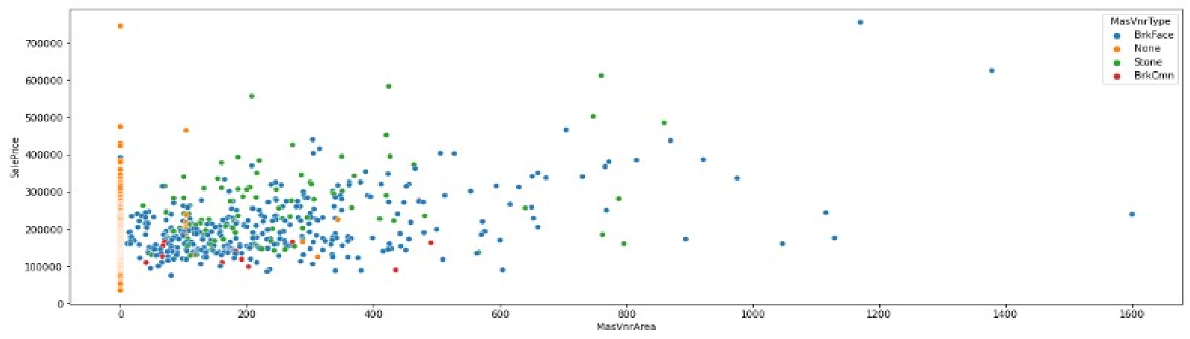
```
<Figure size 1080x576 with 0 Axes>
```



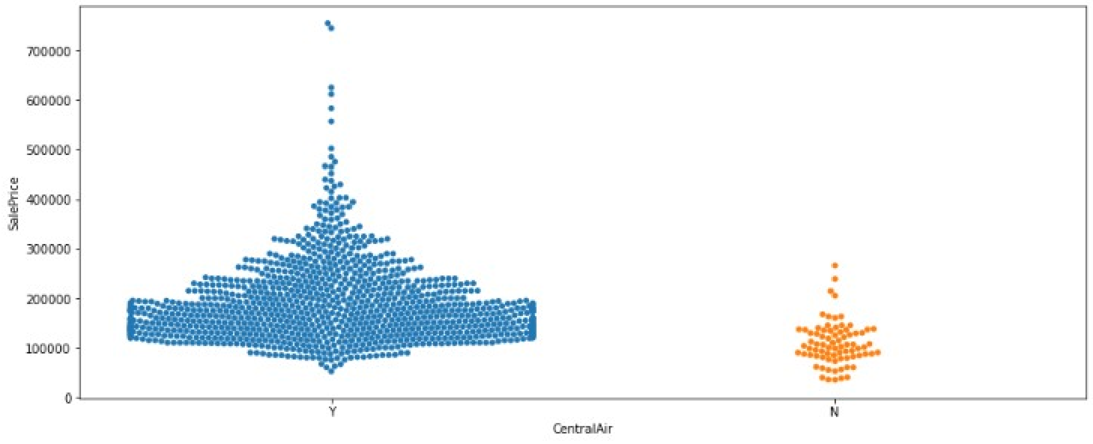
```python
# Plot shows overall condition of house being average and above average will fetch good sale price. whereas highest price
is paid for overall condition rated 9.
```

```python
# Check for the sale price based on the masonry veneer

plt.figure(figsize = [20, 6])

sns.scatterplot(x = 'MasVnrArea', y = 'SalePrice', hue = 'MasVnrType', data = df.sort_values('SalePrice', ascending = False)
```

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

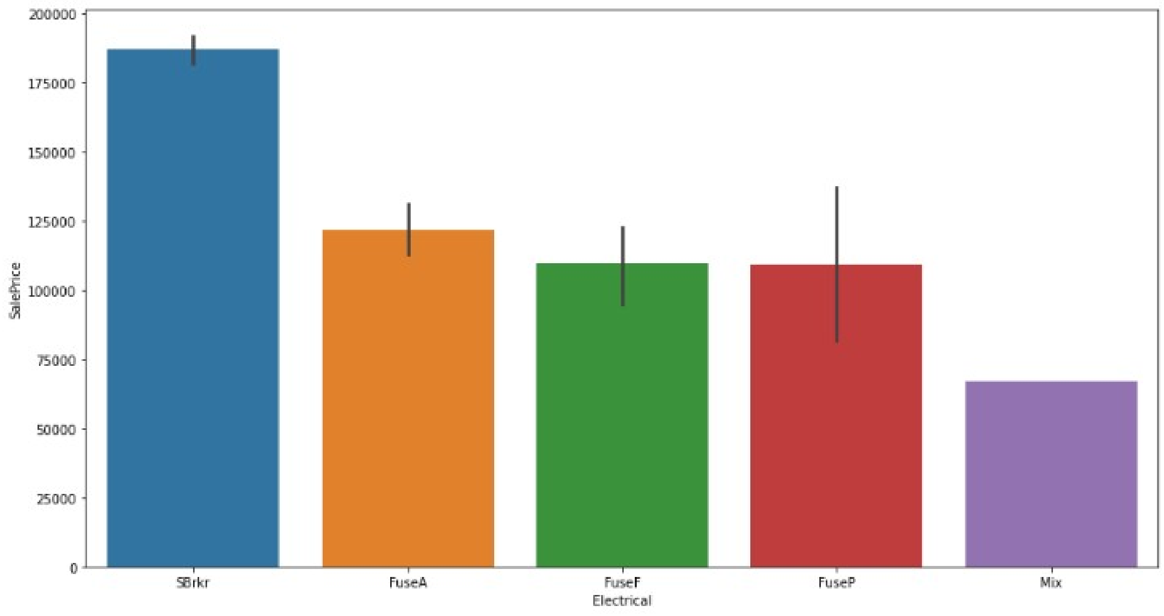


```python
# Masonary Vaneer does not affect much the sale price of houses.
# Check price of the houses on the basis of air conditioning

plt.figure(figsize = [15, 6])

sns.swarmplot(x = 'CentralAir', y = 'SalePrice', data = df.sort_values('SalePrice', ascending = False))

plt.show()
```



```python
# Houses having Central air conditioning have higher sale price as compared to NO Central air conditioning.
```

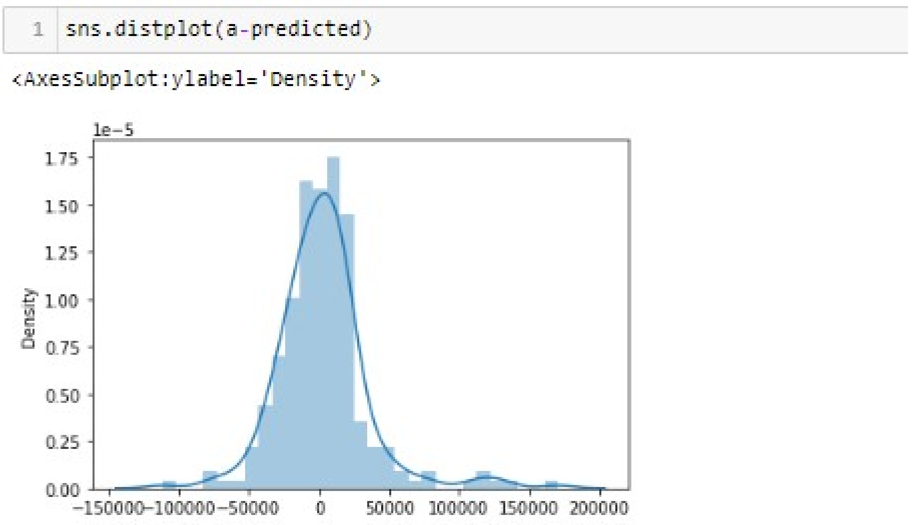
```python
# Check effect of sale price due to electrical system of the house

plt.figure(figsize = [15, 8])

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

plt.show()
```



```python
# Houses having standard circuit breakers and Romex have high sale price.
```

# CONCLUSION

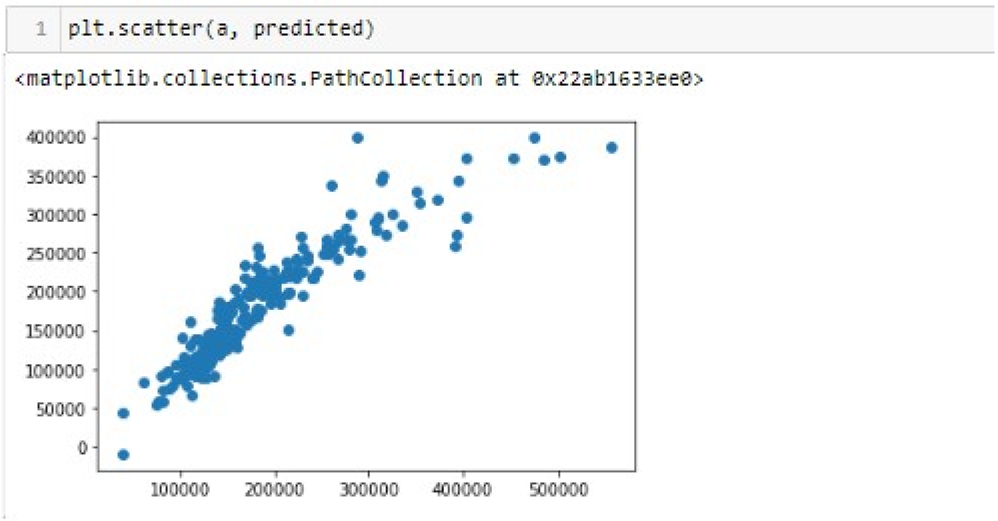
- MSSub Class has the highest impact on House Sale Prices, followed by Basement Full Bath and Basement Half Bath.
- Other than the Basement related features, Condition 2, Exterior Quality and Lot Area are some of the other important features affecting the Sale price of the houses.



```
sns.distplot(a-predicted)
```

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

**Above plot shows almost perfect distribution**



```
plt.scatter(a, predicted)
```

```
<matplotlib.collections.PathCollection at 0x22ab1633ee0>
```

**Above we can see linear relationship throughout.**