HOUSING PROJECT

MACHINE LEARNING

REGRESSION PROBLEM

Submitted by:

RICHARD PRABHAKAR

ACKNOWLEDGMENT

This includes mentioning of all the references, research papers, data sources, professionals and other resources that helped you and guided you in completion of the project.

Links:-

https://stackoverflow.com/questions/72677752/create-x-train-and-y-train-for-csv-dataset-in-python

https://medium.com/analytics-vidhya/zomato-bangalore-restaurant-analysis-and-rating-prediction-101fd635ab15

https://stackoverflow.com/questions/68794590/how-should-i-predict-target-variable-if-it-is-not-included-in-the-test-data-for

https://www.kaggle.com/datasets/nehalbirla/vehicle-dataset-from-cardekho

Medium.com- blogs on EDA

EDA by Analytics Vidya

Regularized Regression Models Kaggle notebook – by CHOWDHURY SALEH AHMED RONY, KERIMCAN ARSLAN, JAKKI SESHAPANPU

https://www.kaggle.com/code/spscientist/a-simple-tutorial-on-exploratory-data-analysis

Geeksforgeeks.com- Dataframe manipulation

Blogs on current housing market industry:-

https://www.forbes.com/advisor/mortgages/real-estate/housing-market-predictions/

https://money.com/housing-market-2023-expert-predictions/

https://fortune.com/2022/12/07/housing-market-forecast-home-prices-2023-housing-crash-prediction/

https://www.noradarealestate.com/blog/housing-market-forecast-for-next-5-years/

INTRODUCTION

- ## Business Problem Framing

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. They want to know:-

i. Which variables are important to predict the price of the house
ii. How do these variables describe the price of the house

So, we are required to create model 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

Intro to the Industry or domain:
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.

SWOT ANALYSIS OF DOMAIN:

Strengths:

1. Value Multiplies Overtime
2. Less Risky Investment
3. Less Risky Investment
4. Inflation Hedge- Especially during Recession
5. Used as Collateral
6. Traded in Open Market

Weakness:

1. Lead cost is high as compared to the conversion ratio

2. Not having an established setup or insight in new markets or a process
3. Abiding with real estate agent laws- the middleman
4. Failing to leverage technology
5. Increasing presence of online estate agents

Opportunities:

1. Global Demand
2. Rapidly Growing-New Areas have Great Potential
3. New sectors offer great potential i.e. student accommodation & care homes.
4. Overall increase in the economic development of that area and the standard of living
5. Can be switched to other forms of industry like tourism, offices , and commercial,etc

Threats:

1. Economic slowdown may affect demand i.e. periods of recession.
2. Competition from other investment asset classes such as equities and bonds.
3. Seasonal demand may affect prices
4. When people stop showing interest
5. The correct features are not attracting them

- Review of Literature

  Points I would like to enumerate regarding the project

  1. Feature selection: Machine learning algorithms can be used to identify which features are most important in predicting the price of a house in a new market. These features might include location, size of the house, number of bedrooms and bathrooms, and proximity to amenities such as schools, shops, and public transportation.
  2. Regression analysis: Machine learning algorithms can be used to perform regression analysis, which helps to understand the relationship between different features and the price of a house. This can be useful for identifying which features have the greatest impact on the price and for developing pricing models that take these features into account.
  3. Feature engineering: Machine learning algorithms can be used to create new features based on existing ones, which can help to improve the accuracy of pricing models. For example, a machine learning algorithm might create a new feature that represents the average price of houses in a

particular neighbourhood, which could be useful in predicting the price of a specific house in that neighbourhood.

4. Dimensionality reduction: Machine learning algorithms can be used to reduce the number of features in a dataset, which can help to improve the accuracy and efficiency of pricing models. This can be particularly useful in a new market where data may be limited.
5. Predictive modeling for housing demand: Machine learning algorithms can be used to predict housing demand in a particular area, taking into account factors such as population growth, economic conditions, and existing housing supply. This can help developers and policymakers determine where to build new housing units and what types of housing to build.
6. Pricing prediction: Machine learning algorithms can also be used to predict housing prices, which can be helpful for buyers and sellers in making informed decisions about the housing market.
7. Affordability assessment: Machine learning can be used to assess the affordability of housing in a particular area, taking into account factors such as income, property taxes, and other costs of living. This can be useful for policymakers looking to address issues of housing affordability.
8. Market analysis: Research has examined the use of data science and machine learning techniques to analyze market trends and forecast demand for housing in specific locations. This can involve the use of machine learning algorithms to analyze historical data on housing prices, sales volumes, and other indicators of market conditions.
9. Customer segmentation: Research has explored the use of data science and machine learning to identify and target specific segments of the housing market, such as first-time homebuyers or investors. This can involve the use of algorithms to analyze customer data and identify patterns that can be used to tailor marketing and sales efforts.

10. Location analysis: Research has examined the use of data science and machine learning to identify locations with the greatest potential for growth in the housing market. This can involve the analysis of data on factors such as population demographics, economic conditions, and local policies to identify areas with the most favourable conditions for the development of housing.

- ## Motivation for the Problem Undertaken

The housing industry involves the construction, design, and management of residential buildings and communities. Research on the housing industry has covered a wide range of topics, including:

- Affordable housing: Research has focused on identifying the factors that contribute to a shortage of affordable housing, such as rising land and construction costs, and on developing strategies to increase the availability of affordable housing, such as inclusionary zoning and housing vouchers.

- Sustainable housing: Research has examined the environmental impact of the housing industry, including the energy efficiency of buildings and the use of sustainable materials in construction.

- Housing finance: Research has examined the role of financial institutions in the housing market, including the impact of mortgage rates on demand for housing and the impact of government policies on the availability of mortgage credit.

- Housing market dynamics: Research has studied the factors that drive changes in housing prices and demand, including economic conditions, demographic trends, and local policies.

- Housing segregation: Research has examined the ways in which racial and economic segregation is perpetuated in the housing market, and the consequences of segregation for individuals and communities.

- Housing policy: Research has analysed the effectiveness of various housing policies, including rent control, affordable housing mandates, and homeownership incentives.

- Market analysis: Data science and machine learning techniques have been used to analyse housing market data, such as real estate listings and transaction records, to identify trends and patterns that can inform investment decisions. This can include predicting price trends, identifying undervalued properties, and analysing the performance of different property types.
- Risk assessment: Machine learning algorithms have been developed to assess the risk of investing in a particular market or property. This can involve analyzing a range of factors, including economic indicators, demographic data, and local market conditions.
- Predictive modeling: Data science and machine learning techniques have been used to build predictive models that can forecast future demand for housing in a particular market. This can help investors to identify promising markets and to optimize their investment strategies.
- Customer segmentation: Data science and machine learning can be used to segment customers into different groups based on their characteristics and preferences, which can inform targeted marketing efforts and help to identify potential buyers in a new market.

Overall, research on the housing industry has sought to understand the complex factors that influence the availability, affordability, and sustainability of housing, and to identify strategies for addressing challenges in the housing market. Therefore using Machine

Learning and leveraging data science in the industry will truly be a benefit to the industry and I would like to undertake such a project to train myself on the future of the housing market analytics

Analytical Problem Framing

- Mathematical/ Analytical Modeling of the Problem

  Various Statistical modelling used and the concept behind their use-case here:-

  Before model building , we used basic statistical tolls to make analytical decisions to do feature engineering

  The Tools we used are mean , median , mode, study of quantile's. we used skewness to see if the data was normally distributed, and we used correlation to see if there is similar relationship between the independent features and we used it to see which features are more related to the label. We also used box-cox which is a transformation technique used to remove outliers and then we started using the models such as :-

  1. Linear regression: This is a basic and widely used technique for modeling the relationship between a dependent variable and one or more independent variables. It involves fitting a straight line (or hyperplane in higher dimensions) to the data such that the distance between the data points and the line is minimized. Linear regression can be used for both simple linear regression (one independent variable) and multiple linear regression (more than one independent variable).
     Reason: as we are predicting a continues target variable
  2. Decision tree regression: This is a type of non-linear regression that involves building a decision tree to make predictions. It is useful for modeling relationships between variables that are not linear.
     Reason: As we are deriving the best features of a house we need to predict the price of a house , the decision tree can narrow it down
  3. Random forest regression: This is an ensemble method that involves training multiple decision tree models and combining

their predictions to make a final prediction. It is often more accurate than a single decision tree, but it is also more computationally expensive
Reason: Similar to Decision tree but multiple trees to get an even better model as with more tree we can cross verify and come up with the best

4. XGBoost (eXtreme Gradient Boosting) is a popular and powerful machine learning algorithm that can be used for both regression and classification tasks. It is an implementation of gradient boosting, which is a type of ensemble method that combines the predictions of multiple weak models to create a strong model.
    a. In the case of regression, XGBoost builds an ensemble of decision trees to make predictions. The algorithm works by training a series of decision tree models in a sequential manner, where each tree is trained to correct the mistakes made by the previous tree. The predictions of all the trees are then combined to make a final prediction.
    b. There are several mathematical, statistical, and analytical techniques that are used in XGBoost regression:
    c. Boosting: As mentioned above, XGBoost is an implementation of gradient boosting, which is a type of boosting algorithm. Boosting algorithms work by training a series of weak models sequentially, where each model is trained to correct the mistakes made by the previous model. The predictions of all the models are then combined to make a final prediction.
    d. Decision trees: XGBoost uses decision trees as the base model for its ensemble. Decision trees are a type of non-linear model that can be used for both regression and classification tasks. They work by dividing the

feature space into regions and making predictions based on which region the data belongs to.

e. Gradient descent: XGBoost uses gradient descent to optimize the loss function and find the optimal values for the model parameters. Gradient descent is an iterative optimization algorithm that works by moving in the direction of the negative gradient of the loss function, which helps to minimize the loss.
f. Regularization: XGBoost includes several regularization techniques that can help to prevent overfitting, including L1 and L2 regularization and early stopping. These techniques help to reduce the complexity of the model and improve its generalization performance.
g. Feature importance: XGBoost includes a feature importance feature that can be used to identify which features are most important for making predictions. This can be useful for feature selection and understanding the underlying relationships in the data.

Reason: One of the best models which as explained above uses many forms of statistical models and get the best or the score with the lowest error to make an effective predictor

- Data Sources and their formats

The company has collected a data set from the sale of houses in Australia.

- Data contains 1460 entries each having 81 variables.
- Derivations of the columns in the dataset

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

20 1-STORY 1946 & NEWER ALL STYLES

30 1-STORY 1945 & OLDER

40 1-STORY W/FINISHED ATTIC ALL AGES

45 1-1/2 STORY - UNFINISHED ALL AGES

50 1-1/2 STORY FINISHED ALL AGES

60 2-STORY 1946 & NEWER

70 2-STORY 1945 & OLDER

75 2-1/2 STORY ALL AGES

80 SPLIT OR MULTI-LEVEL

85 SPLIT FOYER

90 DUPLEX - ALL STYLES AND AGES

120 1-STORY PUD (Planned Unit Development) - 1946 & NEWER

150 1-1/2 STORY PUD - ALL AGES

160 2-STORY PUD - 1946 & NEWER

180 PUD - MULTILEVEL - INCL SPLIT LEV/FOYER

190 2 FAMILY CONVERSION - ALL STYLES AND AGES

MSZoning: Identifies the general zoning classification of the sale.

A Agriculture

C Commercial

FV Floating Village Residential

I Industrial

RH Residential High Density

RL Residential Low Density

RP Residential Low Density Park

RM Residential Medium Density

LotFrontage: Linear feet of street connected to property

LotArea: Lot size in square feet

Street: Type of road access to property

Grvl Gravel

Pave Paved

Alley: Type of alley access to property

Grvl Gravel

Pave Paved

NA No alley access

LotShape: General shape of property

Reg Regular

IR1 Slightly irregular

IR2 Moderately Irregular

IR3 Irregular

LandContour: Flatness of the property

Lvl Near Flat/Level

Bnk Banked - Quick and significant rise from street grade to building

HLS Hillside - Significant slope from side to side

Low Depression

Utilities: Type of utilities available

AllPub All public Utilities (E,G,W,& S)

NoSewr Electricity, Gas, and Water (Septic Tank)

NoSeWa Electricity and Gas Only

ELO Electricity only

LotConfig: Lot configuration

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

LandSlope: Slope of property

Gtl	Gentle slope
Mod	Moderate Slope
Sev	Severe Slope

Neighborhood: Physical locations within Ames city limits

Blmngtn	Bloomington Heights
Blueste	Bluestem
BrDale	Briardale
BrkSide	Brookside
ClearCr	Clear Creek
CollgCr	College Creek
Crawfor	Crawford
Edwards	Edwards
Gilbert	Gilbert
IDOTRR	Iowa DOT and Rail Road
MeadowV	Meadow Village
Mitchel	Mitchell
Names	North Ames
NoRidge	Northridge
NPkVill	Northpark Villa
NridgHt	Northridge Heights
NWAmes	Northwest Ames

OldTown	Old Town
SWISU	South & West of Iowa State University
Sawyer	Sawyer
SawyerW	Sawyer West
Somerst	Somerset
StoneBr	Stone Brook
Timber	Timberland
Veenker	Veenker

Condition1: Proximity to various conditions

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

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

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

RRAe Adjacent to East-West Railroad

BldgType: Type of dwelling

1Fam Single-family Detached

2FmCon Two-family Conversion; originally built as one-family dwelling

Duplx Duplex

TwnhsE Townhouse End Unit

TwnhsI Townhouse Inside Unit

HouseStyle: Style of dwelling

1Story One story

1.5Fin One and one-half story: 2nd level finished

1.5Unf One and one-half story: 2nd level unfinished

2Story Two story

2.5Fin Two and one-half story: 2nd level finished

2.5Unf Two and one-half story: 2nd level unfinished

SFoyer Split Foyer

SLvl Split Level

OverallQual: Rates the overall material and finish of the house

10 Very Excellent

9 Excellent

8 Very Good

7 Good

6 Above Average

5 Average

4 Below Average

3 Fair

2 Poor

1 Very Poor

OverallCond: Rates the overall condition of the house

10 Very Excellent

9 Excellent

8 Very Good

7 Good

6 Above Average

5 Average

4 Below Average

3 Fair

2 Poor

1 Very Poor

YearBuilt: Original construction date

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

RoofStyle: Type of roof

Flat Flat

Gable Gable

Gambrel Gabrel (Barn)

Hip Hip

Mansard Mansard

Shed Shed

RoofMatl: Roof material

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

Exterior1st: Exterior covering on house

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

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

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

MasVnrType: Masonry veneer type

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

MasVnrArea: Masonry veneer area in square feet

ExterQual: Evaluates the quality of the material on the exterior

Ex Excellent

Gd Good

TA Average/Typical

Fa Fair

Po Poor

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

Ex Excellent

Gd Good

TA Average/Typical

Fa Fair

Po Poor

Foundation: Type of foundation

BrkTil Brick & Tile

CBlock Cinder Block

PConc Poured Contrete

Slab Slab

Stone Stone

Wood Wood

BsmtQual: Evaluates the height of the basement

Ex Excellent (100+ inches)

Gd Good (90-99 inches)

TA Typical (80-89 inches)

Fa Fair (70-79 inches)

Po Poor (<70 inches

NA No Basement

BsmtCond: Evaluates the general condition of the basement

Ex Excellent

Gd Good

TA Typical - slight dampness allowed

Fa Fair - dampness or some cracking or settling

Po Poor - Severe cracking, settling, or wetness

NA No Basement

BsmtExposure: Refers to walkout or garden level walls

Gd Good Exposure

Av Average Exposure (split levels or foyers typically score average or above)

Mn Mimimum Exposure

No No Exposure

NA No Basement

BsmtFinType1: Rating of basement finished area

GLQ Good Living Quarters

ALQ Average Living Quarters

BLQ Below Average Living Quarters

Rec Average Rec Room

LwQ Low Quality

Unf Unfinshed

NA No Basement

BsmtFinSF1: Type 1 finished square feet

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

GLQ Good Living Quarters

ALQ Average Living Quarters

BLQ Below Average Living Quarters

Rec Average Rec Room

LwQ Low Quality

Unf Unfinshed

NA No Basement

BsmtFinSF2: Type 2 finished square feet

BsmtUnfSF: Unfinished square feet of basement area

TotalBsmtSF: Total square feet of basement area

Heating: Type of heating

Floor Floor Furnace

GasA Gas forced warm air furnace

GasW Gas hot water or steam heat

Grav Gravity furnace

OthW Hot water or steam heat other than gas

Wall Wall furnace

HeatingQC: Heating quality and condition

Ex Excellent

Gd Good

TA Average/Typical

Fa Fair

Po Poor

CentralAir: Central air conditioning

N No

Y Yes

Electrical: Electrical system

SBrkr Standard Circuit Breakers & Romex

FuseA Fuse Box over 60 AMP and all Romex wiring (Average)

FuseF 60 AMP Fuse Box and mostly Romex wiring (Fair)

FuseP 60 AMP Fuse Box and mostly knob & tube wiring (poor)

Mix Mixed

1stFlrSF: First Floor square feet

2ndFlrSF: Second floor square feet

LowQualFinSF: Low quality finished square feet (all floors)

GrLivArea: Above grade (ground) living area square feet

BsmtFullBath: Basement full bathrooms

BsmtHalfBath: Basement half bathrooms

FullBath: Full bathrooms above grade

HalfBath: Half baths above grade

Bedroom: Bedrooms above grade (does NOT include basement bedrooms)

Kitchen: Kitchens above grade

KitchenQual: Kitchen quality

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

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

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

- Typ Typical Functionality
- Min1 Minor Deductions 1
- Min2 Minor Deductions 2
- Mod Moderate Deductions
- Maj1 Major Deductions 1
- Maj2 Major Deductions 2
- Sev Severely Damaged
- Sal Salvage only

Fireplaces: Number of fireplaces

FireplaceQu: Fireplace quality

- Ex Excellent - Exceptional Masonry Fireplace
- Gd Good - Masonry Fireplace in main level

TA Average - Prefabricated Fireplace in main living area or Masonry Fireplace in basement

Fa Fair - Prefabricated Fireplace in basement

Po Poor - Ben Franklin Stove

NA No Fireplace

GarageType: Garage location

2Types More than one type of garage

Attchd Attached to home

Basment Basement Garage

BuiltIn Built-In (Garage part of house - typically has room above garage)

CarPort Car Port

Detchd Detached from home

NA No Garage

GarageYrBlt: Year garage was built

GarageFinish: Interior finish of the garage

Fin Finished

RFn Rough Finished

Unf Unfinished

NA No Garage

GarageCars: Size of garage in car capacity

GarageArea: Size of garage in square feet

GarageQual: Garage quality

Ex Excellent

Gd Good

TA Typical/Average

Fa Fair

Po Poor

NA No Garage

GarageCond: Garage condition

Ex Excellent

Gd Good

TA Typical/Average

Fa Fair

Po Poor

NA No Garage

PavedDrive: Paved driveway

Y Paved

P Partial Pavement

N Dirt/Gravel

WoodDeckSF: Wood deck area in square feet

OpenPorchSF: Open porch area in square feet

EnclosedPorch: Enclosed porch area in square feet

3SsnPorch: Three season porch area in square feet

ScreenPorch: Screen porch area in square feet

PoolArea: Pool area in square feet

PoolQC: Pool quality

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

Fence: Fence quality

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

MiscFeature: Miscellaneous feature not covered in other categories

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

MiscVal: $Value of miscellaneous feature

MoSold: Month Sold (MM)

YrSold: Year Sold (YYYY)

SaleType: Type of sale

WD Warranty Deed - Conventional

CWD Warranty Deed - Cash

VWD Warranty Deed - VA Loan

New Home just constructed and sold

COD Court Officer Deed/Estate

Con Contract 15% Down payment regular terms

ConLw Contract Low Down payment and low interest

ConLI Contract Low Interest

ConLD Contract Low Down

Oth Other

SaleCondition: Condition of sale

Normal Normal Sale

Abnorml Abnormal Sale - trade, foreclosure, short sale

AdjLand Adjoining Land Purchase

Alloca Allocation - two linked properties with separate deeds, typically condo with a garage unit

Family Sale between family members

Partial Home was not completed when last assessed (associated with New Homes)

SnapShot of Datatypes:-

```
RangeIndex: 1168 entries, 0 to 1167
Data columns (total 81 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Id              1168 non-null   int64
 1   MSSubClass      1168 non-null   int64
 2   MSZoning        1168 non-null   object
 3   LotFrontage     954 non-null    float64
 4   LotArea         1168 non-null   int64
 5   Street          1168 non-null   object
 6   Alley           77 non-null     object
 7   LotShape        1168 non-null   object
 8   LandContour     1168 non-null   object
 9   Utilities       1168 non-null   object
 10  LotConfig       1168 non-null   object
 11  LandSlope       1168 non-null   object
 12  Neighborhood    1168 non-null   object
 13  Condition1      1168 non-null   object
 14  Condition2      1168 non-null   object
 15  BldgType        1168 non-null   object
 16  HouseStyle      1168 non-null   object
 17  OverallQual     1168 non-null   int64
 18  OverallCond     1168 non-null   int64
 19  YearBuilt       1168 non-null   int64
 20  YearRemodAdd    1168 non-null   int64
 21  RoofStyle       1168 non-null   object
 22  RoofMatl        1168 non-null   object
 23  Exterior1st     1168 non-null   object
 24  Exterior2nd     1168 non-null   object
 25  MasVnrType      1161 non-null   object
 26  MasVnrArea      1161 non-null   float64
 27  ExterQual       1168 non-null   object
 28  ExterCond       1168 non-null   object
 29  Foundation      1168 non-null   object
 30  BsmtQual        1138 non-null   object
 31  BsmtCond        1138 non-null   object
 32  BsmtExposure    1137 non-null   object
 33  BsmtFinType1    1138 non-null   object
 34  BsmtFinSF1      1168 non-null   int64
 35  BsmtFinType2    1137 non-null   object
 36  BsmtFinSF2      1168 non-null   int64
 37  BsmtUnfSF       1168 non-null   int64
 38  TotalBsmtSF     1168 non-null   int64
 39  Heating         1168 non-null   object
 40  HeatingQC       1168 non-null   object
 41  CentralAir      1168 non-null   object
 42  Electrical      1168 non-null   object
 43  1stFlrSF        1168 non-null   int64
```

```
44  2ndFlrSF       1168 non-null   int64
45  LowQualFinSF   1168 non-null   int64
46  GrLivArea      1168 non-null   int64
47  BsmtFullBath   1168 non-null   int64
48  BsmtHalfBath   1168 non-null   int64
49  FullBath       1168 non-null   int64
50  HalfBath       1168 non-null   int64
51  BedroomAbvGr   1168 non-null   int64
52  KitchenAbvGr   1168 non-null   int64
53  KitchenQual    1168 non-null   object
54  TotRmsAbvGrd   1168 non-null   int64
55  Functional     1168 non-null   object
56  Fireplaces     1168 non-null   int64
57  FireplaceQu    617 non-null    object
58  GarageType     1104 non-null   object
59  GarageYrBlt    1104 non-null   float64
60  GarageFinish   1104 non-null   object
61  GarageCars     1168 non-null   int64
62  GarageArea     1168 non-null   int64
63  GarageQual     1104 non-null   object
64  GarageCond     1104 non-null   object
65  PavedDrive     1168 non-null   object
66  WoodDeckSF     1168 non-null   int64
67  OpenPorchSF    1168 non-null   int64
68  EnclosedPorch  1168 non-null   int64
69  3SsnPorch      1168 non-null   int64
70  ScreenPorch    1168 non-null   int64
71  PoolArea       1168 non-null   int64
72  PoolQC         7 non-null      object
73  Fence          237 non-null    object
74  MiscFeature    44 non-null     object
75  MiscVal        1168 non-null   int64
76  MoSold         1168 non-null   int64
77  YrSold         1168 non-null   int64
78  SaleType       1168 non-null   object
79  SaleCondition  1168 non-null   object
80  SalePrice      1168 non-null   int64
```

- Data Pre-processing Done

The Steps followed in order to clean the data

1. Using log transform to make the label – SalePrice have a normal distribution
2. Checking for duplicates- found none
3. Checking for null values- found many, So treated
    a. Replacing the nan values with mean for LotFrontage
    b. Removing the features-Alley,PoolQC,Fence,MiscFeature as they had 80% or more null values which cannot be replaced
    c. Replacing the columns-BsmtQual,BsmtQual,BsmtCond,BsmtExposure,BsmtFinType1,BsmtFinType2’s null values with NA as a new class.
    d. Replacing missing values for MasVnrType with None

e. Replacing missing values for MasVnrArea with median
f. Replacing the columns- GarageType,GarageFinish,GarageQual,GarageCond ‘s null values with NA as a new class.
g. Handling the missing value for GarageYrBlt with median
h. Handling the missing value for Electrical with new class
i. Handling the missing values for FireplaceQu with NA as a new class

4. Splitting the categorical and numerical columns and assigning them to a variable for studying relationships.
5. Used transformation techniques –Power Transformer to reduce the skewness and also remove any outliers from the data.

- Data Inputs- Logic- Output Relationships

We have already explained the data inputs and their classes as well as the format of the features in Data Sources and formats, so now we will touch upon the relationships with target and features as well as feature and feature.

1. Some categories seem to more diverse with respect to SalePrice than others. Neighborhood has big impact on house prices. Most expensive seems to be Partial SaleCondition. Having pool on property seems to improve price substantially. There are also differences in variabilities between category values.
2. In correlation matrix . OverallQual is main criterion in establishing house price. Neighborhood has big influence, partially it has some intrisinc value in itself, but also houses in certain regions tend to share same characteristics (confunding) what causes similar valuations.
3. Feature to feature - Garages seem to be built same year as houses, basements have generally same area as first floor which is pretty obvious. Garage area is strongly correlated with number of cars. Neighborhood is

correlated with lots of other variables and this confirms the idea that houses in same region share same characteristics. Dwelling type is negatively correlated with kitchen above grade square feet.

4. Expensive houses have pools, better overall qual and condition, open porch and increased importance of MasVnrArea
5. GrLivArea' and 'TotalBsmtSF' seem to be linearly related with 'SalePrice'. Both relationships are positive, which means that as one variable increases, the other also increases. In the case of 'TotalBsmtSF', we can see that the slope of the linear relationship is particularly high.
6. 'OverallQual' and 'YearBuilt' also seem to be related with 'SalePrice'. The relationship seems to be stronger in the case of 'OverallQual', where the box plot shows how sales prices increase with the overall quality.
7. OverallQual', 'GrLivArea' and 'TotalBsmtSF' are strongly correlated with 'SalePrice'. Check!
8. 'GarageCars' and 'GarageArea' are also some of the most strongly correlated variables. However, as we discussed in the last sub-point, the number of cars that fit into the garage is a consequence of the garage area. 'GarageCars' and 'GarageArea' are like twin brothers. You'll never be able to distinguish them. Therefore, we just need one of these variables in our analysis (we can keep 'GarageCars' since its correlation with 'SalePrice' is higher).
9. 'TotalBsmtSF' and '1stFloor' also seem to be twin brothers. We can keep 'TotalBsmtSF' just to say that our first guess was right (re-read 'So... What can we expect?').'TotRmsAbvGrd' and 'GrLivArea', twin brothers again. 'YearBuilt'... It seems that 'YearBuilt' is slightly correlated with 'SalePrice'

- Hardware and Software Requirements and Tools Used

Listing

```
import pandas as pd ## pandas is used to manupulate the dataframe

import numpy as np ## numpy is used to do scientific calculations

import matplotlib.pyplot as plt ## matplotlib used for visualization or graphs

import seaborn as sns ## seaborn used for visualization or graphs

import missingno as msno ## used to visualize missing values

import warnings ## used to remove warnings

warnings.filterwarnings('ignore')

%matplotlib inline


from sklearn.model_selection import train_test_split,cross_validate, GridSearchCV – ## used to split training data and test , in this case we didn't use as we have already separate files for training and testing

from sklearn.preprocessing import StandardScaler,OrdinalEncoder# to convert or encode the categorical or string values into numbers

from sklearn.metrics import mean_squared_error #it is a metric used to check the error we get with each model , lower the better

from sklearn.linear_model import LinearRegression,Lasso,Ridge,BayesianRidge ## linear regression model used to predict and train data

from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor # Ensemle techniques used to work on model to see which is better at prediction and lower error
```

```
from xgboost import XGBRegressor # another machine learning model but boosting techniques

from lightgbm import LGBMRegressor # another Boosting technique


import math #to do mathematics based functions on the data be it for visualization or for any cleaning as well

from IPython.display import Image # to save and show image

import warnings # to remove the warnings to show clean outputs

warnings.filterwarnings("ignore")

sns.set(rc={"figure.figsize": (20, 15)})

sns.set_style("whitegrid")
```

Model/s Development and Evaluation

- Identification of possible problem-solving approaches (methods)

Also explained in statistical modelling and analytical in previous Analytical Problem Framing Chapter, we use the following methods:-

1. Linear regression: This is a basic and widely used technique for modeling the relationship between a dependent variable and one or more independent variables. It involves fitting a straight line (or hyperplane in higher dimensions) to the data such that the distance between the data points and the line is minimized. Linear regression can be used for both simple linear regression (one independent variable) and multiple linear regression (more than one independent variable).
   Reason: as we are predicting a continues target variable
2. Decision tree regression: This is a type of non-linear regression that involves building a decision tree to make predictions. It is useful for modeling relationships between variables that are not linear.
   Reason: As we are deriving the best features of a house we need to predict the price of a house , the decision tree can narrow it down
3. Random forest regression: This is an ensemble method that involves training multiple decision tree models and combining their predictions to make a final prediction. It is often more accurate than a single decision tree, but it is also more computationally expensive
   Reason: Similar to Decision tree but multiple trees to get an even better model as with more tree we can cross verify and come up with the best

4. XGBoost (eXtreme Gradient Boosting) is a popular and powerful machine learning algorithm that can be used for both regression and classification tasks. It is an implementation of gradient boosting, which is a type of

ensemble method that combines the predictions of multiple weak models to create a strong model.

a. In the case of regression, XGBoost builds an ensemble of decision trees to make predictions. The algorithm works by training a series of decision tree models in a sequential manner, where each tree is trained to correct the mistakes made by the previous tree. The predictions of all the trees are then combined to make a final prediction.
b. There are several mathematical, statistical, and analytical techniques that are used in XGBoost regression:
c. Boosting: As mentioned above, XGBoost is an implementation of gradient boosting, which is a type of boosting algorithm. Boosting algorithms work by training a series of weak models sequentially, where each model is trained to correct the mistakes made by the previous model. The predictions of all the models are then combined to make a final prediction.
d. Decision trees: XGBoost uses decision trees as the base model for its ensemble. Decision trees are a type of non-linear model that can be used for both regression and classification tasks. They work by dividing the feature space into regions and making predictions based on which region the data belongs to.
e. Gradient descent: XGBoost uses gradient descent to optimize the loss function and find the optimal values for the model parameters. Gradient descent is an iterative optimization algorithm that works by moving in the direction of the negative gradient of the loss function, which helps to minimize the loss.
f. Regularization: XGBoost includes several regularization techniques that can help to prevent overfitting, including L1 and L2 regularization and early stopping.

These techniques help to reduce the complexity of the model and improve its generalization performance.

g. Feature importance: XGBoost includes a feature importance feature that can be used to identify which features are most important for making predictions. This can be useful for feature selection and understanding the underlying relationships in the data.

Reason: One of the best models which as explained above uses many forms of statistical models and get the best or the score with the lowest error to make an effective predictor

- Testing of Identified Approaches (Algorithms)

Linear Regression alone and with Tuning techniques,Ridge Regression,Random forest Regressor with and withput hyper parameter tuning techniques,Decision tree Regressor with and without tuning,Xgboost Regressor with and without tuning

- Run and Evaluate selected models
- Model 1-Linear Regression

  Linear regression is a statistical method used to model the linear relationship between a dependent variable and one or more independent variables. In the context of a housing project, linear regression might be used to predict the price of a house based on various features or characteristics of the house, such as its size, location, number of bedrooms and bathrooms, age, and so on.
- To build a linear regression model for this purpose, you would need a dataset that includes the prices of a number of houses, along with their respective features. You could then use this data to fit a linear regression model that could be used to predict the price of a new house based on its features.
- Linear regression is often used in housing projects because it is a relatively simple and straightforward method that can be

used to make predictions based on a limited number of input variables. Additionally, linear regression models can be easily interpreted, which can be useful for understanding the factors that influence the price of a house.

- Snapshot :-

```
In [76]: ## importing the library

         from sklearn.linear_model import LinearRegression

In [77]: LR=LinearRegression()
         LR.fit(x_train,y_train)  ## fitting the training data

         x_test_pred_LR=LR.predict(x_test)  ## predicted x test
```

```
Out[78]: array([3.58225096, 3.51073145, 3.47679707, 3.57476195, 3.43600779,
                3.43058714, 3.64094753, 3.5206022 , 3.79807657, 3.5305856 ,
                3.58903708, 3.5133536 , 3.60258812, 3.49633842, 3.49144597,
                3.52558075, 3.57358263, 3.49816668, 3.53052435, 3.53754812,
                3.55020587, 3.52578819, 3.47791423, 3.55743616, 3.56887349,
                3.55205181, 3.56134753, 3.42981331, 3.66393621, 3.49095835,
                3.47625511, 3.59310777, 3.53806694, 3.62039416, 3.68710741,
                3.55840556, 3.63922834, 3.48062436, 3.62837988, 3.66478797,
                3.61418271, 3.50589042, 3.54898038, 3.63996597, 3.69948546,
                3.48975852, 3.4984038 , 3.50357549, 3.55653948, 3.43066761,
                3.70460304, 3.53847227, 3.55165969, 3.46097382, 3.63578788,
                3.49694587, 3.51018815, 3.6128836 , 3.51817865, 3.48449789,
                3.51651608, 3.5130942 , 3.51238947, 3.56367854, 3.576521  ,
                3.5396283 , 3.50907391, 3.60121882, 3.51841819, 3.60235501,
                3.57183799, 3.48859701, 3.4261631 , 3.69480319, 3.48052507,
                3.6196079 , 3.51211506, 3.41723619, 3.66106128, 3.54063465,
                3.51968591, 3.51480323, 3.47367356, 3.52540617, 3.58330729,
                3.56853036, 3.46915173, 3.58428319, 3.56883095, 3.51400697,
                3.58197672, 3.5732347 , 3.57032333, 3.5820461 , 3.55671323,
                3.53978034, 3.59289366, 3.56884069, 3.49859192, 3.54665398,
                3.62669112, 3.58083639, 3.49267123, 3.54201851, 3.46974523,
                3.62686928, 3.52342785, 3.47185377, 3.56278908, 3.50300373,
                3.44216226, 3.49031176, 3.5731773 , 3.49263031, 3.56788549,
                3.53758586, 3.69443526, 3.51003422, 3.57624159, 3.63328498,
                3.52176262, 3.52741305, 3.50474689, 3.55987565, 3.59371236,
                3.60237066, 3.70620579, 3.57107938, 3.5815039 , 3.56389733,
                3.58655178, 3.57688864, 3.52320004, 3.5671103 , 3.48609073,
                3.59276365, 3.51912756, 3.56206986, 3.47270847, 3.56013216,
                3.54088625, 3.52529047, 3.59665578, 3.53263878, 3.47602014,
                3.54923136, 3.58293527, 3.52640004, 3.6628148 , 3.59455786,
                3.56401991, 3.68584135, 3.61270037, 3.5272529 , 3.544049  ,
                3.61970337, 3.44031593, 3.60581192, 3.52862692, 3.5475291 ,
                3.45428062, 3.52285533, 3.5790118 , 3.50543716, 3.60722735,
                3.48720813, 3.55803355, 3.62990626, 3.58819454, 3.56426177,
                3.55914497, 3.55469179, 3.57908874, 3.56002077, 3.48199834,
                3.51780048, 3.51295809, 3.61231267, 3.48095765, 3.51099515,
                3.65972831, 3.55348944, 3.48708539, 3.65452174, 3.42484538,
                3.55935351, 3.46986325, 3.57740589, 3.57683318, 3.53876757,
                3.51674588, 3.56025968, 3.59342424, 3.52962561, 3.50260996,
                3.5224877 , 3.41902802, 3.5251579 , 3.55461271, 3.49566756,
                3.51845689, 3.57227109, 3.49266637, 3.49949288, 3.54277418,
                3.58131563, 3.54309249, 3.65903763, 3.63622931, 3.38869548,
                3.54151077, 3.62105561, 3.44712886, 3.4768337 , 3.69381325,
                3.50666359, 3.42493551, 3.65111192, 3.61987674, 3.42211709,
                3.57056585, 3.47670268, 3.48759844, 3.50414204, 3.59293721,
                3.64186131, 3.60608134, 3.57323038, 3.51282637, 3.5342849 ,
                3.43620601, 3.53774819, 3.48567275, 3.52391863, 3.61911399,
                3.50989226, 3.51456984, 3.48780673, 3.42611018, 3.54038563,
                3.47706617, 3.56391503, 3.57558134, 3.60101256, 3.56473762,
                3.55334188, 3.57164471, 3.5691913 , 3.54356825, 3.57502938,
                3.56520606, 3.63073228, 3.45997533, 3.71487911, 3.60597829,
                3.59597512, 3.47806699, 3.56000212, 3.74814591, 3.54600285,
                3.55699013, 3.47378385, 3.50927471, 3.47244769, 3.69371869,
                3.68840994, 3.67442007, 3.39041045, 3.62015376, 3.61709831,
                3.49964404, 3.60901986, 3.5263513 , 3.56464162, 3.46661316,
                3.60795847, 3.54740859, 3.59708735, 3.57765169, 3.47415648,
                3.53102064, 3.65994755, 3.56043129, 3.51957543, 3.57679587,
```

```
In [79]: y_test ## tested y

Out[79]: 529    3.58
         491    3.51
         459    3.48
         279    3.57
         655    3.44
                ...
         326    3.66
         440    3.76
         1387   3.51
         1323   3.43
         61     3.46
         Name: SalePrice, Length: 292, dtype: float64
```

```
In [80]: x_train_pred_LR=LR.predict(x_train) ##predicted x train

         x_train_pred_LR

Out[80]: array([3.66059537, 3.47663109, 3.54662469, ..., 3.47932418, 3.60831821,
                3.59878531])
```

```
In [81]: y_train  ## trained y

Out[81]: 618    3.66
         870    3.48
         92     3.55
         817    3.63
         302    3.59
                ...
         763    3.67
         835    3.50
         1216   3.48
         559    3.61
         684    3.60
         Name: SalePrice, Length: 1168, dtype: float64
```

```
In [82]: print('Linear Regression trainind score is',LR.score(x_train,y_train))

         Linear Regression trainind score is 0.9998270692514184
```

```
In [83]: print('Linear Regression testing score is',LR.score(x_test,y_test))

         Linear Regression testing score is 0.9997997133114862
```

Evaluation Metrics for Linear Regression

```
In [84]: from sklearn.metrics import r2_score
         train_score=r2_score(y_train,x_train_pred_LR)
         print('Linear Regression r2_score for training is',train_score)

         Linear Regression r2_score for training is 0.9998270692514184
```

```
In [85]: test_score=r2_score(y_test,x_test_pred_LR)
         print('Linear Regression r2_score for testing is',test_score)

         Linear Regression r2_score for testing is 0.9997997133114862
```

Training score is more than testing score so the model is underfitting.

- Model 2- Random Forest Regressor

  Random Forest Regressor is a machine learning algorithm that can be used to predict a continuous value, such as the price of a house. It is an ensemble method, which means that it builds a model by

training multiple decision trees and combining their predictions.

- In the context of a housing project, Random Forest Regressor might be used to predict the price of a house based on various features or characteristics of the house, such as its size, location, number of bedrooms and bathrooms, age, and so on.

- To build a Random Forest Regressor model for this purpose, you would need a dataset that includes the prices of a number of houses, along with their respective features. You could then use this data to train a Random Forest Regressor model that could be used to predict the price of a new house based on its features.

- Random Forest Regressor is often used in housing projects because it is a powerful and accurate method for making predictions based on a large number of input variables. It is also relatively resistant to overfitting, which means that it can generalize well to new data. Additionally, Random Forest Regressor models can be used to identify the most important features that influence the price of a house, which can be useful for understanding the underlying relationships in the data.

- Snapshot:

In [88]:
```
y_test   ## y test
```

Out[88]:
```
529    3.58
491    3.51
459    3.48
279    3.57
655    3.44
       ...
326    3.66
440    3.76
1387   3.51
1323   3.43
61     3.46
Name: SalePrice, Length: 292, dtype: float64
```

In [89]:
```
x_train_pred_RF=LR.predict(x_train)  ## predicted x train
x_train_pred_RF
```

Out[89]:
```
array([3.66059537, 3.47663109, 3.54662469, ..., 3.47932418, 3.60831821,
       3.59878531])
```

In [90]:
```
y_train  ## y train
```

Out[90]:
```
618    3.66
870    3.48
92     3.55
817    3.63
302    3.59
       ...
763    3.67
835    3.50
1216   3.48
559    3.61
684    3.60
Name: SalePrice, Length: 1168, dtype: float64
```

Random Foresr Regressor Score

In [91]:
```
print('Training score for Random Forest Regressor is',RF.score(x_train,y_train))
```

```
Training score for Random Forest Regressor is 0.9998863520405027
```

In [92]:
```
print('Testing score for Random Forest Regressor is',RF.score(x_test,y_test))
```

```
Testing score for Random Forest Regressor is 0.999059338163104
```

Testing score for Random Forest Regressor is 0.8734354150958835

Training score is more than Testing score,so the Random Forest Regressor model is underfitting.So we can do hyper parametric tuning.

In [93]:
```
from sklearn.model_selection import RandomizedSearchCV

random_grid = {'n_estimators': [100,200,300,400,500,600],  ## no. of trees
               'max_features': ['auto', 'sqrt'],
               'max_depth': [10, 15,20,25], ## maxinum number of levels in trees
               'min_samples_split':  [2, 5, 10], ## minimum number of samples required to split a node
               'min_samples_leaf': [1, 2, 4], ## Minimum number of samples required at each leaf node
               'bootstrap': [True, False]}  ##  Method of selecting samples for training each tree
```

In [94]:
```
hyper_tuning=RandomizedSearchCV(estimator=RF,param_distributions=random_grid,n_iter=10,cv=5,verbose=5,random_state=2)
```

In [95]:
```
hyper_tuning.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV 1/5] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=500;, sc
ore=0.955 total time=   1.2s
[CV 2/5] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=500;, sc
ore=0.899 total time=   1.2s
[CV 3/5] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=500;, sc
ore=0.910 total time=   1.2s
[CV 4/5] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=500;, sc
ore=0.933 total time=   1.2s
[CV 5/5] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_split=10, n_estimators=500;, sc
ore=0.941 total time=   1.3s
[CV 1/5] END bootstrap=True, max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=200;, sco
re=0.959 total time=   0.5s
[CV 2/5] END bootstrap=True, max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=200;, sco
re=0.908 total time=   0.5s
[CV 3/5] END bootstrap=True, max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=200;, sco
re=0.915 total time=   0.5s
[CV 4/5] END bootstrap=True, max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=200;, sco
re=0.937 total time=   0.6s
[CV 5/5] END bootstrap=True, max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=200;, sco
```

In [96]:
```
hyper_tuning.best_params_  ## getting the best parameters
```

Out[96]:
```
{'n_estimators': 200,
 'min_samples_split': 5,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 10,
 'bootstrap': False}
```

In [97]:
```
RF_hyper_tuning=RandomForestRegressor(n_estimators=600,min_samples_split=2,min_samples_leaf= 1,max_features='sqrt',max_depth=30,
```

In [98]:
```
RF_hyper_tuning.fit(x_train,y_train)
```

Out[98]:
```
RandomForestRegressor(bootstrap=False, max_depth=30, max_features='sqrt',
                      n_estimators=600)
```

Score after Random Forest Regressor Hyper Parametric Tuning

```
In [101]: print('Training score for Random Forest Regressor Hyper Parametric Tuning is ',RF_hyper_tuning.score(x_train,y_train))

          Training score for Random Forest Regressor Hyper Parametric Tuning is  1.0

In [102]: print('Testing score for Random Forest Regressor Hyper Parametric Tuning is ',RF_hyper_tuning.score(x_test,y_test))

          Testing score for Random Forest Regressor Hyper Parametric Tuning is  0.9515722883909419
```

- Model 3 – Decision Tree Regressor

  Decision tree regressor is a machine learning algorithm that can be used to predict a continuous value, such as the price of a house. It works by building a tree-like model that makes predictions based on the values of different features or characteristics of the data.

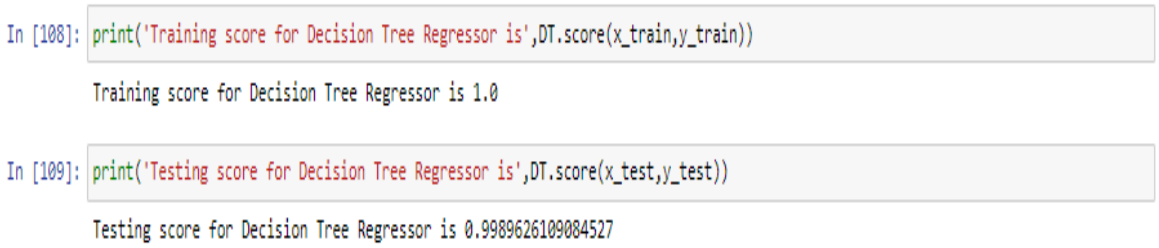
  In the context of a housing project, decision tree regressor might be used to predict the price of a house based on various features or characteristics of the house, such as its size, location, number of bedrooms and bathrooms, age, and so on.

  To build a decision tree regressor model for this purpose, you would need a dataset that includes the prices of a number of houses, along with their respective features. You could then use this data to train a decision tree regressor model that could be

used to predict the price of a new house based on its features.

Decision tree regressor is often used in housing projects because it is a simple and intuitive method for making predictions based on a large number of input variables. It is also relatively easy to interpret, which can be useful for understanding the factors that influence the price of a house. Additionally, decision tree regressor models can handle missing or incomplete data well, which can be a common problem in real-world datasets.

- Snapshot
- Without tuning



Training score is more than the Testing score,so the decision tree model is underfitting,So we can do hyper parametric tuning. ¶

Hyper parametric Tuning-- Decision Tree

```
In [110]: from sklearn.model_selection import RandomizedSearchCV

          parameters={"splitter":["best","random"],
                      "max_depth" : [2,4,6,8],
                     "min_samples_leaf":[1,2,3,4,5,],
                     "max_features":["auto","sqrt"],
                     "max_leaf_nodes":[5,10,15] }
```

```
In [111]: hyper_tuning_DT = RandomizedSearchCV(estimator=DT, param_distributions = parameters,
                                              cv = 2, n_iter = 10, n_jobs=-1)
```

```
In [112]: hyper_tuning_DT.fit(x_train, y_train)

Out[112]: RandomizedSearchCV(cv=2, estimator=DecisionTreeRegressor(), n_jobs=-1,
                             param_distributions={'max_depth': [2, 4, 6, 8],
                                                  'max_features': ['auto', 'sqrt'],
                                                  'max_leaf_nodes': [5, 10, 15],
                                                  'min_samples_leaf': [1, 2, 3, 4, 5],
                                                  'splitter': ['best', 'random']})
```

```
In [113]: hyper_tuning_DT.best_params_

Out[113]: {'splitter': 'best',
           'min_samples_leaf': 3,
           'max_leaf_nodes': 15,
           'max_features': 'auto',
           'max_depth': 6}
```

```
In [114]: hypertuning_DT = DecisionTreeRegressor(splitter= 'best',min_samples_leaf=5,max_leaf_nodes=15,max_features='sqrt',max_depth=6)

          hypertuning_DT

Out[114]: DecisionTreeRegressor(max_depth=6, max_features='sqrt', max_leaf_nodes=15,
                                min_samples_leaf=5)
```

```
In [115]: hypertuning_DT.fit(x_train,y_train)

Out[115]: DecisionTreeRegressor(max_depth=6, max_features='sqrt', max_leaf_nodes=15,
                                min_samples_leaf=5)
```

After Tuning

```
In [120]: print('Training score for hyper parametric tuning of Decision tree regressor is',hypertuning_DT.score(x_train,y_train))

          Training score for hyper parametric tuning of Decision tree regressor is 0.7226957358981887
```

```
In [121]: print('Testing score for hyper parametric tuning of Decision tree regressor is',hypertuning_DT.score(x_test,y_test))

          Testing score for hyper parametric tuning of Decision tree regressor is 0.6817640166246073
```

- Model 4- XG Boost Regressor
  XGBoost (eXtreme Gradient Boosting) is a machine learning algorithm that can be used to predict a continuous value, such as the price of a house. It is an implementation of gradient boosting, which is a

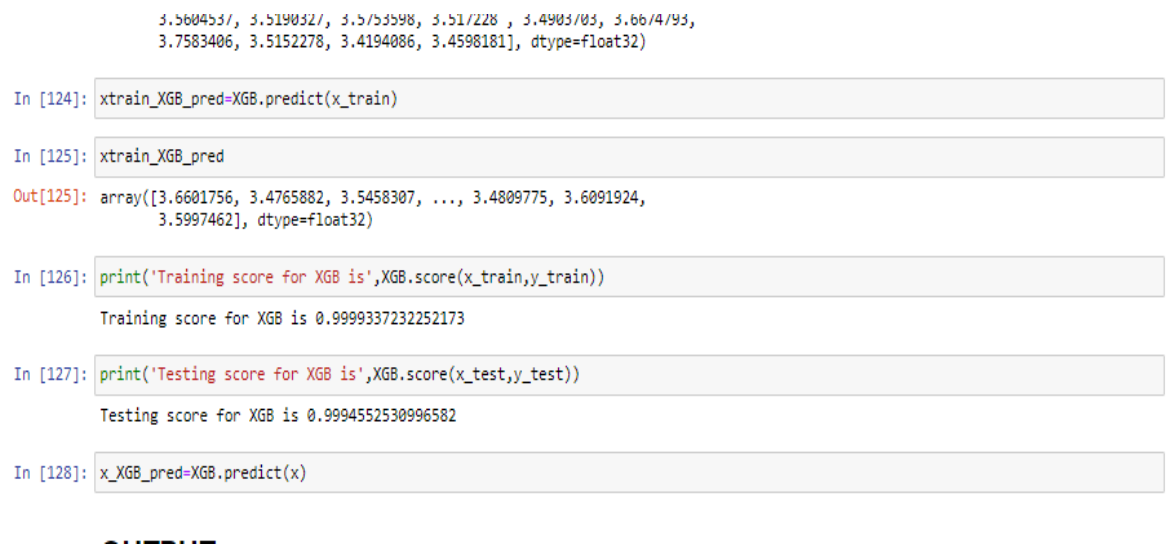
method that builds a model by training multiple weak models and combining their predictions. XGBoost is particularly effective at handling large datasets and has been widely used in a variety of applications, including housing projects.

- In the context of a housing project, XGBoost regressor might be used to predict the price of a house based on various features or characteristics of the house, such as its size, location, number of bedrooms and bathrooms, age, and so on.

- To build an XGBoost regressor model for this purpose, you would need a dataset that includes the prices of a number of houses, along with their respective features. You could then use this data to train an XGBoost regressor model that could be used to predict the price of a new house based on its features.

- XGBoost regressor is often used in housing projects because it is a powerful and accurate method for making predictions based on a large number of input variables. It is also relatively fast to train, which can be useful for large datasets. Additionally, XGBoost regressor models can handle missing or incomplete data well, and they provide a number of hyperparameters that can be tuned to improve their performance.

- Snapshot



```
                3.5604537, 3.5190327, 3.5753598, 3.517228 , 3.4903703, 3.6674793,
                3.7583406, 3.5152278, 3.4194086, 3.4598181], dtype=float32)

In [124]: xtrain_XGB_pred=XGB.predict(x_train)

In [125]: xtrain_XGB_pred

Out[125]: array([3.6601756, 3.4765882, 3.5458307, ..., 3.4809775, 3.6091924,
                3.5997462], dtype=float32)

In [126]: print('Training score for XGB is',XGB.score(x_train,y_train))

          Training score for XGB is 0.9999337232252173

In [127]: print('Testing score for XGB is',XGB.score(x_test,y_test))

          Testing score for XGB is 0.9994552530996582

In [128]: x_XGB_pred=XGB.predict(x)
```

Found that XGBoost Regressor has the best score as random forest had underfitted score

- Key Metrics for success in solving problem under consideration

R2 Score

The R2 score, also known as the coefficient of determination, is a measure of the goodness of fit of a machine learning model. It is used to evaluate the performance of a regression model, and it can range from 0 to 1, with a higher value indicating a better fit.

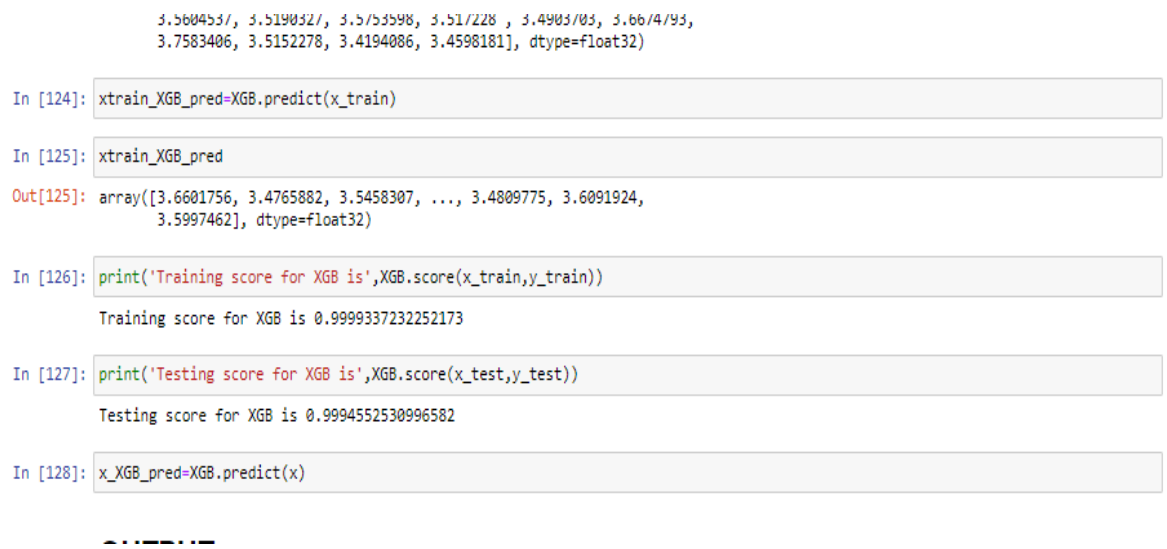
The R2 score is calculated by taking the sum of the squares of the differences between the predicted values and the actual values, and dividing it by the sum of the squares of the differences between the actual values and the mean of the actual values. This results in a value between 0 and 1, with 0 indicating that the model is not a good fit and 1 indicating a perfect fit.

For example, if a model has an R2 score of 0.8, this means that the model explains 80% of the variance in the data. On the other hand,

if a model has an R2 score of 0.2, this means that the model only explains 20% of the variance in the data, which may indicate that the model is not performing well.

In general, the R2 score is a useful metric for evaluating the performance of a machine learning model, especially when the goal is to predict a continuous target variable. However, it is important to keep in mind that the R2 score can be affected by the number of variables in the model, and it may not always be the most appropriate metric to use, depending on the specific characteristics of the data and the goals of the analysis.

Result : we see that we are able to achieve 99% accuracy with Xgboost Regressor model



```
        3.5604537, 3.5190327, 3.57535598, 3.51/228 , 3.4903/03, 3.66/4/95,
        3.7583406, 3.5152278, 3.4194086, 3.4598181], dtype=float32)

In [124]: xtrain_XGB_pred=XGB.predict(x_train)

In [125]: xtrain_XGB_pred

Out[125]: array([3.6601756, 3.4765882, 3.5458307, ..., 3.4809775, 3.6091924,
                 3.5997462], dtype=float32)

In [126]: print('Training score for XGB is',XGB.score(x_train,y_train))

          Training score for XGB is 0.9999337232252173

In [127]: print('Testing score for XGB is',XGB.score(x_test,y_test))

          Testing score for XGB is 0.9994552530996582

In [128]: x_XGB_pred=XGB.predict(x)
```

- Visualizations

Linear regression plots

Plots for X_pred

Out[78]:
```
array([3.58225096, 3.51073145, 3.47679707, 3.57476195, 3.43600779,
       3.43058714, 3.64094753, 3.5206022 , 3.79807657, 3.5305856 ,
       3.58903708, 3.5133536 , 3.60258812, 3.49633842, 3.49144597,
       3.52558075, 3.57358263, 3.49816668, 3.53052435, 3.53754812,
       3.55020587, 3.52578819, 3.47791423, 3.55743616, 3.56887349,
       3.55205181, 3.56134753, 3.42981331, 3.66393621, 3.49095835,
       3.47625511, 3.59310777, 3.53806694, 3.62039416, 3.68710741,
       3.55840556, 3.63922834, 3.48062436, 3.62837988, 3.66478797,
       3.61418271, 3.50589042, 3.54898038, 3.63996597, 3.69948546,
       3.48975852, 3.4984038 , 3.50357549, 3.55653948, 3.43066761,
       3.70460304, 3.53847227, 3.55165969, 3.46097382, 3.63578788,
       3.49694587, 3.51018815, 3.6128836 , 3.51817865, 3.48449789,
       3.51651608, 3.5130942 , 3.51238947, 3.56367854, 3.576521  ,
       3.5396283 , 3.50907391, 3.60121882, 3.51841819, 3.60235501,
       3.57183799, 3.48859701, 3.4261631 , 3.69480319, 3.48052507,
       3.6196079 , 3.51211506, 3.41723619, 3.66106128, 3.54063465,
       3.51968591, 3.51480323, 3.47367356, 3.52540617, 3.58330729,
       3.56853036, 3.46915173, 3.58428319, 3.56883095, 3.51400697,
       3.58197672, 3.5732347 , 3.57032333, 3.5820461 , 3.55671323,
       3.53978034, 3.59289366, 3.56884069, 3.49859192, 3.54665398,
       3.62669112, 3.58083639, 3.49267123, 3.54201851, 3.46974523,
       3.62686928, 3.52342785, 3.47185377, 3.56278908, 3.50300373,
       3.44216226, 3.49031176, 3.5731773 , 3.49263031, 3.56788549,
       3.53758586, 3.69443526, 3.51003422, 3.57624159, 3.63328498,
       3.52176262, 3.52741305, 3.50474689, 3.55987565, 3.59371236,
       3.60237066, 3.70620579, 3.57107938, 3.5815039 , 3.56389733,
       3.58655178, 3.57688864, 3.52320004, 3.5671103 , 3.48609073,
       3.59276365, 3.51912756, 3.56206986, 3.47270847, 3.56013216,
       3.54088625, 3.52529047, 3.59665578, 3.53263878, 3.47602014,
       3.54923136, 3.58293527, 3.52640004, 3.6628148 , 3.59455786,
       3.56401991, 3.68584135, 3.61270037, 3.5272529 , 3.544049  ,
       3.61970337, 3.44031593, 3.60581192, 3.52862692, 3.5475291 ,
       3.45428062, 3.52285533, 3.5790118 , 3.50543716, 3.60722735,
       3.48720813, 3.55803355, 3.62990626, 3.58819454, 3.56426177,
       3.55914497, 3.55469179, 3.57908874, 3.56002077, 3.48199834,
       3.51780048, 3.51295809, 3.61231267, 3.48095765, 3.51099515,
       3.65972831, 3.55348944, 3.48708539, 3.65452174, 3.42484538,
       3.55935351, 3.46986325, 3.57740589, 3.57683318, 3.53876757,
       3.51674588, 3.56025968, 3.59342424, 3.52962561, 3.50260996,
       3.5224877 , 3.41902802, 3.5251579 , 3.55461271, 3.49566756,
       3.51845689, 3.57227109, 3.49266637, 3.49949288, 3.54277418,
       3.58131563, 3.54309249, 3.65903763, 3.63622931, 3.38869548,
       3.54151077, 3.62105561, 3.44712886, 3.4768337 , 3.69381325,
       3.50666359, 3.42493551, 3.65111192, 3.61987674, 3.42211709,
       3.57056585, 3.47670268, 3.48759844, 3.50414204, 3.59293721,
       3.64186131, 3.60608134, 3.57323038, 3.51282637, 3.5342849 ,
       3.43620601, 3.53774819, 3.48567275, 3.52391863, 3.61911399,
       3.50989226, 3.51456984, 3.48780673, 3.42611018, 3.54038563,
       3.47706617, 3.56391503, 3.57558134, 3.60101256, 3.56473762,
       3.55334188, 3.57164471, 3.5691913 , 3.54356825, 3.57502938,
       3.56520606, 3.63073228, 3.45997533, 3.71487911, 3.60597829,
       3.59597512, 3.47806699, 3.56000212, 3.74814591, 3.54600285,
       3.55699013, 3.47378385, 3.50927471, 3.47244769, 3.69371869,
       3.68840994, 3.67442007, 3.39041045, 3.62015376, 3.61709831,
       3.49964404, 3.60901986, 3.5263513 , 3.56464162, 3.46661316,
       3.60795847, 3.54740859, 3.59708735, 3.57765169, 3.47415648,
       3.53102064, 3.65994755, 3.56043129, 3.51957543, 3.57679587,
```

```
In [79]: y_test ## tested y

Out[79]: 529    3.58
         491    3.51
         459    3.48
         279    3.57
         655    3.44
                ...
         326    3.66
         440    3.76
         1387   3.51
         1323   3.43
         61     3.46
         Name: SalePrice, Length: 292, dtype: float64

In [80]: x_train_pred_LR=LR.predict(x_train) ##predicted x train

         x_train_pred_LR

Out[80]: array([3.66059537, 3.47663109, 3.54662469, ..., 3.47932418, 3.60831821,
                3.59878531])

In [81]: y_train  ## trained y

Out[81]: 618    3.66
         870    3.48
         92     3.55
         817    3.63
         302    3.59
                ...
         763    3.67
         835    3.50
         1216   3.48
         559    3.61
         684    3.60
         Name: SalePrice, Length: 1168, dtype: float64

In [82]: print('Linear Regression trainind score is',LR.score(x_train,y_train))

         Linear Regression trainind score is 0.9998270692514184

In [83]: print('Linear Regression testing score is',LR.score(x_test,y_test))

         Linear Regression testing score is 0.9997997133114862
```

We see the model is underfitting but the points are pretty accurate from a holistic view.

Random Forest Regressor

Plots with X_pred

Out[87]:
```
array([3.58226245, 3.51038708, 3.47700858, 3.57465048, 3.43730284,
       3.43168874, 3.64230187, 3.52088946, 3.77692255, 3.53019077,
       3.58912043, 3.51550201, 3.60200024, 3.49653327, 3.49080948,
       3.52553199, 3.57267944, 3.49781417, 3.53076982, 3.53717377,
       3.54894428, 3.52491371, 3.47700695, 3.55733925, 3.56798929,
       3.55117808, 3.56100023, 3.43071084, 3.66224987, 3.49010154,
       3.47703821, 3.59249365, 3.53828091, 3.62016699, 3.6900491 ,
       3.55831147, 3.63850215, 3.48104578, 3.62905434, 3.66601932,
       3.61610342, 3.5063998 , 3.54804228, 3.63992975, 3.70306854,
       3.49076635, 3.49945957, 3.50372578, 3.55583411, 3.43122784,
       3.70598868, 3.53819613, 3.55116306, 3.46023262, 3.63661777,
       3.49635498, 3.50893248, 3.61307586, 3.51805   , 3.48471352,
       3.51625978, 3.51300101, 3.51246884, 3.56389265, 3.57607126,
       3.5394168 , 3.50902324, 3.60194803, 3.51808172, 3.60198806,
       3.57167435, 3.48945915, 3.42596618, 3.6981735 , 3.48020692,
       3.61927442, 3.51244767, 3.42039401, 3.66214466, 3.54048734,
       3.51936601, 3.51502104, 3.47297027, 3.52551387, 3.58168228,
       3.56806914, 3.46895661, 3.5837236 , 3.56888686, 3.51429885,
       3.58166599, 3.57267746, 3.57049655, 3.5816432 , 3.55581874,
       3.53951513, 3.59262199, 3.56807047, 3.49866847, 3.54596451,
       3.62703265, 3.58046222, 3.49227808, 3.54198255, 3.47042532,
       3.62699063, 3.52310254, 3.47177262, 3.56223643, 3.5022303 ,
       3.44170691, 3.49010417, 3.57265116, 3.4924133 , 3.56698185,
       3.53722575, 3.69334648, 3.51042212, 3.57624916, 3.63339434,
       3.52071436, 3.52693508, 3.50449267, 3.55941698, 3.59332706,
       3.59988088, 3.70862367, 3.57073598, 3.58164346, 3.56321917,
       3.58656177, 3.57656883, 3.52310735, 3.5662605 , 3.4867249 ,
       3.59283535, 3.51889403, 3.5623574 , 3.47394195, 3.55943669,
       3.54168552, 3.52554749, 3.59575808, 3.53260607, 3.47532893,
       3.54853933, 3.58240786, 3.52603418, 3.66213919, 3.59429753,
       3.56393587, 3.68865982, 3.61249366, 3.52690607, 3.54410918,
       3.62018252, 3.44041429, 3.60579886, 3.52790428, 3.54706464,
       3.45484774, 3.52192924, 3.57895577, 3.50491648, 3.60742427,
       3.48495028, 3.55835445, 3.63035142, 3.58779382, 3.56431168,
       3.55935195, 3.55443271, 3.57808062, 3.55940347, 3.48117452,
       3.51807761, 3.51299312, 3.6131817 , 3.48036824, 3.51148509,
       3.66103183, 3.55301157, 3.48672948, 3.65488254, 3.42749197,
       3.55833544, 3.47061883, 3.57654123, 3.57665348, 3.53815375,
       3.51697582, 3.560231  , 3.59318893, 3.5292073 , 3.50219118,
       3.52244438, 3.42002449, 3.52553853, 3.55450836, 3.49617543,
       3.51808059, 3.57165018, 3.49305429, 3.49807961, 3.5427959 ,
       3.58153073, 3.54272664, 3.65843445, 3.63660141, 3.38422457,
       3.54166488, 3.62057801, 3.44761167, 3.4761916 , 3.69144141,
       3.50489272, 3.42739029, 3.65287975, 3.6202492 , 3.42209842,
       3.57032858, 3.47707182, 3.48746991, 3.50451259, 3.59281389,
       3.64272692, 3.60581738, 3.57270734, 3.51300359, 3.53376372,
       3.43733535, 3.53723417, 3.48488774, 3.52430082, 3.61927111,
       3.50971055, 3.51430216, 3.48781042, 3.42590834, 3.54027317,
       3.47695601, 3.56430753, 3.57494481, 3.60091013, 3.56440867,
       3.55325678, 3.57042679, 3.56889195, 3.54280176, 3.57453537,
       3.56472259, 3.63092618, 3.46031327, 3.71939406, 3.60504309,
       3.59557849, 3.47773533, 3.55932918, 3.75998246, 3.54270805,
       3.55586851, 3.47377754, 3.50841717, 3.47114979, 3.69190374,
       3.68918259, 3.67586043, 3.38271643, 3.62022439, 3.61704453,
       3.49949175, 3.60850824, 3.52553799, 3.56418186, 3.46623487,
```

In [88]:
```
y_test   ## y test
```

Out[88]:
```
529    3.58
491    3.51
459    3.48
279    3.57
655    3.44
       ...
326    3.66
440    3.76
1387   3.51
1323   3.43
61     3.46
Name: SalePrice, Length: 292, dtype: float64
```

In [89]:
```
x_train_pred_RF=LR.predict(x_train)  ## predicted x train
x_train_pred_RF
```

Out[89]:
```
array([3.66059537, 3.47663109, 3.54662469, ..., 3.47932418, 3.60831821,
       3.59878531])
```

In [90]:
```
y_train  ## y train
```

Out[90]:
```
618    3.66
870    3.48
92     3.55
817    3.63
302    3.59
       ...
763    3.67
835    3.50
1216   3.48
559    3.61
684    3.60
Name: SalePrice, Length: 1168, dtype: float64
```

Random Foresr Regressor Score

In [91]:
```
print('Training score for Random Forest Regressor is',RF.score(x_train,y_train))
```

```
Training score for Random Forest Regressor is 0.9998863520405027
```

In [92]:
```
print('Testing score for Random Forest Regressor is',RF.score(x_test,y_test))
```

```
Testing score for Random Forest Regressor is 0.999059338163104
```

Testing score for Random Forest Regressor is 0.8734354150958835

Training score is more than Testing score so the Random Forest Regressor model is underfitting So we can do hyper parametric tuning

We see the model is underfitting as mentioned

After tuning:

Out[99]:
```
array([3.59391837, 3.51558073, 3.4806793 , 3.58120649, 3.44351968,
       3.45477763, 3.62390255, 3.50982377, 3.7459066 , 3.53366378,
       3.58771614, 3.52138147, 3.60458295, 3.50135228, 3.49553214,
       3.525647  , 3.58676673, 3.48186875, 3.5243325 , 3.53740154,
       3.51967155, 3.52570837, 3.46999534, 3.55393979, 3.57257713,
       3.56380395, 3.55847223, 3.42411573, 3.66293573, 3.48772329,
       3.49942358, 3.58775385, 3.52771134, 3.6369504 , 3.67381399,
       3.56191646, 3.6372357 , 3.48851272, 3.61696603, 3.65594589,
       3.60164479, 3.50730987, 3.55676957, 3.64859834, 3.67412935,
       3.50855536, 3.50096807, 3.50605906, 3.55356597, 3.45468504,
       3.68767201, 3.53226498, 3.55111085, 3.43946028, 3.62339243,
       3.4909323 , 3.50353032, 3.60863888, 3.51764988, 3.47636059,
       3.5225651 , 3.5121223 , 3.52013019, 3.54868635, 3.58389896,
       3.53586209, 3.50487034, 3.60074993, 3.51455222, 3.59040575,
       3.56631012, 3.4926801 , 3.43163791, 3.6189069 , 3.45378814,
       3.62241943, 3.51296936, 3.45747759, 3.64551541, 3.53699307,
       3.51502577, 3.5125274 , 3.48224126, 3.52666389, 3.57379611,
       3.57040403, 3.47041488, 3.58180907, 3.56575464, 3.51746529,
       3.57881028, 3.5700953 , 3.56767024, 3.58963291, 3.56922642,
       3.53599848, 3.59145917, 3.55685973, 3.4971623 , 3.53424362,
       3.62390015, 3.58192194, 3.49545323, 3.53700106, 3.4751741 ,
       3.63178948, 3.5298654 , 3.46688402, 3.56016063, 3.48671292,
       3.45242241, 3.49279503, 3.57340841, 3.49500036, 3.55544428,
       3.53486711, 3.70227512, 3.50776206, 3.5781508 , 3.62616051,
       3.50453508, 3.5396087 , 3.50926545, 3.56745224, 3.59760538,
       3.60895837, 3.6940766 , 3.57506492, 3.57920165, 3.55639986,
       3.5818526 , 3.58073554, 3.51415253, 3.56819329, 3.47946538,
       3.60221647, 3.51686608, 3.56458941, 3.47602354, 3.56528408,
       3.52973079, 3.52379848, 3.59671162, 3.52598867, 3.4913479 ,
       3.55678569, 3.59169008, 3.52419694, 3.64532726, 3.60144181,
       3.55426996, 3.68597088, 3.62054211, 3.52940379, 3.53766209,
       3.61992642, 3.48001115, 3.57530713, 3.48774012, 3.55018602,
       3.47977303, 3.52389116, 3.57607391, 3.49935222, 3.61246333,
       3.48295405, 3.55308324, 3.63995682, 3.582087  , 3.56084243,
       3.55409121, 3.5476725 , 3.5777516 , 3.556705  , 3.4851094 ,
       3.50084947, 3.51239545, 3.61018935, 3.48446849, 3.51788921,
       3.63234063, 3.56162547, 3.48258531, 3.65484581, 3.46689631,
       3.55863841, 3.46205637, 3.57759481, 3.57995249, 3.52621947,
       3.52279047, 3.55073749, 3.59124075, 3.52227433, 3.50063044,
       3.52225739, 3.43061921, 3.52150699, 3.54958266, 3.49881381,
       3.51406652, 3.57394623, 3.48607024, 3.49285442, 3.5184633 ,
       3.56679609, 3.53802284, 3.65841069, 3.6290212 , 3.38952939,
       3.53652195, 3.62276262, 3.44332199, 3.45897133, 3.6667069 ,
       3.54549308, 3.52792887, 3.63162889, 3.61831238, 3.43008063,
       3.57436535, 3.50917386, 3.49283422, 3.50050032, 3.58482924,
       3.63698216, 3.60809519, 3.58477969, 3.51402985, 3.54266293,
       3.44747106, 3.52553776, 3.47191331, 3.52288502, 3.60924547,
       3.50418664, 3.51367462, 3.49044672, 3.43918316, 3.54679621,
       3.47569004, 3.56682652, 3.57954673, 3.60051403, 3.55605285,
       3.54771203, 3.57484784, 3.56964688, 3.52468912, 3.57565507,
       3.56030113, 3.62674096, 3.44826149, 3.68890774, 3.60484419,
       3.60147115, 3.46395967, 3.57457106, 3.70212254, 3.60898172,
       3.56690983, 3.49084172, 3.5072132 , 3.46367688, 3.70254913,
       3.67547962, 3.65393612, 3.40259681, 3.59096387, 3.61768582,
       3.5026133 , 3.59295842, 3.51828541, 3.56127914, 3.4808223 ,
       3.60847916, 3.56906121, 3.57131148, 3.58501726, 3.47125111,
       3.51881815, 3.67682627, 3.54710862, 3.51605064, 3.57807007,
       3.40856655, 3.45774677])
```

In [100]:
```
RF_hyper_pred_train=RF_hyper_tuning.predict(x_train)  ## predicted hyper parametric tuning x train

RF_hyper_pred_train
```

Out[100]:
```
array([3.65995718, 3.47620275, 3.54651016, ..., 3.48018103, 3.60880129,
       3.5989052 ])
```

Score after Random Forest Regressor Hyper Parametric Tuning

In [101]:
```
print('Training score for Random Forest Regressor Hyper Parametric Tuning is ',RF_hyper_tuning.score(x_train,y_train))
```

```
Training score for Random Forest Regressor Hyper Parametric Tuning is  1.0
```

In [102]:
```
print('Testing score for Random Forest Regressor Hyper Parametric Tuning is ',RF_hyper_tuning.score(x_test,y_test))
```

```
Testing score for Random Forest Regressor Hyper Parametric Tuning is  0.9515722883909419
```

We see the test score has reduced to 95%

Decision tree regressor

X_pred plots

Out[104]:
```
array([3.58245295, 3.51039005, 3.47700581, 3.57449452, 3.43755724,
       3.43140396, 3.64190742, 3.52124766, 3.77697651, 3.53028646,
       3.58964119, 3.51558551, 3.60193526, 3.49666629, 3.49085334,
       3.52552538, 3.57267414, 3.49652334, 3.53110617, 3.53662334,
       3.54863144, 3.52552538, 3.47700581, 3.55736315, 3.56803594,
       3.55114258, 3.56034098, 3.43140396, 3.66172115, 3.49011267,
       3.47700581, 3.59169864, 3.53831148, 3.62023614, 3.68762393,
       3.55836156, 3.63855422, 3.48174689, 3.6322379 , 3.6654288 ,
       3.61603821, 3.5063872 , 3.54810359, 3.63978958, 3.70223064,
       3.49085334, 3.49950045, 3.50366565, 3.5557534 , 3.43140396,
       3.70186485, 3.53831148, 3.55124641, 3.46018513, 3.63668404,
       3.49652334, 3.50906614, 3.61318051, 3.51812521, 3.48483639,
       3.51622398, 3.51300751, 3.51226822, 3.56375046, 3.57629559,
       3.53935202, 3.50906614, 3.60193526, 3.51812521, 3.60193526,
       3.57175657, 3.48936878, 3.42502337, 3.69777486, 3.48018103,
       3.61907676, 3.51235689, 3.42041256, 3.66222768, 3.54053665,
       3.51938105, 3.51494465, 3.47295241, 3.52552538, 3.58170928,
       3.56803594, 3.47047367, 3.58374381, 3.56873979, 3.51365566,
       3.58062909, 3.57267414, 3.57037087, 3.58158689, 3.55585445,
       3.5394277 , 3.59169864, 3.56803594, 3.49879629, 3.54597569,
       3.62700884, 3.58150004, 3.49232516, 3.5421144 , 3.47047367,
       3.62700884, 3.5230941 , 3.4713039 , 3.56229762, 3.50228848,
       3.44055353, 3.49011267, 3.57267414, 3.49232516, 3.5670929 ,
       3.53718791, 3.69155946, 3.51039005, 3.57629559, 3.63320056,
       3.52062772, 3.52672816, 3.50435011, 3.55935415, 3.5933268 ,
       3.59968733, 3.70223064, 3.57083403, 3.58158689, 3.56317081,
       3.58671814, 3.57629559, 3.5230941 , 3.56614461, 3.48862165,
       3.59292123, 3.51925588, 3.56249205, 3.47295241, 3.55935415,
       3.54163843, 3.52552538, 3.59653692, 3.53232839, 3.47539591,
       3.54863144, 3.58245295, 3.52582687, 3.66276545, 3.59413505,
       3.56375046, 3.68877845, 3.61245847, 3.52672816, 3.54295124,
       3.62023614, 3.44154085, 3.60581752, 3.52792252, 3.54703232,
       3.45480059, 3.52248084, 3.57852038, 3.50503183, 3.60818647,
       3.48483639, 3.55836156, 3.6322379 , 3.58671814, 3.564232  ,
       3.55935415, 3.55433226, 3.57807775, 3.55935415, 3.48174689,
       3.51812521, 3.51300751, 3.61318051, 3.48018103, 3.51170379,
       3.66005925, 3.55330985, 3.48772078, 3.65563666, 3.42717651,
       3.55836156, 3.46880109, 3.57629559, 3.57629559, 3.53831148,
       3.51686008, 3.56034098, 3.5933268 , 3.52910858, 3.50228848,
       3.52201944, 3.42063493, 3.52552538, 3.55433226, 3.49523143,
       3.51812521, 3.57175657, 3.49305636, 3.49808923, 3.54273314,
       3.58150004, 3.54273314, 3.65730878, 3.63668404, 3.38902932,
       3.54163843, 3.62023614, 3.44829819, 3.47620275, 3.69155946,
       3.50503183, 3.42717651, 3.65281232, 3.62023614, 3.42502337,
       3.57037087, 3.47700581, 3.48636053, 3.50435011, 3.5933268 ,
       3.64223346, 3.60581752, 3.57267414, 3.51300751, 3.53377209,
       3.43755724, 3.53662334, 3.48483639, 3.52431407, 3.61884839,
       3.50972938, 3.51417242, 3.48862165, 3.42502337, 3.54042608,
       3.47700581, 3.56519099, 3.57449452, 3.60193526, 3.56375046,
       3.55320727, 3.57037087, 3.5685055 , 3.54382086, 3.57449452,
       3.56519099, 3.63000989, 3.46018513, 3.71729322, 3.60506328,
       3.59574008, 3.47780514, 3.55935415, 3.75730747, 3.54273314,
       3.55585445, 3.47377078, 3.50773191, 3.4713039 , 3.69155946,
       3.68996892, 3.67716412, 3.38769078, 3.62023614, 3.61674515,
       3.49950045, 3.60731598, 3.52552538, 3.564232  , 3.46574883,
       3.60731598, 3.5475741 , 3.5977252 , 3.57692148, 3.47377078,
```

In [105]:
```
y_test ## y test
```

Out[105]:
```
529    3.58
491    3.51
459    3.48
279    3.57
655    3.44
       ...
326    3.66
440    3.76
1387   3.51
1323   3.43
61     3.46
Name: SalePrice, Length: 292, dtype: float64
```

In [106]:
```
x_train_pred_DT=DT.predict(x_train)  ## predicted x train

x_train_pred_DT
```

Out[106]:
```
array([3.65995718, 3.47620275, 3.54651016, ..., 3.48018103, 3.60880129,
       3.5989052 ])
```

In [107]:
```
y_train ## y train
```

Out[107]:
```
618    3.66
870    3.48
92     3.55
817    3.63
302    3.59
       ...
763    3.67
835    3.50
1216   3.48
559    3.61
684    3.60
Name: SalePrice, Length: 1168, dtype: float64
```

In [108]:
```
print('Training score for Decision Tree Regressor is',DT.score(x_train,y_train))
```

```
Training score for Decision Tree Regressor is 1.0
```

In [109]:
```
print('Testing score for Decision Tree Regressor is',DT.score(x_test,y_test))
```

```
Testing score for Decision Tree Regressor is 0.9989626109084527
```

We see over fitting again so we check the points after hyperparameter tuning

X_pred test plots

```
array([3.57741715, 3.51451257, 3.51451257, 3.57741715, 3.50449037,
       3.50449037, 3.57282868, 3.51451257, 3.67424298, 3.57282868,
       3.57960382, 3.50449037, 3.59671668, 3.51451257, 3.51451257,
       3.50449037, 3.62248204, 3.49607795, 3.50449037, 3.51451257,
       3.47091543, 3.57960382, 3.50449037, 3.57960382, 3.57741715,
       3.54296813, 3.57282868, 3.43221779, 3.67424298, 3.47091543,
       3.50449037, 3.62248204, 3.54296813, 3.62248204, 3.67424298,
       3.57960382, 3.57960382, 3.50449037, 3.62248204, 3.67424298,
       3.57741715, 3.54296813, 3.57960382, 3.67424298, 3.67386839,
       3.50449037, 3.50449037, 3.50449037, 3.57741715, 3.50449037,
       3.67424298, 3.50449037, 3.57741715, 3.51451257, 3.62248204,
       3.50449037, 3.50449037, 3.59671668, 3.50449037, 3.43221779,
       3.50449037, 3.50449037, 3.51451257, 3.54296813, 3.62248204,
       3.54296813, 3.50449037, 3.57282868, 3.57741715, 3.59671668,
       3.57960382, 3.51451257, 3.47091543, 3.57960382, 3.50449037,
       3.62248204, 3.50449037, 3.50449037, 3.59671668, 3.50449037,
       3.50449037, 3.50449037, 3.50449037, 3.57282868, 3.51451257,
       3.57282868, 3.47091543, 3.57960382, 3.57282868, 3.50449037,
       3.57282868, 3.57741715, 3.57960382, 3.57282868, 3.57960382,
       3.51451257, 3.57282868, 3.57741715, 3.50449037, 3.57741715,
       3.57282868, 3.59671668, 3.51451257, 3.50449037, 3.43221779,
       3.62248204, 3.57282868, 3.47091543, 3.57960382, 3.50449037,
       3.51451257, 3.50449037, 3.57741715, 3.54296813, 3.57282868,
       3.57741715, 3.67424298, 3.50449037, 3.51451257, 3.62248204,
       3.49607795, 3.57282868, 3.50449037, 3.57741715, 3.57282868,
       3.57960382, 3.67424298, 3.57960382, 3.57282868, 3.57741715,
       3.57282868, 3.57282868, 3.50449037, 3.57741715, 3.50449037,
       3.67424298, 3.51451257, 3.57282868, 3.50449037, 3.57282868,
       3.51451257, 3.57282868, 3.57741715, 3.50449037, 3.50449037,
       3.57960382, 3.57282868, 3.50449037, 3.67424298, 3.57960382,
       3.57741715, 3.67424298, 3.57282868, 3.50449037, 3.51451257,
       3.57741715, 3.50449037, 3.54296813, 3.49607795, 3.57960382,
       3.50449037, 3.50449037, 3.57282868, 3.50449037, 3.57960382,
       3.50449037, 3.57741715, 3.62248204, 3.57960382, 3.59671668,
       3.54296813, 3.57741715, 3.57960382, 3.57960382, 3.47091543,
       3.50449037, 3.50449037, 3.57741715, 3.47091543, 3.50449037,
       3.54296813, 3.57741715, 3.47091543, 3.67424298, 3.47091543,
       3.54296813, 3.50449037, 3.57282868, 3.57960382, 3.57960382,
       3.57282868, 3.51451257, 3.57282868, 3.57282868, 3.50449037,
       3.57282868, 3.43221779, 3.50449037, 3.54296813, 3.50449037,
       3.50449037, 3.57960382, 3.51451257, 3.50449037, 3.50449037,
       3.50449037, 3.51451257, 3.67424298, 3.59671668, 3.47091543,
       3.57282868, 3.57282868, 3.47091543, 3.43221779, 3.62533785,
       3.62533785, 3.57741715, 3.57741715, 3.57282868, 3.43221779,
       3.57741715, 3.54296813, 3.50449037, 3.50449037, 3.57960382,
       3.62248204, 3.59671668, 3.57741715, 3.51451257, 3.57741715,
       3.47091543, 3.50449037, 3.51451257, 3.49607795, 3.62248204,
       3.51451257, 3.50449037, 3.50449037, 3.50449037, 3.57741715,
       3.47091543, 3.59671668, 3.59671668, 3.59671668, 3.57741715,
       3.57282868, 3.54296813, 3.57282868, 3.51451257, 3.57282868,
       3.54296813, 3.62248204, 3.47091543, 3.67424298, 3.57282868,
       3.59671668, 3.50449037, 3.57960382, 3.67386839, 3.67424298,
       3.57282868, 3.51451257, 3.47091543, 3.50449037, 3.67424298,
       3.67424298, 3.57741715, 3.43221779, 3.51451257, 3.57282868,
```

```
In [117]: y_test

Out[117]: 529    3.58
          491    3.51
          459    3.48
          279    3.57
          655    3.44
                 ...
          326    3.66
          440    3.76
          1387   3.51
          1323   3.43
          61     3.46
          Name: SalePrice, Length: 292, dtype: float64

In [118]: DT_hyper_pred_xtrain=hypertuning_DT.predict(x_train)

          DT_hyper_pred_xtrain

Out[118]: array([3.67424298, 3.50449037, 3.50449037, ..., 3.50449037, 3.57960382,
                 3.57282868])

In [119]: y_train

Out[119]: 618    3.66
          870    3.48
          92     3.55
          817    3.63
          302    3.59
                 ...
          763    3.67
          835    3.50
          1216   3.48
          559    3.61
          684    3.60
          Name: SalePrice, Length: 1168, dtype: float64

In [120]: print('Training score for hyper parametric tuning of Decision tree regressor is',hypertuning_DT.score(x_train,y_train))

          Training score for hyper parametric tuning of Decision tree regressor is 0.7226957358981887

In [121]: print('Testing score for hyper parametric tuning of Decision tree regressor is',hypertuning_DT.score(x_test,y_test))

          Testing score for hyper parametric tuning of Decision tree regressor is 0.6817640166246073
```

We see that the score has drastically reduced

So that is why XG boost was chosen as the best model for the project Scores :

X_pred plots

In [123]:
```
xtest_XGB_pred
```

Out[123]:
```
array([3.5804195, 3.5103054, 3.4764278, 3.5748086, 3.4349697, 3.4344463,
       3.643563 , 3.5222733, 3.8059235, 3.5314527, 3.5899923, 3.515817 ,
       3.6016371, 3.49926  , 3.491044 , 3.5255287, 3.5726929, 3.4991171,
       3.5315993, 3.5365124, 3.5473175, 3.5243623, 3.4761117, 3.5577767,
       3.568932 , 3.551433 , 3.5605288, 3.4326615, 3.658815 , 3.4897733,
       3.4766014, 3.591814 , 3.5388033, 3.6194303, 3.6911347, 3.5589206,
       3.6374202, 3.4811525, 3.6310985, 3.668719 , 3.615722 , 3.5071023,
       3.547313 , 3.6381884, 3.7007473, 3.4914076, 3.4996274, 3.5029616,
       3.5561726, 3.4345105, 3.7138174, 3.539015 , 3.5519536, 3.4583936,
       3.635957 , 3.4984612, 3.509807 , 3.6133192, 3.5179055, 3.4842527,
       3.5162833, 3.5121827, 3.5120094, 3.5627384, 3.575459 , 3.5390275,
       3.5093818, 3.6014178, 3.5179567, 3.601737 , 3.5711436, 3.4908044,
       3.4254634, 3.7006123, 3.4809659, 3.6184475, 3.5118768, 3.4192393,
       3.658722 , 3.5413747, 3.5187583, 3.5153887, 3.4737575, 3.525392 ,
       3.580147 , 3.5689344, 3.4674492, 3.584907 , 3.569075 , 3.5152574,
       3.580663 , 3.5724843, 3.5704672, 3.5806177, 3.5560067, 3.5387404,
       3.5915914, 3.568124 , 3.499072 , 3.5454433, 3.6273654, 3.581936 ,
       3.4920545, 3.5418465, 3.4705794, 3.627245 , 3.5235474, 3.4707537,
       3.5630283, 3.502452 , 3.4380844, 3.4896064, 3.572673 , 3.4921923,
       3.5664792, 3.536183 , 3.693582 , 3.5102835, 3.5753922, 3.634959 ,
       3.521974 , 3.5266805, 3.504948 , 3.5592124, 3.5941536, 3.601072 ,
       3.7160509, 3.5709665, 3.5803459, 3.5633414, 3.5872633, 3.575551 ,
       3.523715 , 3.5663626, 3.4851959, 3.5918458, 3.5184603, 3.5628765,
       3.4735782, 3.559192 , 3.5412524, 3.5255296, 3.5956237, 3.5332205,
       3.473253 , 3.5473769, 3.5806086, 3.5261173, 3.6597946, 3.593967 ,
       3.563212 , 3.69171  , 3.6125534, 3.5269964, 3.5441546, 3.6200974,
       3.4396114, 3.6070495, 3.5277495, 3.5467165, 3.4544554, 3.5225008,
       3.5786095, 3.5046809, 3.607601 , 3.4840162, 3.55866  , 3.630858 ,
       3.5879235, 3.5647502, 3.5594645, 3.5539935, 3.5777464, 3.5591571,
       3.479509 , 3.518231 , 3.5117917, 3.6128259, 3.4810297, 3.5112782,
       3.6593783, 3.55413  , 3.4852467, 3.655505 , 3.4242916, 3.5598526,
       3.4710917, 3.5754883, 3.5754364, 3.5384135, 3.51696  , 3.560347 ,
       3.5942638, 3.527822 , 3.5026805, 3.522861 , 3.4142292, 3.5253623,
       3.5539832, 3.4941607, 3.5178561, 3.571234 , 3.492503 , 3.4991686,
       3.5423772, 3.581958 , 3.5420501, 3.658355 , 3.6364079, 3.3886101,
       3.541293 , 3.6197445, 3.448696 , 3.4757469, 3.688561 , 3.5050821,
       3.426125 , 3.6521063, 3.6193795, 3.4190342, 3.5707612, 3.476498 ,
       3.4881763, 3.5049398, 3.5914528, 3.6434488, 3.6069832, 3.5730648,
       3.5121827, 3.5343082, 3.43897  , 3.5363169, 3.4843442, 3.5245595,
       3.6181283, 3.5100524, 3.5154092, 3.487923 , 3.4228644, 3.5409641,
       3.476589 , 3.5646846, 3.5750382, 3.6007679, 3.5644994, 3.55352  ,
       3.570517 , 3.5687783, 3.54238  , 3.5750892, 3.5665903, 3.6317613,
       3.458817 , 3.7143033, 3.6031985, 3.595088 , 3.4767466, 3.559463 ,
       3.7575817, 3.542195 , 3.5560927, 3.47424  , 3.509104 , 3.4712346,
       3.6868088, 3.6906593, 3.6768086, 3.3903909, 3.6193397, 3.6167872,
       3.499829 , 3.6090796, 3.5254736, 3.564325 , 3.4690983, 3.6087685,
       3.5472298, 3.5967846, 3.5765817, 3.4738598, 3.5310915, 3.6589677,
       3.5604537, 3.5190327, 3.5753598, 3.517228 , 3.4903703, 3.6674793,
       3.7583406, 3.5152278, 3.4194086, 3.4598181], dtype=float32)
```

In [124]:
```
xtrain_XGB_pred=XGB.predict(x_train)
```

In [125]:
```
xtrain_XGB_pred
```

Out[125]:
```
array([3.6601756, 3.4765882, 3.5458307, ..., 3.4809775, 3.6091924,
       3.5997462], dtype=float32)
```

In [126]:
```
print('Training score for XGB is',XGB.score(x_train,y_train))
```

```
Training score for XGB is 0.9999337232252173
```

In [127]:
```
print('Testing score for XGB is',XGB.score(x_test,y_test))
```

```
Testing score for XGB is 0.9994552530996582
```

In [128]:
```
x_XGB_pred=XGB.predict(x)
```

- Interpretation of the Results

From the data set we see that as we have specifically replaced the null values in many of the columns with the right replacements and we have treated the dataset for outliers therefore , we were able to get a good accuracy score , we can actually used other metrics like mean squared error as well and get 0.11 to .14% which is very good , but the accuracy score being 99% + shows that the model is very good and can be used for predictions for house prices , we saw that few features were influencing the price way more than others and having such a huge no of features and achieving this score is a good sign that data science is really going to make a positive impact in the business decisions taken by management on the housing project

CONCLUSION

- Key Findings and Conclusions of the Study

Describe

From the EDA and graphs we saw:

1. Some categories seem to more diverse with respect to SalePrice than others. Neighborhood has big impact on house prices. Most expensive seems to be Partial SaleCondition. Having pool on property seems to improve price substantially. There are also differences in variabilities between category values.
2. In correlation matrix . OverallQual is main criterion in establishing house price. Neighborhood has big influence, partially it has some intrisinc value in itself, but also houses in certain regions tend to share same characteristics (confunding) what causes similar valuations.
3. Feature to feature - Garages seem to be built same year as houses, basements have generally same area as first floor which is pretty obvious. Garage area is strongly correlated with number of cars. Neighborhood is correlated with lots of other variables and this confirms the idea that houses in same region share same characteristics. Dwelling type is negatively correlated with kitchen above grade square feet.
4. Expensive houses have pools, better overall qual and condition, open porch and increased importance of MasVnrArea
5. GrLivArea' and 'TotalBsmtSF' seem to be linearly related with 'SalePrice'. Both relationships are positive, which means that as one variable increases, the other also increases. In the case of 'TotalBsmtSF', we can see that the slope of the linear relationship is particularly high.

6. 'OverallQual' and 'YearBuilt' also seem to be related with 'SalePrice'. The relationship seems to be stronger in the case of 'OverallQual', where the box plot shows how sales prices increase with the overall quality.
7. OverallQual', 'GrLivArea' and 'TotalBsmtSF' are strongly correlated with 'SalePrice'. Check!
8. 'GarageCars' and 'GarageArea' are also some of the most strongly correlated variables. However, as we discussed in the last sub-point, the number of cars that fit into the garage is a consequence of the garage area. 'GarageCars' and 'GarageArea' are like twin brothers. You'll never be able to distinguish them. Therefore, we just need one of these variables in our analysis (we can keep 'GarageCars' since its correlation with 'SalePrice' is higher).
9. 'TotalBsmtSF' and '1stFloor' also seem to be twin brothers. We can keep 'TotalBsmtSF' just to say that our first guess was right (re-read 'So... What can we expect?').'TotRmsAbvGrd' and 'GrLivArea', twin brothers again. 'YearBuilt'... It seems that 'YearBuilt' is slightly correlated with 'SalePrice'.

After model building we see that the model was able to understand the key features and make a really accurate prediction- especially the XGBoost Regressor model . We also see that the people with features like central ac , pool and higher built up area are actually 1-3% as only fewer people can afford expensive homes, we also see that the model works better when the outliers of these expensive homes were transformed which means that in real time , we do not need help n prediction for such expensive homes and we can use other ways to predict them. Overall a really good initiative by the Sunrise housing company, and having such a good model is also key

- Learning Outcomes of the Study in respect of Data Science

As mentioned in the previous part we have many features which have the most influence on the Target and having such a lot of features it was really difficult to see prior to model building to come up with a good score or a good model , but since in Machine learning the computation is not statistical alone and the model actually learns the data as a whole , we are able to see such good scores. We see that in most situations we can go with Xgboost regressor as the model is not having overfitting or underfitting , it's a strong model as well. If we are to use a substitute we can go for Linear regression as well as we saw it has a similar score to Xgboost but the training was lesser , We seed to tune it by using SGB Regressor which is the tuneable substitute for Linear Regression so we can work around the fitting problem , But overall has a better and faster processing time .

- Limitations of this work and Scope for Future Work

The houses which vary greatly in price compared to the actual prices – the expensive ones are not actually outliers so we need a way where we can use the best of both this model and a model to predict those prices as its easy to figure out if the house is greatly higher than the average price based on features. So we can have two models working in the front end to use the model created for maximum effect but when the buyer or speculator shows the need for such and such features which can contribute to purchase or inquiry of an expensive home , the UI should switch over to a different model or approach to predict the price.

Another observation we made is that this model only has features and classes for homes built in the past and with the ever-changing technology and house building methods we need to upgrade the model to train on that and then make a better one .

Also in case the property is going to be used for tourism or commercial use, we need to factor in many many more features such as the cost of maintenance , the surrounding as in locality if its profitable for the business or buyer, and so on

Finally the nulls in some columns were too many that we were not able to use them , but we can safely say that going forward when the model leans more and more data we can use some models which do not need treatment of nulls to be able to make an accurate prediction.