
# HOUSE PRICE PREDICTION

Submitted by:

MUSSAVEER SHARIFF H

# INTRODUCTION

- ## Business Problem Framing

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

  A US-based housing company named Surprise Housing has decided to enter the Australian market. The company uses data analytics to purchase houses at a price below their actual values and flip them at a higher price. For the same purpose, the company has collected a data set from the sale of houses in Australia. The data is provided in the CSV file below.

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

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

  • How do these variables describe the price of the houses

# Analytical Problem Framing

## Models used to solve this project

## Models tried in this project:

1. KNeighborsRegressor (KNR)
2. Decision tree Regressor (DTR)
3. Support vector Regressor (SVR)
4. Random Forest Regressor (RFR)
5. Gradient Boosting Regressor (GBR)
6. AdaBoost Regressor (ABR)
7. XGB Regressor (XGBR)
8. Artificial neural network(ANN)

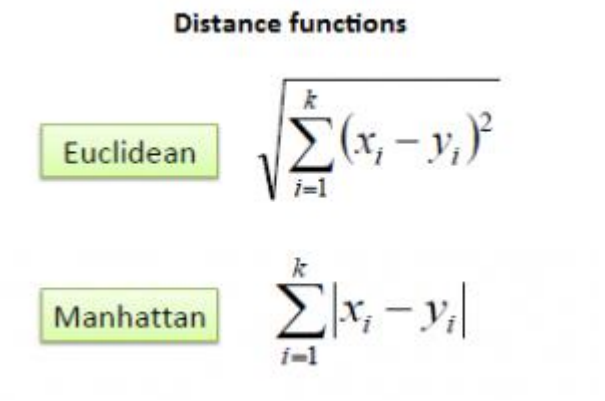
## KNEIGHBORS REGRESSOR:

As we saw above, KNN algorithm can be used for both classification and regression problems. The KNN algorithm uses 'feature similarity' to predict the values of any new data points. This means that the new point is assigned a value based on how closely it resembles the points in the training set

**Methods for calculating distances between points**

The first step is to calculate the distance between the new point and each training point. There are various methods for calculating this distance, of which the most commonly known methods are – Euclidian, Manhattan (for continuous) and Hamming distance (for categorical).

1. Euclidean Distance: Euclidean distance is calculated as the square root of the sum of the squared differences between a new point (x) and an existing point (y).
2. Manhattan Distance: This is the distance between real vectors using thesum of their absolute difference.




3. Hamming Distance: It is used for categorical variables. If the value (x) and the value (y) are the same, the distance D will be equal to 0. Otherwise, D=1.

$$D_H = \sum_{i=1}^{k}|x_i - y_i|$$

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

Once the distance of a new observation from the points in our training set has been measured, the next step is to pick the closest points. The number of points to be considered is defined by the value of k.

## DECISION TREE REGRESSOR

A **decision tree** is an efficient algorithm for describing a way to traverse a dataset while also defining a tree-like path to the expected outcomes. This branching in a tree is based on control statements or values, and the data points lie on either side of the **splitting node,** depending on the value of a specific feature.

**Algorithm: Generate_decision_tree.** Generate a decision tree from the training tuples of data partition *D*.

**Input:**

- Data partition, *D*, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split point* or *splitting subset*.

**Output:** A decision tree.

**Method:**

```
(1)  create a node N;
(2)  if tuples in D are all of the same class, C then
(3)      return N as a leaf node labeled with the class C;
(4)  if attribute_list is empty then
(5)      return N as a leaf node labeled with the majority class in D; // majority voting
(6)  apply Attribute_selection_method(D, attribute_list) to find the "best" splitting_criterion;
(7)  label node N with splitting_criterion;
(8)  if splitting_attribute is discrete-valued and
         multiway splits allowed then // not restricted to binary trees
(9)      attribute_list ← attribute_list − splitting_attribute; // remove splitting_attribute
(10) for each outcome j of splitting_criterion
     // partition the tuples and grow subtrees for each partition
(11)     let Dj be the set of data tuples in D satisfying outcome j; // a partition
(12)     if Dj is empty then
(13)         attach a leaf labeled with the majority class in D to node N;
(14)     else attach the node returned by Generate_decision_tree(Dj, attribute_list) to node N;
     endfor
(15) return N;
```

---

## How do we know where to split?

An **attribute selection measure** is a heuristic for selecting the splitting criterion that's most capable of deciding how to partition data in such a way that would result in individual classes. Attribute selection measures are also known as splitting rules because they define how the data points are to be split on a certain level in a decision tree. The attribute that has the best score for the measure is chosen as the splitting attribute for the given data points.

***Here are some major attribute selection measures:***

9. **Information Gain:** This attribute provides us the splitting attribute in terms of the amount of information required to further describe the tree. Information gain minimizes the information needed to classify the data points into respective partitions and reflects the least randomness or "impurity" in these partitions.

$$Info(D) = -\sum_{i}^{m} p_i \log_2(p_i),$$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j).$$

pi is the probability that an arbitrary tuple in dataset D belongs to class Ci and is estimated by|Ci,D|/|D|. **Info(D) is simply the mean amount of information needed to identify the class/category of a data point in D**. A log function to base 2 is used, since in most cases, information is encoded in bits.

***Info(D) is also known as the entropy of the dataset D.***

The information gain can be calculated as follows:

$$Gain(A) = Info(D) - Info_A(D).$$

**2. Gain Ratio:** The information gain measure is biased toward tests with many outcomes. Thus, it prefers selecting attributes that have a large number of values. Gain ratio is an attempt to improve this problem.

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right).$$

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)}.$$

**3. Gini Index:** The Gini index is calculated in the following manner:

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2,$$

where pi is the probability that a tuple in D belongs to class Ci and is estimated by |Ci,D|/|D|. The sum is computed over m classes.

The Gini index considers a binary split for each attribute. For example, if age has three possible values, namely {low,medium,high},then the possible subsets are{low,medium,high},{low,medium},{low, high}, {medium, high}, {low}, {medium}, {high}, and {} (ignoring the power and super set in our calculations).

$$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2).$$

$$\Delta Gini(A) = Gini(D) - Gini_A(D).$$

# SUPPORT VECTOR REGRESSOR

Suppose we have a set of training data where $x_n$ is a multivariate set of $N$ observations with observed response values $y_n$.

To find the linear function

$f(x)=x'\beta+b$,

and ensure that it is as flat as possible, find $f(x)$ with the minimal norm value ($\beta'\beta$). This is formulated as a convex optimization problem to minimize

$J(\beta)=_{12}\beta'\beta$

subject to all residuals having a value less than ε; or, in equation form:

$\forall n: y_n-(x_n'\beta+b)\leq\varepsilon$ .

It is possible that no such function $f(x)$ exists to satisfy these constraints for all points. To deal with otherwise infeasible constraints, introduce slack variables $\xi_n$ and $\xi^*_n$ for each point. This approach is similar to the "soft margin" concept in SVM classification, because the slack variables allow regression errors to exist up to the value of $\xi_n$ and $\xi^*_n$, yet still satisfy the required conditions.

Including slack variables leads to the objective function, also known as the primal formula:

$J(\beta)=_{12}\beta'\beta+C_N\sum_{n=1}(\xi_n+\xi_{*n})$ ,

subject to:

$\forall n: yn-(xn'\beta+b)\leq\varepsilon+\xi n \forall n: (xn'\beta+b)-yn\leq\varepsilon+\xi*n \forall n: \xi*n\geq 0 \forall n: \xi n\geq 0$ .

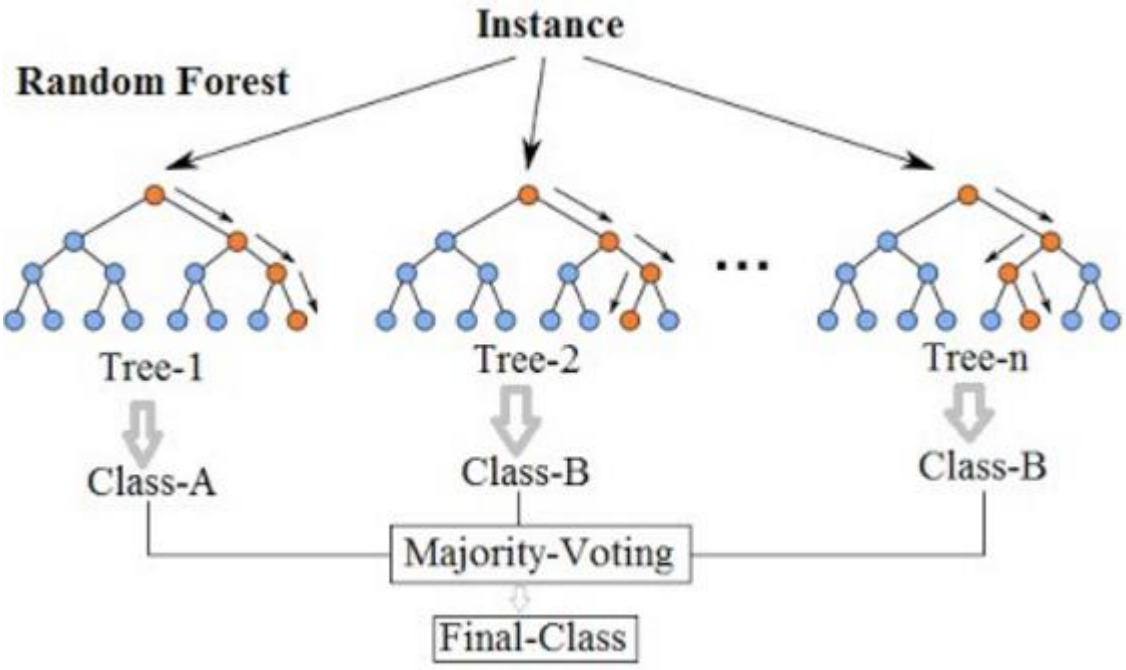
The constant $C$ is the box constraint, a positive numeric value that controls the penalty imposed on observations that lie outside the epsilon margin ($\varepsilon$) and helps to prevent overfitting (regularization). This value determines the trade-off between the flatness of $f(x)$ and the amount up to which deviations larger than $\varepsilon$ are tolerated.

The linear ε-insensitive loss function ignores errors that are within $\varepsilon$ distance of the observed value by treating them as equal to zero. The loss is measured based on the distance between observed value $y$ and the $\varepsilon$ boundary. This is formally described by

$L_\varepsilon = \begin{cases} 0 & \text{if } |y - f(x)| < \varepsilon \\ |y - f(x)| & \text{otherwise} \end{cases}$

## RANDOM FOREST REGRESSOR

Random-forest does both row sampling and column sampling with Decision tree as a base. Model h1, h2, h3, h4 are more different than by doing only bagging because of column sampling.




As you increase the number of base learners (k), the variance will decrease. When you decrease k, variance increases. But bias remains constant for the whole process. k can be found using cross-validation.

Random forest= DT(base learner)+ bagging(Row sampling with replacement)+ feature bagging(column sampling) + aggregation(mean/median, majority vote)

Here we want our base learner as low bias and high variance. so train DT to full depth length. we are not worried about depth, we let them grow because at end variance decrease in aggregation.

For model h1, (D-D') dataset not used in modeling are out of bag dataset. they used for cross-validation for the h1 model.

*Let's look at the steps taken to implement a Random forest:*

**1**. Suppose there are N observations and M features in the training data set. First, a sample from the training data set is taken randomly with replacement.

**2**. A subset of M features is selected randomly and whichever feature gives the best split is used to split the node iteratively.

**3**. The tree is grown to the largest.

**4**. The above steps are repeated and prediction is given based on the aggregation of predictions from n number of trees.

## Train and run-time complexity

Training time = O(log(nd)*k)

Run time = O(depth*k)

Space = O(store each DT*K)

As the number of base model increases, training run time increases so always use Cross-validation to find optimal hyperparameter.

## BOOSTING TECHNIQUES:

***Boosting*** *is another ensemble technique to create a collection of predictors. In this technique, learners are learned sequentially with early learners fitting simple models to the data and then analyzing data for errors. In other words, we fit consecutive trees (random sample) and at every step, the goal is to solve for net error from the prior tree.*

This works in case of high bias and low variance base model and additively combine. We will try to reduce bias.

When an input is misclassified by a hypothesis, its weight is increased so that the next hypothesis is more likely to classify it correctly. By combining the whole set at the end converts weak learners into a better performing model.

# ADABOOST:

## The Mathematics Behind AdaBoost

Here comes the hair-tugging part. Let's break AdaBoost down, step-by-step and equation-by-equation so that it's easier to comprehend.

Let's start by considering a dataset with N points, or rows, in our dataset.

$$x_i \in \mathbb{R}^n, y_i \in \{-1, 1\}$$

In this case,

- *n* is the dimension of real numbers, or the number of attributes in our dataset
- *x* is the set of data points
- *y* is the target variable which is either -1 or 1 as it is a binary classification problem, denoting the first or the second class (e.g. Fit vs Not Fit)

We calculate the weighted samples for each data point. AdaBoost assigns weight to each training example to determine its significance in the training dataset. When the assigned weights are high, that set of training data points are likely to have a larger say in the training set. Similarly, when the assigned weights are low, they have a minimal influence in the training dataset.

Initially, all the data points will have the same weighted sample *w*:

$$w = 1/N \in [0, 1]$$

where N is the total number of data points.

The weighted samples always sum to 1, so the value of each individual weight will always lie between 0 and 1. After this, we calculate the actual influence for this classifier in classifying the data points using the formula:

$$\alpha_t = \frac{1}{2} ln \frac{(1 - TotalError)}{TotalError}$$

*Alpha* is how much influence this stump will have in the final classification. *Total Error* is nothing but the total number of misclassifications for that training set divided by the training set size. We can plot a graph for *Alpha* by plugging in various values of *Total Error* ranging from 0 to 1.

Notice that when a Decision Stump does well, or has no misclassifications (a perfect stump!) this results in an error rate of 0 and a relatively large, positive alpha value.

If the stump just classifies half correctly and half incorrectly (an error rate of 0.5, no better than random guessing!) then the alpha value will be 0. Finally, when the stump ceaselessly gives misclassified results (just do the opposite of what the stump says!) then the alpha would be a large negative value.

After plugging in the actual values of *Total Error* for each stump, it's time for us to update the sample weights which we had initially taken as *1/N* for every data point. We'll do this using the following formula:

$$w_i = w_{i-1} * e^{\pm \alpha}$$

In other words, the new sample weight will be equal to the old sample weight multiplied by Euler's number, raised to plus or minus alpha (which we just calculated in the previous step).

The two cases for alpha (positive or negative) indicate:

- Alpha is positive when the predicted and the actual output agree (the sample was classified correctly). In this case we decrease the sample weight from what it was before, since we're already performing well.
- Alpha is negative when the predicted output does not agree with the actual class (i.e. the sample is misclassified). In this case we need to increase the sample weight so that the same misclassification does not repeat in the next stump. This is how the stumps are dependent on their predecessors.

## GRADIENTBOOSTING

Many implementations of Gradient Boosting follow approach 1 to minimize the objective function. At every iteration, we fit a base learner to the negative

gradient of the loss function and multiply our prediction with a constant and add it to the value from previous iteration.

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations $M$.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For $m$ = 1 to $M$:
    1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

    2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
    3. Compute multiplier $\gamma_m$ by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

    4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

The intuition is by fitting a base learner to the negative gradient at each iteration is essentially performing gradient descent on the loss function. The negative gradients are often called as pseudo residuals, as they indirectly help us to minimize the objective function.

## XGBOOSTREGRESSOR

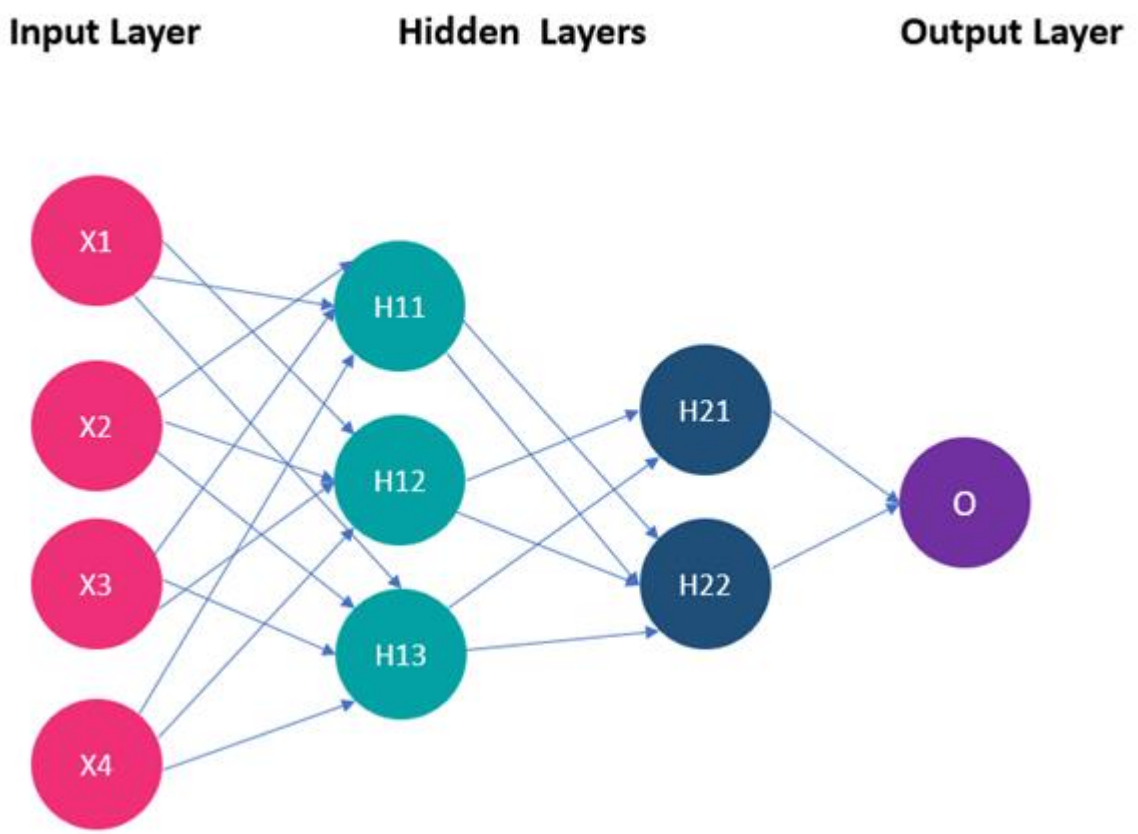
XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree based algorithms are considered best-in-class right now. Please see the chart below for the evolution of tree-based algorithms over the year

XGBoost algorithm was developed as a research project at the University of Washington. Tianqi Chen and Carlos Guestrin presented their paper at SIGKDD Conference in 2016 and caught the Machine Learning world by fire. Since its introduction, this algorithm has not only been credited with winning numerous Kaggle competitions but also for being the driving force under the hood for several cutting-edge industry applications. As a result, there is a strong community of data scientists contributing to the XGBoost open source projects with ~350 contributors and ~3,600 commits on GitHub. The algorithm differentiates itself in the following ways:

1. A wide range of applications: Can be used to solve regression, classification, ranking, and user-defined prediction problems.
2. Portability: Runs smoothly on Windows, Linux, and OS X.
3. Languages: Supports all major programming languages including C++, Python, R, Java, Scala, and Julia.
4. Cloud Integration: Supports AWS, Azure, and Yarn clusters and works well with Flink, Spark, and other ecosystems.

## ARTIFICIAL NEURAL NETWORK (ANN)

> Just like the brain consists of billions of highly connected neurons, a basic operating unit in a neural network is a neuron-like node. It takes input from other nodes and sends output to others. — Fei-Fei Li




# Working of Neural Network

A neural network works based on two principles

1. Forward Propagation
2. Backward Propagation

Let’s understand these building blocks with the help of an example. Here I am considering a single input layer, hidden layer, output layer to make the understanding clear.

**Forward Propagation**

- Considering we have data and would like to apply binary classification to get the desired output.

- Take a sample having features as X1, X2, and these features will be operated over a set of processes to predict the outcome.
- Each feature is associated with a weight, where X1, X2 as features and W1, W2 as weights. These are served as input to a neuron.
- *A neuron performs both functions. a) Summation b)Activation.*
- In the summation, all features are multiplied by their weights and bias are summed up. (Y=W1X1+W2X2+b).
- This summed function is applied over an Activation function. The output from this neuron is multiplied with the weight W3 and supplied as input to the output layer.
- 


  , but we vary the activation functions in hidden layer neurons, not in the output layer.

*We just randomly initialized the weights and continued the process. There are many techniques for initializing the weights. But, you may be having a doubt how these weights are getting updated right??? This will be answered using back propagation.*

**Backward Propagation**

Let us get back to our calculus basics and we will be using chain rule learned in our school days to update the weights.

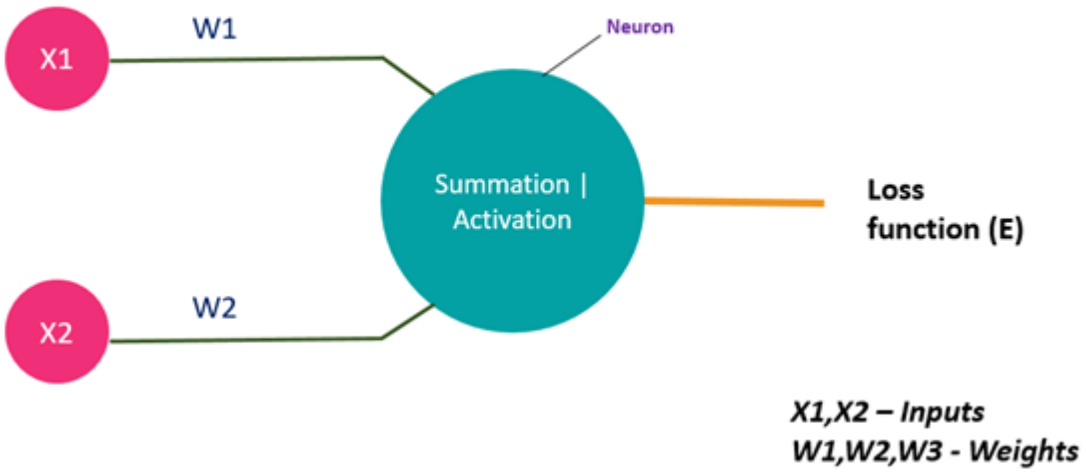
**Chain Rule**

The **chain rule** provides us a technique for finding the derivative of composite functions, with the number of functions that make up the composition determining how many differentiation steps are necessary. For example, if a composite function $f(x)$ is defined as

$$f(x) = (g \circ h)(x) = g[h(x)]$$
$$\text{then } f'(x) = g'[h(x)] \cdot h'(x)$$

Chain rule

Let us apply the chain rule to a single neuron,




Chain Rule

In neural networks, our main goal will be on reducing the error, to make it possible we have to update all the weights by doing backpropagation. We need to find a change in weights such that error should be minimum. To do so we calculate dE/dW1 and dE/dW2.

Considering S (Summation) = x1W1 + x2W2

A (Activation) = sigmoid = $e^x$ / $(1 + e^x)$

***Using Chain Rule***

$$\frac{dE}{dW1} = \frac{dE}{dA} \times \frac{dA}{dS} \times \frac{dS}{dW1}$$

$$\frac{dE}{dW2} = \frac{dE}{dA} \times \frac{dA}{dS} \times \frac{dS}{dW2}$$

Change in Weights


Backward Propagation

Once you have calculated changes in weights concerning error our next step will be on updating the weights using gradient descent procedure. Please check more details on gradient descent here.

$$W1new = W1old - \eta \frac{dE}{dW1}$$

$$W2new = W2old - \eta \frac{dE}{dW2}$$

New weights

Forward propagation and backward propagation will be continuous for all samples until the error reaches minimum value.

## Data Sources and their formats

- The data sources were provided by flip-robo technologies
- Two data sets: train & test
- Format of data is `coma separated values (csv)`
- The data has 1460 entries
- 81 features

The data is:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 1158 | 1159 | 1160 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Id** | 127 | 889 | 793 | 110 | 422 | 1197 | 561 | 1041 | 503 | 576 | ... | 673 | 943 | 551 | |
| **MSSubClass** | 120 | 20 | 60 | 20 | 20 | 60 | 20 | 20 | 20 | 50 | ... | 20 | 90 | 120 | |
| **MSZoning** | RL | RL | RL | RL | RL | RL | RL | RL | RL | RL | ... | RL | RL | RL | |
| **LotFrontage** | NaN | 95 | 92 | 105 | NaN | 58 | NaN | 88 | 70 | 80 | ... | NaN | 42 | 53 | |
| **LotArea** | 4928 | 15865 | 9920 | 11751 | 16635 | 14054 | 11341 | 13125 | 9170 | 8480 | ... | 11250 | 7711 | 4043 | |
| **Street** | Pave | Pave | Pave | Pave | Pave | Pave | Pave | Pave | Pave | Pave | ... | Pave | Pave | Pave | |
| **Alley** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | |
| **LotShape** | IR1 | IR1 | IR1 | IR1 | IR1 | IR1 | IR1 | Reg | Reg | Reg | ... | IR1 | IR1 | Reg | |
| **LandContour** | Lvl | Lvl | Lvl | Lvl | Lvl | Lvl | Lvl | Lvl | Lvl | Lvl | ... | Lvl | Lvl | Lvl | |
| **Utilities** | AllPub | AllPub | AllPub | AllPub | AllPub | AllPub | AllPub | AllPub | AllPub | AllPub | ... | AllPub | AllPub | AllPub | |
| **LotConfig** | Inside | Inside | CulDSac | Inside | FR2 | Inside | Inside | Corner | Corner | Inside | ... | Inside | Inside | Inside | |
| **LandSlope** | Gtl | Mod | Gtl | Gtl | Gtl | Gtl | Gtl | Gtl | Gtl | Gtl | ... | Gtl | Gtl | Gtl | |
| **Neighborhood** | NPkVill | NAmes | NoRidge | NWAmes | NWAmes | Gilbert | Sawyer | Sawyer | Edwards | NAmes | ... | Veenker | Edwards | NPkVill | |
| **Condition1** | Norm | Norm | Norm | Norm | Norm | Norm | Norm | Norm | Feedr | Norm | ... | Norm | Norm | Norm | |
| **Condition2** | Norm | Norm | Norm | Norm | Norm | Norm | Norm | Norm | Norm | Norm | ... | Norm | Norm | Norm | |
| **BldgType** | TwnhsE | 1Fam | 1Fam | 1Fam | 1Fam | 1Fam | 1Fam | 1Fam | 1Fam | 1Fam | ... | 1Fam | Duplex | TwnhsE | |
| **HouseStyle** | 1Story | 1Story | 2Story | 1Story | 1Story | 2Story | 1Story | 1Story | 1Story | 1.5Fin | ... | 1Story | 1Story | 1Story | |
| **OverallQual** | 6 | 8 | 7 | 6 | 6 | 7 | 5 | 5 | 5 | 5 | ... | 6 | 4 | 6 | |
| **OverallCond** | 5 | 6 | 5 | 6 | 7 | 5 | 6 | 4 | 7 | 5 | ... | 6 | 3 | 6 | |
| **YearBuilt** | 1976 | 1970 | 1996 | 1977 | 1977 | 2006 | 1957 | 1957 | 1965 | 1947 | ... | 1977 | 1977 | 1977 | |
| **YearRemodAdd** | 1976 | 1970 | 1997 | 1977 | 2000 | 2006 | 1996 | 2000 | 1965 | 1950 | ... | 1977 | 1977 | 1977 | |
| **RoofStyle** | Gable | Flat | Gable | Hip | Gable | Gable | Hip | Gable | Hip | Gable | ... | Gable | Gable | Gable | |
| **RoofMatl** | CompShg | Tar&Grv | CompShg | CompShg | CompShg | CompShg | CompShg | CompShg | CompShg | CompShg | ... | CompShg | CompShg | CompShg | C |
| **Exterior1st** | Plywood | Wd Sdng | MetalSd | Plywood | CemntBd | VinylSd | Wd Sdng | Wd Sdng | MetalSd | MetalSd | ... | Plywood | MetalSd | Plywood | |
| **Exterior2nd** | Plywood | Wd Sdng | MetalSd | Plywood | CmentBd | VinylSd | Wd Sdng | Wd Sdng | MetalSd | MetalSd | ... | Plywood | MetalSd | Plywood | |
| **MasVnrType** | None | None | None | BrkFace | Stone | None | BrkFace | BrkCmn | None | None | ... | None | None | None | |
| **MasVnrArea** | 0 | 0 | 0 | 480 | 126 | 0 | 180 | 67 | 0 | 0 | ... | 0 | 0 | 0 | |
| **ExterQual** | TA | Gd | Gd | TA | Gd | Gd | TA | TA | TA | TA | ... | Gd | TA | TA | |
| **ExterCond** | TA | Gd | TA | TA | TA | TA | TA | TA | TA | TA | ... | TA | TA | TA | |
| **Foundation** | CBlock | PConc | PConc | CBlock | CBlock | PConc | CBlock | CBlock | CBlock | CBlock | ... | CBlock | PConc | CBlock | |
| **BsmtQual** | Gd | TA | Gd | Gd | Gd | Gd | Gd | TA | TA | TA | ... | Gd | Gd | Gd | |
| **BsmtCond** | TA | Gd | TA | TA | TA | TA | TA | TA | TA | TA | ... | TA | TA | TA | |
| **BsmtExposure** | No | Gd | Av | No | No | Av | No | No | No | No | ... | No | Gd | No | |
| **BsmtFinType1** | ALQ | ALQ | GLQ | BLQ | ALQ | Unf | ALQ | Rec | ALQ | Rec | ... | ALQ | GLQ | ALQ | |
| **BsmtFinSF1** | 120 | 351 | 862 | 705 | 1246 | 0 | 1302 | 168 | 698 | 442 | ... | 767 | 1440 | 559 | |
| **BsmtFinType2** | Unf | Rec | Unf | Unf | Unf | Unf | Unf | BLQ | GLQ | Unf | ... | Unf | Unf | Unf | |
| **BsmtFinSF2** | 0 | 823 | 0 | 0 | 0 | 0 | 0 | 682 | 96 | 0 | ... | 0 | 0 | 0 | |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LowQualFinSF | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 |
| GrLivArea | 958 | 2217 | 2013 | 1844 | 1602 | 1863 | 1392 | 1803 | 1214 | 1216 | ... | 1208 | 1440 | 1069 |
| BsmtFullBath | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | ... | 1 | 2 | 0 |
| BsmtHalfBath | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 |
| FullBath | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 1 | 1 | ... | 1 | 2 | 2 |
| HalfBath | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | ... | 1 | 0 | 0 |
| BedroomAbvGr | 2 | 4 | 3 | 3 | 3 | 4 | 3 | 3 | 2 | 2 | ... | 3 | 4 | 2 |
| KitchenAbvGr | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | ... | 1 | 2 | 1 |
| KitchenQual | TA | Gd | TA | TA | Gd | Gd | TA | TA | TA | TA | ... | TA | TA | TA |
| TotRmsAbvGrd | 5 | 8 | 8 | 7 | 8 | 9 | 5 | 8 | 6 | 6 | ... | 6 | 8 | 4 |
| Functional | Typ | Typ | Typ | Typ | Typ | Typ | Mod | Maj1 | Typ | Typ | ... | Typ | Typ | Typ |
| Fireplaces | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | ... | 1 | 0 | 0 |
| FireplaceQu | TA | TA | TA | TA | TA | Gd | Gd | TA | NaN | NaN | ... | TA | NaN | NaN |
| GarageType | Attchd | Attchd | Attchd | Attchd | Attchd | BuiltIn | Detchd | Attchd | Detchd | Detchd | ... | Attchd | NaN | Attchd |
| GarageYrBlt | 1977 | 1970 | 1997 | 1977 | 1977 | 2006 | 1957 | 1957 | 1965 | 1947 | ... | 1977 | NaN | 1977 |
| GarageFinish | RFn | Unf | Unf | RFn | Fin | Fin | Unf | RFn | Unf | Unf | ... | RFn | NaN | RFn |
| GarageCars | 2 | 2 | 2 | 2 | 2 | 3 | 2 | 2 | 2 | 1 | ... | 2 | 0 | 2 |
| GarageArea | 440 | 621 | 455 | 546 | 529 | 660 | 528 | 484 | 461 | 336 | ... | 546 | 0 | 440 |
| GarageQual | TA | TA | TA | TA | TA | TA | TA | TA | Fa | TA | ... | TA | NaN | TA |
| GarageCond | TA | TA | TA | TA | TA | TA | TA | TA | Fa | TA | ... | TA | NaN | TA |
| PavedDrive | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | ... | Y | N | Y |
| WoodDeckSF | 0 | 81 | 180 | 0 | 240 | 100 | 0 | 0 | 0 | 158 | ... | 198 | 321 | 0 |
| OpenPorchSF | 205 | 207 | 130 | 122 | 0 | 17 | 0 | 0 | 0 | 0 | ... | 42 | 0 | 55 |
| EnclosedPorch | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 184 | 102 | ... | 0 | 0 | 0 |
| 3SsnPorch | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 |
| ScreenPorch | 0 | 224 | 0 | 0 | 0 | 0 | 95 | 0 | 0 | 0 | ... | 0 | 0 | 200 |
| PoolArea | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 |
| PoolQC | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN |
| Fence | NaN | NaN | NaN | MnPrv | NaN | NaN | NaN | GdPrv | GdPrv | NaN | ... | NaN | NaN | NaN |
| MiscFeature | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | Shed | NaN | ... | NaN | NaN | NaN |
| MiscVal | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 400 | 0 | ... | 0 | 0 | 0 |
| MoSold | 2 | 10 | 6 | 1 | 6 | 11 | 5 | 1 | 4 | 10 | ... | 6 | 8 | 10 |
| YrSold | 2007 | 2007 | 2007 | 2010 | 2009 | 2006 | 2010 | 2006 | 2007 | 2008 | ... | 2006 | 2007 | 2008 |
| SaleType | WD | WD | WD | COD | WD | New | WD | WD | WD | COD | ... | WD | Oth | COD |
| SaleCondition | Normal | Normal | Normal | Normal | Normal | Partial | Normal | Normal | Normal | Abnorml | ... | Normal | Abnorml | Abnorml |
| SalePrice | 128000 | 268000 | 269790 | 190000 | 215000 | 219210 | 121500 | 155000 | 140000 | 118500 | ... | 165000 | 150000 | 140000 |

81 rows × 1168 columns

# Data dictionary:

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

GdGood

TA Average/Typical

Fa Fair

Po Poor

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

Ex Excellent

GdGood

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)

GdGood (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

GdGood

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

AvAverage Exposure (split levels or foyers typically score average or above)

Mn Mimimum Exposure

NoNo 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

GdGood

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

GdGood

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

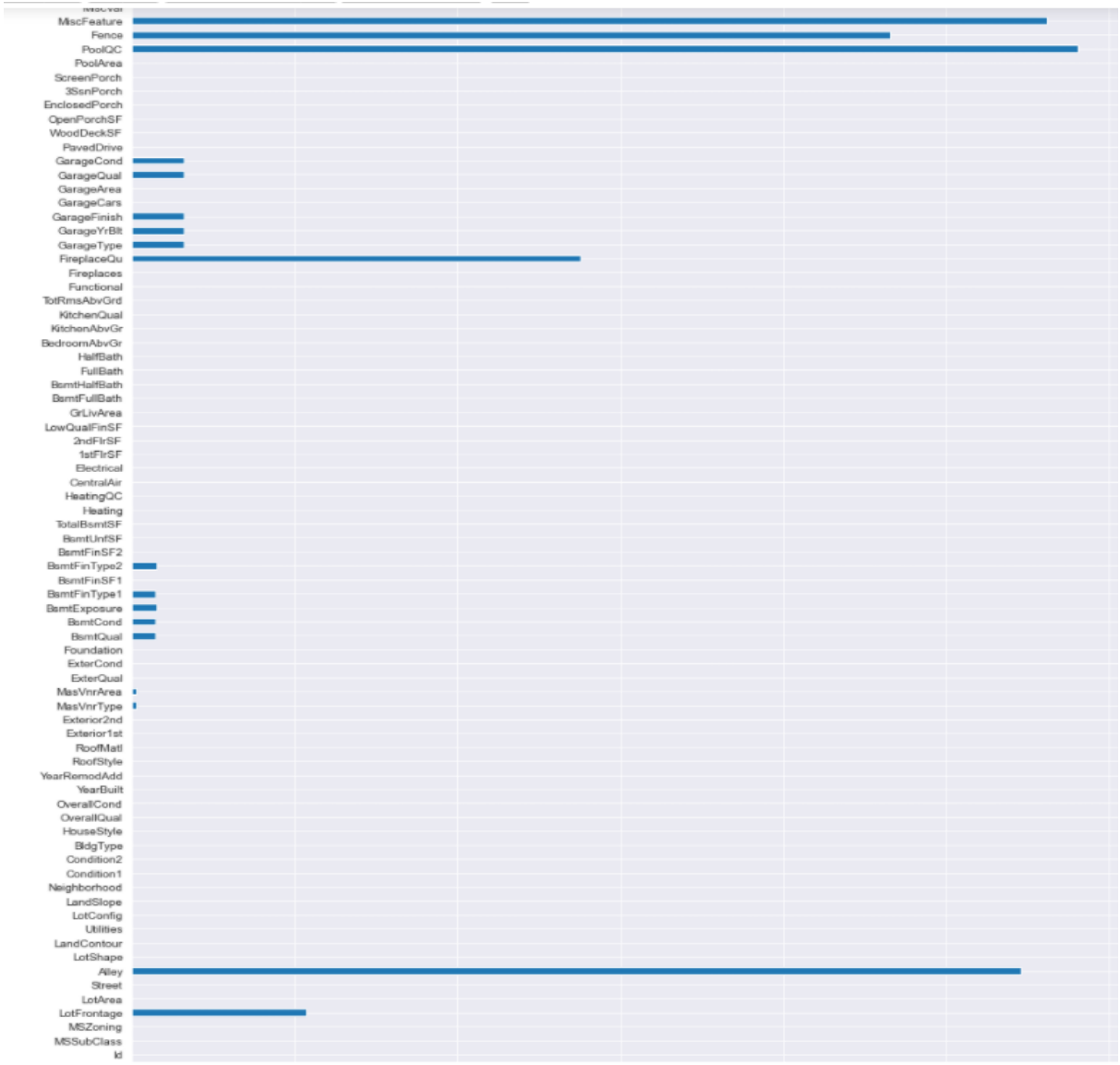
Alloca Allocation - two linked properties with separate deeds, typically condo with a garage unit

Family Sale between family members

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

# Data Preprocessing

- The First step was to check if any missing values by visualizing it



**Assumption:** from this graph we can see that some features has more number of missing values considered dropping the features with more than 40% of missing values.

```
def drop_features(data):
    """
    It drops the features which contains more than `40%` of missing values
    """
    percentage = dict(round(data.isna().sum()/len(data)*100,2))
    for key, values in percentage.items():
        if values > 40:
            print(f'The {key} has {values} % of missing values.')
            data = data.drop(key, axis = 1)
    return data
```

```
df_new = drop_features(df)
```

```
The Alley has 93.41 % of missing values.
The FireplaceQu has 47.17 % of missing values.
The PoolQC has 99.4 % of missing values.
The Fence has 79.71 % of missing values.
The MiscFeature has 96.23 % of missing values.
```

The features like : ALLEY, FIREPLACEQU,POOLQC,FENCE,MISCFEATURE were removed from the data as there were more than 40% of missing data.

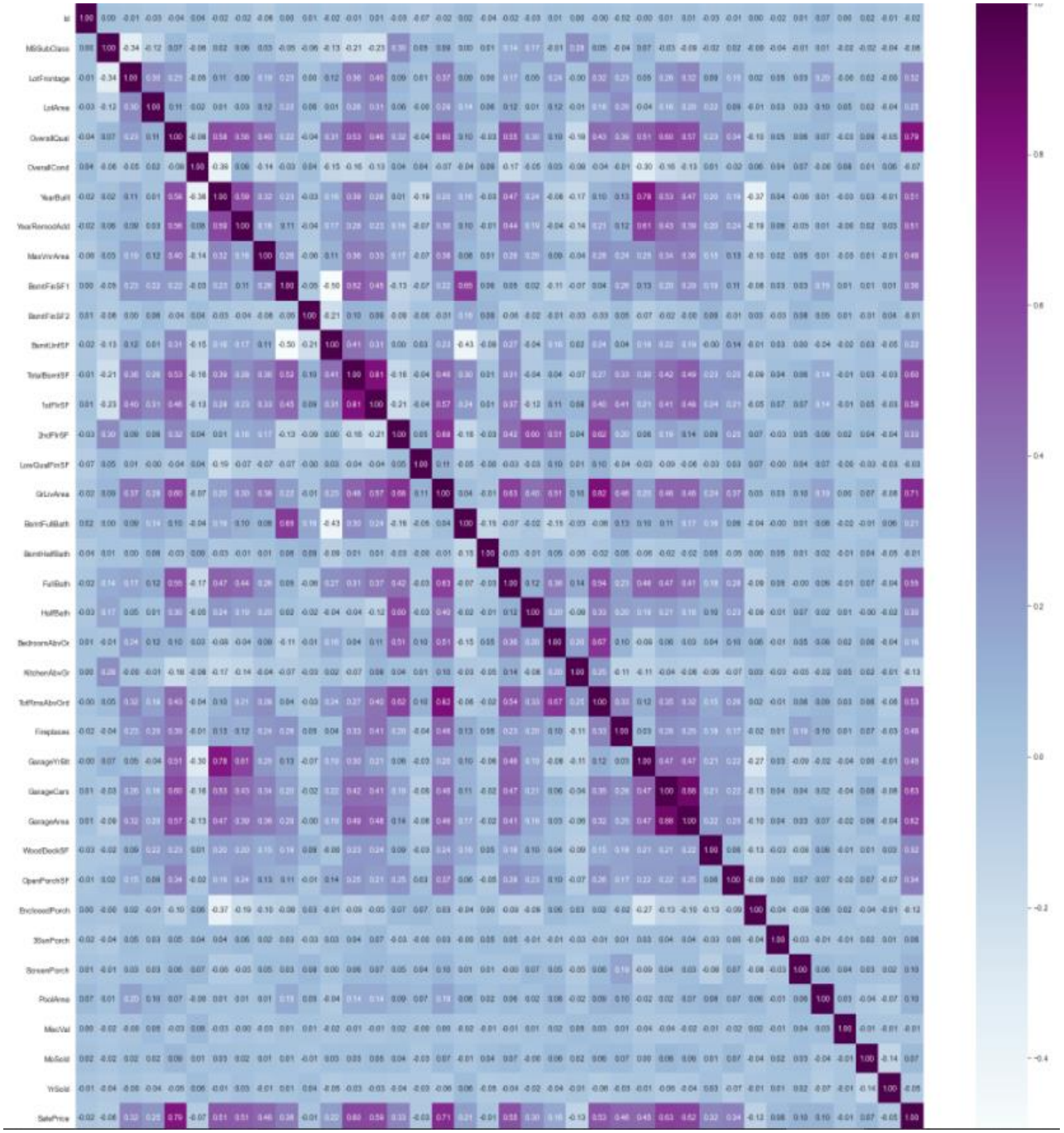
Still we got the missing values in columns less than 40% next filled them with the simple Imputer library

```
def fill_values(data):
    """
    Fills all the missing values using simple imputer,Categorical values with the 'most frequent' values,Numerical values with th
    """

    for labels, content in data.items():
         # Filling the missing values of object dtype(categorical) with 'most frequent'
        if pd.api.types.is_object_dtype(content):
            if pd.isnull(content).any():
                Imputer = SimpleImputer(strategy = 'most_frequent')
                data[labels] = Imputer.fit_transform(data[labels].values.reshape(-1,1))[:,0]

        # Filling the missing values of the numerical dtype
        else:
            if pd.isnull(content).any():
                Imputer1= SimpleImputer(strategy = 'median')
                data[labels] = Imputer1.fit_transform(data[labels].values.reshape(-1,1))[:,0]
    return data
```

After filling the missing data with simple imputer librabry

**Feature selection:**

Used correlation method for feature selection



Removed the features which are correlated more than 90%

**Encoding:**

As the entire data should be in the numerical format for giving in model

The label Encoder method was applied on data to convert the categorical data types to numerical data type.

```
le = LabelEncoder()

#creating function for converting the features into numerical
def convert_features(data):
    """
    Converts the categorical features into numerical features
    """
    for cols in data.columns:
        if not pd.api.types.is_numeric_dtype(data[cols]):
            data[cols] = le.fit_transform(data[cols])
    return data
```

# The data after cleaning:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 1158 | 1159 | 1160 | 11( |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MSSubClass | 120.0 | 20.0 | 60.0 | 20.0 | 20.0 | 60.0 | 20.0 | 20.0 | 20.0 | 50.0 | ... | 20.0 | 90.0 | 120.0 | 60 |
| MSZoning | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | ... | 3.0 | 3.0 | 3.0 | 3 |
| LotFrontage | 70.0 | 95.0 | 92.0 | 105.0 | 70.0 | 58.0 | 70.0 | 88.0 | 70.0 | 80.0 | ... | 70.0 | 42.0 | 53.0 | 70 |
| LotArea | 4928.0 | 15865.0 | 9920.0 | 11751.0 | 16635.0 | 14054.0 | 11341.0 | 13125.0 | 9170.0 | 8480.0 | ... | 11250.0 | 7711.0 | 4043.0 | 10762 |
| Street | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | ... | 1.0 | 1.0 | 1.0 | 1 |
| LotShape | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 3.0 | 3.0 | ... | 0.0 | 0.0 | 3.0 | 0 |
| LandContour | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | ... | 3.0 | 3.0 | 3.0 | 3 |
| Utilities | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0 |
| LotConfig | 4.0 | 4.0 | 1.0 | 4.0 | 2.0 | 4.0 | 4.0 | 0.0 | 0.0 | 4.0 | ... | 4.0 | 4.0 | 4.0 | 1 |
| LandSlope | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0 |
| Neighborhood | 13.0 | 12.0 | 15.0 | 14.0 | 14.0 | 8.0 | 19.0 | 19.0 | 7.0 | 12.0 | ... | 24.0 | 7.0 | 13.0 | 8 |
| Condition1 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 1.0 | 2.0 | ... | 2.0 | 2.0 | 2.0 | 2 |
| Condition2 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | ... | 2.0 | 2.0 | 2.0 | 2 |
| BldgType | 4.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 2.0 | 4.0 | 0 |
| HouseStyle | 2.0 | 2.0 | 5.0 | 2.0 | 2.0 | 5.0 | 2.0 | 2.0 | 2.0 | 0.0 | ... | 2.0 | 2.0 | 2.0 | 5 |
| OverallQual | 6.0 | 8.0 | 7.0 | 6.0 | 6.0 | 7.0 | 5.0 | 5.0 | 5.0 | 5.0 | ... | 6.0 | 4.0 | 6.0 | 7 |
| OverallCond | 5.0 | 6.0 | 5.0 | 6.0 | 7.0 | 5.0 | 6.0 | 4.0 | 7.0 | 5.0 | ... | 6.0 | 3.0 | 6.0 | 5 |
| YearBuilt | 1976.0 | 1970.0 | 1996.0 | 1977.0 | 1977.0 | 2006.0 | 1957.0 | 1957.0 | 1965.0 | 1947.0 | ... | 1977.0 | 1977.0 | 1977.0 | 1999 |
| YearRemodAdd | 1976.0 | 1970.0 | 1997.0 | 1977.0 | 2000.0 | 2006.0 | 1996.0 | 2000.0 | 1965.0 | 1950.0 | ... | 1977.0 | 1977.0 | 1977.0 | 1999 |
| RoofStyle | 1.0 | 0.0 | 1.0 | 3.0 | 1.0 | 1.0 | 3.0 | 1.0 | 3.0 | 1.0 | ... | 1.0 | 1.0 | 1.0 | 1 |
| RoofMatl | 1.0 | 5.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | ... | 1.0 | 1.0 | 1.0 | 1 |
| Exterior1st | 8.0 | 12.0 | 7.0 | 8.0 | 4.0 | 11.0 | 12.0 | 12.0 | 7.0 | 7.0 | ... | 8.0 | 7.0 | 8.0 | 11 |
| Exterior2nd | 9.0 | 13.0 | 7.0 | 9.0 | 4.0 | 12.0 | 13.0 | 13.0 | 7.0 | 7.0 | ... | 9.0 | 7.0 | 9.0 | 12 |
| MasVnrType | 2.0 | 2.0 | 2.0 | 1.0 | 3.0 | 2.0 | 1.0 | 0.0 | 2.0 | 2.0 | ... | 2.0 | 2.0 | 2.0 | 2 |
| MasVnrArea | 0.0 | 0.0 | 0.0 | 480.0 | 126.0 | 0.0 | 180.0 | 67.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 344 |
| ExterQual | 3.0 | 2.0 | 2.0 | 3.0 | 2.0 | 2.0 | 3.0 | 3.0 | 3.0 | 3.0 | ... | 2.0 | 3.0 | 3.0 | 2 |
| ExterCond | 4.0 | 2.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | ... | 4.0 | 4.0 | 4.0 | 4 |
| Foundation | 1.0 | 2.0 | 2.0 | 1.0 | 1.0 | 2.0 | 1.0 | 1.0 | 1.0 | 1.0 | ... | 1.0 | 2.0 | 1.0 | 2 |
| BsmtQual | 2.0 | 3.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 3.0 | 3.0 | 3.0 | ... | 2.0 | 2.0 | 2.0 | 2 |
| BsmtCond | 3.0 | 1.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | ... | 3.0 | 3.0 | 3.0 | 3 |

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BsmtFinSF1 | 120.0 | 351.0 | 862.0 | 705.0 | 1246.0 | 0.0 | 1302.0 | 168.0 | 698.0 | 442.0 | ... | 767.0 | 1440.0 | 559.0 | 694 |
| BsmtFinType2 | 5.0 | 4.0 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 1.0 | 2.0 | 5.0 | ... | 5.0 | 5.0 | 5.0 | 5 |
| BsmtFinSF2 | 0.0 | 823.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 682.0 | 96.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0 |
| BsmtUnfSF | 958.0 | 1043.0 | 255.0 | 1139.0 | 356.0 | 879.0 | 90.0 | 284.0 | 420.0 | 390.0 | ... | 441.0 | 0.0 | 510.0 | 284 |
| TotalBsmtSF | 1078.0 | 2217.0 | 1117.0 | 1844.0 | 1602.0 | 879.0 | 1392.0 | 1134.0 | 1214.0 | 832.0 | ... | 1208.0 | 1440.0 | 1069.0 | 978 |
| Heating | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | ... | 1.0 | 1.0 | 1.0 | 1 |
| HeatingQC | 4.0 | 0.0 | 0.0 | 0.0 | 2.0 | 0.0 | 4.0 | 0.0 | 0.0 | 4.0 | ... | 4.0 | 4.0 | 4.0 | 0 |
| CentralAir | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | ... | 1.0 | 1.0 | 1.0 | 1 |
| Electrical | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | ... | 4.0 | 4.0 | 4.0 | 4 |
| 1stFlrSF | 958.0 | 2217.0 | 1127.0 | 1844.0 | 1602.0 | 879.0 | 1392.0 | 1803.0 | 1214.0 | 832.0 | ... | 1208.0 | 1440.0 | 1069.0 | 1005 |
| 2ndFlrSF | 0.0 | 0.0 | 886.0 | 0.0 | 0.0 | 984.0 | 0.0 | 0.0 | 0.0 | 384.0 | ... | 0.0 | 0.0 | 0.0 | 978 |
| LowQualFinSF | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0 |
| GrLivArea | 958.0 | 2217.0 | 2013.0 | 1844.0 | 1602.0 | 1863.0 | 1392.0 | 1803.0 | 1214.0 | 1216.0 | ... | 1208.0 | 1440.0 | 1069.0 | 1983 |
| BsmtFullBath | 0.0 | 1.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | 0.0 | ... | 1.0 | 2.0 | 0.0 | 0 |
| BsmtHalfBath | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0 |
| FullBath | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 1.0 | 2.0 | 1.0 | 1.0 | ... | 1.0 | 2.0 | 2.0 | 2 |
| HalfBath | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 1.0 | 0.0 | 0.0 | 0.0 | ... | 1.0 | 0.0 | 0.0 | 1 |
| BedroomAbvGr | 2.0 | 4.0 | 3.0 | 3.0 | 3.0 | 4.0 | 3.0 | 3.0 | 2.0 | 2.0 | ... | 3.0 | 4.0 | 2.0 | 3 |
| KitchenAbvGr | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | ... | 1.0 | 2.0 | 1.0 | 1 |
| KitchenQual | 3.0 | 2.0 | 3.0 | 3.0 | 2.0 | 2.0 | 3.0 | 3.0 | 3.0 | 3.0 | ... | 3.0 | 3.0 | 3.0 | 2 |
| TotRmsAbvGrd | 5.0 | 8.0 | 8.0 | 7.0 | 8.0 | 9.0 | 5.0 | 8.0 | 6.0 | 6.0 | ... | 6.0 | 8.0 | 4.0 | 9 |
| Functional | 6.0 | 6.0 | 6.0 | 6.0 | 6.0 | 6.0 | 4.0 | 0.0 | 6.0 | 6.0 | ... | 6.0 | 6.0 | 6.0 | 6 |
| Fireplaces | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | 0.0 | ... | 1.0 | 0.0 | 0.0 | 1 |
| GarageType | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 3.0 | 5.0 | 1.0 | 5.0 | 5.0 | ... | 1.0 | 1.0 | 1.0 | 1 |
| GarageYrBlt | 1977.0 | 1970.0 | 1997.0 | 1977.0 | 1977.0 | 2006.0 | 1957.0 | 1957.0 | 1965.0 | 1947.0 | ... | 1977.0 | 1980.0 | 1977.0 | 1999 |
| GarageFinish | 1.0 | 2.0 | 2.0 | 1.0 | 0.0 | 0.0 | 2.0 | 1.0 | 2.0 | 2.0 | ... | 1.0 | 2.0 | 1.0 | 0 |
| GarageArea | 440.0 | 621.0 | 455.0 | 546.0 | 529.0 | 660.0 | 528.0 | 484.0 | 461.0 | 336.0 | ... | 546.0 | 0.0 | 440.0 | 490 |
| GarageQual | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 1.0 | 4.0 | ... | 4.0 | 4.0 | 4.0 | 4 |
| GarageCond | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 1.0 | 4.0 | ... | 4.0 | 4.0 | 4.0 | 4 |
| PavedDrive | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | ... | 2.0 | 0.0 | 2.0 | 2 |
| WoodDeckSF | 0.0 | 81.0 | 180.0 | 0.0 | 240.0 | 100.0 | 0.0 | 0.0 | 0.0 | 158.0 | ... | 198.0 | 321.0 | 0.0 | 0 |
| OpenPorchSF | 205.0 | 207.0 | 130.0 | 122.0 | 0.0 | 17.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 42.0 | 0.0 | 55.0 | 0 |

The entire data was in numerical format

Now, used standard scaler method for scaling the data,

```
sd = StandardScaler()
def scale_data(data):
    """
    It scales the data using Standard scaler
    """
    data = sd.fit_transform(data)
    return data
```

**Outliers**: The sale price may varies for different reasons like selling person, need for sell ,Area, land value differs on market price

So I not considered the outliers here

After this data is splitted into X and Y

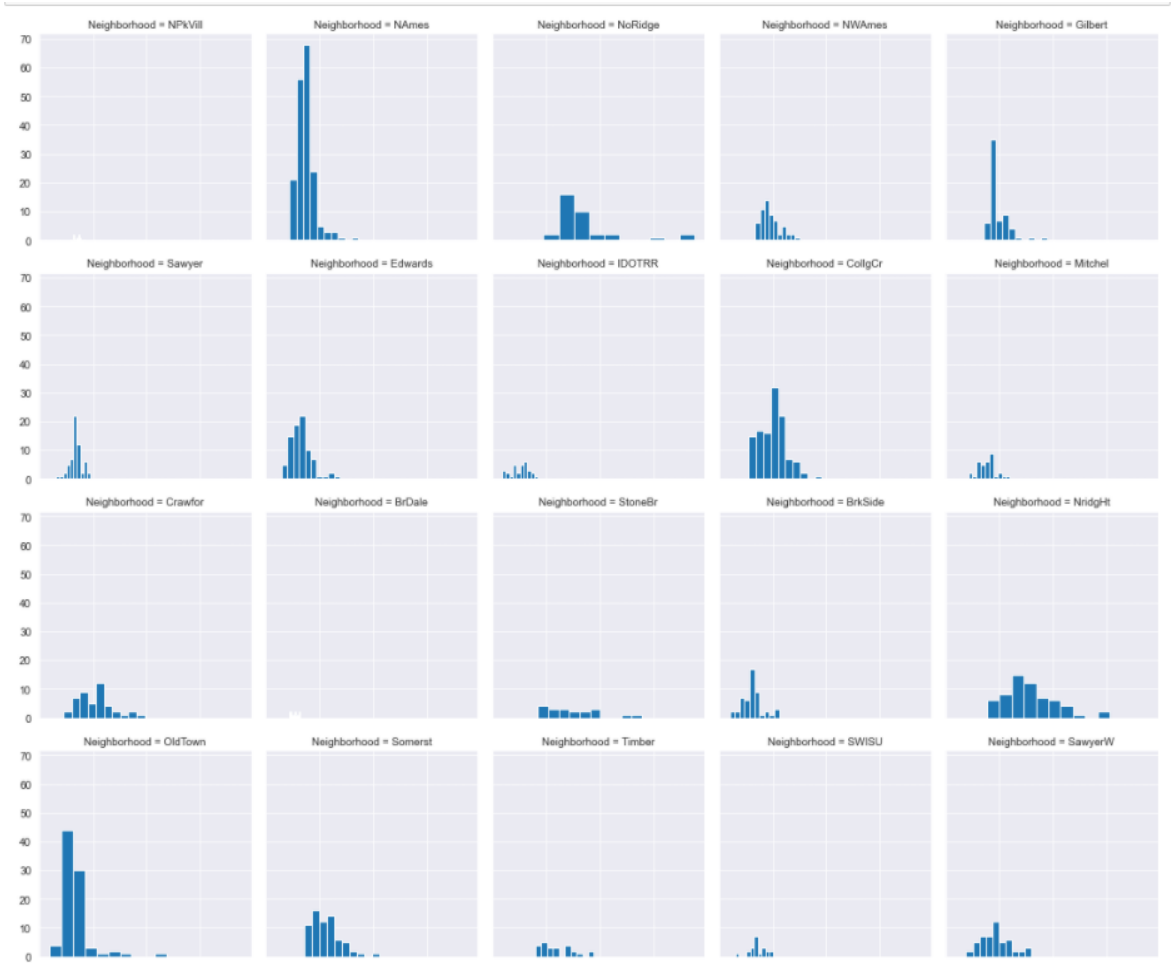
And further splitted into train and test for training purpose.

# Data Inputs- Logic- Output Relationships

Started with the distribution of sale price:


We can see that Most of the sale price lies between 100000 to 450000.

I thought neighbourhood has more impact on sale price so compared the distribution of sale price with the neighbourhood

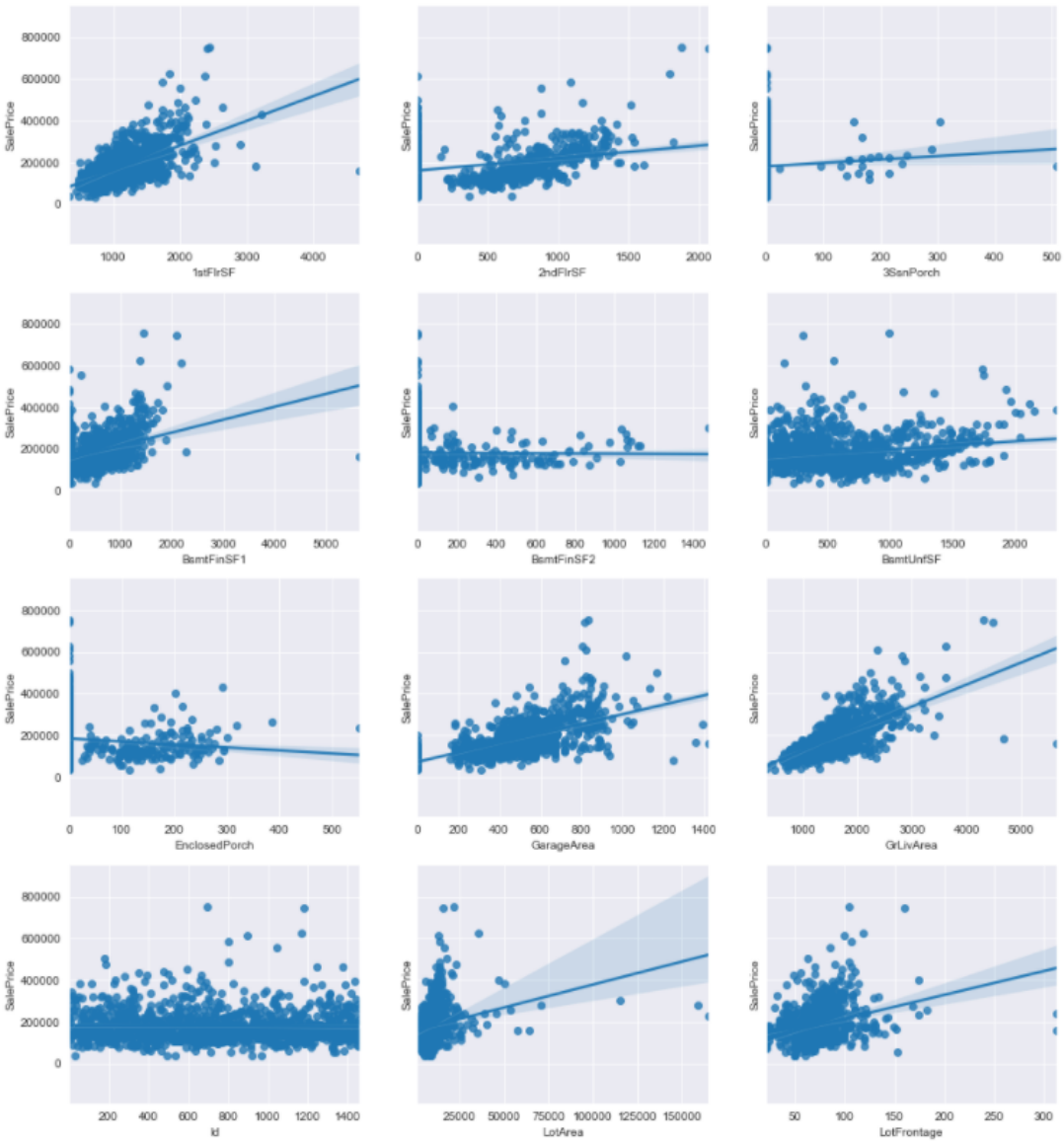


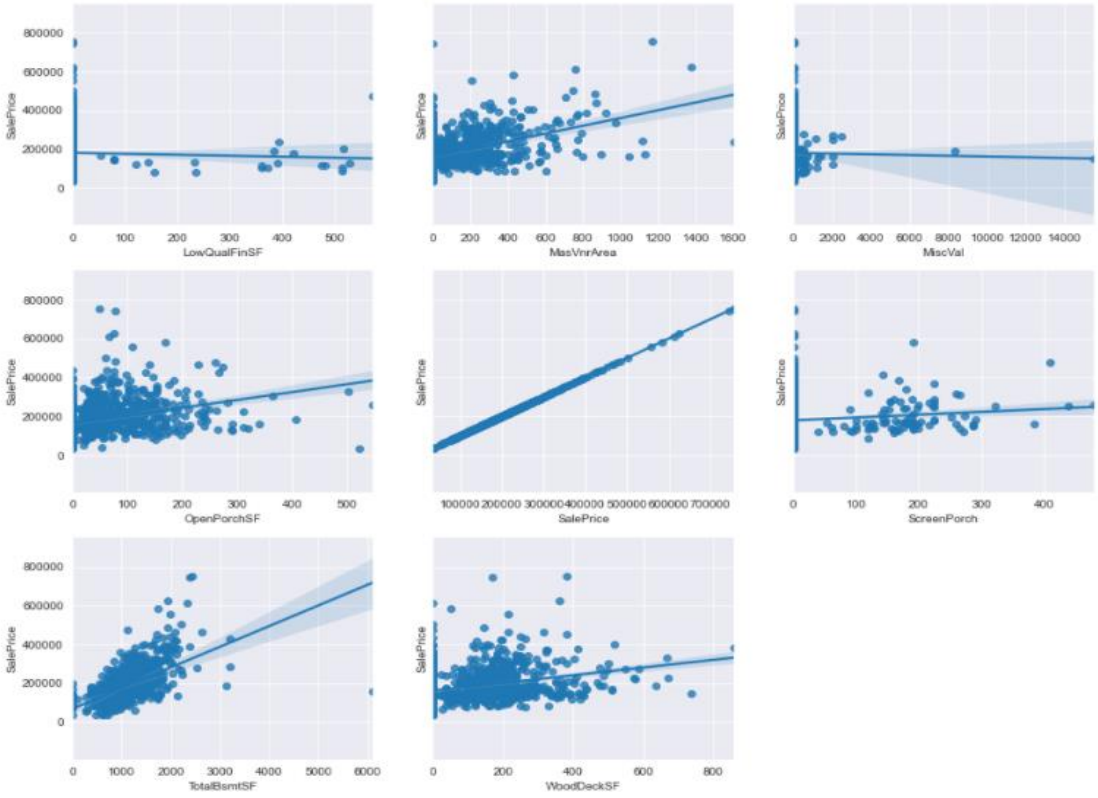

- Neighborhood like NAmes, Collgcr, oldtown has sale price between 150000 and 250000.
- Neighborhood Nridght has more spread of sale price.
- (All the other neighborhood has a smaller number of sales

To check the relationships between features divided the features into

- Numeical features
- Categorical features
- Time series features

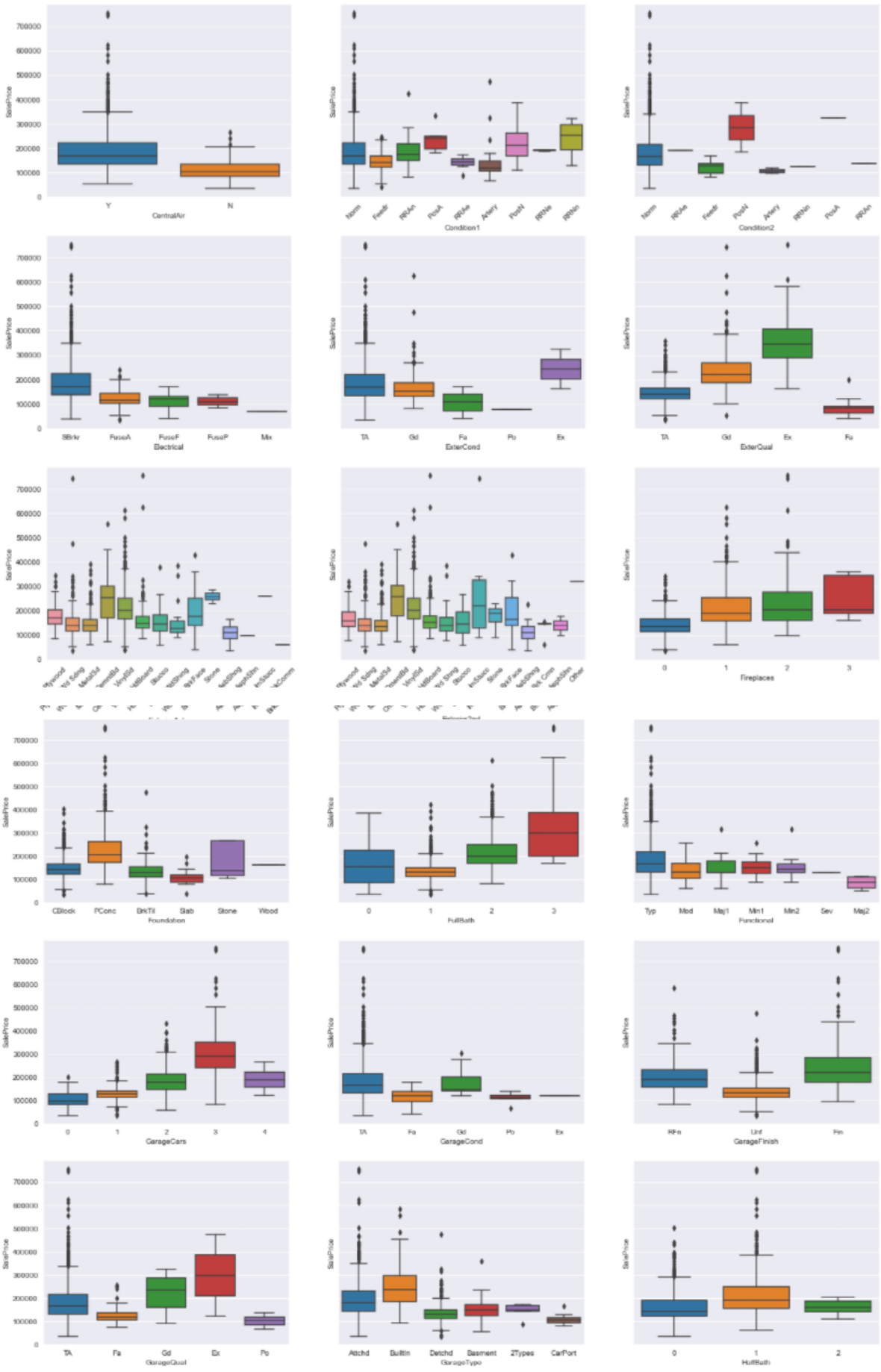
**Plotting the numerical features with the sale price:**

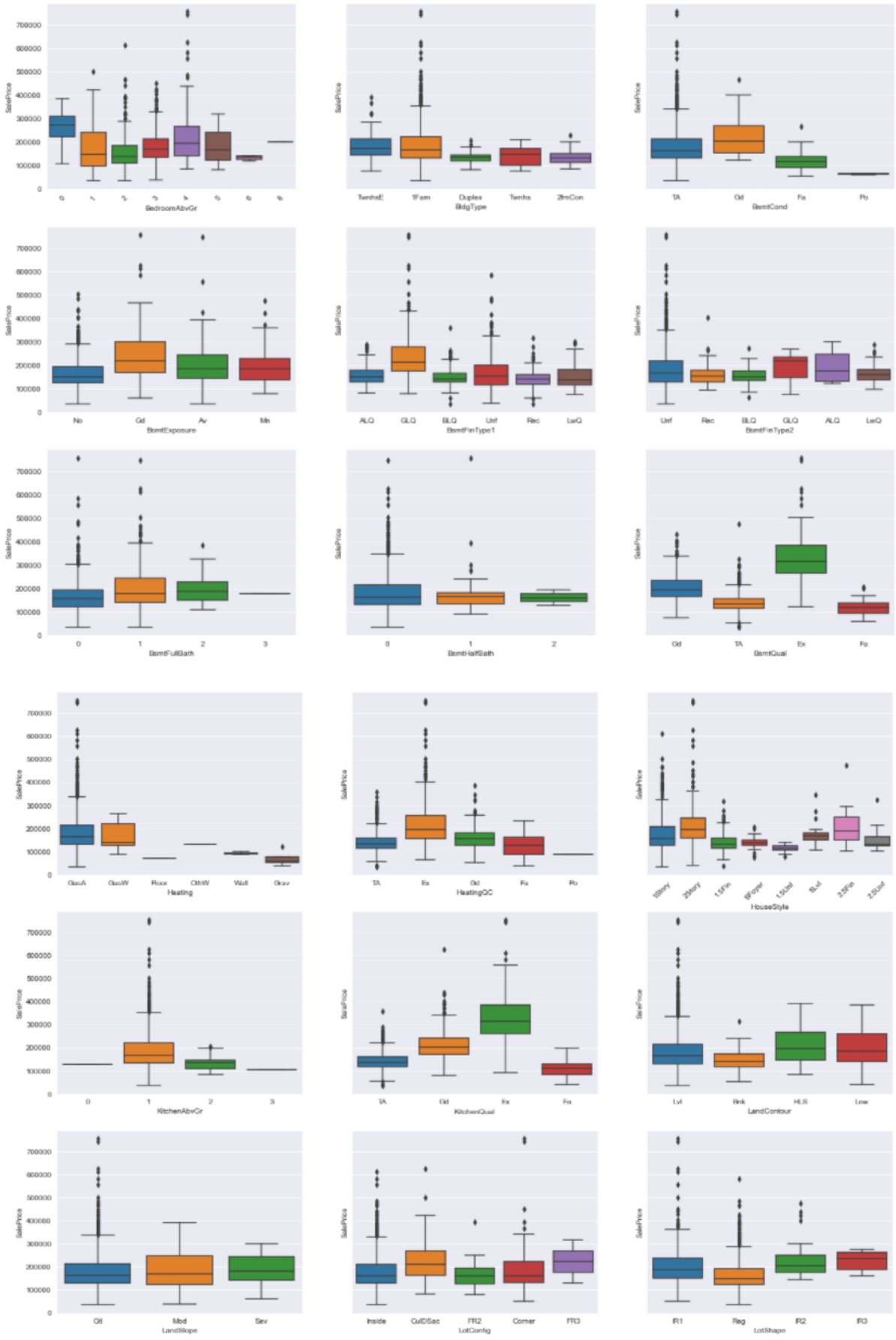

SalePrice
1stFlrSF
2ndFlrSF
3SsnPorch
BsmtFinSF1
BsmtFinSF2
BsmtUnfSF
EnclosedPorch
GarageArea
GrLivArea
Id
LotArea
LotFrontage

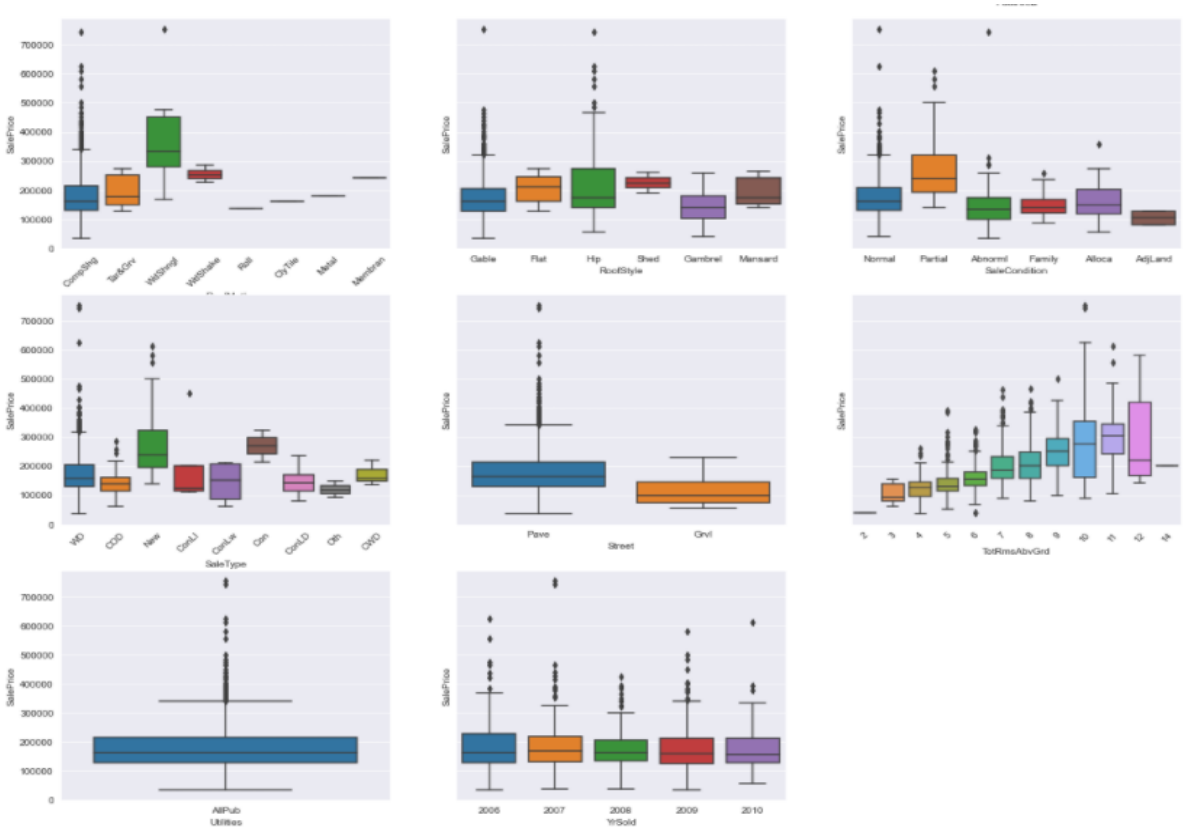



This gives the idea of most related features with sale prices

**Plotting the Categorical features:**

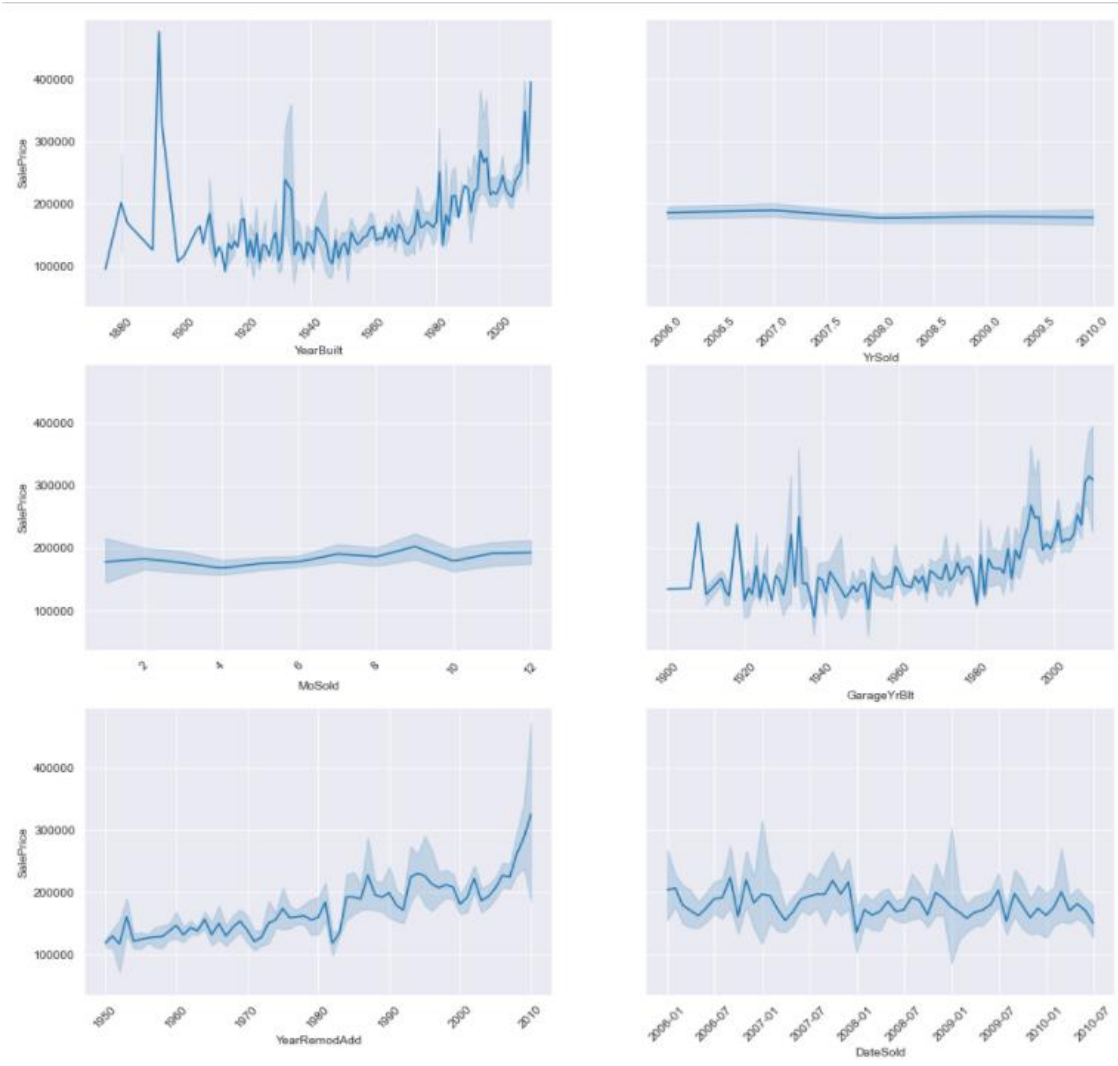

SalePrice
CentralAir
Y
N
Condition1
Norm
Feedr
RRAn
PosA
RRAe
Artery
PosN
RRNe
RRNn
Condition2
Norm
RRAe
Feedr
PosN
Artery
RRNn
PosA
RRAn
Electrical
SBrkr
FuseA
FuseF
FuseP
Mix
ExterCond
TA
Gd
Fa
Po
Ex
ExterQual
TA
Gd
Ex
Fa
Fireplaces
0
1
2
3
Foundation
CBlock
PConc
BrkTil
Slab
Stone
Wood
FullBath
0
1
2
3
Functional
Typ
Mod
Maj1
Min1
Min2
Sev
Maj2
GarageCars
0
1
2
3
4
GarageCond
TA
Fa
Gd
Po
Ex
GarageFinish
RFn
Unf
Fin
GarageQual
TA
Fa
Gd
Ex
Po
GarageType
Attchd
BuiltIn
Detchd
Basment
2Types
CarPort
HalfBath
0
1
2


SalePrice
BedroomAbvGr
TwnhsE
1Fam
Duplex
Twnhs
2fmCon
BldgType
TA
Gd
Fa
Po
BsmtCond
No
Gd
Av
Mn
BsmtExposure
ALQ
GLQ
BLQ
Unf
Rec
LwQ
BsmtFinType1
BsmtFinType2
BsmtFullBath
BsmtHalfBath
Ex
BsmtQual
GasA
GasW
Floor
OthW
Wall
Grav
Heating
HeatingQC
1Story
2Story
1.5Fin
SFoyer
1.5Unf
SLvl
2.5Fin
2.5Unf
HouseStyle
KitchenAbvGr
KitchenQual
Lvl
Bnk
HLS
Low
LandContour
Gtl
Mod
Sev
LandSlope
Inside
CulDSac
FR2
Corner
FR3
LotConfig
IR1
Reg
IR2
IR3
LotShape




This gives the clear idea on how categorical featurs are dependent on the sale price.

**Plotting the time-series features:**




This gives the idea on how sale price is varying based on time series data.

## Hardware and Software Requirements and Tools Used

**Pandas:** is a software library written for the **Python** programming language for data manipulation and analysis.

**Numpy:** is a **Python** library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices.

**Matplotlib:** is a plotting library for the **Python** programming language and its numerical mathematics extension NumPy.

**Seaborn:** is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

**Sklearn: Scikit**-learn (formerly scikits. learn and also known as **sklearn**) is a free software machine learning library for the Python programming language.

**LabelEncoder:** In **label encoding** in **Python**, we replace the categorical value with a numeric value between 0 and the number of classes minus 1. If the categorical variable value contains 5 distinct classes, we use (0, 1, 2, 3, and 4).

**StandardScaler :** In **label encoding** in **Python**, we replace the categorical value with a numeric value between 0 and the number of classes minus 1. If the categorical variable value contains 5 distinct classes, we use (0, 1, 2, 3, and 4).

**Simple Imputer**: **impute** class SimpleImputer to **impute**/replace missing values for both numerical and categorical features. For numerical missing values, a strategy such as mean, median, most frequent, and constant **can** be used. For categorical features, a strategy such as the most frequent and constant **can** be used.

**Train test split:** The **train-test split** is a technique for evaluating the performance of a machine learning algorithm. It can be used for classification or regression problems and can be used for any supervised learning algorithm. The procedure involves taking a dataset and dividing it into two subsets.

**GridSearchCV: Grid search** is an approach to parameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a **grid.**

**Stratified k Fold:** is an extension of regular **kfold cross validation** but specifically for classification problems where rather than the splits being completely random, the ratio between the target classes is the same in each fold as it is in the full dataset.

**Tensorflow:** is an open source library for numerical computation and large-scale machine learning. **TensorFlow** bundles together a slew of machine learning and deep learning (aka neural networking) models and algorithms and makes them useful by way of a common metaphor.

**Keras:** is a powerful and easy-to-use free open source Python library for developing and evaluating deep learning models. It wraps the efficient numerical computation libraries Theano and TensorFlow and allows you to define and train neural network models in just a few lines of code.

# Model/s Development and Evaluation

## Identification of possible problem-solving approaches

Regression methods

## Testing of Identified Approaches (Algorithms)

1. KNeighborsregressor
2. Supportvectorregressor
3. Decisiontreeregressor
4. RandomForestRegressor
5. Adaboostregressor
6. Gradientboosting regressor
7. XGBregressor
8. Sequential model(ANN)

## Run and evaluate selected models

First made a for loop for three models:

1.SVR

2.Decision tree regressor

3.Kneighborsregressor

```
#general model performances
models = [SVR(), DecisionTreeRegressor(), KNeighborsRegressor()]
for model in models:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)
    print(model)
    print('rscore :',r2_score(y_val, y_pred))
    print('MAE :', mean_absolute_error(y_val, y_pred))
    print('MSE :', mean_squared_error(y_val, y_pred))
    print('RMSE :',  np.sqrt(mean_squared_error(y_val, y_pred)))
    print('\n')
```

```
SVR()
rscore : -0.03914925369073452
MAE : 57129.05451168578
MSE : 7250158210.188264
RMSE : 85147.86086677847


DecisionTreeRegressor()
rscore : 0.5760591974974903
MAE : 28890.05982905983
MSE : 2957840636.4444447
RMSE : 54386.03346857026


KNeighborsRegressor()
rscore : 0.6316165933133218
MAE : 26009.13247863248
MSE : 2570215944.4377775
RMSE : 50697.297210381716
```

Ensemble technique:

```
model = RandomForestRegressor()
model.fit(X_train,y_train)
y_pred = model.predict(X_val)
print('r2score :',r2_score(y_val, y_pred))
print('MAE :', mean_absolute_error(y_val, y_pred))
print('MSE :', mean_squared_error(y_val, y_pred))
print('RMSE :',  np.sqrt(mean_squared_error(y_val, y_pred)))
```

```
r2score : 0.7867279489694878
MAE : 19515.571923076925
MSE : 1488001946.1022928
RMSE : 38574.62826913945
```

**Boosting techniques:**

```
models = [AdaBoostRegressor(base_estimator = RandomForestRegressor()), GradientBoostingRegressor(), XGBRegressor()]
for model in models:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)
    print(model)
    print('r2score :',r2_score(y_val, y_pred))
    print('MAE :', mean_absolute_error(y_val, y_pred))
    print('MSE :', mean_squared_error(y_val, y_pred))
    print('RMSE :',  np.sqrt(mean_squared_error(y_val, y_pred)))
    print('\n')
```

```
AdaBoostRegressor(base_estimator=RandomForestRegressor())
r2score : 0.8290629484098919
MAE : 18175.450683760686
MSE : 1192630090.0565686
RMSE : 34534.47683195112


GradientBoostingRegressor()
r2score : 0.800592778148234
MAE : 18602.15772563482
MSE : 1391266847.8994884
RMSE : 37299.689648836065


XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=8, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
r2score : 0.8167743844817914
MAE : 19601.010082799145
MSE : 1278367564.5707517
RMSE : 35754.26638277944
```

---

# HYPERPARAMETER TUNING

### DECISION TREE REGRESSOR

```
cv = StratifiedKFold(n_splits = 5, shuffle = True)
params_dtr = {'min_samples_split': np.arange(2, 10),
          'min_samples_leaf': np.arange(.05, .2),
          'max_leaf_nodes': np.arange(2, 30)}

dtr_grd = GridSearchCV(estimator = DecisionTreeRegressor(random_state = 42),
                       param_grid = params_dtr,
                       cv= cv,
                       verbose = True,
                       scoring = 'r2',
                       n_jobs = -1)
dtr_grd.fit(X_train, y_train)

y_pred_dtr = dtr_grd.predict(X_val)

metric_dtr = []

metric_dtr.append(r2_score(y_val, y_pred_dtr))
metric_dtr.append(mean_absolute_error(y_val, y_pred_dtr))
metric_dtr.append(mean_squared_error(y_val, y_pred_dtr))
metric_dtr.append(np.sqrt(mean_squared_error(y_val, y_pred_dtr)))
```

```
Fitting 5 folds for each of 224 candidates, totalling 1120 fits
```

**KNEIGHBORSREGRESSOR**

```
params_knr = {'n_neighbors': [2,3,4,5,6],
              'weights': ['uniform','distance']}

knr_grd = GridSearchCV(estimator = KNeighborsRegressor(),
                       param_grid = params_knr,
                       cv= cv,
                       verbose = False,
                       scoring = 'r2',
                       n_jobs = -1)
knr_grd.fit(X_train, y_train)

y_pred_knr = knr_grd.predict(X_val)

metric_knr = []

metric_knr.append(r2_score(y_val, y_pred_knr))
metric_knr.append(mean_absolute_error(y_val, y_pred_knr))
metric_knr.append(mean_squared_error(y_val, y_pred_knr))
metric_knr.append(np.sqrt(mean_squared_error(y_val, y_pred_knr)))
```

**RANDOMFORESTREGRESSOR**

```
params_rfr = {'n_estimators' : [50,100,200,400,500],
             'max_features' : ["auto", "sqrt", "log2"],
             'min_samples_split' : np.linspace(0.1, 1.0, 10),
              'max_depth' : [x for x in range(1,20)]}

rfr_grd = GridSearchCV(estimator = RandomForestRegressor(random_state = 42),
                       param_grid = params_rfr,
                       cv= cv,
                       verbose = False,
                       scoring = 'r2',
                       n_jobs = -1)
rfr_grd.fit(X_train, y_train)

y_pred_rfr = rfr_grd.predict(X_val)

metric_rfr = []

metric_rfr.append(r2_score(y_val, y_pred_rfr))
metric_rfr.append(mean_absolute_error(y_val, y_pred_rfr))
metric_rfr.append(mean_squared_error(y_val, y_pred_rfr))
metric_rfr.append(np.sqrt(mean_squared_error(y_val, y_pred_rfr)))
```

```
C:\Users\mussa\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:670: UserWarning: The least populated class in y h
as only 1 members, which is less than n_splits=5.
  warnings.warn(("The least populated class in y has only %d"
```

**ADABOOSTREGRESSOR**

```
params_abr = {'base_estimator' : [RandomForestRegressor()],
              'n_estimators': [50, 100],
              'learning_rate' : [0.01,0.05,0.1,0.3,1],
              'loss' : ['linear', 'square', 'exponential']}

abr_grd = GridSearchCV(estimator = AdaBoostRegressor(random_state = 42),
                       param_grid = params_abr,
                       cv = cv,
                       verbose = True,
                       scoring = 'r2',
                       n_jobs = -1)
abr_grd.fit(X_train, y_train)

y_pred_abr = abr_grd.predict(X_val)

metric_abr = []

metric_abr.append(r2_score(y_val, y_pred_abr))
metric_abr.append(mean_absolute_error(y_val, y_pred_abr))
metric_abr.append(mean_squared_error(y_val, y_pred_abr))
metric_abr.append(np.sqrt(mean_squared_error(y_val, y_pred_abr)))
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
```

**GRADIENTBOOSTINGREGRESSOR**

```
params_gbr = {'learning_rate' : [1 ,0.1, 0.05, 0.01],
              'n_estimators' : [50, 100, 200],
              'max_depth' : np.linspace(1, 10, 20, endpoint=True),
              'min_samples_split' : np.linspace(0.1, 1.0, 10, endpoint=True)}

gbr_grd = GridSearchCV(estimator = GradientBoostingRegressor(random_state = 42),
                       param_grid = params_gbr,
                       cv = cv,
                       verbose = True,
                       scoring = 'r2',
                       n_jobs = -1)
gbr_grd.fit(X_train, y_train)

y_pred_gbr = gbr_grd.predict(X_val)

metric_gbr = []

metric_gbr.append(r2_score(y_val, y_pred_gbr))
metric_gbr.append(mean_absolute_error(y_val, y_pred_gbr))
metric_gbr.append(mean_squared_error(y_val, y_pred_gbr))
metric_gbr.append(np.sqrt(mean_squared_error(y_val, y_pred_gbr)))
```

```
Fitting 5 folds for each of 2400 candidates, totalling 12000 fits
```

**XGBREGRESSOR**

```
params_xgbr = {'learning_rate': [0.01, 0.1,1],
        'learning_rate': [0.01, 0.1],
        'max_depth': [3, 5, 7, 10],
        'min_child_weight': [1, 3, 5],
        'subsample': [0.5, 0.7],
        'colsample_bytree': [0.5, 0.7],
        'n_estimators' : [100, 200, 500],
        }
xgbr_grd = GridSearchCV(estimator = XGBRegressor(random_state = 42),
                       param_grid = params_xgbr,
                       cv = cv,
                       verbose = True,
                       scoring = 'r2',
                       n_jobs = -1)
xgbr_grd.fit(X_train, y_train)

y_pred_xgbr = xgbr_grd.predict(X_val)

metric_xgbr = []

metric_xgbr.append(r2_score(y_val, y_pred_xgbr))
metric_xgbr.append(mean_absolute_error(y_val, y_pred_xgbr))
metric_xgbr.append(mean_squared_error(y_val, y_pred_xgbr))
metric_xgbr.append(np.sqrt(mean_squared_error(y_val, y_pred_xgbr)))
```

```
Fitting 5 folds for each of 288 candidates, totalling 1440 fits
```

# Results:

| | Metrics | Decision Tree Regressor | KNeighbors Regressor | Random Forest Regressor | Ada Boost Regressor | Gradient Boosting Regressor | XGBRegressor |
|---|---|---|---|---|---|---|---|
| 0 | R2_score | 6.549225e-01 | 6.444579e-01 | 7.019122e-01 | 8.411609e-01 | 7.661486e-01 | 8.161143e-01 |
| 1 | Mean_absolute_error | 3.039652e+04 | 2.572184e+04 | 2.584308e+04 | 1.778250e+04 | 1.913224e+04 | 1.747958e+04 |
| 2 | Mean_squared_error | 2.407611e+09 | 2.480622e+09 | 2.079763e+09 | 1.108223e+09 | 1.631584e+09 | 1.282973e+09 |
| 3 | Root_mean_squared_error | 4.906741e+04 | 4.980584e+04 | 4.560441e+04 | 3.328998e+04 | 4.039287e+04 | 3.581861e+04 |

# Finalized model:

## AdaBoostRegressor(*base_estimator': RandomForestRegressor(), 'learning_rate': 1, 'loss': 'linear', 'n_estimators': 50*)

### Training data with the best model

```
%%time
ideal_model = AdaBoostRegressor(base_estimator= RandomForestRegressor(),
                                learning_rate= 1,
                                loss= 'linear',
                                n_estimators= 50)
ideal_model.fit(X_train,y_train)
```

```
Wall time: 39.4 s
```

```
AdaBoostRegressor(base_estimator=RandomForestRegressor(), learning_rate=1)
```

# Metrics used:

**R2_score:** regression score function. Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a score of 0.0.

**Mean_absolute_error:** In statistics, **mean absolute error** (**MAE**) is a measure of errors between paired observations expressing the same phenomenon. Examples of *Y* versus *X* include comparisons of predicted versus observed, subsequent time versus initial time, and one technique of measurement versus an alternative technique of measurement. MAE is calculated as:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} |\, x_i - x \,|$$

**Mean_squared_error:** In statistics, the **mean squared error** (**MSE**) or **mean squared deviation** (**MSD**) of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value. MSE is a risk function, corresponding to the expected value of the squared error loss. The fact that MSE is almost always strictly positive (and not zero) is because of randomness or because the estimator does not account for information that could produce a more accurate estimate.

The MSE is a measure of the quality of an estimator—it is always non-negative, and values closer to zero are better.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n} (y_i - \tilde{y}_i)^2$$

**Root mean squared error:** The **root-mean-square deviation** (**RMSD**) or **root-mean-square error** (**RMSE**) is a frequently used measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed. The RMSD represents the square root of the second sample moment of the differences between predicted values and observed values or the quadratic mean of these differences. These deviations are called *residuals* when the calculations are performed over the data sample that was used for estimation and are called *errors* (or prediction errors) when computed out-of-sample. The RMSD serves to aggregate the magnitudes of the errors in predictions for various data points into a single measure of predictive power. RMSD is a measure of accuracy, to compare forecasting errors of different models for a particular dataset and not between datasets, as it is scale-dependent.[1]

RMSD is always non-negative, and a value of 0 (almost never achieved in practice) would indicate a perfect fit to the data. In general, a lower RMSD is better than a higher one. However, comparisons across different types of data would be invalid because the measure is dependent on the scale of the numbers used.

RMSD is the square root of the average of squared errors. The effect of each error on RMSD is proportional to the size of the squared error; thus larger errors have a disproportionately large effect on RMSD. Consequently, RMSD is sensitive to outliers

$$RMS = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (d_i - p_i)^2}$$

# CONCLUSION

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

As a recommendation, I advise to use this model (or a version of it trained with more recent data) by people who want to buy a house in the area covered by the dataset to have an idea about the actual price. The model can be used also with datasets that cover different cities and areas provided that they contain the same features. I also suggest that people take into consideration the features that were deemed as most important as seen in the previous section; this might help them estimate the house price better.