FINAL YEAR PROJECT

COMPARISON OF PERFORMANCE OF SOME MACHINE LEARNING MODELS FOR HOUSE PRICE PREDICTION

PRANAB KUMAR PANDA

University Roll No. 7473U196013

College Roll No. 2019B006

Institute of Mathematics & Applications
Andharua, Bhubaneswar- 751029
Odisha

COMPARISON OF PERFORMANCE OF SOME MACHINE LEARNING MODELS FOR HOUSE PRICE PREDICTION

A Project Report submitted to

Institute of Mathematics & Applications, Bhubaneswar

in partial fulfilment of the requirements

of the degree of

B. Sc. (Hons)

in

Mathematics & Computing

by

PRANAB KUMAR PANDA
(7473U196013)

under the supervision of

Prof. Sudarsan Padhy

INSTITUTE OF MATHEMATICS AND APPLICATIONS, BHUBANESWAR
Odisha, India

Institute of Mathematics & Applications
Andharua, Bhubaneswar- 751 029, Odisha.

Prof. Sudarsan Padhy
Guest Faculty
Institute of Mathematics and Applications

Date 04/08/2022

Supervisor's Certificate

This is to certify that the work presented in this project entitled *"COMPARISON OF PERFORMANCE OF SOME MACHINE LEARNING MODELS FOR HOUSE PRICE PREDICTION"* by *PRANAB KUMAR PANDA*, 7473U196013, is a record of original research/ review work carried out by him/her under my supervision and guidance in partial fulfilment of the requirements of the B. Sc. (Hons) in Mathematics and Computing. Neither this project nor any part of it has been submitted for any degree or diploma to any institute or university in India or abroad.

Prof. Sudarsan Padhy

Dedicated

To Maa and Bapa

Declaration

I, *PRANAB KUMAR PANDA*, University Roll Number 7473U196013 hereby declare that this project entitled *“COMPARISON OF PERFORMANCE OF SOME MACHINE LEARNING MODELS FOR HOUSE PRICE PREDICTION*" represents my original work carried out as a B. Sc. (Hons) of IMA Bhubaneswar and, to the best of my knowledge, it is not a complete copy of previously published or written by another person, nor any material presented for award of any other degree or diploma of Institute of Mathematics and Applications, Bhubaneswar or any other institution. Any contribution made to this research by others, with whom I have worked at Institute of Mathematics and Applications, Bhubaneswar or elsewhere, is explicitly acknowledged in the dissertation. Works of other authors cited in this dissertation have been duly acknowledged under the section Bibliography.

Date 04/08/2022 Pranab Kumar Panda

Acknowledgment

This project is the document to support my candidature for the academic degree Bachelor of Science (B.Sc.), in which is detailed explanation of my research and findings on the topic COMPARISON OF PERFORMANCE OF SOME MACHINE LEARNING MODELS FOR HOUSE PRICE PREDICTION. Mentioned study was developed in Institute of Mathematics and Applications, Bhubaneswar at Utkal University, India. I would like to thank Prof. Sudarsan Padhy, who has supervised the academic and professional development of investigation of implementation of various Machine Learning Algorithms and comparing their accuracy. Also, I appreciate my family and personal friends; especially to Suman Sourav Biswal, Prateek Kumar for their enthusiastic support, constant motivation, and constant encouragement, and to my seniors Sampa Mondal and Swati Sucharita for their constructive comments that improved the presentation of these findings.

Abstract

This project proposes a performance comparison between some machine learning algorithms. The machine learning models used in this study are Multiple Linear regression, Ridge regression, Least Absolute Shrinkage and Selection Operator (Lasso), XGBoost, Random Forest, SVR(Support Vector Regression), ANN(Artificial Neural Network). Moreover, this study attempts to analyse the correlation between variables to determine the most important factors that affect house prices in a dataset taken from kaggle.com containing data of residential homes in Ames, Iowa.

The accuracy of the prediction is evaluated by checking the Mean Absolute Error (MAE), Mean Squared Error (MSE), RMSE (Root Mean Square Error), R^2 Square of all the applied model. The test is performed after applying the required pre-processing methods and splitting the data into two parts. However, one part will be used in the training and the other in the test phase. The correlation graphs show the variables' level of dependency. The empirical results show that 'EnclosedPorch', 'KitchenAbvGr', 'OverallCond' influence the house prices negatively whereas 'GarageCars', 'GarageArea', 'GrLivArea', 'OverallQual' impact the house prices positively to a great extent. This project shows that Random Forest Regressor gives the best prediction among other algorithms. At the end of the project, we will compare the accuracy of prediction by the machine learning models mentioned earlier by the help of bar graphs.

Keywords: Multiple Linear regression,Ridge regression, Lasso(Least Absolute Shrinkage and Selection Operator), XGBoost(Express Gradient Boosting), SVR(Support Vector Regression), ANN(Artificial Neural Network), EDA(Exploratory Data Analysis), RMSE(Root Mean Square Error)

Contents

Chapter 1

Introduction

" Data science is the art of extracting meaningful insights from various data sources and deriving useful information from them! "

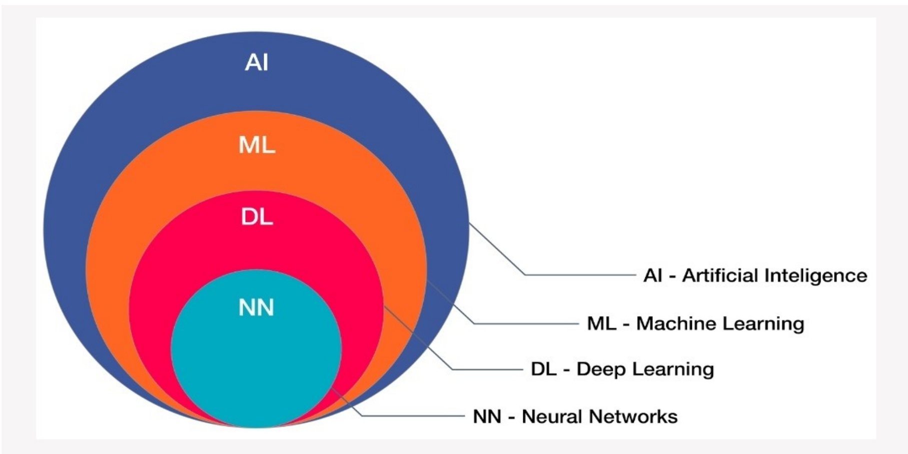
In this project, we will compare some of the well-known machine learning models like Multiple Linear regression, Ridge regression, Lasso, XGBoost, SVM(Support Vector Machine), Random Forest regressor, ANN(Artificial Neural Network) after applying them on a dataset imported from the kaggle competition "House Prices - Advanced Regression Techniques". Here, our aim is to find the best model that will fit our problem (predicting the price of a house). We begin by EDA (Exploratory Data Analysis), where we try to visualize various parameters by analyzing the datasets and summarizing their main characteristics. It helps us to get an initial impression of the data we have before making any assumptions. In fact, it can help us to identify obvious errors, as well as understand patterns within the data, detect outliers or anomalous events, find interesting relations among the variables. One of the results that we get is that, 'EnclosedPorch', 'KitchenAbvGr', 'OverallCond' influence the house prices negatively whereas 'GarageCars', 'GarageArea', 'GrLivArea', 'OverallQual' impact the house prices positively to a great extent. Then, we prepare the data for modelling, by cleaning the data, processing the missing data, selecting the relevant variables, deducing some features, running statistical tests, defining the machine learning models and finally choosing the best price prediction model for the test set. For comparison between models we will use four common evaluation metrics i.e, Mean Absolute Error (MAE), Mean Squared Error (MSE), RMSE (Root Mean Square Error), R^2 Square. We have used Python programming language in Jupyter notebook for implementing all the tasks. Please find the github link and code attached in the end of this report to see the codes and the respective outputs.

Chapter 2

Background

2.1 Artificial Intelligence

In the simplest terms, AI which stands for artificial intelligence refers to systems or machines that mimic human intelligence to perform tasks and can iteratively improve themselves based on the information they collect. AI is the ability of a machine to perform cognitive functions that we associate with human minds, such as perceiving, reasoning, learning, interacting with the environment, problem solving, and even exercising creativity. [1]
Five basic components of artificial intelligence include learning, reasoning, problem-solving, perception, and language understanding. [2]



2.2 Machine Learning

Machine learning is a subfield of AI. In a nutshell, we can say that ML makes computer learn from experience. With the use of machine learning (ML), software programmes can predict outcomes more accurately without having to be explicitly instructed to do so. In order to forecast new output values, machine learning algorithms use historical data as input.

MACHINE LEARNING IS OF BASICALLY THREE TYPES:-

1. SUPERVISED LEARNING
2. UNSUPERVISED LEARNING
3. REINFORCEMENT LEARNING

1. **Supervised Learning**
   Supervised comes from the word supervise which means to observe and direct the execution of a task. In supervised learning, a supervisor is required for training the machine to learn from the data.
   In supervised learning we do predictions and classification (image classification, audio classification) mainly.

2. **Unsupervised Learning**
   Unsupervised learning is a type of machine learning that involves algorithms that train on unlabeled data.In this we can do clustering and recommendation.Clustering means grouping smiliar data in one group and unsimilar data in others.Here we can see that provide recommendations like TSNE, PCA(for dimensionality reduction),FACTORING etc.

3. **Reinforcement Learning**
   Reinforcement learning helps us to make decision.It works on the goal and prescribed set of rules for accomplishing the goal.In Reinforcement learning we train robot(agent)to takes decision(action) so that it will earn maximum reward .We teach the robot to take decision by giving them reward.Reward in general we can say it's a token of appreciation for doing correct things or to justify the performance.

2.3 Models used in the project

In this project the machine learning models that we are going to use, come under supervised learning. The models that we'll use are

1. Multiple Linear Regression
2. Ridge Regression
3. Lasso(Least Absolute Shrinkage and Selection Operator)
4. XGBoost(Express Gradient Boosting)
5. SVM(Support Vector Machine)

6. Random Forest Regressor
7. ANN(Artificial Neural Network)

2.3.1 Multiple Linear Regression

Multiple linear regression (MLR), or simply multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression is to model the linear relationship between the explanatory (independent) variables and response (dependent) variables.

It comes under supervised machine learning and is used to estimate the relationship between one dependent variable and more than one independent variables. Identifying the correlation and its cause-effect helps to make predictions by using these relations. To estimate these relationships, the prediction accuracy of the model is essential; the complexity of the model is of more interest. However, Multiple Linear Regression is prone to many problems such as multicollinearity, noises, and overfitting, which effect on the prediction accuracy.

Regularised regression plays a significant part in Multiple Linear Regression because it helps to reduce variance at the cost of introducing some bias, avoid the overfitting problem and solve ordinary least squares (OLS) problems. There are two types of regularisation techniques L1 norm (least absolute deviations) and L2 norm (least squares). L1 and L2 have different cost functions regarding model complexity .

2.3.2 Ridge Regression

The Ridge Regression is an L2-norm regularised regression technique that was introduced by Hoerl in 1962. It is an estimation procedure to manage collinearity without removing variables from the regression model. In multiple linear regression, the multicollinearity is a common problem that leads least square estimation to be unbiased, and its variances are far from the correct value. Therefore, by adding a degree of bias to the regression model, Ridge Regression reduces the standard errors, and it shrinks the least square coefficients towards the origin of the parameter space.

Ridge formula is:

$$R = Min(\ sum\ of\ squared\ residuals\ +\ \alpha\ *\ slope^2\ ) \tag{2.3.1}$$

When Least Squared Error determines the values of parameters, it minimises the sum of squared residuals. However, when Ridge determines the values of parameters, it reduces the sum of squared residuals. It adds a penalty term, where α determines the severity of the penalty and the length of the slope. In addition, increasing the α makes the slope asymptotically close to zero. Like Lasso, α is determined by applying the Cross-validation method. Therefore, Ridge helps to reduce variance by shrinking parameters and make the prediction less sensitive.

2.3.3 Lasso(Least Absolute Shrinkage and Selection Operator)

LASSO (least absolute shrinkage and selection operator) is a regression analysis method in machine learning that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the resulting statistical model. It was originally introduced in geophysics, and later by Robert Tibshirani, who coined the term in 1996. It is an L1-norm regularised regression technique that can also perform regularisation and feature selection.

Lasso introduces a bias term, but instead of squaring the slope like Ridge regression, the absolute value of the slope is added as a penalty term.

Lasso is defined as:

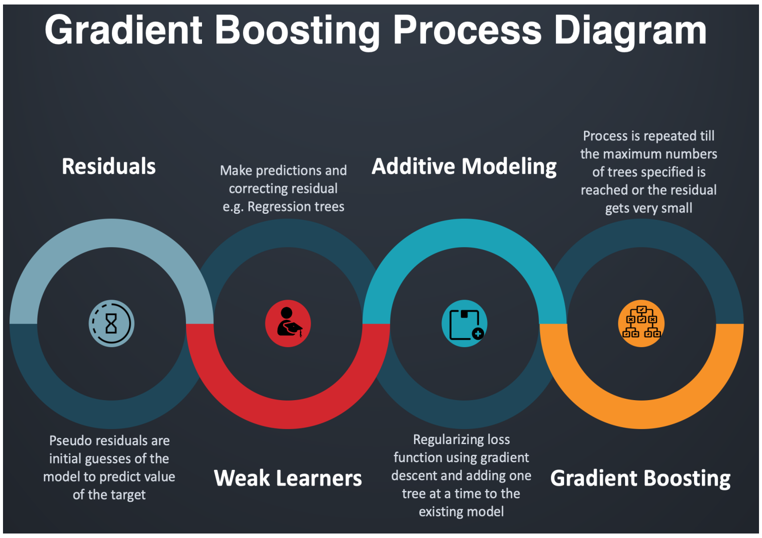
$$L = Min(\ sum\ of\ squared\ residuals\ +\ \alpha\ *\ |slope|\ ) \tag{2.3.2}$$

Where $Min(\ sum\ of\ squared\ residuals)$ is the Least Squared Error, and $\alpha * \mid slope \mid$ is the penalty term. However, α is the tuning parameter which controls the strength of the penalty term. In other words, α the tuning parameter is the value of shrinkage. $\mid slope \mid$ the sum of the absolute value of the coefficients.

2.3.4 XGBoost(Express Gradient Boosting)

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. [9]

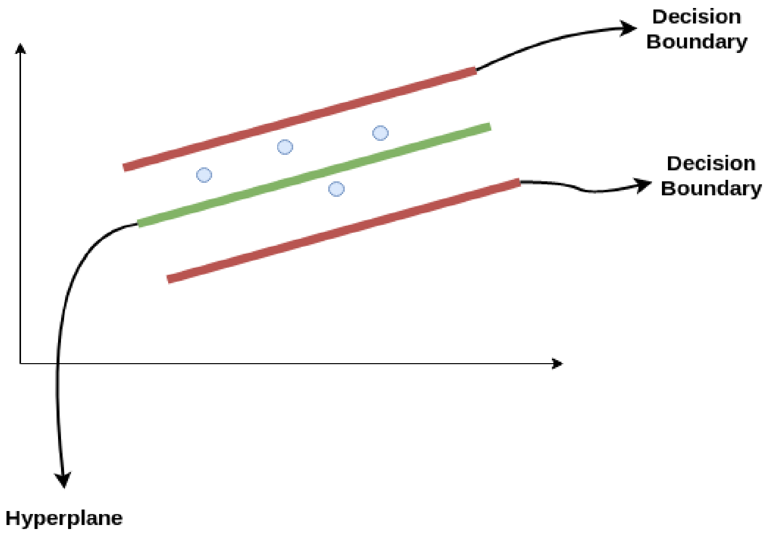
Thus it is a scalable Tree Boosting System. It is actually the implementation of gradient boosting trees designed for speed and performance.The gradient boosting decision tree algorithm is implemented in the XGBoost library. Boosting is an ensemble technique that adds new models to correct errors made by existing models. Models are added in a sequential order until no further advancements can be made. Gradient boosting is a method in which new models are created that predict the residuals or errors of prior models and are then combined to make the final prediction. It is called gradient boosting because it employs a gradient descent algorithm to reduce the loss while adding new models. [8]



2.3.5 SVM(Support Vector Machine)

It is a supervised machine learning algorithm used for both classification and regression. [6]It is very effective in high dimensional cases. In classification problem,the objective of SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

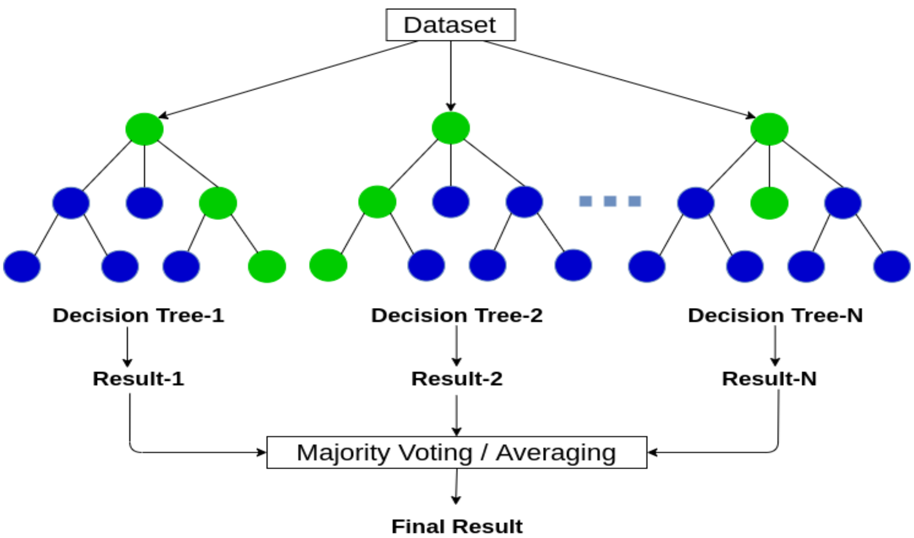
SVR(Support Vector Regression) is the implementation of SVM in regression problems. The problem of regression is to find a function that approximates mapping from an input domain to real numbers on the basis of a training sample



In the above figure the two red lines are the decision boundary and the green line is the hyperplane. The objective of SVR is to basically consider the points that are within the decision boundary line. Our best fit line is the hyperplane that has a maximum number of points. [7]

2.3.6 Random Forest Regressor

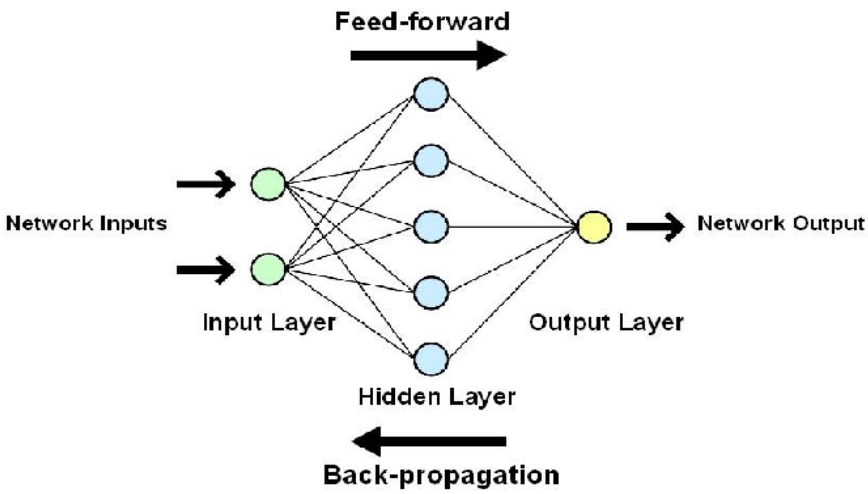
Random forest is an ensemble of decision trees. This is to say that many trees, constructed in a certain "random" way form a Random Forest. [5] In a random forest regressor each tree is created from a different sample of rows and at each node, a different sample of features is selected for splitting. Each of the trees makes its own individual prediction. These predictions are then averaged to produce a single result. The averaging makes a Random Forest better than a single Decision Tree hence improves its accuracy and reduces overfitting.



2.3.7 ANN(Artificial Neural Network)

Artificial neural network (ANN) is a simulation of the work of a biological brain.Just as the brain learns and evolves through the experiments that it faces through time to make decisions and predict the result of particular actions, ANN also tries to simulate the brain to learn the pattern in a given data to predict the output of that data.

ANN is based on an collection of connected elements or nodes called neurons. Neurons act as channels that take an input, process it, and then pass it to next neurons for further processing. This transaction or the process of transferring data between neurons is handled in various layers. A simple neural network consists of at least three layers that are input layer, one or more of hidden layers and an output layer. Each layer holds a set of neurons that takes input and process data and finally pass the output to other neurons in the next layer. This process is repetitive until the output layer has been reached, so that eventually, the result can be presented.
An ANN architecture is shown in the following figure



The data that is being held in each neuron is called activation. Activation value ranges from 0 to 1. As shown in the above figure, each neuron is linked to all neurons in the previous layer. Together, all activations from the first layer will decide if the activation will be triggered or not, which is done by taking all activations from the first layer and computing their weighted sum

$$w_1\ a_1 + w_2\ a_2 + w_3\ a_3 ... + w_n\ a_n \tag{2.3.3}$$

However, the output can be any number,although it should be only between 0 and 1. Thus, specifying the range of the output value to be within the accepted range. It can be done by using the Sigmoid function that will put the output to be ranging from 0 to 1. Then the bias is added for inactivity to the equation so it can limit the activation to when it is meaningfully active

$$\sigma(w_1\, a_1 + w_2\, a_2 + w_3\, a_3 ... + w_n\, a_n - b) \tag{2.3.4}$$

Where a_i is activation, w_i presents the weight, b is the bias and σ is the sigmoid function.

Nevertheless, after getting the final activation, its predicted value needs to be compared with the actual value. The difference between these values is considered as an error, and it is calculated with the cost function. The cost function helps to detect the error percentage in the model, which needs to be reduced. Applying back-propagation on the model reduces the error percentage by running the procedures backwards to check on how the weights and bias are affecting the cost function.
Back-propagation is simply the process of reversing the whole activations transference among neurons. The method calculates the gradient of the cost function concerning the weight. It is performed in the training stage of the feed-forward for supervised learning

Chapter 3

Experiment

3.1 Dataset Used

The dataset used for this project is taken from kaggle. It is uploaded there in the name of "House Prices - Advanced Regression Techniques". [3]In the project, we have only taken the train.csv file from the site and divided it into two parts, one for training purpose and another one for prediction and testing purpose.The dataset is split into 3:1 ratio.
It has 81 columns including "Id" that is being used to identify each house uniquely and the "Sale Price" that we will try to predict using various machine learning models. A screenshot of the dataset is attached below

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0

https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data

3.2 Evaluation metrics

The prediction accuracy will be evaluated by four common evaluation metrics i.e, Mean Absolute Error (MAE), Mean Squared Error (MSE), RMSE (Root Mean Square Error), R^2 Square. R^2 will show if the model is overfitted, whereas MAE, MSE, RMSE shoW the error percentage between the actual and predicted data, which in this case, the house prices.

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE) is the mean of the squared errors:

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

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

NOTE: Here y_i represents the actual value (SalePrice) and $\hat{y}_i$ represents the predicted value (SalePrice).

R^2 **Square** is the proportion of the variation in the dependent variable that is predictable from the independent variable(s). It is also known as the coefficient of determination.

If $R^2 = 0$, it indicates that the dependent variable cannot be predicted from the independent variable(s).

If $R^2 = 1$, it indicates the dependent variable can be predicted without error from the independent variable(s).

If R^2 lies in between 0 to 1 , it indicates the extent to which the dependent variable is predictable.

Comparing these metrics:

MAE is the easiest to understand, because it's the average error.

MSE is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.

RMSE is even more popular than MSE, because RMSE is interpretable in the "y" units.

The first three are loss functions, so we want to minimize them. And more is the value of R^2 Square better and more accurate will be our predictions.

3.3 Computer Specification

The needed time to train the model depends on the capability of the used system during the experiment. Some libraries use GPU resources over the CPU to take a shorter time to train a model.
The specification of the computer system on which this code is being implemented is given below:

1. **OPERATING SYSTEM** Windows 11
2. **PROCESSOR** Intel i5 10th generation
3. **RAM** 8 GB
4. **GRAPHICS CARD** NVIDIA MX110

3.4 Program Design

The algorithms used in this study have different properties that will be used during the implementation. The experiment is done with the Jupyter Notebook using Python programming language. However, in all algorithms, the data is split into four variables, namely; X_train, X_test, y_train and y_test, by using train_test_split class from the library sklearn.model_selection. In addition, in all algorithms, the train_test_split class takes as parameters the independent variables, which is the data, the dependent variable, which is the SalePrice, test_size = 0.25, and random_state = 0.

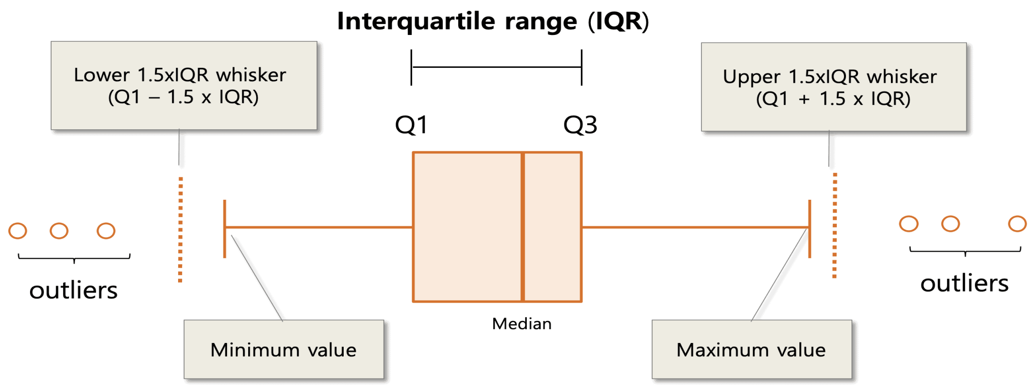
3.4.1 EDA (Exploratory Data Analysis)

The main motive of EDA is to get an overview of the numerical and categorical features of a dataset. In this project, python libraries like **pandas**, **matplotlib**, **seaborn** are imported and pre-defined functions are used for graphical representation of various features and to see the relationships in between them. Also we have used **heat map** to see the covariance between various numerical features. In the program we have used various graphs like scatter plot, box plot to give a pictorial description dataset and compare the accuracy of different models.

3.4.2 Outliers

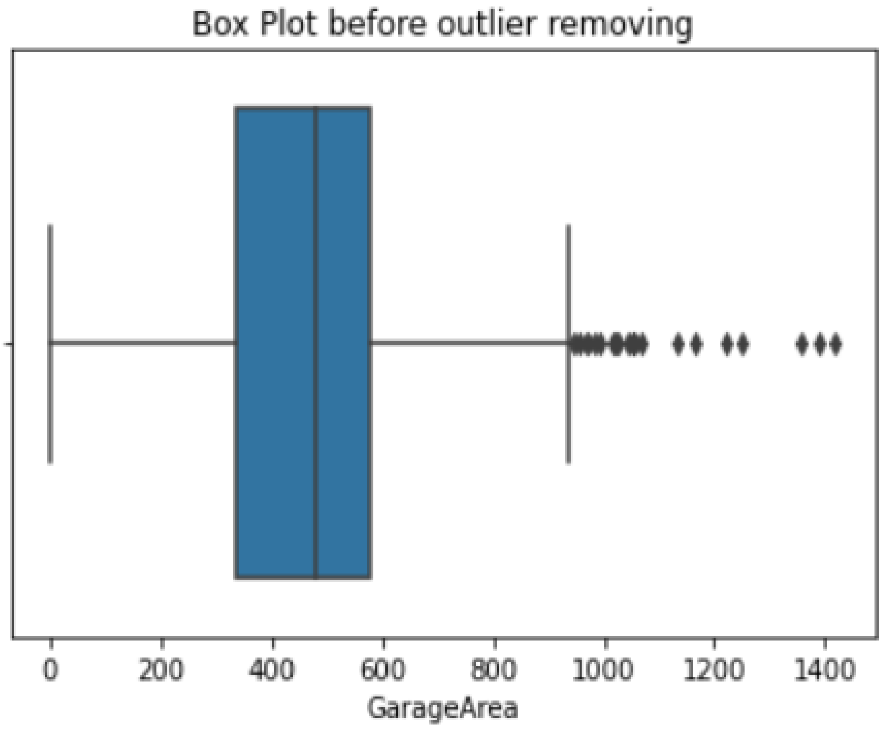
Outliers are those data points which differs significantly from other observations present in given dataset. It can occur because of variability in measurement and due to misinterpretation in filling data points. [4] They can be natural or because of entry errors, or due to measurement errors as well. In the project we have used **box plots** for visualising the outliers. Box plot is a graphical display for describing the distributions of the data.

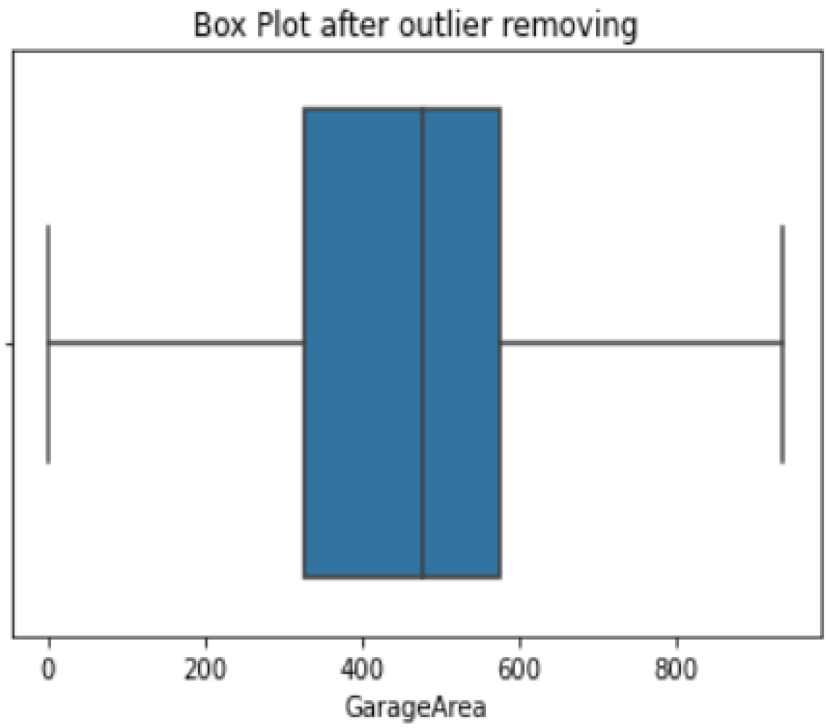
Box plot uses the median and the lower and upper quartiles to showcase the distribution of data points .




```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

ghar = pd.read_csv("...PROJECT\OTHERS\house-prices-advanced-regression-techniques\train.csv")
sns.boxplot(ghar['GarageArea'])
plt.title("Box Plot before removing outlier")
plt.show()

def drop_outliers(df, field_name):
    iqr = 1.5 * (np.percentile(df[field_name], 75) - np.percentile(df[field_name], 25))
    df.drop(df[df[field_name] > (iqr + np.percentile(df[field_name], 75))].index, inplace=True)
    df.drop(df[df[field_name] < (np.percentile(df[field_name], 25) - iqr)].index, inplace=True)

drop_outliers(ghar, 'GarageArea')
sns.boxplot(ghar['GarageArea'])
plt.title("Box Plot after outlier removing")
plt.show()
```

Listing 3.1: Python example





3.4.3 Train-Test Split of the dataset

The procedure of spliting a dataset involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

Train Dataset: Used to fit the machine learning model.

Test Dataset: Used to evaluate the fit machine learning model.

In the project we have used **sklearn.model_selection** library and **train_test_split** function to split the data into 3:1 ratio, which means we have used 75% of data for training the ML models and 25% for testing the accuracy of the predictions made by the ML models.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
    random_state=0, train_size = .75)
```

Listing 3.2: Program code for dataset split

Chapter 4

Results and Disscussion

In the beginining, we got to see that the dataset on which we were going to work had 81 columns including 'Id' that is used for identifying houses and 'SalePrice' which depicts the price of a house. On evaluation, we get to know that the used dataset has **38 Numerical features** and **43 Categorical features**.

```
In [6]:
numerical_feats = ghar.dtypes[ghar.dtypes != "object"].index
print("Number of Numerical features: ", len(numerical_feats))

categorical_feats = ghar.dtypes[ghar.dtypes == "object"].index
print("Number of Categorical features: ", len(categorical_feats))
```

```
Number of Numerical features:  38
Number of Categorical features:  43
```

We then drop all the columns having categorical features, since we wish to work on numerical features to predict the Sale Price of house.

Now, we have 38 columns which means 37 features (including Id) to predict the house price. But we can't use them without pre-processing, since on checking we get to see that there are three columns **(LotFrontage, GarageYrBlt, MasVnrArea**) having some missing values under them(**NULL VALUES**).

```
total = ghar.isnull().sum().sort_values(ascending=False)
percent = (ghar.isnull().sum()/ghar.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(10)
```

	Total	Percent
LotFrontage	259	0.177397
GarageYrBlt	81	0.055479
MasVnrArea	8	0.005479
Id	0	0.000000
OpenPorchSF	0	0.000000
KitchenAbvGr	0	0.000000
TotRmsAbvGrd	0	0.000000
Fireplaces	0	0.000000
GarageCars	0	0.000000
GarageArea	0	0.000000

Numerical features having null values

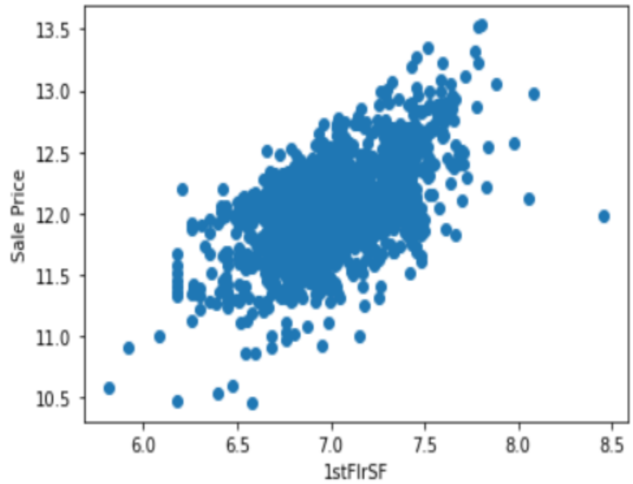
As we can see here, 'LotFrontage' has 259 null missing row entries under it out of 1460 entries that is around 17.7 % missing entries. Similarly, 'GarageYrBlt', 'MasVnrArea' have 81, 8 missing entries respectively. So, we drop these three columns. So, finally we have 34 features to predict the Sale Price of a house.

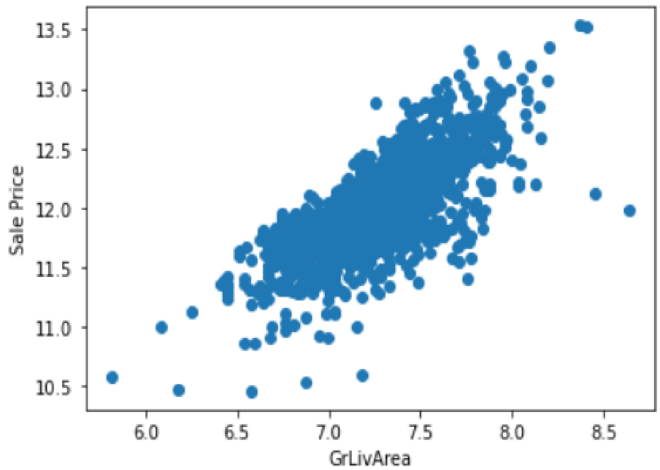
```
In [13]: ghar.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 35 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   LotArea        1460 non-null   int64
 3   OverallQual    1460 non-null   int64
 4   OverallCond    1460 non-null   int64
 5   YearBuilt      1460 non-null   int64
 6   YearRemodAdd   1460 non-null   int64
 7   BsmtFinSF1     1460 non-null   int64
 8   BsmtFinSF2     1460 non-null   int64
 9   BsmtUnfSF      1460 non-null   int64
 10  TotalBsmtSF    1460 non-null   int64
 11  1stFlrSF       1460 non-null   int64
 12  2ndFlrSF       1460 non-null   int64
 13  LowQualFinSF   1460 non-null   int64
 14  GrLivArea      1460 non-null   int64
 15  BsmtFullBath   1460 non-null   int64
 16  BsmtHalfBath   1460 non-null   int64
 17  FullBath       1460 non-null   int64
 18  HalfBath       1460 non-null   int64
 19  BedroomAbvGr   1460 non-null   int64
 20  KitchenAbvGr   1460 non-null   int64
 21  TotRmsAbvGrd   1460 non-null   int64
 22  Fireplaces     1460 non-null   int64
 23  GarageCars     1460 non-null   int64
 24  GarageArea     1460 non-null   int64
 25  WoodDeckSF     1460 non-null   int64
 26  OpenPorchSF    1460 non-null   int64
 27  EnclosedPorch  1460 non-null   int64
 28  3SsnPorch      1460 non-null   int64
 29  ScreenPorch    1460 non-null   int64
 30  PoolArea       1460 non-null   int64
 31  MiscVal        1460 non-null   int64
 32  MoSold         1460 non-null   int64
 33  YrSold         1460 non-null   int64
 34  SalePrice      1460 non-null   int64
dtypes: int64(35)
memory usage: 399.3 KB
```

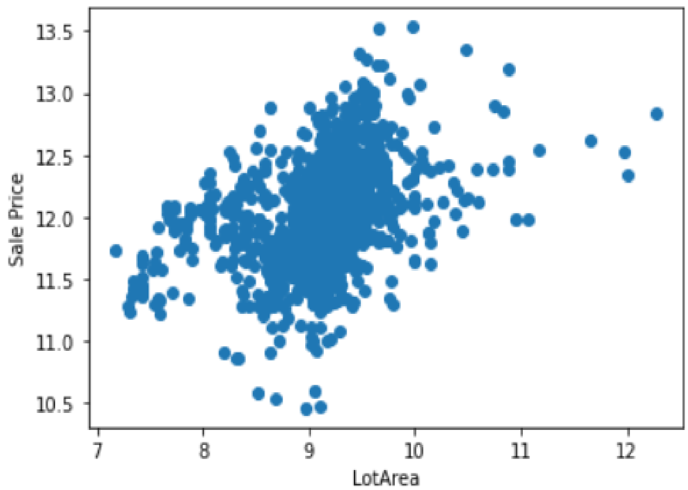
Features in the final dataset after droping unwanted features

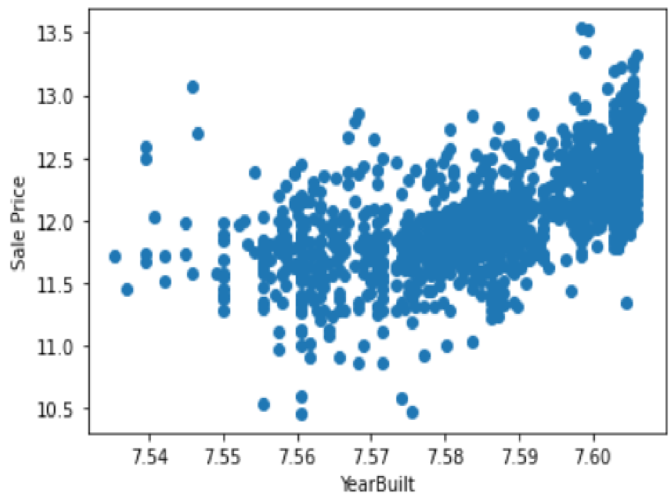
The code and it's output given above shows the features that we have retained for our work. Now we can see we have 1460 non-null entries.

During EDA, we plot several graphs to see the co-relations in between Sale Price and other features. The following scatter plots represent the relationship of '1stFlrSF', 'GrLivArea', 'LotArea', 'YearBuilt' with 'Sale Price'.

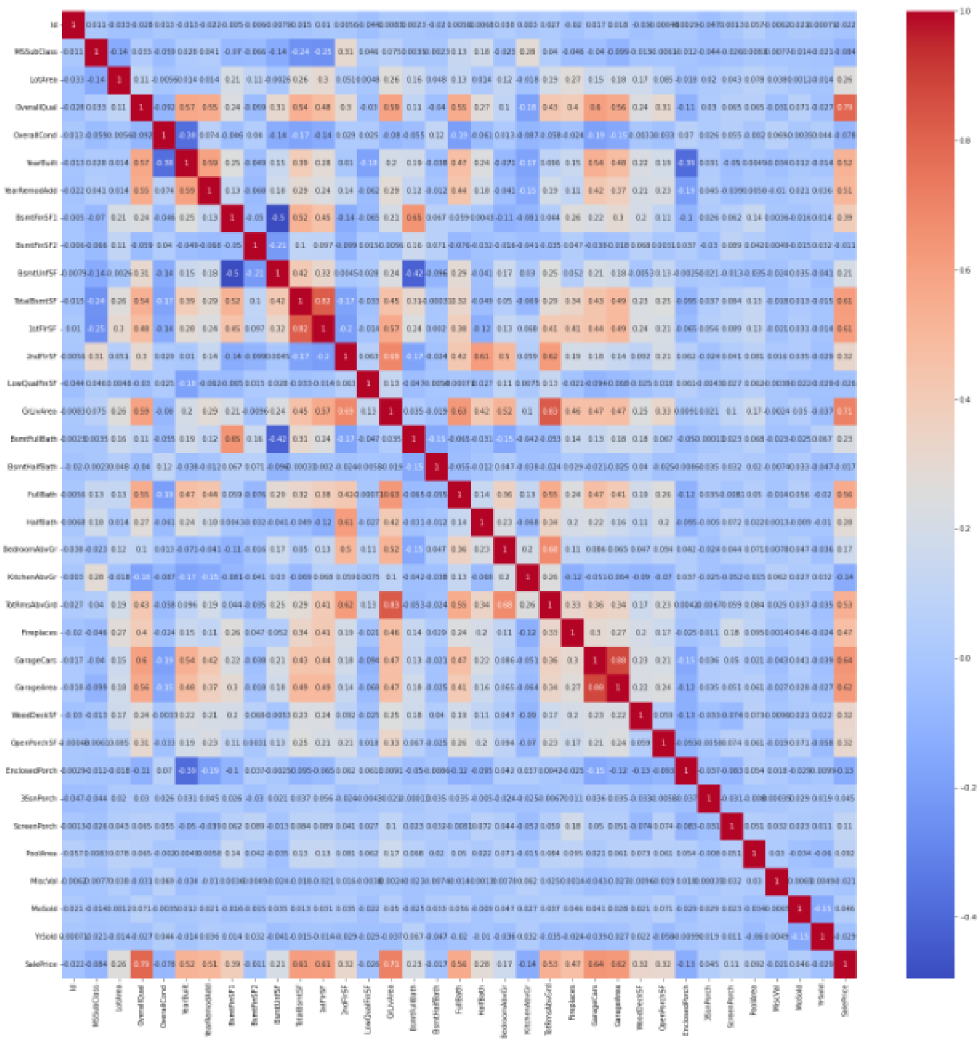









```
In [15]: plt.figure(figsize=(25,25))
         sns.heatmap(ghar.corr(),annot=True,cmap='coolwarm')
Out[15]: <AxesSubplot:>
```



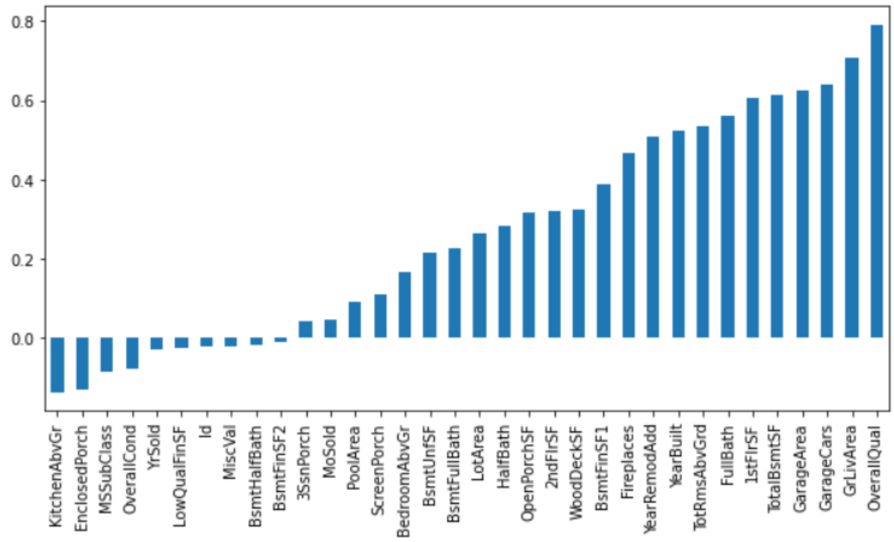
Heat Map of features in the dataset

The above figure is known as heat-map, it can be seen as a **variance-covariance matrix** between the features in the dataset. Deeper the red color more positive is the correlation between the features and deeper the blue color more will be the negative correlation between the features

Since we are interested in predicting the Sale Price of houses, we also plotted a **bar graph** to see the dependence of Sale Price with other features. The following graph depicts the same.

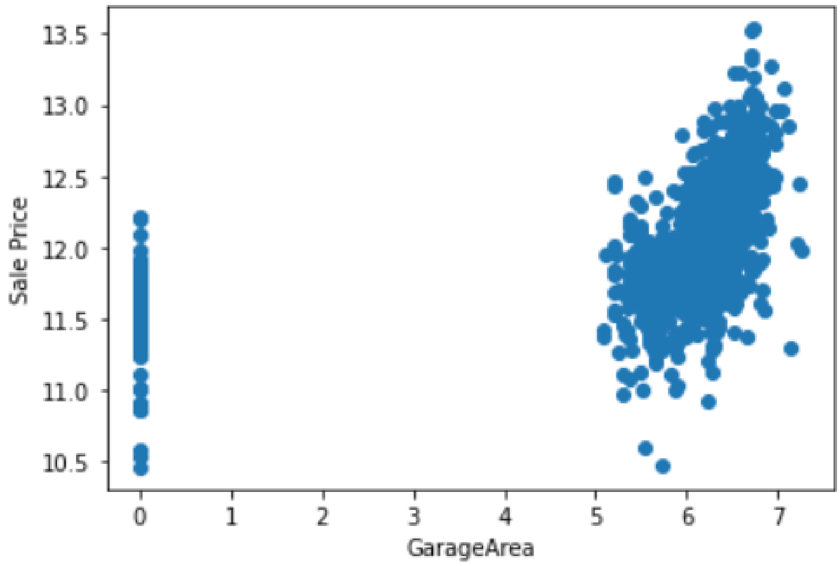
In [16]:
```
plt.figure(figsize=(10,5))
ghar.corr()['SalePrice'].sort_values().drop('SalePrice').plot(kind='bar')
```

Out[16]: <AxesSubplot:>

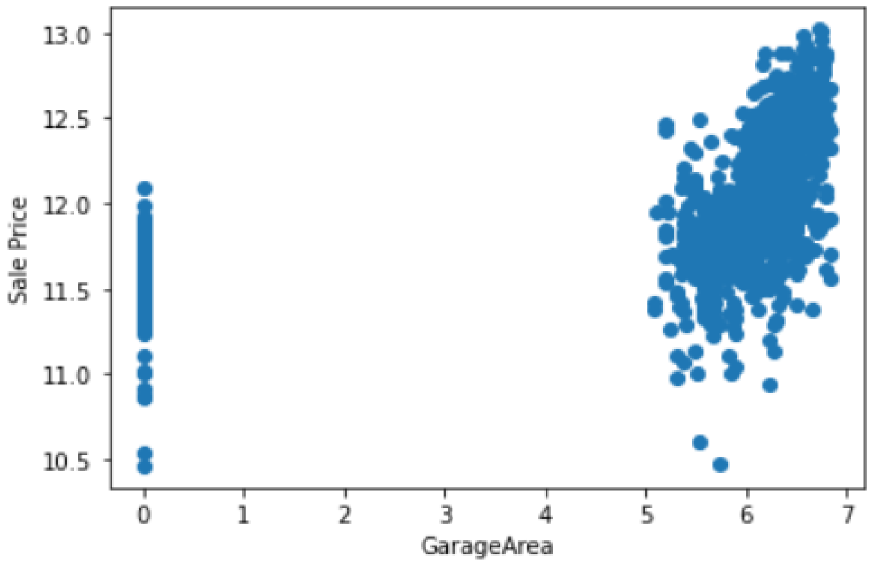


Bar graph of dependence of Sale Price upon features

Then we have removed outliers from certain features like 'GarageArea', 'GrLivArea', '1stFlrSF', 'TotalBsmtSF' and have visualised them by the help of box plots. Now, we have 1347 entries to work on with.



Before removing outliers



After removing outliers

Then, we split the dataset into two parts for training and testing ML models in the ratio of 3:1. Thus, 75% of data will be used for training the ML models and 25% for testing the accuracy of the predictions made by the ML models

Then we define some functions for our evaluation metrics using predefined fuctions from **sklearn** library. This helps us to aviod repeatation of same block of codes again and again in the program.

evaluate() function is defined to evaluate the all the four required metrics that we are going to use to compare the accuracy of ML models. **print_evaluate** function is defined to print the results.

```
import numpy as np
from sklearn import metrics

def print_evaluate(true, predicted):
    mae = metrics.mean_absolute_error(true, predicted)
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    r2_square = metrics.r2_score(true, predicted)
    print('MAE:', mae)
    print('MSE:', mse)
    print('RMSE:', rmse)
    print('R2 Square', r2_square)
    print('__________________________________')

def evaluate(true, predicted):
    mae = metrics.mean_absolute_error(true, predicted)
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    r2_square = metrics.r2_score(true, predicted)
    return mae, mse, rmse, r2_square
```

Listing 4.1: Code for defining evaluation metrics

ML MODELS IMPLEMENTATION

We begin by **Multiple linear regression**. The code given below fits multiple linear regression by the help of training data that is X_train and y_train then, it is used to predict the Sale Price of the houses present in the testing dataset that is X_test. Finally by the help of our user-defined functions, we evaluate the accuracy of the multiple linear regression model and store it into a dataframe results_df so that we can compare all the results later, altogether.

```python
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression(normalize=True)
lin_reg.fit(X_train,y_train)

test_pred = lin_reg.predict(X_test)
train_pred = lin_reg.predict(X_train)

print('Test set evaluation:\n_________________________________')
print_evaluate(y_test, test_pred)
print('Train set evaluation:\n_________________________________')
print_evaluate(y_train, train_pred)

results_df = pd.DataFrame(data=[["Linear Regression", *evaluate(y_test, test_pred) ]],
            columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
```

Listing 4.2: Code for Multiple linear regression

The output that we got is as follows:

```
Test set evaluation:
_____________________________________
MAE: 18031.09994805906
MSE: 654754851.4198682
RMSE: 25588.17796209547
R2 Square 0.8616161915916168
__________________________________
Train set evaluation:
_____________________________________
MAE: 16594.941404038913
MSE: 508797862.3608397
RMSE: 22556.548103839817
R2 Square 0.8728078724387939
__________________________________
```

Then, we have applied **Ridge regression** after importing predefined fuctions from **sklearn.linear_model** library. Again we apply it to the training data and check the accuracy by applying our user-defined fuctions on testing data.

```python
from sklearn.linear_model import Ridge, RidgeCV

alphas = np.geomspace(1e-9, 5, num=100)

ridgecv = RidgeCV(alphas = alphas, scoring = 'neg_mean_squared_error', normalize = True)
ridgecv.fit(X_train, y_train)

ridge = Ridge(alpha = ridgecv.alpha_, normalize = True)
ridge.fit(X_train, y_train)

print('Ridge Regression:')
print("Alpha =", ridgecv.alpha_)

test_pred = ridge.predict(X_test)
train_pred = ridge.predict(X_train)

print('Test set evaluation:\n_________________________________')
print_evaluate(y_test, test_pred)
print('Train set evaluation:\n_________________________________')
print_evaluate(y_train, train_pred)

results_df_2 = pd.DataFrame(data=[["Ridge", *evaluate(y_test, test_pred)]],
              columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df = results_df.append(results_df_2, ignore_index=True)
```

Listing 4.3: Code for Ridge regression

The output that we got is as follows:

```
Ridge Regression:
Alpha = 0.03496578933827813
Test set evaluation:
_____________________________________
MAE: 17941.203909029748
MSE: 659197222.3203398
RMSE: 25674.836364042123
R2 Square 0.8606772872028419
__________________________________
Train set evaluation:
_____________________________________
MAE: 16598.578018622953
MSE: 510812130.3043013
RMSE: 22601.1532958896
R2 Square 0.8723043345032797
__________________________________
```

Then, we applied **LASSO** by importing predefined fuctions from **sklearn.linear_model** library. Again we apply it to the training data and check the accuracy by applying our user-defined fuctions on testing data.

```python
from sklearn.linear_model import Lasso, LassoCV

lasso = Lasso(max_iter = 100000, normalize = True)

lassocv = LassoCV(alphas = None, cv = 10, max_iter = 100000, normalize = True)
lassocv.fit(X_train, y_train)

lasso.set_params(alpha=lassocv.alpha_)
lasso.fit(X_train, y_train)

test_pred = lasso.predict(X_test)
train_pred = lasso.predict(X_train)

print('Test set evaluation:\n_________________________________')
print_evaluate(y_test, test_pred)
print('Train set evaluation:\n_________________________________')
print_evaluate(y_train, train_pred)

print("Alpha =", lassocv.alpha_)

results_df_3 = pd.DataFrame(data=[["Lasso", *evaluate(y_test, test_pred)]],
                            columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df = results_df.append(results_df_3, ignore_index=True)
```

Listing 4.4: Code of LASSO

The output that we got is as follows:

```
Test set evaluation:
_____________________________________
MAE: 17924.41488047078
MSE: 654162960.59647
RMSE: 25576.609638426864
R2 Square 0.8617412889561131
__________________________________
Train set evaluation:
_____________________________________
MAE: 16622.69823507287
MSE: 513233502.30069155
RMSE: 22654.657408592422
R2 Square 0.871699026425905
__________________________________
Alpha = 11.21941092502285З
```

Then, we applied **XGBRegressor** by importing predefined fuctions from **xgboost** library. Again we apply it to the training data and check the accuracy by applying our user-defined fuctions on testing data.

```python

from numpy import loadtxt
from xgboost import XGBRegressor

model = XGBRegressor()
model.fit(X_train, y_train)

test_pred = model.predict(X_test)
train_pred = model.predict(X_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)

print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)

results_df_4 = pd.DataFrame(data=[["XGBRegressor", *evaluate(y_test, test_pred), 0]],
                            columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square', 'Cross Validation'])
results_df = results_df.append(results_df_4, ignore_index=True)
```

Listing 4.5: Code of XGBRegressor

The output that we got is as follows:

```
Test set evaluation:
_____________________________________
MAE: 17178.648901149852
MSE: 613590323.5517366
RMSE: 24770.755409388235
R2 Square 0.8703164007238927
__________________________________
Train set evaluation:
_____________________________________
MAE: 574.0484452351485
MSE: 628525.4767664843
RMSE: 792.7959868506426
R2 Square 0.9998428776955834
__________________________________
```

Then, we applied **Random Forest Regressor** by importing predefined fuctions from **sklearn.linear_model** library. We apply it to the training data and check the accuracy by applying our user-defined fuctions on testing data.

```
from sklearn.ensemble import RandomForestRegressor

rf_reg = RandomForestRegressor(n_estimators=1000)
rf_reg.fit(X_train, y_train)

test_pred = rf_reg.predict(X_test)
train_pred = rf_reg.predict(X_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)

print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)

results_df_5 = pd.DataFrame(data=[["Random Forest Regressor", *
    evaluate(y_test, test_pred)]],
                columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df = results_df.append(results_df_2, ignore_index=True)
```

Listing 4.6: Code of Random Forest Regressor

The output that we got is as follows:

```
Test set evaluation:
_____________________________________
MAE: 15986.833023738873
MSE: 556869679.7667505
RMSE: 23598.086358150962
R2 Square 0.8823044275179215
__________________________________
Train set evaluation:
_____________________________________
MAE: 5893.06822079208
MSE: 72930242.0443673
RMSE: 8539.920494030803
R2 Square 0.9817684913098197
__________________________________
```

Then, we have applied **Support Vector Machine** (**Support Vector Regression**, since here it is used for regression) after importing predefined fuctions from **sklearn.linear_model** library. Again we apply it to the training data and check the accuracy by applying our user-defined fuctions on testing data.

```
from sklearn.svm import SVR

svm_reg = SVR(kernel='rbf', C=1000000, epsilon=0.001)
svm_reg.fit(X_train, y_train)

test_pred = svm_reg.predict(X_test)
train_pred = svm_reg.predict(X_train)

print('Test set evaluation:\n_______________________________')
print_evaluate(y_test, test_pred)

print('Train set evaluation:\n_______________________________')
print_evaluate(y_train, train_pred)

results_df_3 = pd.DataFrame(data=[["SVM Regressor", *evaluate(
    y_test, test_pred)]],
            columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df = results_df.append(results_df_3, ignore_index=True)
```

Listing 4.7: Code of SVM

The output that we got is as follows:

```
Test set evaluation:
_____________________________________
MAE: 23228.650822315394
MSE: 1136136310.904815
RMSE: 33706.62117306947
R2 Square 0.7598752124812613
__________________________________
Train set evaluation:
_____________________________________
MAE: 21057.707781303285
MSE: 913885321.5834868
RMSE: 30230.536243730225
R2 Square 0.7715418499208946
__________________________________
```

Finally, we used Artificial Neural Network

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense, Activation,
    Dropout
from tensorflow.keras.optimizers import Adam
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)

model = Sequential()

model.add(Dense(X_train.shape[1],activation='relu'))
model.add(Dense(32,activation='relu'))
# model.add(Dropout(0.2))

model.add(Dense(64,activation='relu'))
# model.add(Dropout(0.2))

model.add(Dense(128,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1))

model.compile(optimizer=Adam(0.001), loss='mse')

r = model.fit(X_train, y_train,
              validation_data=(X_test,y_test),
              batch_size=128,
              epochs=50)

test_pred = model.predict(X_test)
train_pred = model.predict(X_train)

print('Test set evaluation:\n_____________________________')
print_evaluate(y_test, test_pred)

print('Train set evaluation:\n_____________________________')
print_evaluate(y_train, train_pred)

results_df_4 = pd.DataFrame(data=[["Artficial Neural Network", *
    evaluate(y_test, test_pred)]],
                            columns=['Model', 'MAE', 'MSE', 'RMSE',
     'R2 Square'])
results_df = results_df.append(results_df_4, ignore_index=True)
```

Listing 4.8: Code of ANN

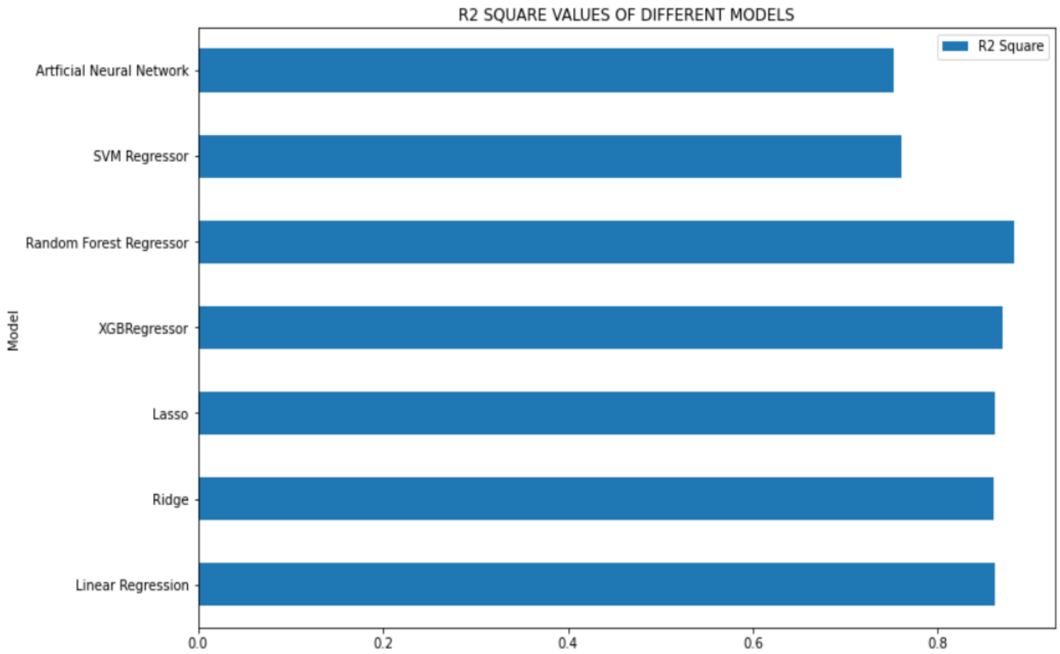
The output that we got is as follows:

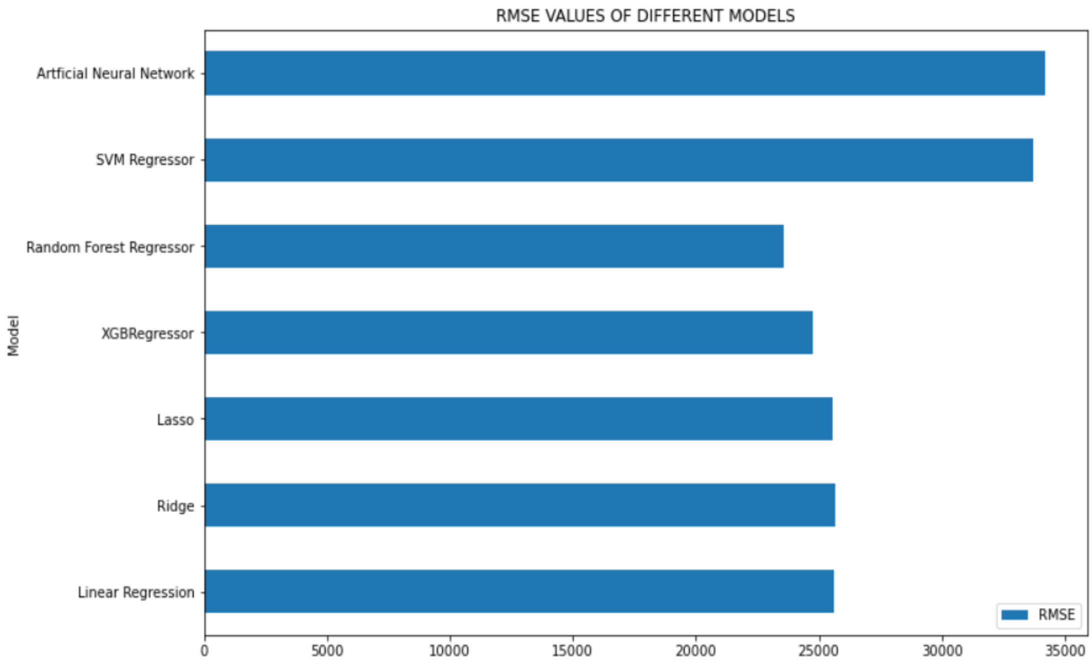
```
Test set evaluation:
_____________________________________
MAE: 24946.108772255193
MSE: 1232085008.6104596
RMSE: 35101.068482461604
R2 Square 0.7395962543772652
__________________________________
Train set evaluation:
_____________________________________
MAE: 22945.703991336635
MSE: 985160327.001907
RMSE: 31387.263770547233
R2 Square 0.7537241265149027
__________________________________
```

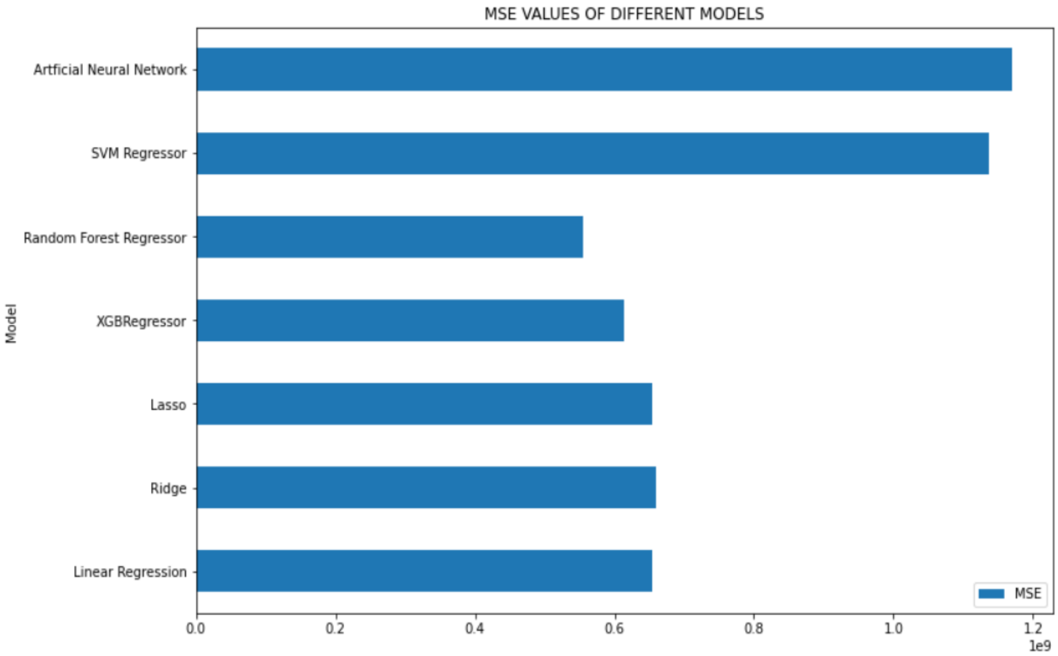
After using these ML models, we finally pictorially represent the values of evaluation metrics that we obtained. If we compare them, we can see that **Random Forest Regressor** has the **maximum** R^2 **Square value** and **minimum Mean Absolute Error (MAE), Mean Squared Error (MSE), RMSE (Root Mean Square Error)**.

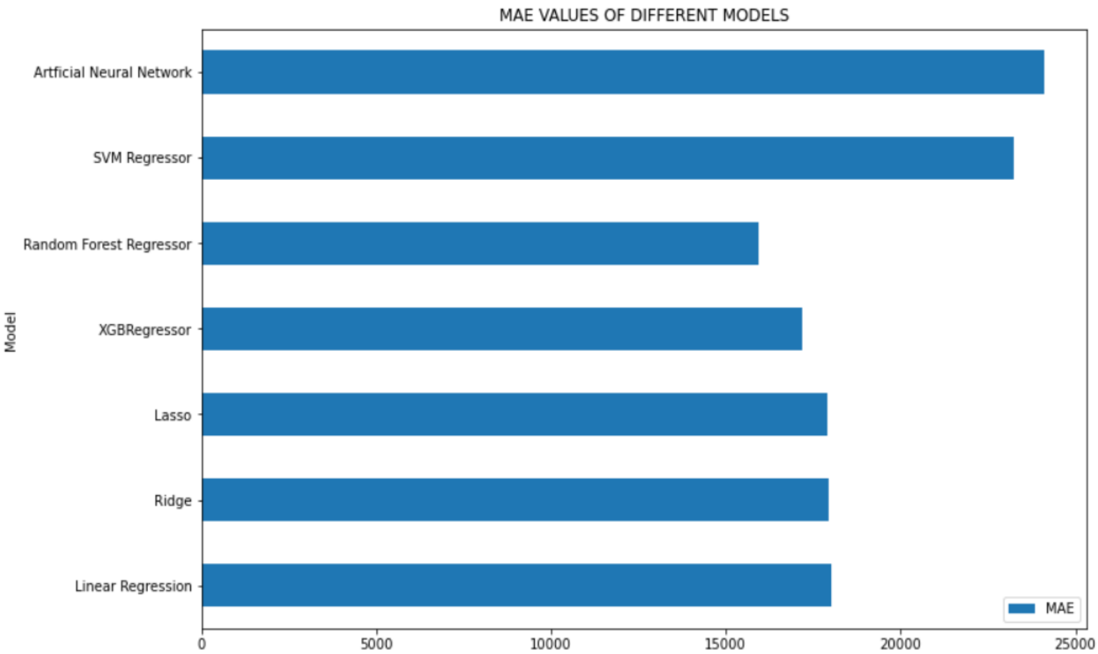
```
results_df.plot(x='Model', y='R2 Square', kind='barh', figsize=(12, 8),title="R2 SQUARE VALUES OF DIFFERENT MODELS")

results_df.plot(x='Model', y='RMSE', kind='barh', figsize=(12, 8), title="RMSE VALUES OF DIFFERENT MODELS")

results_df.plot(x='Model', y='MSE', kind='barh', figsize=(12, 8), title="MSE VALUES OF DIFFERENT MODELS")

results_df.plot(x='Model', y='MAE', kind='barh', figsize=(12, 8), title="MAE VALUES OF DIFFERENT MODELS")
```

Listing 4.9: Code for pictorial representation of evaluation metrics


R2 SQUARE VALUES OF DIFFERENT MODELS
R2 Square
Model
Artficial Neural Network
SVM Regressor
Random Forest Regressor
XGBRegressor
Lasso
Ridge
Linear Regression
0.0
0.2
0.4
0.6
0.8


RMSE VALUES OF DIFFERENT MODELS
RMSE
Model
Artficial Neural Network
SVM Regressor
Random Forest Regressor
XGBRegressor
Lasso
Ridge
Linear Regression
0
5000
10000
15000
20000
25000
30000
35000





From the above bar graphs, we can see that **Random Forest Regressor** performs better than other ML model used in the project.

Chapter 5

Conclusion

After running the python program, we compared the ML models that we had taken. Finally, the result we got is as follows:

	Model	MAE	MSE	RMSE	R2 Square
0	Linear Regression	18031.099948	6.547549e+08	25588.177962	0.861616
1	Ridge	17941.203909	6.591972e+08	25674.836364	0.860677
2	Lasso	17924.414880	6.541630e+08	25576.609638	0.861741
3	XGBRegressor	17178.648901	6.135903e+08	24770.755409	0.870316
4	Random Forest Regressor	15921.487059	5.546648e+08	23551.321735	0.882770
5	SVM Regressor	23228.650822	1.136136e+09	33706.621173	0.759875
6	Artficial Neural Network	24120.632789	1.170360e+09	34210.526267	0.752642

From the results, we are getting a MAE of about \$ 15,921 in the prediction by Random Forest regressor that is the least among all models used in this project. This means on an average the predictions of house price differ from the actual price by \$ 15,921. Whereas the highest MAE we got is of \$ 24,120 , that was from ANN which makes it the poorest ML model used for prediction, at least under these conditions that were taken in this project. Usually, ANN requires a lot of training data to get a good accuracy during testing.

The final results of this project showed that Random Forest regressor makes better prediction as compared to other used algorithms since it gives us the least MAE, MSE, RMSE and the highest R^2 Square values.

Although this study has shown that **Random forest Regressor** makes the best prediction, one cannot guarantee that it will perform the same when used for other purposes than the ones that have been presented in this study. Infact, if we use different parameters for ML models like SVM, ANN, then their performance might increase too. Also, we can model the data, instead of deleting the features having some NULL entries, we could fill them up with the mean or mode of all the observed values so that we get more number of predictors. This can also boost the accuracy of some ML models.

5.1 Ethics

This project is taking into consideration the ethical part, where the dataset is downloaded from a public website called Kaggle. Kaggle allows users to find and publish data sets, explore and build models in a web-based data-science environment, and work with other data scientists.In this project, the algorithms are public and open source. In addition, the algorithms are trained and tested on the same public dataset.

5.2 Future Work

There is a great scope for improvement in accuracy of prediction. Future work on this project could be done by:

- Changing the way we have handled missing values. We can fill them up by mean or mode of observations present in that column. By this we will have more number of predictors which can also boost the accuracy of some ML models.
- Making use of categorical features which have not used in this project.
- Tweeking with the ANN model by using different parameters that can help it train the model better.
- The used pre-processing methods do help in the prediction accuracy. However, experimenting with different combinations of pre-processing methods to achieve better prediction accuracy.

Appendix A

Features

The list of all features present in the dataset is as follows:

FEATURE	DESCRIPTION
SalePrice	the property's sale price in dollars
MSSubClass	The building class
MSZoning	The general zoning classification
LotFrontage	Linear feet of street connected to property
LotArea	Lot size in square feet
Street	Type of road access
Alley	Type of alley access
LotShape	General shape of property
LandContour	Flatness of the property
Utilities	Type of utilities available
LotConfig	Lot configuration
LandSlope	Slope of property
Neighborhood	Physical locations within Ames city limits
Condition1	Proximity to main road or railroad
Condition2	Proximity to main road or railroad (if a second is present)
BldgType	Type of dwelling
HouseStyle	Style of dwelling
OverallQual	Overall material and finish quality
OverallCond	Overall condition rating
YearBuilt	Original construction date
YearRemodAdd	Remodel date
RoofStyle	Type of roof
RoofMatl	Roof material

FEATURE	**DESCRIPTION**
Exterior1st	Exterior covering on house
Exterior2nd	Exterior covering on house (if more than one material)
MasVnrType	Masonry veneer type
MasVnrArea	Masonry veneer area in square feet
ExterQual	Exterior material quality
ExterCond	Present condition of the material on the exterior
Foundation	Type of foundation
BsmtQual	Height of the basement
BsmtCond	General condition of the basement
BsmtExposure	Walkout or garden level basement walls
BsmtFinType1	Quality of basement finished area
BsmtFinSF1	Type 1 finished square feet
BsmtFinType2	Quality of second finished area (if present)
BsmtFinSF2	Type 2 finished square feet
BsmtUnfSF	Unfinished square feet of basement area
TotalBsmtSF	Total square feet of basement area
Heating	Type of heating
HeatingQC	Heating quality and condition
CentralAir	Central air conditioning
Electrical	Electrical system
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	Number of bedrooms above basement level
Kitchen	Number of kitchens
KitchenQual	Kitchen quality
TotRmsAbvGrd	Total rooms above grade (does not include bathrooms)
Functional	Home functionality rating
Fireplaces	Number of fireplaces

FEATURE	**DESCRIPTION**
FireplaceQu	Fireplace quality
GarageType	Garage location
GarageYrBlt	Year garage was built
GarageFinish	Interior finish of the garage
GarageCars	Size of garage in car capacity
GarageArea	Size of garage in square feet
GarageQual	Garage quality
GarageCond	Garage condition
PavedDrive	Paved driveway
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
Fence	Fence quality
MiscFeature	Miscellaneous feature not covered in other categories
MiscVal	Value of miscellaneous feature
MoSold	Month Sold
YrSold	Year Sold
SaleType	Type of sale
SaleCondition	Condition of sale

Appendix B

Python Code

Below is the full python code that we have made to implement and evaluate the accuracy of various ML models.

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

ghar = pd.read_csv(r"C:\Users\PRANAB\Desktop\B.Sc FINAL YR PROJECT\
    OTHERS\house-prices-advanced-regression-techniques\train.csv")
ghar.head()

ghar.info()

ghar.describe()

ghar.columns


numerical_feats = ghar.dtypes[ghar.dtypes != "object"].index
print("Number of Numerical features: ", len(numerical_feats))

categorical_feats = ghar.dtypes[ghar.dtypes == "object"].index
print("Number of Categorical features: ", len(categorical_feats))



print(ghar[numerical_feats].columns)
print("="*100)
print(ghar[categorical_feats].columns)

```

```
for i in categorical_feats:
    ghar.drop(i, inplace=True, axis=1)

ghar.columns

ghar.shape

total = ghar.isnull().sum().sort_values(ascending=False)
percent = (ghar.isnull().sum()/ghar.isnull().count()).sort_values(
    ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', '
    Percent'])
missing_data.head(10)

ghar.drop(['LotFrontage','GarageYrBlt','MasVnrArea'],axis=1,inplace
    =True)

ghar.info()

#EDA

features= []
for i in ghar.columns:
    features.append(i)

for feature in features:
    data=ghar.copy()
    data['SalePrice']=np.log(data['SalePrice']+1)
    data[feature]=np.log(data[feature]+1)
    plt.scatter(data[feature], data['SalePrice'])
    plt.xlabel(feature)
    plt.ylabel('Sale Price')
    plt.show()

plt.figure(figsize=(25,25))
sns.heatmap(ghar.corr(),annot=True,cmap='coolwarm')

plt.figure(figsize=(10,5))
ghar.corr()['SalePrice'].sort_values().drop('SalePrice').plot(kind=
    'bar')

import warnings
warnings.filterwarnings("ignore")

#DATA PREPROCESSING AGAIN OUTLIER

```

```
sns.boxplot(ghar['GarageArea'])
plt.title("Box Plot before outlier removing")
plt.show()

def drop_outliers(df, field_name):
    iqr = 1.5 * (np.percentile(df[field_name], 75) - np.percentile(
   df[field_name], 25))
    df.drop(df[df[field_name] > (iqr + np.percentile(df[field_name
   ], 75))].index, inplace=True)
    df.drop(df[df[field_name] < (np.percentile(df[field_name], 25)
   - iqr)].index, inplace=True)
drop_outliers(ghar, 'GarageArea')
sns.boxplot(ghar['GarageArea'])
plt.title("Box Plot after outlier removing")
plt.show()



sns.boxplot(ghar['GrLivArea'])
plt.title("Box Plot before outlier removing")
plt.show()

def drop_outliers(df, field_name):
    iqr = 1.5 * (np.percentile(df[field_name], 75) - np.percentile(
   df[field_name], 25))
    df.drop(df[df[field_name] > (iqr + np.percentile(df[field_name
   ], 75))].index, inplace=True)
    df.drop(df[df[field_name] < (np.percentile(df[field_name], 25)
   - iqr)].index, inplace=True)
drop_outliers(ghar, 'GrLivArea')
sns.boxplot(ghar['GrLivArea'])
plt.title("Box Plot after outlier removing")
plt.show()



sns.boxplot(ghar['1stFlrSF'])
plt.title("Box Plot before outlier removing")
plt.show()

def drop_outliers(df, field_name):
    iqr = 1.5 * (np.percentile(df[field_name], 75) - np.percentile(
   df[field_name], 25))
    df.drop(df[df[field_name] > (iqr + np.percentile(df[field_name
   ], 75))].index, inplace=True)
    df.drop(df[df[field_name] < (np.percentile(df[field_name], 25)
   - iqr)].index, inplace=True)
```

```
drop_outliers(ghar, '1stFlrSF')
sns.boxplot(ghar['1stFlrSF'])
plt.title("Box Plot after outlier removing")
plt.show()



sns.boxplot(ghar['TotalBsmtSF'])
plt.title("Box Plot before outlier removing")
plt.show()

def drop_outliers(df, field_name):
    iqr = 1.5 * (np.percentile(df[field_name], 75) - np.percentile(
    df[field_name], 25))
    df.drop(df[df[field_name] > (iqr + np.percentile(df[field_name
    ], 75))].index, inplace=True)
    df.drop(df[df[field_name] < (np.percentile(df[field_name], 25)
    - iqr)].index, inplace=True)
drop_outliers(ghar, 'TotalBsmtSF')
sns.boxplot(ghar['TotalBsmtSF'])
plt.title("Box Plot after outlier removing")
plt.show()


ghar.info()

#AGAIN VISUALISATION

plt.figure(figsize=(10,5))
ghar.corr()['SalePrice'].sort_values().drop('SalePrice').plot(kind=
    'bar')



X = ghar.loc[:, ghar.columns!='SalePrice']

y = ghar['SalePrice']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
    random_state=0, train_size = .75)

from sklearn import metrics

def print_evaluate(true, predicted):
    mae = metrics.mean_absolute_error(true, predicted)
```

```
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    r2_square = metrics.r2_score(true, predicted)
    print('MAE:', mae)
    print('MSE:', mse)
    print('RMSE:', rmse)
    print('R2 Square', r2_square)
    print('__________________________________')

def evaluate(true, predicted):
    mae = metrics.mean_absolute_error(true, predicted)
    mse = metrics.mean_squared_error(true, predicted)
    rmse = np.sqrt(metrics.mean_squared_error(true, predicted))
    r2_square = metrics.r2_score(true, predicted)
    return mae, mse, rmse, r2_square



#MODELS
#LINEAR REGRESSION
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression(normalize=True)
lin_reg.fit(X_train,y_train)

test_pred = lin_reg.predict(X_test)
train_pred = lin_reg.predict(X_train)

print('Test set evaluation:\n_____________________________________'
   )
print_evaluate(y_test, test_pred)
print('Train set evaluation:\n_____________________________________
   ')
print_evaluate(y_train, train_pred)

results_df = pd.DataFrame(data=[["Linear Regression", *evaluate(
   y_test, test_pred) ]],
                          columns=['Model', 'MAE', 'MSE', 'RMSE', '
   R2 Square'])

#RIDGE
from sklearn.linear_model import Ridge, RidgeCV

alphas = np.geomspace(1e-9, 5, num=100)

ridgecv = RidgeCV(alphas = alphas, scoring = '
   neg_mean_squared_error', normalize = True)
```

```
ridgecv.fit(X_train, y_train)

ridge = Ridge(alpha = ridgecv.alpha_, normalize = True)
ridge.fit(X_train, y_train)

print('Ridge Regression:')
print("Alpha =", ridgecv.alpha_)

test_pred = ridge.predict(X_test)
train_pred = ridge.predict(X_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)
print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)



results_df_2 = pd.DataFrame(data=[["Ridge", *evaluate(y_test, test_pred)]],
                            columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df = results_df.append(results_df_2, ignore_index=True)



#LASSO
from sklearn.linear_model import Lasso, LassoCV

lasso = Lasso(max_iter = 100000, normalize = True)

lassocv = LassoCV(alphas = None, cv = 10, max_iter = 100000, normalize = True)
lassocv.fit(X_train, y_train)

lasso.set_params(alpha=lassocv.alpha_)
lasso.fit(X_train, y_train)


test_pred = lasso.predict(X_test)
train_pred = lasso.predict(X_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)
```

```
print('Train set evaluation:\n_____________________________________
    ')
print_evaluate(y_train, train_pred)


print("Alpha =", lassocv.alpha_)

results_df_3 = pd.DataFrame(data=[["Lasso", *evaluate(y_test,
    test_pred)]],
                            columns=['Model', 'MAE', 'MSE', 'RMSE',
     'R2 Square'])
results_df = results_df.append(results_df_3, ignore_index=True)


#XGB
from numpy import loadtxt
from xgboost import XGBRegressor

# fit model no training data
model = XGBRegressor()
model.fit(X_train, y_train)
# make predictions for test data
#test_pred = model.predict(X_test)
#predictions = [round(value) for value in y_pred]
# evaluate predictions


test_pred = model.predict(X_test)
train_pred = model.predict(X_train)

print('Test set evaluation:\n_____________________________________'
    )
print_evaluate(y_test, test_pred)

print('Train set evaluation:\n_____________________________________
    ')
print_evaluate(y_train, train_pred)

results_df_4 = pd.DataFrame(data=[["XGBRegressor", *evaluate(y_test
    , test_pred)]],
                            columns=['Model', 'MAE', 'MSE', 'RMSE',
     'R2 Square'])
results_df = results_df.append(results_df_4, ignore_index=True)

#RANDOM FOREST REGRESSOR
from sklearn.ensemble import RandomForestRegressor

```

```
rf_reg = RandomForestRegressor(n_estimators=1000)
rf_reg.fit(X_train, y_train)

test_pred = rf_reg.predict(X_test)
train_pred = rf_reg.predict(X_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)

print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)

results_df_5 = pd.DataFrame(data=[["Random Forest Regressor", *evaluate(y_test, test_pred)]],
                            columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df = results_df.append(results_df_5, ignore_index=True)

#SVM
from sklearn.svm import SVR

svm_reg = SVR(kernel='rbf', C=1000000, epsilon=0.001)
svm_reg.fit(X_train, y_train)

test_pred = svm_reg.predict(X_test)
train_pred = svm_reg.predict(X_train)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)

print('Train set evaluation:\n_____________________________________')
print_evaluate(y_train, train_pred)

results_df_6 = pd.DataFrame(data=[["SVM Regressor", *evaluate(y_test, test_pred)]],
                            columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df = results_df.append(results_df_6, ignore_index=True)

#ANN

from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Input, Dense, Activation,
    Dropout
from tensorflow.keras.optimizers import Adam
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)

model = Sequential()

model.add(Dense(X_train.shape[1],activation='relu'))
model.add(Dense(32,activation='relu'))
# model.add(Dropout(0.2))

model.add(Dense(64,activation='relu'))
# model.add(Dropout(0.2))

model.add(Dense(128,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1))

model.compile(optimizer=Adam(0.001), loss='mse')

r = model.fit(X_train, y_train,
              validation_data=(X_test,y_test),
              batch_size=128,
              epochs=150)

test_pred = model.predict(X_test)
train_pred = model.predict(X_train)

print('Test set evaluation:\n_____________________________________'
    )
print_evaluate(y_test, test_pred)

print('Train set evaluation:\n_____________________________________
    ')
print_evaluate(y_train, train_pred)

results_df_7 = pd.DataFrame(data=[["Artficial Neural Network", *
    evaluate(y_test, test_pred)]],
                            columns=['Model', 'MAE', 'MSE', 'RMSE',
     'R2 Square'])
results_df = results_df.append(results_df_7, ignore_index=True)

#Models comparision

```

```
results_df.plot(x='Model', y='R2 Square', kind='barh', figsize=(12,
    8),title="R2 SQUARE VALUES OF DIFFERENT MODELS")

results_df.plot(x='Model', y='RMSE', kind='barh', figsize=(12, 8),
   title="RMSE VALUES OF DIFFERENT MODELS")

results_df.plot(x='Model', y='MSE', kind='barh', figsize=(12, 8),
   title="MSE VALUES OF DIFFERENT MODELS")

results_df.plot(x='Model', y='MAE', kind='barh', figsize=(12, 8),
   title="MAE VALUES OF DIFFERENT MODELS")



results_df
```

Listing B.1: Complete Python code

Appendix C

Github Link

The link to GitHub repository to access the source code https://github.com/git-pkp/MLpro1

Bibliography

[1] ai explained "what is ai? learn about artificial intelligence". https://www.oracle.com/in/artificial-intelligence/what-is-ai/.

[2] Artificial Intelligence (AI) Explained "what is ai? learn about artificial intelligence". https://www.oracle.com/in/artificial-intelligence/what-is-ai/.

[3] Dataset "house prices - advanced regression techniques". https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data.

[4] Outlier "what are outliers and how to deal with them?". https://medium.com/analytics-vidhya/how-to-remove-outliers-for-machine-learning-24620c4657e8.

[5] RFR "random forest regression: When does it fail and why?". https://neptune.ai/blog/random-forest-regression-when-does-it-fail-and-why.

[6] SVR. https://www.geeksforgeeks.org/support-vector-machine-algorithm/#:~:text=Support%20Vector%20Machine(SVM)%20is,distinctly%20classifies%20the%20data%20points.

[7] SVR. https://www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/.

[8] XGB "xgboost: A deep dive into boosting". https://dzone.com/articles/xgboost-a-deep-dive-into-boosting.

[9] XGBoost Documentation. https://xgboost.readthedocs.io/en/stable/.