

Building and Simulating Neural Network Based on House Prices

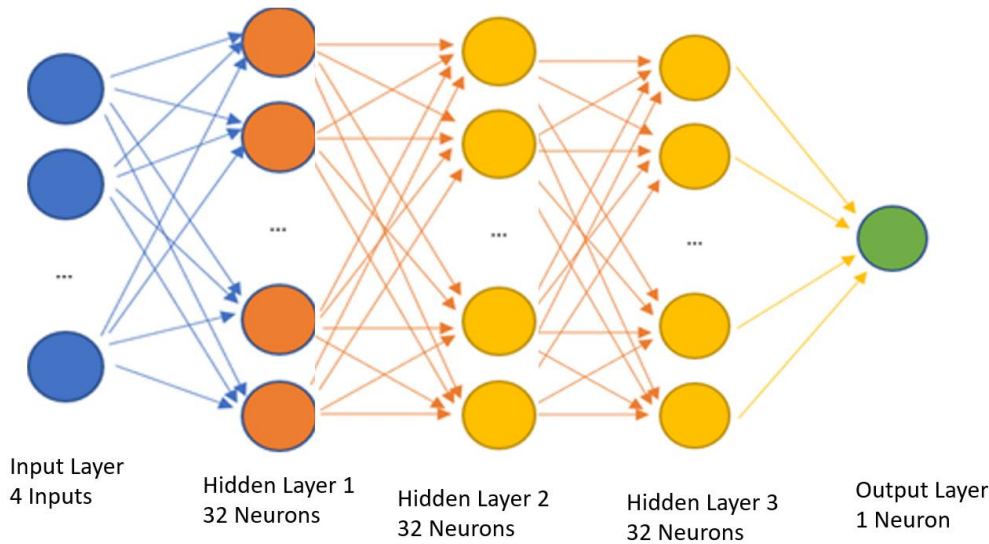
Development of a Neural Network Prediction Model for House Sales Price

The first step in coding any ML algorithm is to prepare the given data in the format that the algorithm requires. The excel file has to be read in specific. Divide the provided dataset next into the input features, represented by x, and the label, represented by y. In this instance, there are four inputs: garage cars, garage area, overall quality, and ground living area. This dataset was then divided into a training set and a test set. Importing the panda's excel function for reading will only import the necessary functions from the panda's library. The following step involves reading the excel file, which must be located in the same directory as the notebook and storing the data in the variable df. Simply type df into the grey box to learn what it contains.

Divide the dataset into input features (x) and the feature (y) that we want to predict. To accomplish that split, we only assign the SalePrice to a variable named y and the GarageCars, GarageArea, OverallQual, and GrLivArea to a variable called x. The process of dividing our dataset into a training set and a test set is the final step in data processing. 80% of the data must be used for training and 20% for testing. As a result, the training set contains 1168 data points, whereas each test set contains 292 data points. While the y variables only have one feature to forecast, the x variables have four input features. As part of the data processing, we read the Excel file, transformed the data into data frames, and divided the dataset into input features and labels. Separate the training set from the test set in our dataset.

Building, Training, and Fitting the Neural Network

The first stage entails defining a template (an architecture), and the second involves selecting the best data points to fill out the template with. Setting up the architecture or structure is the first thing that needs to be done. The neural network with three hidden layers, 32 neurons in the first layer, 32 neurons in the second hidden layer, and 32 neurons in the third layer, as shown in Figure 1, was decided upon using the lowest error received, which was 14. In other words, we wish to have the following layers: output layer: 1 neuron, hidden layer 1: 32 neurons, ReLU activation, hidden layer 2: 32 neurons, ReLU activation and hidden layer 3: 32 neurons, ReLU activation. This architecture needs to be explained to Keras. Since it will use the sequential model, we only need to list the layers above in order.

Figure 1.*Structure of the Model*

Note. Made using word and paint software.

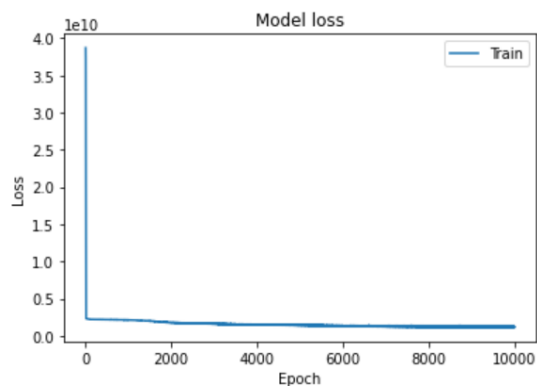
Import the appropriate Keras code first. In our Keras sequential model, we then declare it as `House model = Sequential ()`. This indicates that our model will be stored in the `House model` variable. We will apply `.add()` to the sequential model to add the layers. We have our first layer as a dense layer with 32 neurons, ReLU activation, and the input shape is four since we have four input features. Our second layer, ReLU activation, is similarly a dense layer with 32 neurons and the third layer, ReLU activation, is similarly a dense layer with 32 neurons. As Keras can deduce from the output of our first layer, we do not need to provide the input shape in this case. A dense layer with one neuron makes up our third layer.

We need to identify the best numbers for our structure now that it has been defined. The model must be configured before we begin our training by informing it of the optimization technique we intend to use and the loss function we want it to employ. Calling the function `House model.compile` will configure the model with these settings. The extended stochastic gradient descent algorithm known as the Adam optimizer may be used in the future in a variety of deep learning applications, including computer vision and natural language processing. The mean of the squared errors between labels and predictions is computed via loss. Writing one line of code is all that is necessary to perform training on the data. Since we are fitting the parameters

to the data, the function is called fit. We must supply the `x_train` and `y_train` training data that we will be using. The length of time we wish to train it must then be specified (epochs). A history will be produced by this function, which we save in the variable `hist`. One should be able to watch the loss decline and the accuracy rise over time by looking at the figures. One can now experiment around with neural network architecture and hyper-parameters to see the rationalization of the model. After adjusting to any hyperparameters, run the cells once again to check how your training has altered.

Figure 2

The plot of Loss during Training of Model with 10000 Epoch



Note. Taken a snapshot from the colab.

As a result, we use the Keras Sequential model to specify the architecture. With the model, we define a few of our settings (optimizer, loss function, metrics to track) with the model. Compile function. Using `model.fit`, we train our model using the training data to get the ideal architecture parameters. Now, predictions can be made using the model. Absolute percentage error and its mean can be used to find errors between predicted and actual data. The model performs better, the lower the error.

Rationalizing the Developed Neural Network

In rationalizing a model, we want to determine which activation function, number of neurons, and hidden layers are best for the given data by minimizing the error. There are some rule of thumb to guide us; first, we know that for a regression model, the Relu function works both for linear and non-linear models; however, we also tested if a linear model would be more

appropriate since understand through initial data visualization that the relationship between house sales prices and the other parameters are linear. Below is the screenshot of the results.

Rationalization with Various Activation Function

Figure 3

Rationalization with Activation Function as Linear

```
[25] #Building Model
      from keras.models import Sequential

[66] from keras.layers import Dense, Activation

[67] House_model = Sequential()

[68] House_model.add(Dense(32, activation="linear", input_dim = 4))

[69] House_model.add(Dense(64, activation = "linear"))

[70] House_model.add(Dense(1))

[71] #Compiling the Mmodel with loss and optimizer
      House_model.compile(loss = "mean_squared_error", optimizer = "adam")

[72] #Fitting the Model
      hist = House_model.fit(x_train, y_train, epochs= 10000)

[79] y_test.shape
      (292,)

[80] ## reshaping
      y_predicted_1 = y_predicted.reshape(1,292)

[81] import numpy
      y_test_1 = y_test.to_numpy()

[82] APE = abs( numpy.subtract(y_predicted_1, y_test_1) )/y_test_1*100

[83] MAPE = numpy.mean(APE)

MAPE
18.300572594933843
```

Note. Model structure with linear activation function showing and error of 18%. Screenshots from Colab.

Figure 4

Rationalization with Activation Function as Relu

```
[27] #Building Model
      from keras.models import Sequential

[28] from keras.layers import Dense, Activation

[29] House_model = Sequential()

[30] House_model.add(Dense(32, activation="relu", input_dim = 4))

[31] House_model.add(Dense(64, activation = "relu"))

[32] House_model.add(Dense(1))

[33] #Compiling the Mmodel with loss and optimizer
      House_model.compile(loss = "mean_squared_error", optimizer = "adam")

[52] y_test.shape
      (292,)

[41] ## reshaping
      y_predicted_1 = y_predicted.reshape(1,292)

[53] import numpy
      y_test_1 = y_test.to_numpy()

[54] APE = abs( numpy.subtract(y_predicted_1, y_test_1) )/y_test_1*100

[55] MAPE = numpy.mean(APE)

MAPE
15.820286451242708
```

Note. Model structure with relu activation function showing and error of 15.8%. Screenshots from Colab

The Linear model shows a Mean Average Percentage Error (MAPE) of approximately 18% i.e., an accuracy of 82%, as shown in Figure 3, while the Relu activation function shows a MAPE of 15.8%, as shown in Figure 4. Other activation functions, such as tanh and sigmoid, were tested, but the result shows very little accuracy as expected see Figures 5 and 6 below. These activation functions are better suited for classification or normalized data.

Figure 5

Rationalization with Activation Function as tanh

```
[27] House_model = Sequential()
[28] House_model.add(Dense(32, activation="tanh", input_dim = 4))
[30] House_model.add(Dense(64, activation = "tanh"))
[31] House_model.add(Dense(1))
[32] #Compiling the Mmodel with loss and optimizer
House_model.compile(loss = "mean_squared_error", optimizer = "adam")
[33] #Fitting the Model
hist = House_model.fit(x_train, y_train, epochs= 10000)

[40] ## reshaping
y_predicted_1 = y_predicted.reshape(1,292)
[41] import numpy
y_test_1 = y_test.to_numpy()
[42] APE = abs( numpy.subtract(y_predicted_1, y_test_1) )/y_test_1*100
[43] MAPE = numpy.mean(APE)
[44] MAPE
84.64540199690789
```

Note. Model structure with tanh activation function showing an error of approximately 84% Screenshots from Colab.

Figure 6

Rationalization with Activation Function as sigmoid

```
[26] from keras.layers import Dense, Activation
[47] House_model = Sequential()
[48] House_model.add(Dense(32, activation="sigmoid", input_dim = 4))
[49] House_model.add(Dense(64, activation = "sigmoid"))
[50] House_model.add(Dense(1))
[51] #Compiling the Mmodel with loss and optimizer
House_model.compile(loss = "mean_squared_error", optimizer = "adam")
#Fitting the Model
hist = House_model.fit(x_train, y_train, epochs= 10000)

[59] ## reshaping
y_predicted_1 = y_predicted.reshape(1,292)
[60] import numpy
y_test_1 = y_test.to_numpy()
[61] APE = abs( numpy.subtract(y_predicted_1, y_test_1) )/y_test_1*100
[62] MAPE = numpy.mean(APE)
[63] MAPE
84.6725698978152
```

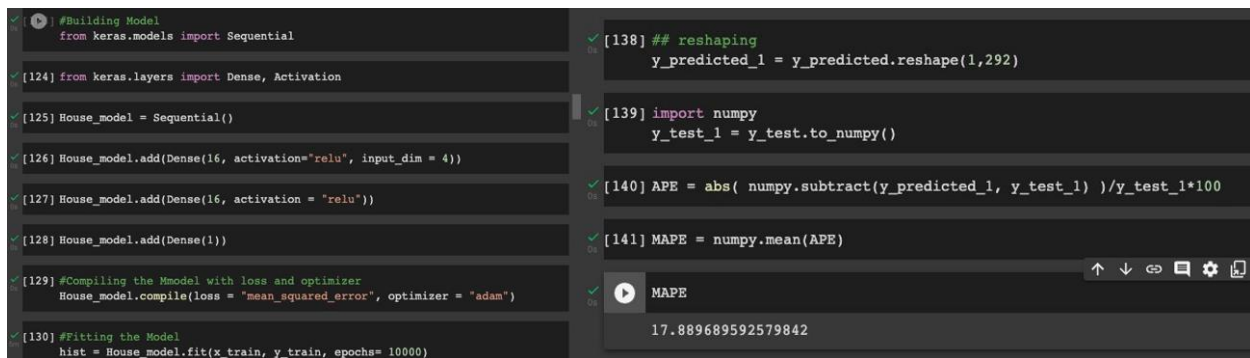
Note. Model structure with Sigmoid activation function showing an error of approximately 85% Screenshots from Colab.

Rationalization with Varying Number of Neurons

Increasing the number of neurons to 64 and 128 in the first and second hidden layers, respectively, made the processing slow but did not improve the accuracy; the accuracy was slightly lower, and reducing the number of neurons to 32 each only improved the accuracy slightly at 15.2% and decreasing the neurons to 16 increases the error up to 17%, this shows that 32 neurons are the optimum solution. These results indicate that a Neural network model with 32 neurons are the optimum solution for the given dataset.

Figure 7

Rationalization with 16 Neurons and 2 Hidden Layers and Relu Activation



```
[124] from keras.models import Sequential
[125] House_model = Sequential()
[126] House_model.add(Dense(16, activation="relu", input_dim = 4))
[127] House_model.add(Dense(16, activation = "relu"))
[128] House_model.add(Dense(1))
[129] #Compiling the Mmodel with loss and optimizer
House_model.compile(loss = "mean_squared_error", optimizer = "adam")
[130] #Fitting the Model
hist = House_model.fit(x_train, y_train, epochs= 10000)

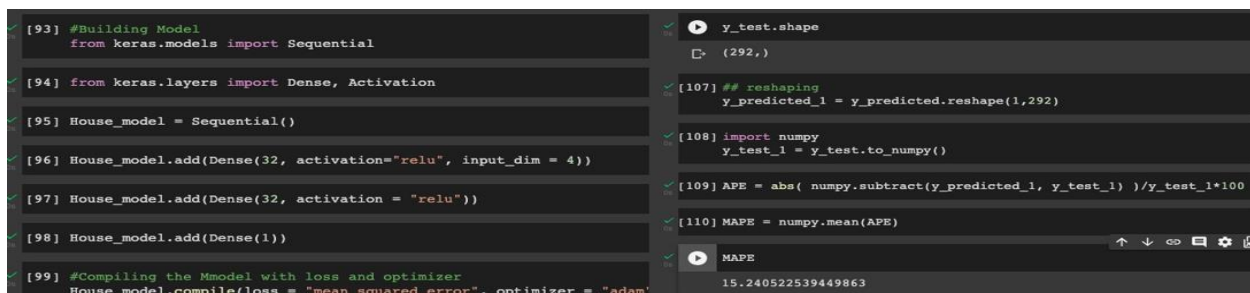
[138] ## reshaping
y_predicted_1 = y_predicted.reshape(1,292)
[139] import numpy
y_test_1 = y_test.to_numpy()
[140] APE = abs( numpy.subtract(y_predicted_1, y_test_1) )/y_test_1*100
[141] MAPE = numpy.mean(APE)

MAPE
17.889689592579842
```

Note. The model with 16 neurons and 2 hidden layers with relu activation function showing an error of approximately 18% Screenshots from Colab.

Figure 8

Rationalization with 32 Neurons and 2 Hidden Layers and Relu Activation



```
[93] #Building Model
from keras.models import Sequential
[94] from keras.layers import Dense, Activation
[95] House_model = Sequential()
[96] House_model.add(Dense(32, activation="relu", input_dim = 4))
[97] House_model.add(Dense(32, activation = "relu"))
[98] House_model.add(Dense(1))
[99] #Compiling the Mmodel with loss and optimizer
House_model.compile(loss = "mean_squared_error", optimizer = "adam")

y_test.shape
(292,)
[107] ## reshaping
y_predicted_1 = y_predicted.reshape(1,292)
[108] import numpy
y_test_1 = y_test.to_numpy()
[109] APE = abs( numpy.subtract(y_predicted_1, y_test_1) )/y_test_1*100
[110] MAPE = numpy.mean(APE)

MAPE
15.240522539449863
```

Note. Model with 32 neurons and 2 hidden layers with relu activation function showing an error of approximately 15% Screenshots from Colab.

Figure 9

Rationalization with 64 Neurons and 2 Hidden Layers and Relu Activation

```
[69] #Building Model
from keras.models import Sequential

[70] from keras.layers import Dense, Activation

[71] House_model = Sequential()

[72] House_model.add(Dense(64, activation="relu", input_dim = 4))

[73] House_model.add(Dense(64, activation = "relu"))

[74] House_model.add(Dense(1))

[75] #Compiling the Mmodel with loss and optimizer
House_model.compile(loss = "mean_squared_error", optimizer = "adam")

[82] y_test.shape
(292,)

[83] ## reshaping
y_predicted_1 = y_predicted.reshape(1,292)

[84] import numpy
y_test_1 = y_test.to_numpy()

[85] APE = abs( numpy.subtract(y_predicted_1, y_test_1) )/y_test_1*100

[86] MAPE = numpy.mean(APE)

[87] MAPE
16.033057636815904
```

Note. Model with 64 neurons and 2 hidden layers with relu activation function showing an error of approximately 16% Screenshots from Colab.

The Rationalization for the Number of Hidden Layers

Next, we consider changing the number of hidden layers and observe how the model behaves. Below, Figures 10 to 12, are the screenshot of the results. In all these sensitivities, we found that using 3 hidden layers with 32 neurons and a Relu activation function gave the minimum error and thus represents the optimum solution for the given dataset.

Figure 10

Rationalization with 64 Neurons and 3 Hidden Layers and Relu Activation

```
[68] #Building Model
from keras.models import Sequential

[69] from keras.layers import Dense, Activation

[70] House_model = Sequential()

[71] House_model.add(Dense(64, activation="relu", input_dim = 4))

[72] House_model.add(Dense(64, activation = "relu"))

[73] House_model.add(Dense(64, activation = "relu"))

[74] House_model.add(Dense(1))

[75] #Compiling the Mmodel with loss and optimizer
House_model.compile(loss = "mean_squared_error", optimizer = "adam")

[82] y_test.shape
(292,)

[83] ## reshaping
y_predicted_1 = y_predicted.reshape(1,292)

[84] import numpy
y_test_1 = y_test.to_numpy()

[85] APE = abs( numpy.subtract(y_predicted_1, y_test_1) )/y_test_1*100

[86] MAPE = numpy.mean(APE)

[87] MAPE
16.544497962862856
```

Note. Model with 64 neurons and 3 hidden layers showing an error of approximately 17% Screenshots from Colab.

Figure 11

Rationalization with 32 Neurons and 4 Hidden Layers and Relu Activation

```
[88] #Building Model
from keras.models import Sequential

[89] from keras.layers import Dense, Activation

[90] House_model = Sequential()

[91] House_model.add(Dense(32, activation="relu", input_dim = 4))

[92] House_model.add(Dense(32, activation = "relu"))

[93] House_model.add(Dense(32, activation = "relu"))

[94] House_model.add(Dense(32, activation = "relu"))

[95] House_model.add(Dense(1))

[96] #Compiling the Mmodel with loss and optimizer
House_model.compile(loss = "mean_squared_error", optimizer = "adam")

[97] #Fitting the Model
hist = House_model.fit(x_train, y_train, epochs= 10000)

Epoch 9552/10000
37/37 [=====] - 0s 3ms/step - loss: 1069155968.0000

[112] #Predicting the House sales Price Based on the Q2 of Inputs provided
House_model.predict([[2,600,7, 2200]])

1/1 [=====] - 0s 30ms/step
array([[222082.47]], dtype=float32)

[113] y_test.shape

(292,)

[114] ## reshaping
y_predicted_1 = y_predicted.reshape(1,292)

[115] import numpy
y_test_1 = y_test.to_numpy()

[116] APE = abs( numpy.subtract(y_predicted_1, y_test_1) )/y_test_1*100

[117] MAPE = numpy.mean(APE)

MAPE
15.74579312556299
```

Note. Model with 32 neurons and 4 hidden layers showing an error of approximately 16% Screenshots from Colab.

Figure 12

Rationalization with 32 Neurons and 3 Hidden Layers and Relu Activation

```
[48] #Building Model
from keras.models import Sequential

[49] from keras.layers import Dense, Activation

[50] House_model = Sequential()

[51] House_model.add(Dense(32, activation="relu", input_dim = 4))

[52] House_model.add(Dense(32, activation = "relu"))

[53] House_model.add(Dense(32, activation = "relu"))

[54] House_model.add(Dense(1))

[55] #Compiling the Mmodel with loss and optimizer
House_model.compile(loss = "mean_squared_error", optimizer = "adam")

[56] #Fitting the Model
hist = House_model.fit(x_train, y_train, epochs= 10000)

[61] #Predicting the House sales Price Based on the Q2 of Inputs provided
House_model.predict([[2,600,7, 2200]])

1/1 [=====] - 0s 29ms/step
array([[237118.75]], dtype=float32)

[62] y_test.shape

(292,)

[63] ## reshaping
y_predicted_1 = y_predicted.reshape(1,292)

[64] import numpy
y_test_1 = y_test.to_numpy()

[65] APE = abs( numpy.subtract(y_predicted_1, y_test_1) )/y_test_1*100

[66] MAPE = numpy.mean(APE)

MAPE
14.978669246991826
```

Note. Model with 32 neurons and 3 hidden layers showing an error of approximately 15% Screenshots from Colab.

Forecast of House Sales Price Based on the Data Given

Neural Network with the given four variables representing the input layer (Garage cars=2, Garage Area=600, OverallQual=7, GrLivArea=2200) forecasted or predicted a value of 239859.8 for the sales price, which is the output layer.

Table 1

Data set for house price and Forecasted sales price

SalePrice	GarageCars	GarageArea	OverallQual	GrLivArea	
235128	2	830	8	1580	
236000	2	554	7	2256	
236000	2	574	7	1949	
236500	3	608	8	1616	
236500	3	522	8	1788	
237000	1	613	7	1829	
237000	1	577	8	2031	
237500	2	550	6	2117	
238000	3	833	7	2127	
239000	2	504	7	1767	
239000	1	576	7	1950	
239000	2	864	6	2624	
239000	3	702	9	1800	
239000	1	360	6	2345	
239000	0	722	8	1571	
239500	1	466	8	2020	
239686	1	670	7	2149	
239799	1	868	7	1962	
239859.8	2	600	7	2200	Forecasted Value
239900	0	800	8	1801	

Note: Created in Excel.

Table 1 represents the forecasted value of the sale price of the given data set, with a predicted value of \$239,859.8. In terms of Garage cars provide almost the same number of vehicles in the garage at similar prices from the given data set, whereas there are four prices which are providing more than 3 garage cars. The garage Area for the predicted price of \$239,859.8 is smaller than the area at the price of \$239,799, which provides 868 sq. ft of the garage area, which may be better for clients looking for a more spacious garage for cars that are bigger in size. In contrast, the GrLivArea of 2354 sq. ft comes at \$239,000, whereas the forecasted sales price provides 2200 sq. ft space. Considering the Overall Qual of 7, the forecasted Sale Price is \$239,859.8, providing an attractive house price for a living area of 2,200 sq. ft and 600 sq. ft of garage space which is above average for most of the houses, which

provided a similar price but the best option would be to go for the cost of 238,000 providing three garage cars, 833 sq. ft of garage space with OverQual of 7 and a GrLivArea of 2127 sq. Ft.

Simulating the Impact of the Model Variables on House Sales Price

For simulating purposes, the manual simulation of around eight readings was taken, and the combined graph was plotted in the colab using matplotlib.pyplot library in python. Also, the data of inputs was converted to a list of arrays, and using for loop, the automated simulation of all parameters of inputs and predicted house sales prices were plotted. As per the figure of automated simulation of garage cars with the predicted house prices, there seems not much effect of garage cars on the sales price of the house. The prices are lowered for fewer and more cars and vice versa. Thus it can be inferred that garage cars have no major significance on house sales prices.

Table 2

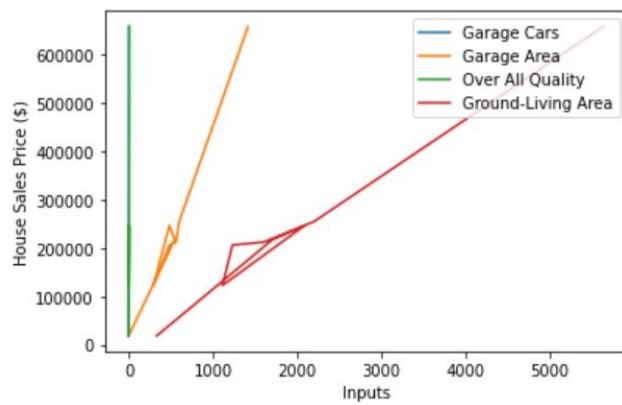
Manual Simulation of Inputs Variables with the Predicted House Sales Price

Garage Cars	Garage Area	Overall Quality	Ground-Living Area	Predicted Sales Price
0	0	1	334	18550.143
2	548	7	1710	217722.67
0	484	7	2090	246725.17
1	294	5	1114	122850.11
3	484	8	1234	206385.2
2	564	7	1604	212439.78
3	600	7	2200	254709.16
3	1418	10	5642	658764.44

Note. Created in Excel.

Figure 13

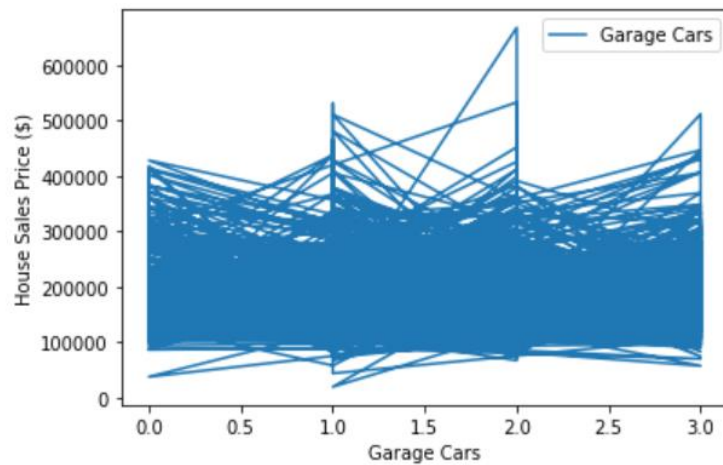
Manual Simulation Plot of Inputs vs. the Predicted Value of Sales



Note. Taken a snapshot from the Colab.

Figure 14

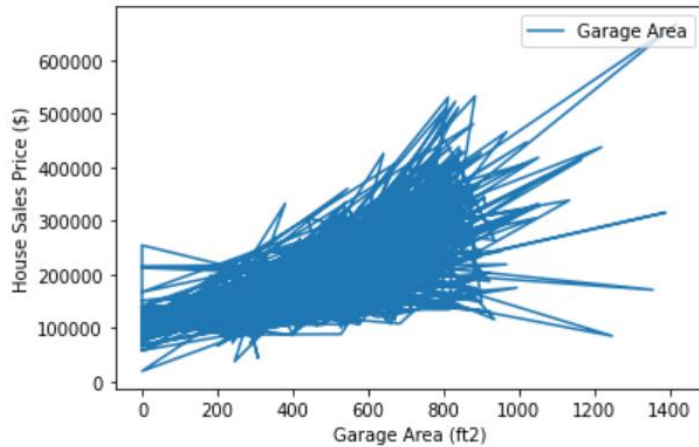
Automated Simulation using for Loop with Impact of Garage Cars on the Predicted Sales Prices



Note. Taken a snapshot from the Colab.

Figure 15

Automated Simulation using for loop with Impact of Garage Area on the Predicted Sales Price

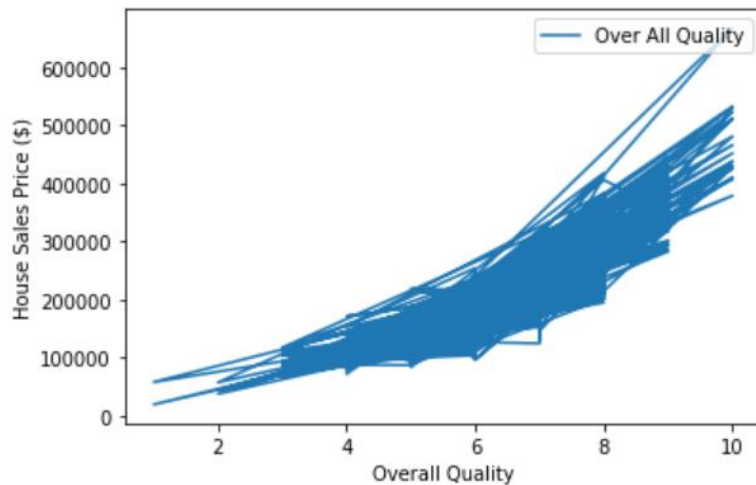


Note. Taken a snapshot from the Colab.

As per the figure on automated simulation of the garage area and predicted house sales price, there seems to be a relationship between the input and the predicted sales price. It can be seen that as the area increase, the house sales prices seem to be increased. Thus, this parameter needs to be considered for evaluating the sales price of houses as it has a significant impact on the sales prices.

Figure 16

Automated Simulation using for loop with Impact of Overall Quality on the Predicted Sales Price

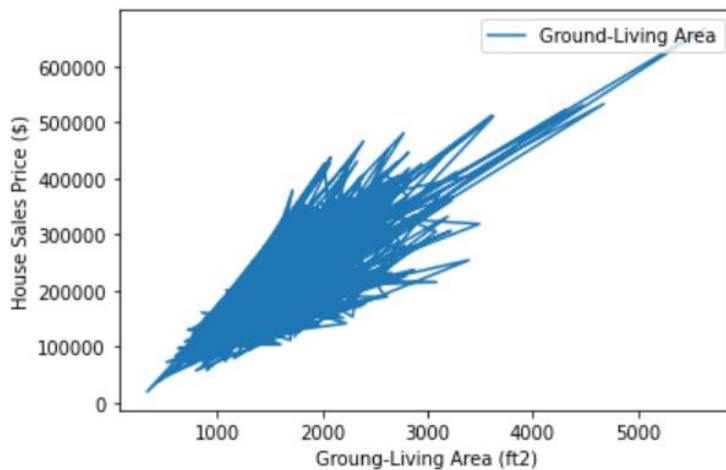


Note. Taken a snapshot from the Colab.

From the automated simulation of overall quality with the predicted sales price, it can be seen that as the quality increases, the sales prices increase. Thus, there seems a linear relationship with the parameters, and so there seems to be a positive impact of overall quality on the sales price. Therefore, overall quality has a significant impact on the sales prices and has to be considered for the prediction of the sales prices.

Figure 17

Automated Simulation with Impact of Ground-Living Area on the Predicted Sales Price

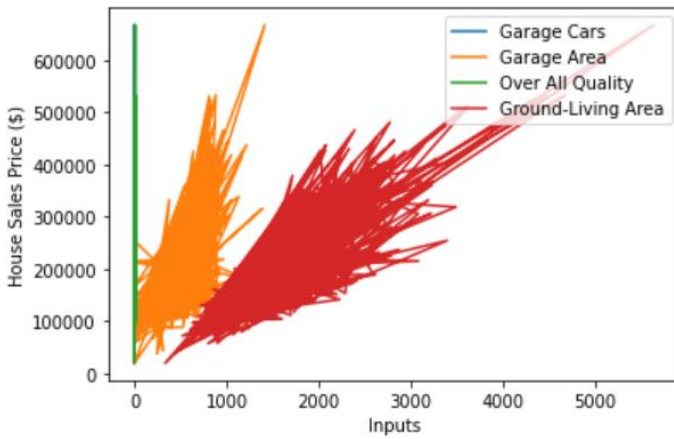


Note. Taken a snapshot from the Colab.

According to the plot generated using an automated simulation of the ground living area and the predicted house sales price, it can be seen that as the area increases, the sales prices increase as so there is a positive relationship between the parameters. Therefore, there is a significant impact of the ground living area on predicting the sales prices of houses. Thus this parameter needs to be considered while making the prediction of the sales prices.

Figure 18

Automated Simulation with Impact of ALL Input Variables on the Predicted Sales Price

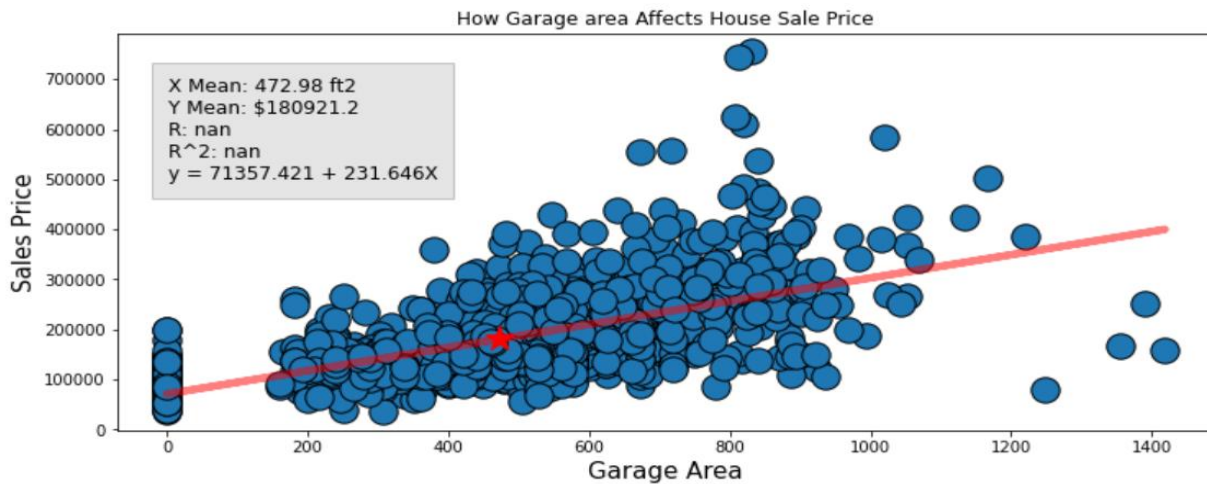


Note. Taken a snapshot from the Colab.

The following figures show the regression of the individual inputs on the output. From the plot, it can be seen that, again, the garage cars have no impact on the sales prices. In contrast, the other three parameters, namely garage area, overall quality, and the ground-living area, have a significant impact on the prediction of house sales prices. Which has a positive relationship with the output and should be considered when predicting the output of sales prices for houses.

Figure 19

How Garage Area Affects the House Sales Price



Note. Taken a snapshot from the Colab.

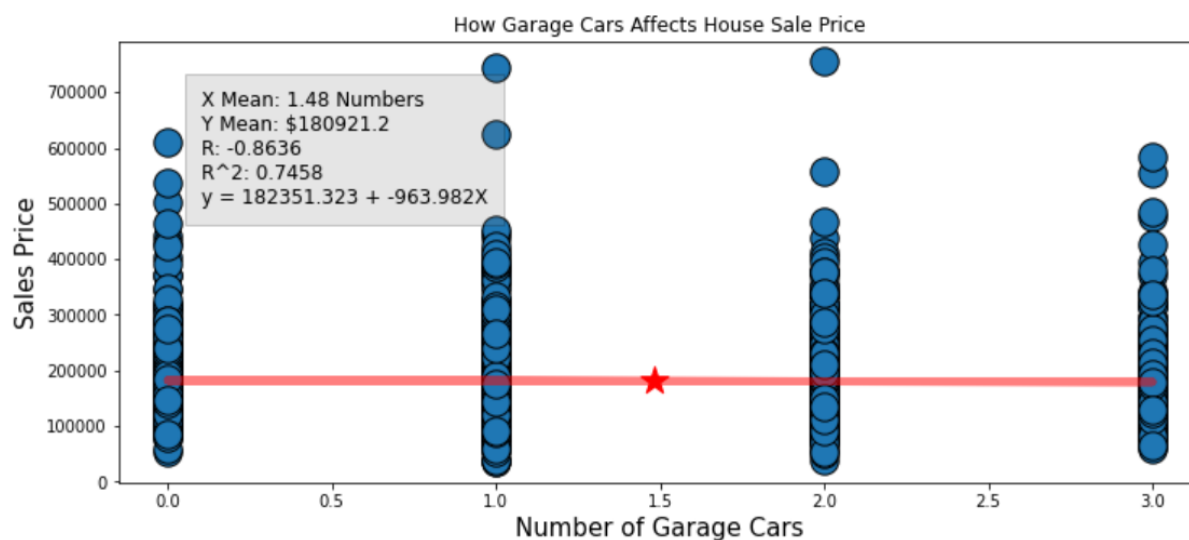
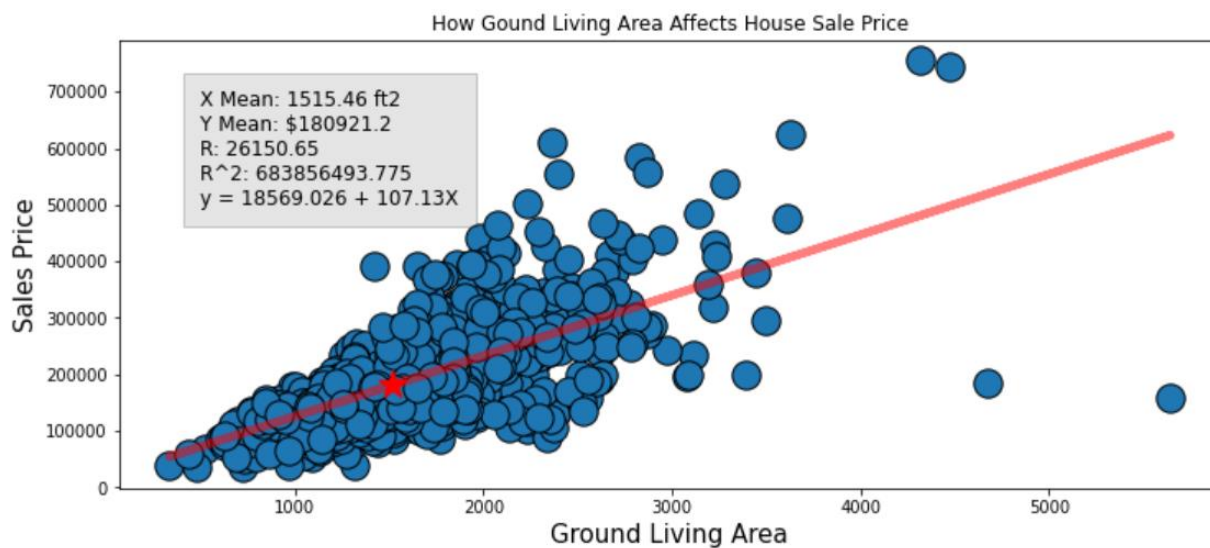
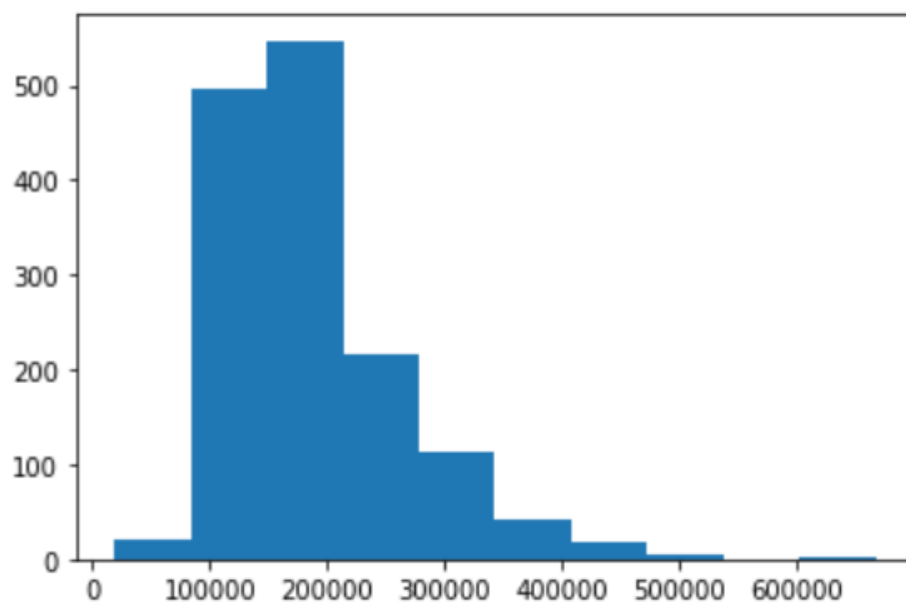
Figure 20*How Garage Cars Affects House Sales Price**Note. Taken a snapshot from the Colab***Figure 21***How Overall Quality Affects House Sales Price**Note. Taken a snapshot from the Colab.*

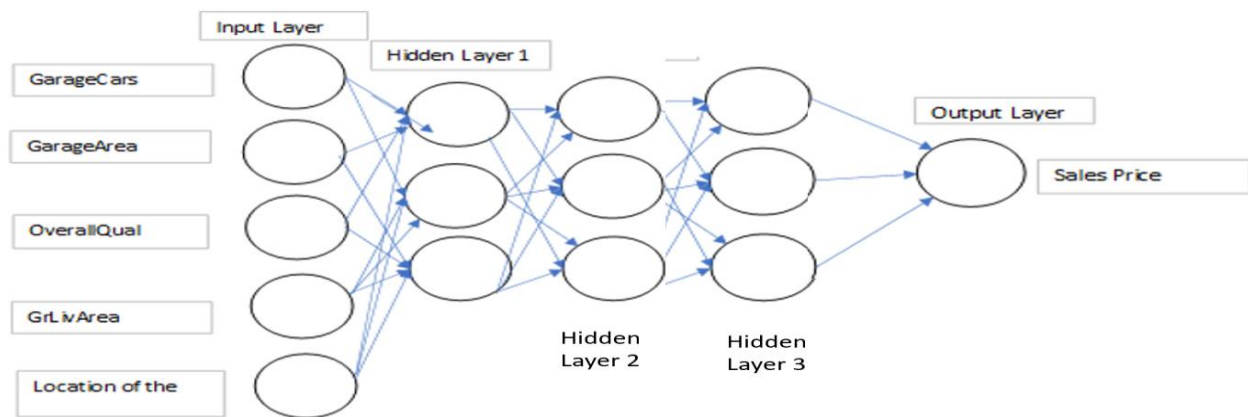
Figure 22*How Gound Living Area Affects House Sales Price**Note.* Taken a snapshot from the Colab.**Figure 23***Histogram of Predicted House Sales Prices**Note.* Taken a snapshot from the Colab.

As from the figure of the histogram of the predicted house sales prices, the majority of the density is found between 100k and 250k, although there seem to be many outliers on the more expensive side.

Highlights of Limitation of Model

Figure 25

Neural Network with Limited Variables and Addition of Location Variable for the Sales Price



Note: Created in Word.

The model has limited variables to make a more accurate prediction, such as the place's location. For example, if the house is in a remote town, it will be cheaper than a home in the city close to nearby facilities like schools, malls, and hospitals.

While the model can predict accurate sale price predictions for the given variables, The data set consists of 1460 columns for the limited four variables, which may not be enough to predict accurate house prices as other factors play a role, such as a house's location and the consumers' purchasing power (Newrez, 2021).

The smaller dataset can also lead the neural network models to tend to overfit as the model can only learn from the training set but fails to predict values when the model is run through the test set and is not able to effectively apply and classify the pattern for accurately predicting the model for the test case (Deep Lizard, 2017).

The model requires trial and error to create a neural network based on the number of layers, neurons, and which activation function to consider making an accurate prediction model as well as to reduce the mean absolute percentage error and is time-consuming to provide a

satisfactory result which is more based on experience as well as hit and trial method (MCMAHAN, 2021).

In our model, training on the data is uncomplicated by using the function fit, which makes it easy to fit the parameters to the data. However, there is insufficient data to find an optimal set of data for a defined function that fits the best in each set of observations. Moreover, it requires building different models to address the overfitting. Furthermore, splitting data sets can result in a different look of numbers and graphs at each time while the test accuracy is getting high. Even though constructing visualization is easy to check the loss and accuracy in overfitting, the running time to train it for epochs is quite long. The model provides a more apparent divergence between training and validation accuracy only when the training loss decreases. Still, the validation loss is increasing and way above the training loss.

References

DeepLizard. (2017). *Overfitting In A Neural Network Explained*.

<https://deeplizard.com/learn/video/DEMmkFC6IGM>

MCMAHAN, S. (2021, December 9). *Artificial Neural Network Disadvantages*.

<https://www.datascienceexamples.com/artificial-neural-network-disadvantages/>

Newrez. (2021, October 8). *The 5 Major Factors That Cause Home Prices to Fluctuate*.

<https://www.newrez.com/blog/home-buying-selling/5-things-that-influence-home-prices-in-2021/>