House Price Prediction

Amro Habboush
School Of Applied Technical Sciences
German Jordanian University
Amman, Jordan
A.Habboush@gju.edu.jo

Hashem Al Taha
School Of Applied Technical Sciences
German Jordanian University
Amman, Jordan
H.Altaha@gju.edu.jo

Abstract*—Nowadays, large amount of data is available everywhere. Therefore, it is very important to analyze this data in order to extract some useful information and to develop an algorithm based on this analysis. This can be achieved through data mining and machine learning. Machine learning is an integral part of artificial intelligence, which is used to design algorithms based on the data trends and historical relationships between data. Machine learning is used in various fields such as bioinformatics, intrusion detection, Information retrieval, game playing, marketing, malware detection, image deconvolution and so on. This paper presents the work done by various authors in the field of machine learning in house price prediction.

I. INTRODUCTION

The purpose of this article is to make the best possible prediction of house prices by using appropriate algorithms and finding out which among them is best suitable for predicting the price with low error rate. This is an interesting problem because most of the people will eventually buy/sell a home. This problem allows us to learn more about the housing market and helps with making more informed decisions. The analysis that were done in this paper is mainly based on the datasets of Washington DC because of unexpected changes in price of houses in and around it.

In this paper, we try to demonstrate all the possible Regression techniques which are suitable to our problem. The brief overview of all the reference taken are as follows: [SVR] support vector machine regression ,[KNN]K-nearest neighbors and[RF] Random forest are used.

DATA DESCRIPTION

A. Data

We obtained a dataset from Kaggle, it contains around 21,000 house data, and 21 features.

id :a notation for a house

date: Date house was sold

price: Price is prediction targe

bedrooms: Number of Bedrooms/House

bathrooms: Number of bathrooms/bedrooms

sqft_living: square footage of the home

sqft_lot: square footage of the lot

floors :Total floors (levels) in house

waterfront :House which has a view to a waterfront

view: Has been viewed

condition :How good the condition is Overall

grade: overall grade given to the housing unit, based on King

County grading system

sqft_above :square footage of house apart from basement

sqft_basement: square footage of the basement

yr_built :Built Year

yr_renovated :Year when house was renovated

zipcode:zip code

lat: Latitude coordinate

long: Longitude coordinate

sqft_living15 :Living room area in 2015(implies-- some renovations) This might or might not have affected the lotsize area

sqft_lot15 :lotSize area in 2015(implies-- some renovations)

B. Reading the Data

We imported pandas library as pd and used it to read the data from the excel sheet ‘housingproj’; which we obtained from Kaggle, we used pd.read_csv().

III. Analyzing the data The

things we wanted to analyze are:

A. What is the shape of the Dataset?

We got a notion of the whole dataset from the following code in Figure 1; we have approximately 22K samples and a total of 21 columns, the most of which are numerical and a few of which are categorical.

```
df.describe()
df.shape
✓ 0.2s
(21613, 21)
```

Fig 1

B. What features are we working with?

```
df.columns
✓ 0.6s
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'sqft_living15', 'sqft_lot15', 'was_renovated',
       'yr_built_0', 'yr_built_1', 'yr_built_2', 'yr_built_3'],
      dtype='object')
```

Fig. 2.

By using the code “df.columns” we notice 5 columns that has nothing to add to our project, so we dropped them (id, date, zip code, lat, long) as you can see in the code (df.drop(['id', 'date', 'id', 'zipcode', 'lat', 'long'], axis=1, inplace=True)

C. What is the type of each feature?

From df.info() code output we can know which of the data we have to encode in fig3.

```
df.info()
✓ 0.4s
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21552 entries, 0 to 21612
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   price          21552 non-null  float64
 1   bedrooms       21552 non-null  float64
 2   bathrooms      21552 non-null  float64
 3   sqft_living    21552 non-null  int64
 4   sqft_lot       21552 non-null  int64
 5   floors         21552 non-null  float64
 6   waterfront     21552 non-null  int64
 7   view           21552 non-null  int64
 8   condition      21552 non-null  int64
 9   grade          21552 non-null  int64
 10  sqft_above     21552 non-null  int64
 11  sqft_basement  21552 non-null  int64
 12  sqft_living15  21552 non-null  int64
 13  sqft_lot15     21552 non-null  int64
 14  was_renovated  21552 non-null  int64
 15  yr_built_0     21552 non-null  int64
 16  yr_built_1     21552 non-null  int64
 17  yr_built_2     21552 non-null  int64
 18  yr_built_3     21552 non-null  int64
dtypes: float64(4), int64(15)
```

Fig. 3.

IV. Visualizing the Data

We imported from the seaborn library the countplot function to plot the attrition flag with most features, such as:

Bathroom (fig4) , Bedrooms (fig5)
And by that we could conclude that the most important features due to the figures are:

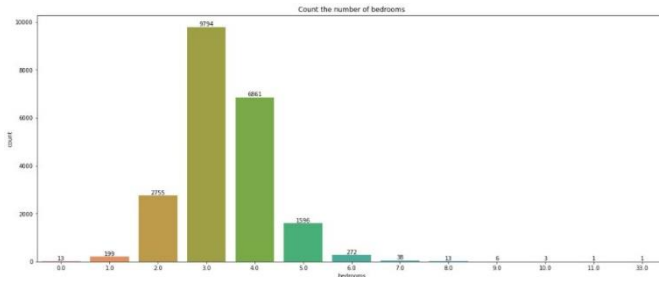

Fig4

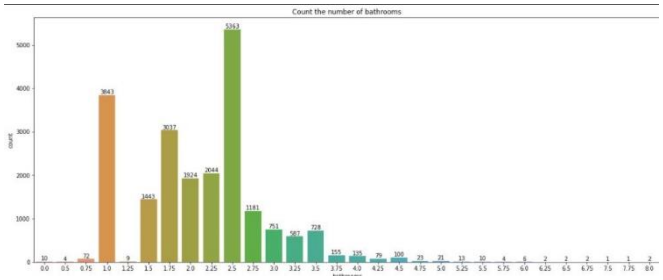

Fig. 5.

We can see that most houses have 3 bedrooms and 2.5 bathrooms.

Also, we can see that most features are not useable as they have low values-variation such as :

Views (fig6), Waterfront (fig7), Renovation (fig8).

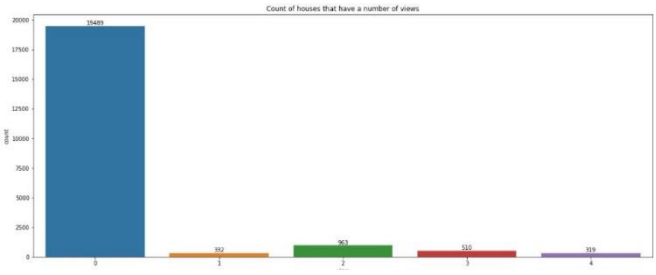

Fig 6

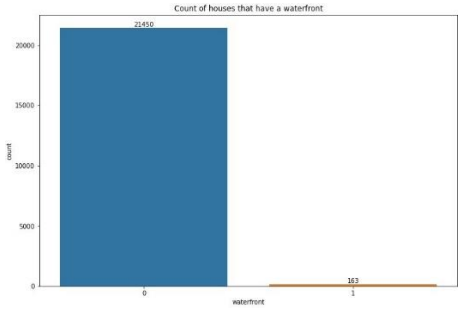

Fig 7

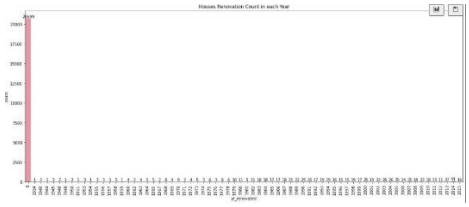

Fig 8

V. METHODOLOGY

A. Apply Data Processing Based on the Investigation Results

There are some features that can be useful if we change the way we look at them, such as:
the house renovation, as we can create a new feature that is the indication of the house renovation status: 1 if renovation is done, 0 otherwise.
So we change it from when question to an if question (fig9)

```
# Create a new feature that is the indication of the house renovation status: 1 if renovation is done, 0 otherwise.
df['was_renovated'] = df['yr_renovated'].apply(lambda x: 1 if x > 0 else 0)
df.drop(['yr_renovated'], axis=1, inplace=True) # Drop the old feature


# Apply One-Hot Encoding to the feature yr_built, to create 4 new features:
# 1. yr_built_0: houses built between 1900 and 1951
# 2. yr_built_1: houses built between 1952 and 1975
# 3. yr_built_2: houses built between 1976 and 1997
# 4. yr_built_3: houses built after 1998

df['yr_built_0'] = df['yr_built'].apply(lambda x: 1 if x >= 1900 and x <= 1951 else 0)
df['yr_built_1'] = df['yr_built'].apply(lambda x: 1 if x >= 1952 and x <= 1975 else 0)
df['yr_built_2'] = df['yr_built'].apply(lambda x: 1 if x >= 1976 and x <= 1997 else 0)
df['yr_built_3'] = df['yr_built'].apply(lambda x: 1 if x >= 1998 else 0)

df.drop(['yr_built'], axis=1, inplace=True) # Drop the old feature
```

Fig 9

B.Check NaN values

We check the amount of Nan (unknown) values that we have (fig10) then we can either drop them or impute them (replace them with the mean of the column) as in (fig11)

```
# Check NaN values
df.isnull().sum()
```

```
price            0
bedrooms        29
bathrooms       32
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
sqft_living15    0
sqft_lot15       0
was_renovated    0
yr_built_0       0
yr_built_1       0
yr_built_2       0
yr_built_3       0
dtype: int64
```

Fig 10

```
# Drop the houses that have NaN values in the features
df.dropna(inplace=True)


mean=df['bedrooms'].mean() #Imputation (replace the missing values of the column bedrooms with the mean of the column bedrooms)
df['bedrooms'].replace(np.nan,mean, inplace=True)


mean=df['bathrooms'].mean() #Imputation (replace the missing values of the column bathrooms with the mean of the column bathrooms)
df['bathrooms'].replace(np.nan,mean, inplace=True)
```

Fig 11

C. Splitting the Data

We use the train split for actual training and the test split to measure the model performance and in order to measure how a model performs on new instances we have split the data into 80% train set and 20% test set (fig12). From sklearn.model_selection library we imported train_test_split to split the data which led that the train set has 17291 data, and the test set has 4322 data.
The data is divided into:
1.train input (all features except the price)
2.train output (the price)
3.test input (all features except the price)
4.test output (the price)
Then we apply the standardscaler to the features to resize the distribution of values, decrease the error and to get better performance.

```
# Split data into training and test sets
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(df, test_size=0.2, shuffle=True,random_state=42)

# Get the features in a dataframe and the target in a separate dataframe
train_input = train_data.drop(['price'], axis=1)
train_output = train_data['price']

test_input = test_data.drop(['price'], axis=1)
test_output = test_data['price']



# Apply StadardScaler to the features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(train_input) # Fit the scaler only to the training data

# Get the scaled data
scaled_train_input = scaler.transform(train_input)
scaled_test_input = scaler.transform(test_input)
```

Fig. 12

VI. TRAINING MACHINE LEARNING MODELS

A. Support vector machine regression (SVR)

Support Vector Regression is a supervised learning algorithm that is used to predict discrete values. Support Vector Regression uses the same principle as the SVMs. The basic idea behind SVR is to find the best fit line. In SVR, the best fit line is the hyperplane that has the maximum number of points (fig13).
SVR is a powerful algorithm that allows us to choose how tolerant we are of errors.

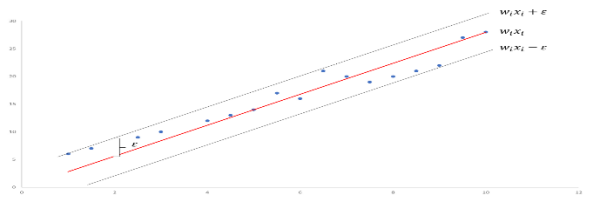

Fig. 13

We used both the original and scaled data so that we can compare the error later on

As we can see the code of the SVR is (fig 14).
Also, we chose (kernel=rbf, C=1000, gamma=auto)

```
# Create a SVR model and fit it to the scaled training data
svr_model_scaled = SVR(kernel='rbf', C=1e3, gamma='auto')
svr_model_scaled.fit(scaled_train_input, train_output)
✓ 23.6s
SVR
SVR(C=1000.0, gamma='auto')

# Create a SVR model and fit it to the original (not scaled) training data
svr_model_original = SVR(kernel='rbf', C=1e3, gamma='auto')
svr_model_original.fit(train_input, train_output)
✓ 41.5s
SVR
SVR(C=1000.0, gamma='auto')
```

Fig 14

B. K-Nearest Neighbor

Next, we used another simple but efficient algorithm called K-Nearest Neighbor.
K nearest neighbors is a supervised learning algorithm, it is a simple algorithm for classifying data items based on their closest neighbors. It makes a "educated guess" on what an unclassified point should be classed as based on test results. The KNN contains a few hyperparameter. The most important one is:
n_neighbors: When voting, this is the number of neighbors to utilize. The greater the number, the more precise it becomes (sometimes), but at the cost of speed, the default number of n_neighbors is 5. An odd number is preferred to avoid any tie (fig 15)

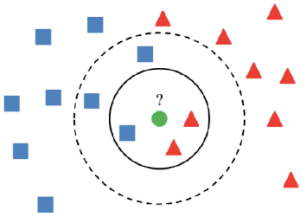

Fig 15

We can see the model code (fig 16)
Also, we used the scaled and original data to compare the error later on.

```
# Create a KNN model and fit it to the scaled training data
knn_model_scaled = KNeighborsRegressor(n_neighbors=5)
knn_model_scaled.fit(scaled_train_input, train_output)
✓ 0.8s
KNeighborsRegressor
KNeighborsRegressor()

# Create a KNN model and fit it to the original training data
knn_model_original = KNeighborsRegressor(n_neighbors=5)
knn_model_original.fit(train_input, train_output)
✓ 0.1s
KNeighborsRegressor
KNeighborsRegressor()
```

Fig 16

C. Ensemble Model - Random Forest

Next, we used another simple but efficient algorithm called decision tree. Random forest is a supervised learning algorithm, it creates many decision trees and then combines them to provide a more accurate and stable estimate (fig17). Random Forest contains may hyperparameter, but the most important ones are:

1) N_estimators: This is the total number of decision trees that will be made. As one might expect, a larger number of trees leads to a stronger model, but at the cost of longer training time.

2) Max_depths: This is the particular tree's maximum depth. The higher the number, the simpler the trees and therefore the random forest ensemble.

3) Random_state: Controls both the randomness of the bootstrapping of the samples used when building trees and the sampling of the features to consider when looking for the best split at each node

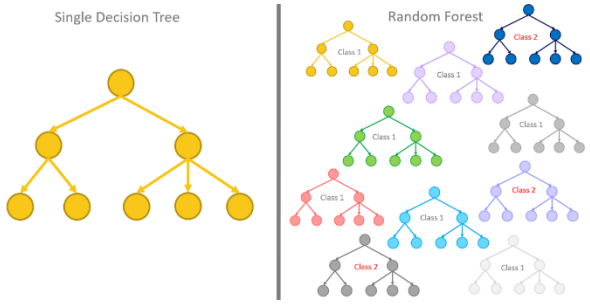

Fig 17

We can see the code of the random forest (fig 18)

We used both the original and scaled data so that we can compare the error later on

We used 500 as the value of the n_estimators and 42 as the value of random_state.

```
# Create a RandomForest ensemble model (of decision trees)  and fit it to the scaled training
rf_model_scaled = RandomForestRegressor(n_estimators=500, random_state=42)
rf_model_scaled.fit(scaled_train_input, train_output)
✓ 1m 18.3s
RandomForestRegressor
RandomForestRegressor(n_estimators=500, random_state=42)

# Create a RandomForest ensemble model (of descion trees)  and fit it to the scaled training d
rf_model_original = RandomForestRegressor(n_estimators=500, random_state=42)
rf_model_original.fit(train_input, train_output)
✓ 1m 29.1s
RandomForestRegressor
RandomForestRegressor(n_estimators=500, random_state=42)
```

Fig 18

D. *Deep Learning – keras (Neural Network)*

Deep learning is a type of machine learning and artificial intelligence (AI) that imitates the way humans gain certain types of knowledge. Deep learning is an important element of data science, which includes statistics and predictive modeling.

Deep learning utilizes both structured and unstructured data for training. Practical examples of deep learning are Virtual assistants, vision for driverless cars, money laundering, face recognition and many more.

There are many examples on deep learning methods such as Classic Neural Networks, Convolutional Neural Networks, Recurrent Neural Networks and many others.

We used Keras deep learning package with Tensorflow backend.

First we transform dates into year, month and day and select columns, Split data for training and validation (fig19) then we Prepare data for training and validation of the Keras model, we should Normalise training and validation predictors using the stats from training data only.

```
kc_data_org['sale_yr'] = pd.to_numeric(kc_data_org.date.str.slice(0, 4))
kc_data_org['sale_month'] = pd.to_numeric(kc_data_org.date.str.slice(4, 6))
kc_data_org['sale_day'] = pd.to_numeric(kc_data_org.date.str.slice(6, 8))

kc_data = pd.DataFrame(kc_data_org, columns=[
        'sale_yr','sale_month','sale_day',
        'bedrooms','bathrooms','sqft_living','sqft_lot','floors',
        'condition','grade','sqft_above','sqft_basement','yr_built',
        'zipcode','lat','long','sqft_living15','sqft_lot15','price'])
label_col = 'price'

print(kc_data.describe())
```

```
def train_validate_test_split(df, train_part=.6, validate_part=.2, test_part=.2, seed=None):
    np.random.seed(seed)
    total_size = train_part + validate_part + test_part
    train_percent = train_part / total_size
    validate_percent = validate_part / total_size
    test_percent = test_part / total_size
    perm = np.random.permutation(df.index)
    m = len(df)
    train_end = int(train_percent * m)
    validate_end = int(validate_percent * m) + train_end
    train = perm[:train_end]
    validate = perm[train_end:validate_end]
    test = perm[validate_end:]
    return train, validate, test
```

Fig 19

We have three functions to define alternative Keras models, the first is very simple, consisting of three layers and Adam optimizer. The second with Adam optimizer consists of 4 layers and the first uses 10% dropouts. The third is the most complex, it extends the previous model with Nadam optimizer, dropouts and L1/L2 regularisers.

Then we fit/ train the model. At the end we Evaluate and report performance of the trained model.

IV. Models Evaluation

A. Get the predictions

we start with getting the predictions based on the test set inputs and compare results with the test set actual outputs and Finding the mean absolute error for each model (fig20).

```
# SVR Model
scaled_svr_predictions = svr_model_scaled.predict(scaled_test_input)
original_svr_predictions = svr_model_original.predict(test_input)

# KNN Model
scaled_knn_predictions = knn_model_scaled.predict(scaled_test_input)
original_knn_predictions = knn_model_original.predict(test_input)

# RandomForest Model
scaled_rf_predictions = rf_model_scaled.predict(scaled_test_input)
original_rf_predictions = rf_model_original.predict(test_input)
```

Fig 20

B. mean absolute error

Calculate the mean absolute error for the scaled and unscaled data for all models

For SVR (fig21)

```
# Calculate the mean absolute error for the scaled and unscaled models (SVR)
mae_svr_scaled = int(mean_absolute_error(test_output, scaled_svr_predictions))
mae_svr_original = int(mean_absolute_error(test_output, original_svr_predictions))

print('Mean Absolute Error for the scaled SVR model:', mae_svr_scaled, '$')
print('Mean Absolute Error for the original SVR model:', mae_svr_original, '$')
✓ 0.1s
Mean Absolute Error for the scaled SVR model: 155004 $
Mean Absolute Error for the original SVR model: 221227 $
```

Fig 21

For KNN (fig22)

```
# Calculate the mean absolute error for the scaled and unscaled models (KNN)
mae_knn_scaled = int(mean_absolute_error(test_output, scaled_knn_predictions))
mae_knn_original = int(mean_absolute_error(test_output, original_knn_predictions))


print('Mean Absolute Error for the scaled KNN model:', mae_knn_scaled, '$')
print('Mean Absolute Error for the original KNN model:', mae_knn_original, '$')
✓ 0.1s
Mean Absolute Error for the scaled KNN model: 132902 $
Mean Absolute Error for the original KNN model: 160034 $
```

Fig 22

For Random Forest (fig23)

```
# Calculate the mean absolute error for the scaled and unscaled models (RandomForest)
mae_rf_scaled = int(mean_absolute_error(test_output, scaled_rf_predictions))
mae_rf_original = int(mean_absolute_error(test_output, original_rf_predictions))

print('Mean Absolute Error for the scaled RandomForest model:', mae_rf_scaled, '$')
print('Mean Absolute Error for the original RandomForest model:', mae_rf_original, '$')
✓ 0.1s
Mean Absolute Error for the scaled RandomForest model: 117725 $
Mean Absolute Error for the original RandomForest model: 117696 $
```

Fig 23

For Deep Learning (fig24)

```
train_score = model.evaluate(arr_x_train, arr_y_train, verbose=0)
valid_score = model.evaluate(arr_x_valid, arr_y_valid, verbose=0)

print('Train MAE: ', round(train_score[1], 4), ', Train Loss: ', round(train_score[0], 4))
print('Val MAE: ', round(valid_score[1], 4), ', Val Loss: ', round(valid_score[0], 4))
```

✓ 1.1s

```
Train MAE:  nan , Train Loss:  nan
Val MAE:  nan , Val Loss:  nan
```

Fig 24

C. Plot and compare

We plot each model original and scaled value and then we compare all the models with each other to see what is the best model we have (fig25).

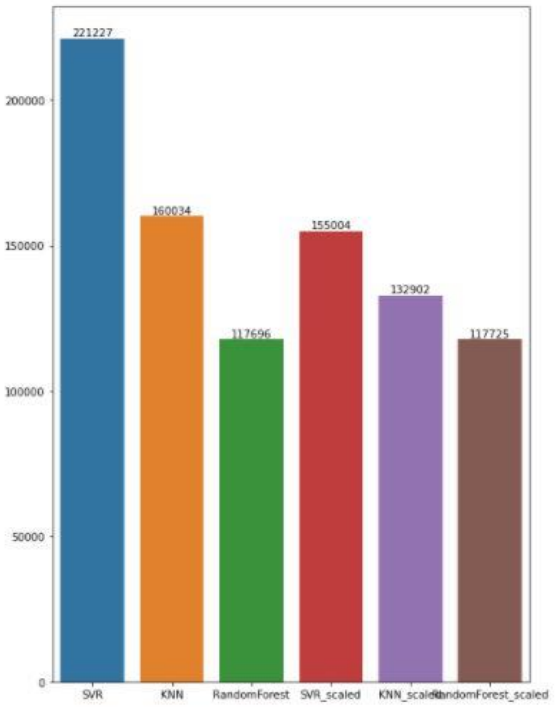

Fig 25

By evaluating models, we can infer that:

- Random Forest (ensemble) model has the lowest errors

- Scaling the data for the non-ensemble models enhanced their performance dramatically.

- Ensemble model did not greatly affect by data scaling, because (in this case) it was based on non-parametric algorithm (Decision Trees).

VII. CONCLUSION

In conclusion, the Random Forest model has the lowest errors and based in the error between the original and scaled data it has the least difference (117,725 $ - 117,696$) = 29 $ where the other models have higher error as we saw the SVR model has (221,227 $ - 155,004 $) = 66,223 $

And the KNN model has (160,034 $ - 132,902 $)= 27 132 US$.

REFERENCES

1. https://www.kaggle.com/code/burhanykiyakoglu/predicting-house-prices
2. https://www.upgrad.com/blog/top-deep-learning-techniques-you-should-know-about/
3. https://www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/
4. https://library.gju.edu.jo:2085/document/8882834
5. https://itsmycode.com/
6. https://stackoverflow.com/
7. https://www.askpython.com/
8. https://www.w3schools.com/
9. https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/
10. https://library.gju.edu.jo:2085/document/9688839
11. https://github.com/vishnu123sai/house-price-prediction-using-neural-network
12. https://hackernoon.com/build-your-first-neural-network-to-predict-house-prices-with-keras-3fb0839680f4