Prediction of House Prices

CS3AM | Dr Muhammad Shahzad
30013556 | Imole Adebayo

Module Code: CS3AM
Assignment report Title: Prediction of House Prices
Date (when the work completed): 06/12/2024
Actual hrs spent for the assignment: ~20 (est.)

Summary of Work & Results

For my project of data analytics and model training, I have prepared and tested two different models to predict house prices based on a variety of corresponding features in the dataset. I have then taken performance scores for each model and experimented with tweaking the values to improve the overall performance scores. Within the creation of the programs and algorithms, this includes data sourcing, preprocessing, model definition within functions, and compilation. The testing of these implementations has proved to be generally successful overall, with accuracy scores reaching as high as the >80th or even >90th percentiles.

Why address this problem?

Shelter and security are fundamental to the lives of every person in the world, regardless of if these needs are actually met or not, on an individual basis and within groups. For most people, shelter and security are considered to be as basic of a need as food and water. This is well-illustrated by including shelter and property on the two highest priorities, 'Basic Needs' levels conceptualised by Maslow's Hierarchy of Needs[1].

Various factors may affect the price of a house or other property. This may include immediate factors, such as its size, its number of rooms, garage & exterior, and its location. External features such as the nature of the property market in the current year, or the availability of alternatives, are more difficult to control and quantify but will also influence the purchase of a property.

Consequently, housing prices are critical to consider in everyday life for individuals, for landlords, for couples, and for families, in the case of letting a place out or finding a place to stay. This can be particularly daunting when reflecting on restrictive factors such as the constant increases of house prices yearly, which are increasingly impacting homeownership ability[2]. '…Changes in residential housing prices can dramatically affect many aspects of our modern economy. An accurate prediction model of the housing price as well as the related beneficial data analysis are in the interest of all real estate market participants such as home sellers, home buyers, investors, banks, builders and also the government.'[6]

However, this project takes the focus to the variance of housing prices in a vacuum, considering factors such as house quality and build year, and the relative influence these have on the sale prices of the houses. 'In an ever-evolving real estate landscape… accurate price predictions empower landlords to make sound investment choices, (and) enable buyers to negotiate effectively'[3].

The Data

The data is formatted as a CSV file containing 81 columns. It is a House Purchases dataset from Kaggle [4].

To model the dataset, I first used a Seaborn heatmap to show the correlations of different values in comparison with each other. While many cells in the correlation map remained relatively neutral, there are some stronger mappings too, which are relevant for creating predictive models. I then created a ***second*** heatmap when processing the **non-numerical values** in the dataframes. I will touch on the pre-processing techniques for transforming textual values into numerical ones later in this document.

Here are the two concatenated heatmaps of the original numerical values in the table. All values above 'MSZoning' are the original Numerical features, while 'MSZoning' and below were transformed from text into numbers.

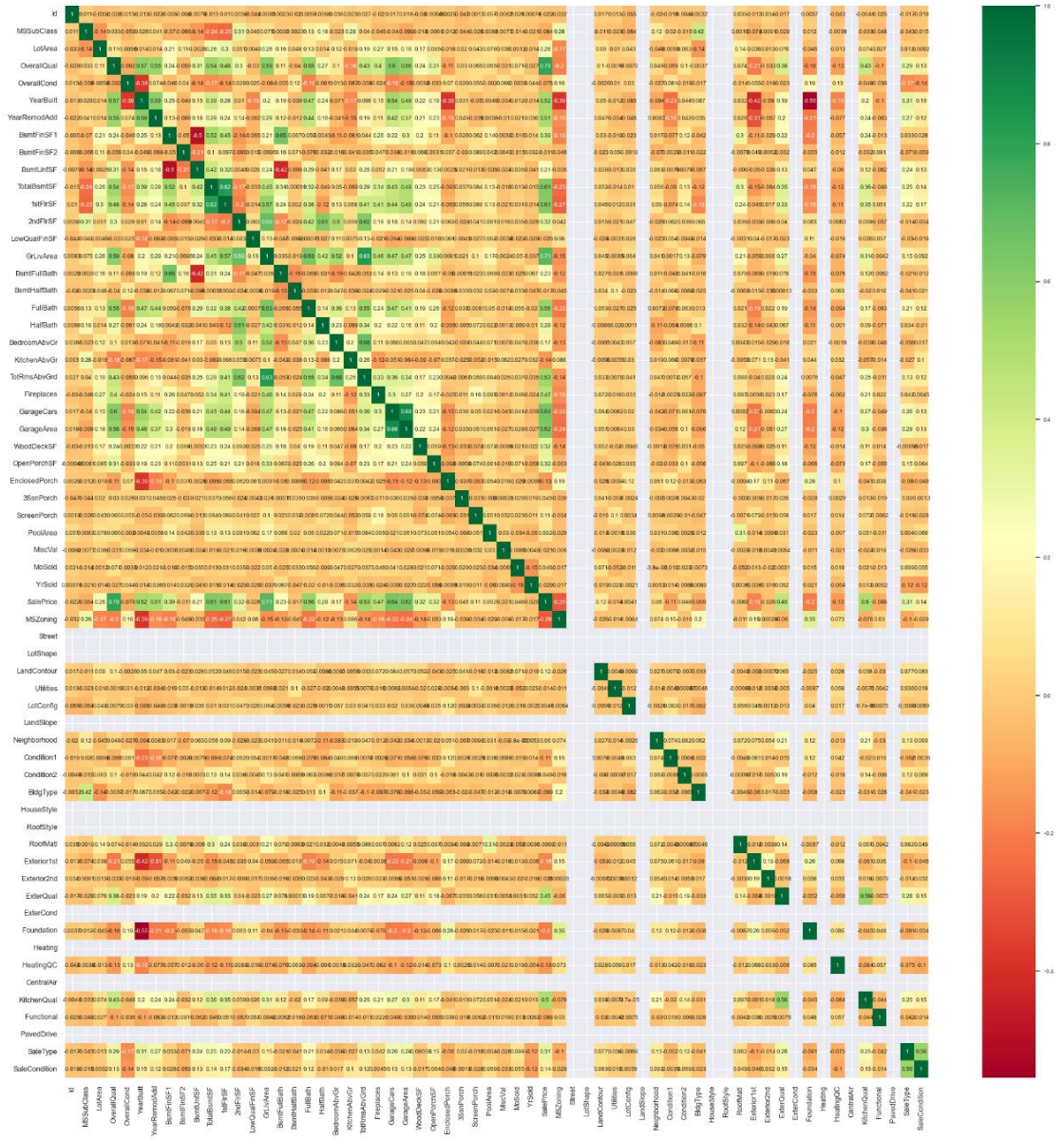

Figure 1.0 – heatmap. '*MSZoning*' is near the divide in the middle.

The **target** column to predict here is the '***SalePrice***' column. Notable features in the dataset with a prominent **positive** or **negative correlation** with this feature (>=0.5 or <=-0.25) include *OverallQual*, *YearBuilt*, *YearRemodAdd*, *1stFlrSF*, *GrLivArea*, *FullBath*, *TotRmsAbvGrd*,

GarageCars, *GarageArea*, *MSZoning*, and *KitchenQual*. These helped me to predict which features were most important for doing predictions, though all features were left in.

I also used Matplotlib to do other variance correlations. This included a Pair Plot for SalePrice, OverallQual, GrLiveArea, and YearBuilt. The Pair Plot can be found in the notebook.

Lastly, here is a graph I modelled to show the changes in house prices over time:

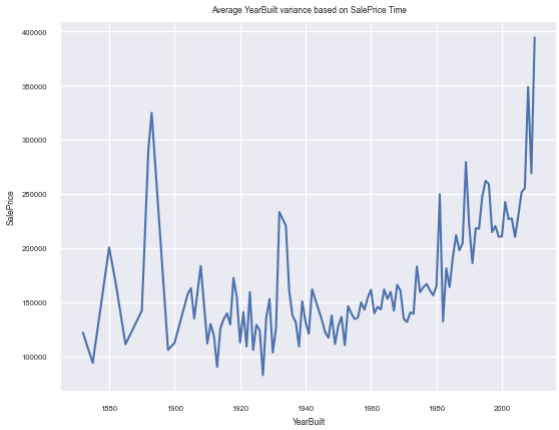

Figure 1.1 – modelled variance of prices across years using the data

Here we can see that, though there are outliers and fluctuations, house prices do trend towards increasing over time.

Machine Learning Models

For my training and testing I used two models to predict the **SalePrice** column based on other features: **Sklearn Linear Regression**, and **Tensorflow & Keras Sequential/Neural Network**.

Linear regression methods would enable me to manipulate and predict values based on data retrieved from the dataframe whilst testing and amending my algorithms continuously. Specifically, I would use Linear Regression in the case of house price prediction based on a continuous set of values. Linear regression would be a better choice than logistic regression in this case, as 'Linear regression is used for continuous outcome variables (e.g., days of hospitalization or FEV1), and logistic regression is used for categorical outcome variables, such as death.' [5]

The Keras Sequential API allows for the creation of artificial neural networks for deep learning. Use of this in conjunction with TensorFlow would allow me to feed in the house data and use it for training and prediction. Keras' Sequential model is quite clear and straightforward to use, so it would enable me to write efficient functions to model my data predictions accurately, training the neural network on a continuous set of values within the dataframe.

Pre-Processing & Feature Selection

For each model, training & testing was completed using two different versions of the same dataframe. Initially, the dataframe's numerical data only was used with all textual representations removed. For instance, columns like 'YearBuilt' and 'LotArea' were left in as

these contained pure numerical data, while columns such as ‘Utilities’ and ‘Condition1’ were dropped, as these contained textual values (e.g. ‘AllPub’ or ‘Norm’ respectively).

A **separate table** was created for these textual values. I then ran an algorithm to convert them back into numerical values. This algorithm added every unique value in the column into a **dictionary**, and then inserted the corresponding dictionary number back into the dataframe in place of the original textual value. This new dataframe of numerically converted values was then concatenated with the original dataframe, with improved accuracy of predictions as a result.

Training

Training for both models was pretty simple. Both models were callable by a primary function, along with similar utilities for calculating accuracy averages and/or printing the accuracy itself. The functions each contained a number of optional arguments to tweak the values used in the model. They also contained the options to print various features of the model, such as the accuracy evaluations.

For the Regression model, I created a set of functions to predict the prices using a LinearRegression object from Sklearn. The inserted dataframe was split into train and test data for both the prediction (Y) and predictive variables (X). A random state was included as an argument in the function (which would be later used for evaluating accuracy). A default data split size of 0.2:0.8 was included as an argument. However, having the ratio as 0.05:0.95 (with the training data as around 70 values) when testing the functions, proved to yield more accurate results later on. The model was then evaluated with accuracy metrics including the Mean Absolute Error, R^2 score, and the similar Explained Variance Score.

For the Keras model the implementation was a little bit more complex, but was still fairly straightforward to implement. For pre-processing, a MinMaxScaler was applied to the predictive data along with removing dummy values. Both the predictive and target data was once again split into training data and testing data. A Sequential object was then added containing an Input layer, followed by Layer 1 (dimension size 128 by default) and Layer 2

(size 64 by default). Two more layers half size each followed. The final Layer was of dimension 1. The model’s function also contains a default epoch size of 50 and a default batch size of 32. All these were used in fitting and testing the data. The default train-test size is 0.2, but upon further testing I tended to use 0.05 as this proved to yield far more accurate results when examined, potentially due to less overfitting. Like the Regression model, the accuracy was evaluated using metrics including Mean Absolute Error, Explained Variance Score, and the R^2 score. This was also returned as a percentage.

Later I would experiment with inserting the predicted values into the dataframe to compare with the actual price, as seen below:

SalePrice	Predicted Price
208500	64040.895667
181500	259139.625332
223500	192564.486726
140000	212414.918362
250000	130132.340359
143000	153496.697926
307000	269936.340879
200000	295910.078894
129900	117898.178215
118000	269224.858672

Figure 2.0 – prediction vs reality (Regression model)

SalePrice	Predicted Price
208500	123462.218750
181500	148513.703125
223500	229041.843750
140000	129412.476562
250000	123766.812500
143000	109852.507812
307000	148690.218750
200000	65091.843750
129900	177288.468750
118000	239137.687500

Figure 2.1 – prediction vs reality (Keras model)

Evaluation, Results

Evaluations for both models were calculated as averages to ensure general representation instead of looking at a single test value. The models were each run several times on the **entire dataset**, with the **random state** being changed each time to a random value between 0 and 999, and an average was calculated across the percentages of each run. For instance the Keras model was run using a customisable default of 8 times. The 8 Explained Variance Scores, one per run, were then added together and the mean was calculated, giving the overall average estimate. This was easier to do for the Regression model due to its lower computing demand and faster execution. I also created functions to calculate the **highest and lowest** accuracy values achieved by both models, with similar customisable data sizes. In both cases, the **whole** dataframe yielded **higher accuracy results** on average than the dataframe with the textual values split off.

Here are some example Explained Variance Scores of accuracy evaluations measured. Assume all other variables (train-test split ratio, epochs, batch size) are optimal.

- Average optimal accuracy for the dataframe with numerical values only (**Sklearn**): 75.85
- Average optimal accuracy for the dataframe with numerical values only (**Keras**): 84.30
- Average optimal accuracy for the dataframe with ALL values including converted text to numbers (**Sklearn**): 78.80
- Average optimal accuracy for the dataframe with ALL values including converted text to numbers (**Keras**): 85.19
- Highest optimal single accuracy value (**Sklearn**): 95.41
- Highest optimal single accuracy value (**Keras**): 94.53

Here are the output histograms for each model:

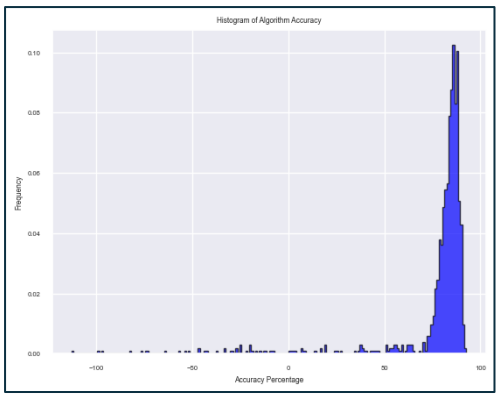

Figure 2.2 – Accuracy for Regression model, though a few outlying accuracies go below 0.

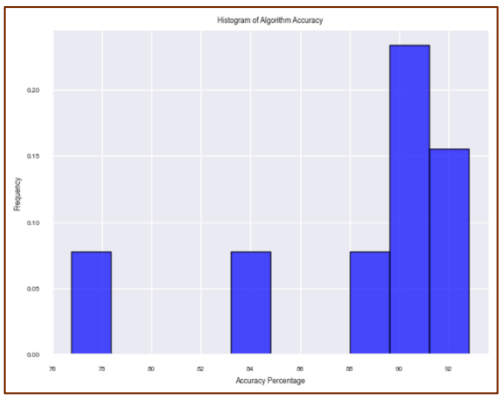

Figure 2.3 – Accuracy for Keras model, lower due to slow performance speed

Which model is better?

Overall both models had their strengths and weaknesses, and it was interesting to compare their results using similar metrics. The Sklearn Linear Regression model generally executed quite quickly, even on more complex tasks such as calculating accuracy averages. This made it quite easy to test and tweak, and to run on the entire dataset. However, it had a low level of customisability as the only changeable metrics were the random state, sample weight, and the train:test ratio. This meant that optimising the model was quite limited, and there was a relatively low level of trial-and-error that could be completed before exhausting the options.

The Keras TensorFlow Sequential model was much more customisable as it contained tweakable dimensions for random state, train:test ratio, number of epochs, batch size, and the number of layers. This made it easier to experiment and improve the model's accuracy and efficiency, allowing for analysis of overfitting as well. However, execution was much slower for this model, to the point where I had to measure it with a timer using Python's `time` module. This also included a much higher level of power consumption which was less efficient to work with (my laptop's heat fans consciously sped up power usage when running the model accuracy averages, which routinely took up to dozens of minutes to calculate). This made it much tougher to test and tweak the model's algorithms and running parameters.

As far as actual results went, Sklearn's Linear Regression presented better accuracy values at higher train:test ratios, e.g. 0.2:0.8. However, it also had a lower range for improvement. Meanwhile, Keras' accuracy assessments performed better at a lower train:test ratio, e.g. 0.05:0.95.

Based on these evaluations, and the accuracy scores provided in the previous section, I would say that the **Keras** model was slightly better for my overall work purpose due to its customisability and technicality, as well as accelerated accuracy. However, the **Sklearn** model might be more useful in situations where the accuracy of results can be compromised in favour of performance and execution speed. Lower accuracy scores with faster performance open the door for more trial-and-error to optimise the model iteratively, even if the actual parameters that may be tweaked are quite limited in comparison.

Conclusion/Reflection

Overall, while I had a lower or shaky level of understanding regarding regression, neural networks & their layers, and the tools needed to implement these theories within an applied programming environment, the use of these on a real-world data application has helped to solidify my knowledge.

Much of the theory behind this project was quite challenging to pick up at first due to a lack of prior knowledge. While I have had the experience of programming in Python for data analysis prior, in the CS2PP22 module two years back, had to put in a lot of extra research to understand the topic the second time around. This helped me to understand many theories including neural networks, regression modelling, and underfitting and overfitting data.

Finding an adequate dataset also proved to be slightly difficult as many datasets online suffered from one or more issues which made them less than desirable to work with. A lack of measurable features was of the more frequent issues I ran into, for instance. Another prominent issue was that many of the datasets I found suffered from a lack of correlation in their variables. Thankfully the dataset I have managed to find here has 81 columns and varying correlations across all variables, as seen in the heatmap and graphs featured.

In the future, I could potentially improve my models by allowing for an even higher level of customisability. For instance, the TensorFlow functions could benefit from a customisable number of layers when calling the function, instead of the default 4. Speed of execution for many of these functions was also a recurrent issue, especially for the TensorFlow functions, and so a form of execution optimisation (such as multithreading) might be desirable in the future. Overall, progression and completion of this project has helped me to better understand many aspects of data analysis and AI model training, allowing me to apply this theory to my work.

References

[1] McLeod, S., 2007. Maslow's hierarchy of needs. *Simply psychology*, *1*(1-18). https://www.simplypsychology.org/maslow.html

[2] Hick, R., Pomati, M. and Stephens, M. (2024) 'Housing affordability and poverty in Europe: on the deteriorating position of market renters', *Journal of Social Policy*, pp. 1–24. doi:10.1017/S0047279423000703. https://www.cambridge.org/core/journals/journal-of-social-policy/article/housing-affordability-and-poverty-in-europe-on-the-deteriorating-position-of-market-renters/F1843A654FE38E4B3E5ED1619C52F6BB

[3] Sankar, M., Chithambaramani, R., Sivaprakash, P., Ithayan, V., Charaan, R.D. and Marichamy, D., 2024, September. Analysis of Landlord's Land Price Prediction using Machine Learning. In *2024 5th International Conference on Smart Electronics and Communication (ICOSEC)* (pp. 1514-1518). IEEE. https://ieeexplore.ieee.org/document/10722300

[4] (Srikanth, 2019) https://www.kaggle.com/datasets/srikanthladda/house-price-prediction

[5] Castro HM, Ferreira JC. Linear and logistic regression models: when to use and how to interpret them? J Bras Pneumol. 2023 Jan 13;48(6):e20220439. doi: 10.36416/1806-3756/e20220439. PMID: 36651441; PMCID: PMC9747134. https://pmc.ncbi.nlm.nih.gov/articles/PMC9747134/

[6] K. He and C. He, "Housing Price Analysis Using Linear Regression and Logistic Regression: A Comprehensive Explanation Using Melbourne Real Estate Data," 2021 IEEE International Conference on Computing (ICOCO), Kuala Lumpur, Malaysia, 2021, pp. 241-246, doi: 10.1109/ICOCO53166.2021.9673533. keywords: {Training;Machine learning algorithms;Computational modeling;Linear regression;Machine learning;Predictive models;Prediction algorithms;Data Science;Housing price model;Linear regression;Logistic regression}, https://ieeexplore.ieee.org/document/9673533