

|  |
| --- |
| 2023/24 |
| Regression Task |
| 5CS037 – Concepts and Technologies of AI |

|  |
| --- |
| Leigh Clarke  2245151 |

Contents

**1.Introduction3**

1.1 Approach3

**2. Dataset4**

2.1 Data preparation4

2.1.1 Head of the Dataset5

2.1.1 Checking for NULL Values5

2.2 Visualisation of Data6

2.2.1 Correlation Heatmap6

2.2.2 Scatterplot Matrix8

**3. Multivariate Regression9**

3.1 Overview9

3.2 Model Predictions10

3.3 Review and Improvements11

3.3.1 Adjusting Random State11

3.3.2 Adjusting Test Size12

3.3.3 Dropping Features13

**4. Decision Tree Regression14**

4.1 Overview14

4.2 Model Predictions15

4.3 Review and Improvements16

4.3.1 Adjusting Random State16

4.3.2 Adjusting Test Size17

4.3.3 Adjusting Max Depth18

**5. Neural Network Regression20**

5.1 Overview20

5.1.1 TensorFlow21

5.2 Model Predictions22

5.3 Review and Improvements23

5.3.1 Adjusting Layers23

5.3.2 Adjusting Neurons24

**6. References26**

# Introduction

This report aims to predict the housing prices in King County, USA by employing two regression models using machine learning (ML) and one neural network (NN) model. The primary objective is to dissect the dataset and analyse the variables influence property prices and form correlations between them.

For each ML and NN model that is used a report will be created to review the accuracy of each model, compare them with one another and review where improvements can be made.

## Approach

Create two regression models: Multivariate Regression (MR) and Decision Tree Regression (DTR), alongside a neural network model.

Using MR and DRT models we aim to use the predictive power of traditional statistical techniques coupled with powerful Python libraries. Simultaneously, using neural networks, we will look to unlock deeper layers of understanding within the dataset. Specifically trying to achieve a high predictive accuracy but also highlight underlying variables that correlate to property values in King County.

# Dataset

The dataset I’ve chosen to use for the regression models is: KC\_house\_data.csv Source: [https://www.kaggle.com/datasets/shivachandel/kc-house-data].

The dataset compromises of 15 keys and 21613 instances.

#### Keys:

**price** - Price of the property

**bedrooms** - Total amount of bedrooms

**sqft\_living** - Square footage of living space

**sqft\_lot** - Total Square Foot Area of the lot.

**floors** - Total amount of floors.

**condition** - Condition of the property from a range of 1-5.

**grade** - An index from 1 to 13, where 1-3 falls short of building construction and design, 7 has an average level of construction and design, and 11-13 have a high-quality level of construction and design.

**sqft\_above** – square footage of housing space above ground level.

**sqft\_base** – square footage of housing space below ground level.

**yr\_built** – year property was built.

**yr\_renovated** – the year of property last renovation.

**lat** – Latitude coordinates.

**long** – Longitude coordinates.

**sqft\_living15** – Square footage of the housing space for the nearest 15 neighbours.

**sqft\_lot15** – Square footage of the lots of the nearest 15 neighbours.

## Data Preparation

I have removed latitude and longitude coordinates from the dataset as I believe it will have a negative impact on the accuracy of results and could potentially cause issues such as potential overfitting.

I wanted to focus on features and characteristics that directly relate to the housing characteristics and market dynamics. Simplifying the dataset will reduce any risk of negative impact on results. Real estate is an immovable product that is permanently attached to a parcel of land (Wang and Chang, 2024).

Therefore, real estate data and model constructs must contain spatial attributes. This supports the decision to exclude latitude and longitude coordinates, as they may not adequately capture the spatial dynamics and could overlook important factors influencing house prices.

### Head of the dataset

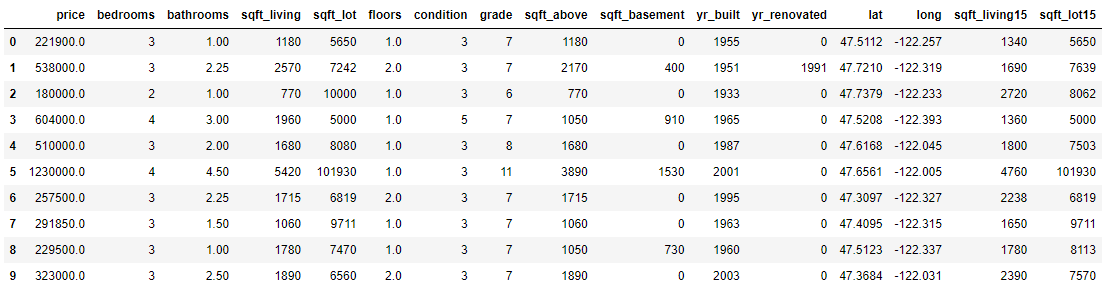


Figure 1: Head of dataset.

A screenshot of a table

Description automatically generated

Figure 2: Head of dataset after latitude and longitude removal.

### Checking for NULL values

The dataset that is being used has already been checked for any NULL values, however I did a check using a pandas function.

A screenshot of a computer

Description automatically generated

Figure 3 Check for NULL values.

## Visualisation of Data

### Correlation Heatmap

A blue and green squares with white text

Description automatically generated

Figure 4 Correlation heatmap

A graph of a triangle

Description automatically generated with medium confidence

Figure 5 Triangular correlation heatmap

A screenshot of a graph

Description automatically generated

Figure 6 Correlation between price

Examining the correlation heatmaps reveals that the most significant correlation to house prices is **sqft\_living** (0.7), followed closely by **grade** (0.67). The three images provided represent different heatmap styles, all displaying the same data.

I will remove the sqft\_lot15 feature from the dataset as the correlation is very low and I don’t believe this will have much of an impact on the prediction accuracy.

### Scatterplot Matrix

A grid of dots

Description automatically generated with medium confidence

Figure 7 Scatterplot Matrix

The above image is of a scatterplot matrix, also known as a pairs plot graph. To help visualise the relationships between the multiple features (variables) within the dataset.

# Multivariate Regression

### Overview

Linear regression is a simple and popular machine learning method used to model the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data (Daneshfar et al., 2023).

Linear regression formula:

In simple linear regression, only one independent variable is used, while in multivariate regression, multiple independent variables are used to predict a single dependent variable, allowing for a more complex understanding of the relationships between variables and potentially more accurate predictions(Tabachnick et al., 2006).

Multivariate regression formula:

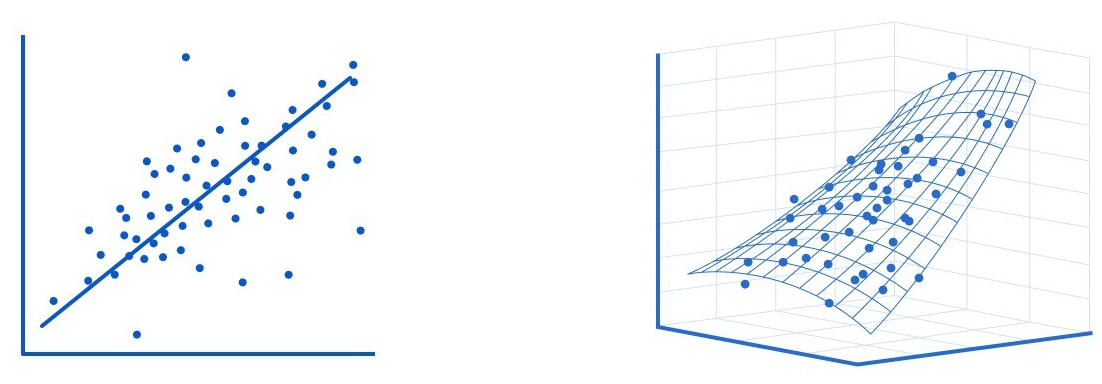


Figure 8 Linear Regression (Left) and Multivariate Regression (Right)

### Model Predictions

We employed a multivariate regression model developed with powerful Python libraries, including the popular sklearn. Central to the module’s creation is the use of a function from sklearn named ‘train\_test\_learn’. Presently, the function will operate with default parameters, with the exception of the arguments used from our dataset (X, y).

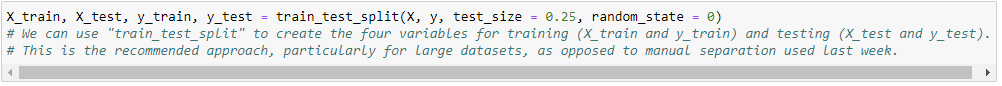


Figure 9 sklearn function: train\_test\_split.

Prediction Accuracy = **62%** (61.90302737287912)

A graph showing a blue and red line

Description automatically generated

Figure 10 ‘Actual vs Predicted House Price’ - Results.

Using a multivariate regression model with default parameters, we attained an accuracy of 62%. A 100% accuracy result would result in a perfectly straight linear line on our graph (highlighted by the red markers). However, upon visual inspection, it’s evident that the blue markers deviate from this line, particularly in predicting higher house prices, showing a noticeable drop-off.

To improve accuracy, we will explore various methods to enhance our model and also consider modifications to the dataset.

### Review and Improvements

#### Adjusting Random State

As previously mentioned, within this model we used a function from sklearn called ‘train\_test\_split’, in our initial model two of the parameters (or arguments) were using default settings (‘test\_size’ and ‘random\_state’).

Firstly, we will look at changing the random\_state arguments and comparing the results. I will use a range from 0-9.

|  |  |
| --- | --- |
| Random State | Accuracy: |
| 0 | 61.90302737287912 |
| 1 | 61.89554201834233 |
| 2 | 62.02484429123055 |
| 3 | 61.91049721760866 |
| 4 | 61.50324887706167 |
| 5 | 62.24842328239185 |
| 6 | 61.28969605956085 |
| 7 | 60.91080325928466 |
| 8 | 62.05975516950469 |
| 9 | 63.54388542513352 |

A graph with a line and a line

Description automatically generated

Figure 11 Results of the Random State improvements.

Changing the random\_state parameter to a ‘9’ has resulted in an improvement in accuracy, increasing it from 62% to 63.5%. However, this increase is still relatively marginal. Therefore, we will explore what changes we can achieve by adjusting the test\_size parameter.

#### Adjusting Test Size

We will now examine how adjusting the ‘test\_size’ parameter within the ‘train\_test\_split’ function impacts the model’s performance. This parameter ranges from 0 (indicating 100% of the dataset used) to 1 (indicating 0% of the dataset used).

We will conduct 8 tests, incrementing the value by 0.10 for each test, starting from 0.10 and ending with 0.90.

|  |  |
| --- | --- |
| Test Size | Accuracy |
| 0.10 | 62.61195769690471 |
| 0.20 | 63.36207857323972 |
| 0.30 | 62.61541350222973 |
| 0.40 | 62.51095721327535 |
| 0.50 | 61.9602842581331 |
| 0.60 | 61.87087477195071 |
| 0.70 | 61.82213146799912 |
| 0.80 | 61.74682610726452 |

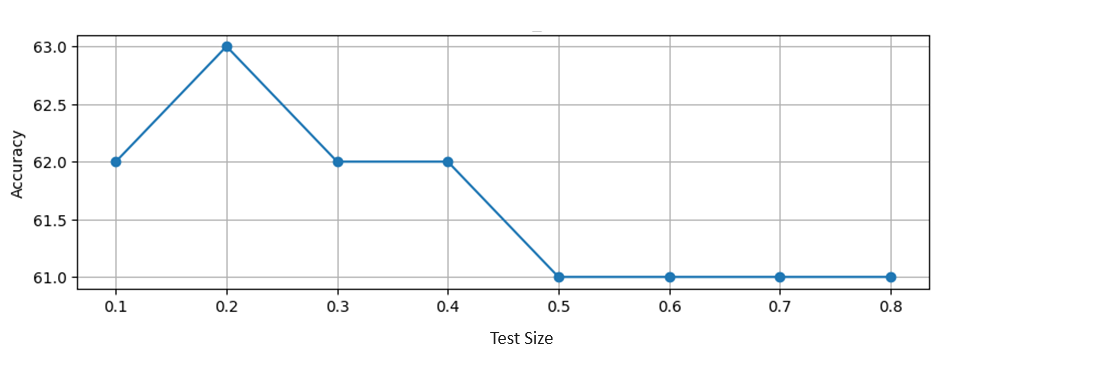


Figure 12 Results of changing test\_size.

Unfortunately, altering the values within test\_size parameter did not lead to any improvements. Our initial test\_size setting of 0.25 results gave an accuracy of 63.54388542513352 , and the highest accuracy achieved during our testing variation was 63.36207857323972.

We will now look at dropping features from our dataset to see if this will positively impact our accuracy.

Earlier we looked at the correlation between features to the price. I will drop the features that have a correlation score lower than 0.3 and see how this impacts the results.

#### Dropping Features

Earlier we looked at the correlation between features to the price. I will drop the features that have a correlation score lower than 0.3 and see how this impacts the results.

Dropped features: condition, yr\_built, sqft\_lot15, sqft\_lot, yr\_renovated, floors.

Prediction Accuracy = **57%** (57.98770669840994)

A graph showing a red line and blue dots

Description automatically generated

Figure 13 Results after dropping features.

After dropping the selected features the prediction accuracy dropped considerably down to **57%**.

Concluding the best prediction accuracy we can achieve using this model is with the following parameters within train\_test\_split: test\_size = 0.25, random\_state = 9 and using a dataset with all features to achieve a prediction accuracy of **63.5%.**

# Decision Tree Regression

### Overview

Decision tree regression (DTR) is a widely recognized and potent algorithm in the realm of machine learning, finding its application in popular libraries such as Scikit-Learn and PySpark (Testas, 2023). This method offers a structured approach to regression models, the data is split recursively into sub-sections (decision nodes and terminal nodes) based on the features, ultimately leading to the prediction of a targeted variable (feature) within a dataset.

Furthermore, the DTR model operates by analysing the dataset information and will create appropriate splits within the data itself. Meaning we can actually create the model with a predefined number of inquiries, such as stipulating a minimum of 3 or 5 questions.

However, decision tree regressors come with their own set of limitations. For example, their vulnerability to overfitting is a known issue, especially if their hyperparameters are not set up properly (Testas, 2023).

A diagram of a decision tree

Description automatically generated

Figure 14 Decision Tree Regression (DTR).

In predictive modelling, decision trees iterate through a dataset by posing true or false questions until they can confidently make a prediction. While decision trees are often employed for classification tasks, they can also be utilized for making predictions.

### Model Predictions

We are going to develop a Decision Tree Regression (DTR) model using powerful Python libraries such as sklearn, similar to the previous model (MVR). Our initial approach involves employing default parameters across all functions and analysing the results.

Afterwards, we will explore potential parameter modifications to improve the prediction accuracy.

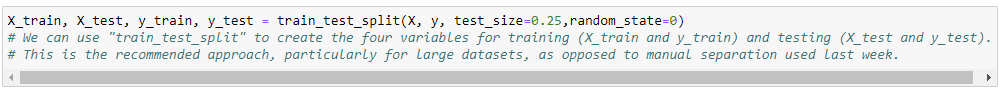


Figure 15 train\_test\_split function with default parameters.

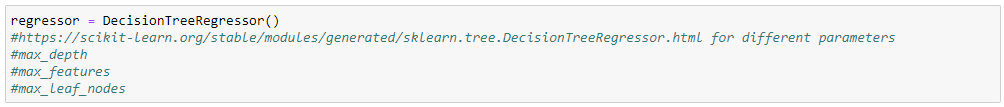


Figure 16 DecisionTreeRegressor function with no parameters.

Prediction Accuracy = **50%** (0.5032583966814992)

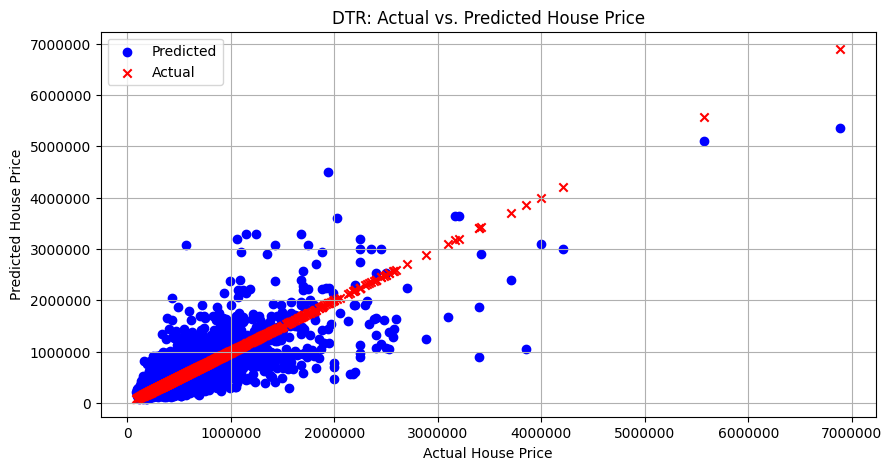


Figure 17 DTR results.

Using a DTR model with default parameters, we attained a prediction accuracy of **50%**. This is quite a poor result and significantly lower than our MVR model which achieved a result of 63.5%.

As previously mentioned a 100% accuracy result would result in a perfectly straight linear line on our graph (highlighted by the red markers). However, upon visual inspection, it’s evident that the blue markers deviate from this line, even more than our previous MVR mode.

To improve accuracy, we will explore various methods to enhance our model and look deeper into changing parameters within the functions of our DTR model.

### Review and Improvements

#### Adjusting Random State

In this section we will look into changing the ‘random\_state’ parameter within the function ‘train\_test\_learn’. We will perform the exact same tests as we did previously with the MVR model. By changing the ‘random\_state’ parameter value from 0-9, in increments of 1.

Below are the results from this test:

|  |  |
| --- | --- |
| Random State | Accuracy: |
| 0 | 50.32583966814992 |
| 1 | 53.57835000288398 |
| 2 | 47.32057786216529 |
| 3 | 46.757809685242724 |
| 4 | 50.29823132355533 |
| 5 | 49.357596953775396 |
| 6 | 50.93442543179753 |
| 7 | 41.5579602931338 |
| 8 | 51.28843478594989 |
| 9 | 47.70867796610859 |

A graph with a line and a line

Description automatically generated

Figure 18 Results of random\_state test.

Using a random\_state value of 1 has yielded the best prediction accuracy result with **53.6%** (53.57835000288398). An increase of 3.6% from the original prediction accuracy.

**Previous prediction accuracy result:** 50%.

**Result after random\_state changes:** 53.6%.

#### Adjusting Test Size

We will now look into testing the effectiveness of changing the value of ‘test\_size’. We will conduct 8 tests going up in increments of 0.10 starting with 0.10 and concluding at 0.80.

|  |  |
| --- | --- |
| Test Size | Accuracy |
| 0.10 | 55.78701499431886 |
| 0.20 | 54.57441426944631 |
| 0.30 | 47.772739742668036 |
| 0.40 | 46.695721872262075 |
| 0.50 | 52.35561475649939 |
| 0.60 | 44.73725639754317 |
| 0.70 | 46.341159786311503 |
| 0.80 | 38.75368885646523 |

A graph with a line and a line

Description automatically generated

Figure 19 Results of test\_size test.

After experimenting with different values for the 'test\_size' parameter, it was found that using a value of 0.1 resulted in the highest prediction accuracy, achieving 55.8%. Despite these adjustments to the 'random\_state' and 'test\_size' parameters leading to improvements in prediction accuracy, the overall result remains relatively low.

In the next phase, we will delve into the analysis of the DecisionTreeRegressor() function within our model to identify potential modifications.

**Previous prediction accuracy result:** 53.6%.

**Result after test\_size changes:** 55.8%.

#### Adjusting Max Depth

We will now look at adjusting one of the parameters within the function DecisionTreeRegressor, which is a function from the sklearn library.

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than ‘min\_samples\_split’ samples. (scikit-learn, 2024).

The ‘max\_depth’ value represents the maximum depth of the decision tree, and will enable us to control the complexity of the model and prevent overfitting.

We will conduct 10 tests, incrementing the 'max\_depth' value by 1 starting from 0 and continuing until we reach 10.

|  |  |
| --- | --- |
| Max Depth | Accuracy |
| 1 | 30.02738858375955 |
| 2 | 44.854463838175895 |
| 3 | 54.51027655216746 |
| 4 | 60.86230562517452 |
| 5 | 64.99631133925194 |
| 6 | 63.90412103423462 |
| 7 | 66.76486401864099 |
| 8 | 64.61316972356175 |
| 9 | 68.46662287711782 |
| 10 | 66.63861364860395 |

A graph with a line and a blue line

Description automatically generated

Figure 20 Result of Max Depth test.

Using a 'max\_depth' value of 9 yielded a prediction accuracy of **68.5%** (68.46662287711782), marking a significant improvement compared to our previous results. The previous prediction accuracy stood at 55.8%.

**Previous prediction accuracy result:** 55.8%.

**Result after test\_size changes:** 68.5%.

With adjustments made to the 'max\_depth' parameter and other modifications, we achieved a remarkable increase in prediction accuracy from the initial 50% to the current 68.5%. This enhancement reflects an overall improvement of 18.5% in prediction accuracy.

A graph showing a line of blue dots and red dots

Description automatically generated

Figure 21 Final graph of DTR model.

# Neural Network Regression

### Overview

This part of the report aims to predict the housing prices in King County, USA by employing a Neural Network Regression (NNR) model.

A neural network is a model inspired by the structure of the human brain. It consists of interconnected nodes (or neurons) which are organised in layers. Information flows through the network from input to output layers, with each neuron processing and passing on signals using weighted connections.

A NNR model compromises of an input layer, hidden layers and finally an output layer, the input layer and hidden layers compromise of interconnected nodes that apply various mathematical transformations (such as weighted sum and activation functions) to the input data before passing it to the next layer.

A diagram of a network

Description automatically generated

Figure 22 Neural Network diagram.

#### TensorFlow

To achieve the creation of this NNR model we will be using the robust and powerful machine learning library TensorFlow 2.x Platform. The following diagram provides a simplified conceptual overview of the entire TensorFlow platform.

A diagram of a software system

Description automatically generated

Figure 23 TensorFlow 2.x platform

As typical of any machine learning project, it consists of three distinct phases. During the first phase, also called the training phase, we define our artificial neural network model and train it on the given data. Also, we test the model using test data and retrain it until we are satisfied with its performance. In the next phase, we save the model to a file, which can later be deployed on a production server. In the third phase of our development, we deploy the saved model on a production server ready to make predictions on unseen data (Sarang 2020).

Within the model that we have created there are 4 main attributes that will have a significant effect on the performance of the model: Amount of hidden layers, Amount of neurons per layer, Activation type and Optimizer.

A screenshot of a computer

Description automatically generated

Figure 24 Excerpt of NNR code.

The last part of the model which is important is the optimizer, this will determine how the model learns. Within our particular model we are using the Adam optimizer from Keras library. We then use the validation function fit, this functions similar to how humans would train themselves by means of revision and then testing. One draw back with validations is training too much as this can cause overfitting. The function below operates in the following manor: the higher the batch\_size the quicker it trains but less detail, the lower the batch\_size the slower it trains but the more detail.

model.fit(x=X\_train, y=y\_train.values, epochs=120, validation\_data=(X\_test, y\_test), batch\_size=32)

### Model Predictions

Initially our NNR model is with 5 layers (including input and output), with input and hidden layers each having 30 neurons and “relu” activation type

Using the Adam optimizer and a batch\_size of “32” we were able to achieve the following results:

Prediction accuracy of: **75%** (0.7456405828806422)

A graph showing a red line and blue dots

Description automatically generated

Figure 25 Results of NNR model

Although, 75% is a good result and far better than the MVR(63.5%) and DTR (68.5%) models, I believe the prediction accuracy of the NNR model can be greatly increased with some improvements.

### Review and Improvements

#### Adjusting Layers

This section explores how the number of layers in a neural network (NNR) affects its performance and prediction accuracy. We'll investigate the impact of increasing layers from the current configuration of five (one input, three hidden, and one output) to two alternative options:

**Test 1:** Seven total layers with five hidden layers.

**Test 2:** Nine total layers with seven hidden layers.

**Constraints:** Due to limitations in computational power and time, we can only evaluate these two variations.

**Test 1 Results:**

A graph showing a red line and blue dots

Description automatically generated

Figure 26 NNR seven layer test 1.

Prediction Accuracy: 83% (0.8309396751249429)

**Test 2 Results:**

A graph with blue and red dots

Description automatically generated

Figure 27 NNR layer test 2.

Prediction Accuracy: 86% (0. 8625874220838263)

In our evaluation, Test 2 demonstrated superior performance, achieving an accuracy of 86%. This represents an 11% improvement over the baseline model and a 3% increase compared to Test 1. While further parameter tuning might be possible, the cost-benefit analysis suggests diminishing returns. We're satisfied with the substantial 11% accuracy gain achieved by Test 2.

#### Adjusting Neurons

The number of neurons in a Dense layer determines the model's capacity to learn complex relationships in the data. With more neurons, the model can potentially capture intricate patterns, but it also increases the risk of overfitting (learning noise in the data).

I will again create two tests, each one with less neurons than the other.

**Test 1 Results:**

20 neurons per layer

Prediction Accuracy: **83.8%** (0.8382066791762157)

**Test 2 Results:**

15 neurons per layer

Prediction Accuracy: **79%** (0. 790494150476628)

Initial observations suggest that reducing the number of neurons have negatively impacted prediction accuracy. To explore this further, let's test a new model configuration. We'll increase the neurons per layer to 50 while keeping the total number of layers manageable due to hardware limitations. This approach aims to balance model complexity with computational efficiency while potentially boosting prediction accuracy.

**New Test Results:**  
50 neurons per layer. 7 layers.  
Prediction Accuracy: **85%** (0.854701301679798)

Based on above findings, increasing the number of neurons in a neural network may lead to improved performance. However, there's also a **limit where adding more neurons might have negative consequences**. To investigate this further, I propose an experiment with two stages. In the first stage, we will increase the neuron count to 100 and measure the resulting performance improvement. If an improvement is observed, we will then conduct a second stage where we increase the number of layers to 9. This final test will solidify the impact of modifying both neuron count and layer count on the network's performance.

The following results were generated using a modified model with 9 layers and 100 neurons:

Prediction Accuracy: **86%** (0.8601351302282261)

A graph showing a red line and blue dots

Description automatically generated

Figure 28 NNR New Test Results

Increasing the number of neurons from 50 to 100 in our model yielded a negligible performance improvement. Even with 9 layers, which we believe is near the optimal range, the gain was only 0.5% compared to previous tests. While further parameter tuning, such as adjusting batch size, might provide marginal improvements, the overall impact is likely to be minimal. Encouragingly, the combined modifications have resulted in a significant accuracy boost from 75% to 86%.

# References

An Application of the Spatial Autocorrelation Method on the Change of Real Estate Prices in Taitung City by Wen-Ching Wang 1,\*ORCID,Yu-Ju Chang 1 andHsueh-Ching Wang 2ORCID. Print

Daneshfar, R., Esmaeili, M., Mohammadi-Khanaposhtani, M., Baghban, A., Habibzadeh, S., & Eslamian, S. (2023). Advanced machine learning techniques: Multivariate regression. In S. Eslamian & F. Eslamian (Eds.), Handbook of Hydroinformatics (pp. 1-38). Elsevier. <https://doi.org/10.1016/B978-0-12-821285-1.00017-8>

Tabachnick, Barbara G., and Linda S. Fidell. Using Multivariate Statistics. 5th ed. Boston, Mass. ; Allyn and Bacon, 2006. Print.

Testas, A. (2023). Distributed Machine Learning with PySpark Migrating Effortlessly from Pandas and Scikit-Learn. 1st ed. Berkeley, CA: Apress. Available at: [https://learning.oreilly.com/library/view/distributed-machine-learning/9781484297513/html/604220\_1\_En\_4\_Chapter.xhtml] (Accessed: 05 January 2024).

scikit-learn. (2024). DecisionTreeRegressor. Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html> (Accessed: 05 January 2024).

Sarang, Poornachandra. Artificial Neural Networks with TensorFlow 2: ANN Architecture Machine Learning Projects. 1st Edition. Berkeley, CA: Apress L. P, 2020. Web.