Machine Learning Solution for House Price Prediction

By

Kypros Tsolakis

F328778

Abstract:

This project explores the application of machine learning techniques to predict house prices. We leverage a comprehensive dataset and employ various machine-learning models to achieve accurate predictions. The report summarises the methodology, experiments, and key findings.

Table of Contents

Part 1: Dataset

1.1 Introduction:

This project looks into how to estimate the value of a property, concentrating on Sindian Dist., New Taipei City, Taiwan, China. We had to go through 400 records and our aim was to make a computer program that can guess how much a house costs based on certain criteria. The dataset we used has 7 main attributes: Data about buying and selling, is the age of the house, the distance to the closest bus stop, the number of nearby convenience stores, the geographical location of the house given in latitude and longitude, and finally, price of house according to its square area. This is also our target label. Looking at the data carefully and using different machine learning methods, this study aims to make correct and helpful predictions as to what the price of a house is, given its characteristics. It helps us understand better the factors affecting real estate appraisals.

```
First few rows of the dataset:
   No  transaction date  house age  distance to the nearest bus station  \
0   1          2012.917       32.0                              84.87882
1   2          2012.917       19.5                             306.59470
2   3          2013.583       13.3                             561.98450
3   4          2013.500       13.3                             561.98450
4   5          2012.833        5.0                             390.56840

   number of convenience stores  latitude  longitude  house price of unit area
0                            10  24.98298  121.54024                      37.9
1                             9  24.98034  121.53951                      42.2
2                             5  24.98746  121.54391                      47.3
3                             5  24.98746  121.54391                      54.8
4                             5  24.97937  121.54245                      43.1
```

Figure 1

1.2 Visualization of Samples and Overall Statistics and Analysis of the Dataset:

To give a brief and quick view of our dataset, and see how the various attributes affect the price of a house individually, we constructed the following plots:

To provide a quick view and realize how the various attributes of the dataset affect the price of a house individually, we constructed the following plots:

1. A graph showing the correlation between the price of a house and its age. (Figure 2)
2. A graph showing how the price of a house changes concerning the distance one must travel to get to a nearby bus station. (Figure 3)
3. A graph showing how the price of a house changes in case there are convenience stores nearby. (Figure 4)

4. A graph showing how the price of a house changes according to its geographic location. (Figure 5) shows the correlation between the house price and its latitude, while (Figure 6) shows the correlation between the house price and its longitude.

The plots for all the aforementioned cases can be seen below.

From the graphs, we conclude that the dominant factor affecting the price of a house is its distance to a nearby bus station. Its price is more inclined to go up the closer it is to a nearby station. However, a hint exists that shows that the price of the house increases the northern it is located (latitude increase). The same is also hinted at the Eastern where the house is located (longitude increase). Additionally, there is a potential correlation between higher latitude and increased house prices.

To derive the price of a house though we will look at all the variables in unison.

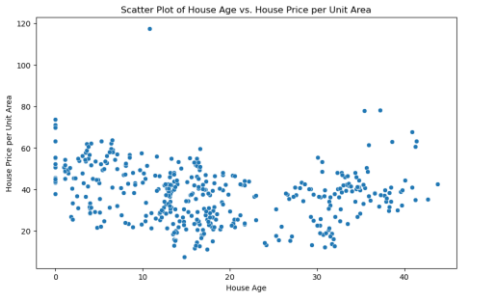

Figure 2

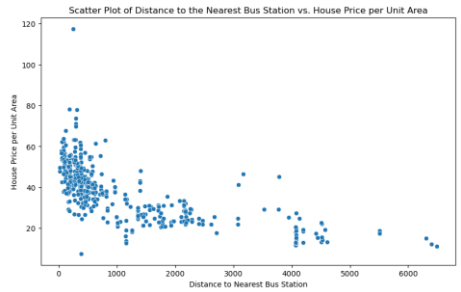

Figure 3

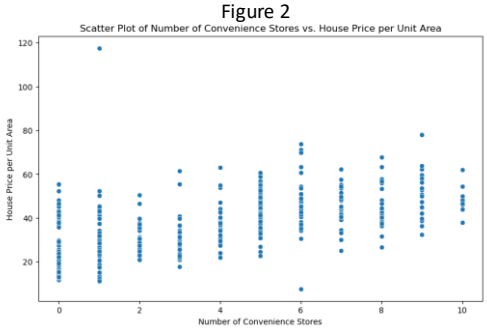

Figure 4

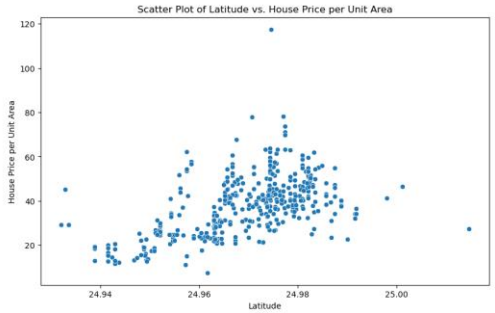

Figure 5

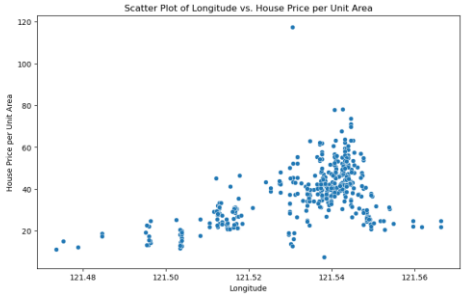

Figure 6

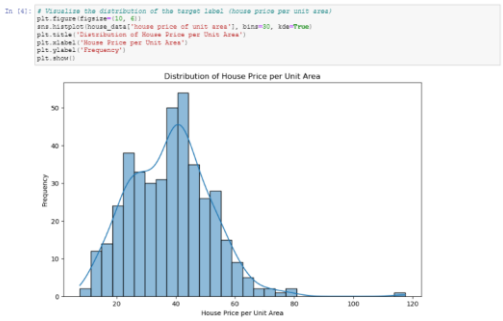

Figure 7

Figure 7 shows the price distribution of the houses among the dataset. Although it is not exactly a Gaussian distribution, we can see that the house prices are fairly normally distributed among our data with the prices at the two extreme points far less frequent.

The following plots show the relationships between the various variables within our dataset. By quickly looking at the plots we can make assumptions as to which variables affect the house price more. This can help us later on to try some variations and improvements to our model by removing some variables from the dataset and seeing how this affects our model.

The information was made ready for machine learning models using methods to find extreme values and normalise numbers. Boxplots showed possible exceptions, which helped decide later treatment choices. This good study of attributes helps to make data ready and correctly build models.

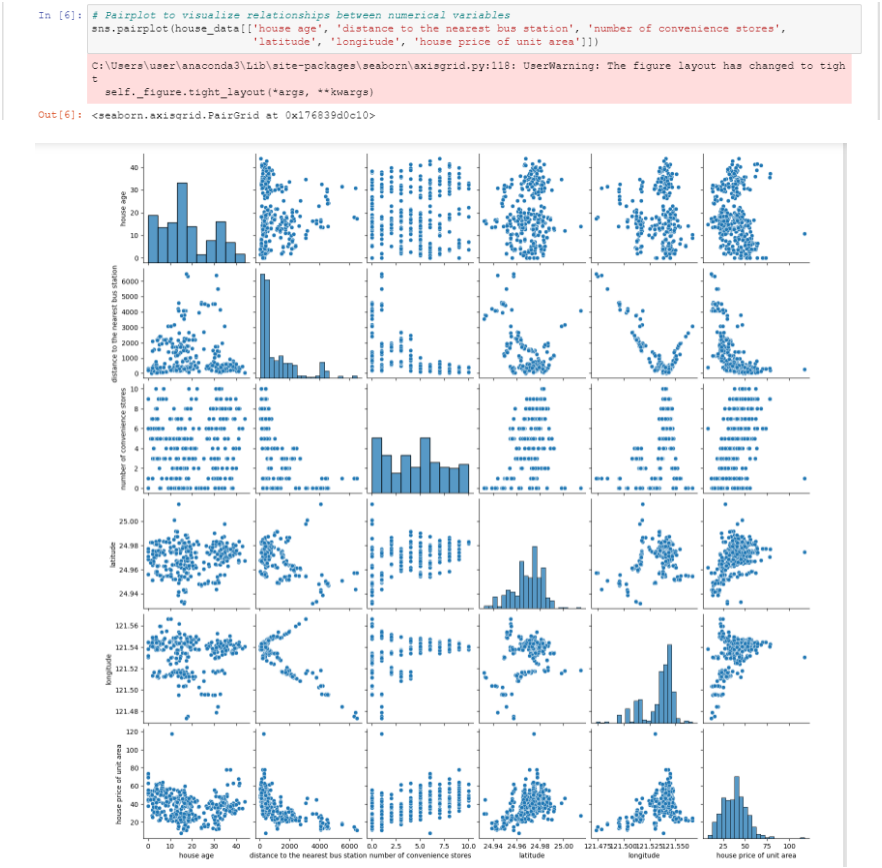

Figure 8

Part 2: Methods

This part is about the steps used in the project. It covers making data ready, choosing and building machine learning models, and setting up a well-made machine learning process.

2.1 Data Pre-processing:

Preparing data before it is used is very important to make sure the information in the data set is good and trustworthy. In this project, several essential pre-processing steps were implemented which are analyzed and explained below:

Checking the dataset for missing values:

The first thing we did was to check whether our data had any missing values so that we proceeded with the necessary steps into rectifying those. This can be achieved by either removing the data which have missing values from the dataset or use data prediction techniques like linear regression to predict the missing values.

Fortunately, as it can be seen below, we did not have any data which was missing any values.

```
# Check for missing values
print("\nMissing values in the dataset:")
print(house_data.isnull().sum())
```

Figure 9

```
Missing values in the dataset:
No                                      0
transaction date                        0
house age                               0
distance to the nearest bus station     0
number of convenience stores            0
latitude                                0
longitude                               0
house price of unit area                0
dtype: int64
```

Figure 10

Outlier Handling:

Outliers are data points which are very different from the rest and if used within our Machine Learning models can skew the results of any prediction leading us into a less accurate model.

The Boxplot method was used to find outliers which provides a strong way of detecting any outliers within the dataset.

By looking at Figure 8, and the correlation between the various attributes of our dataset, we decided to construct boxplots for the different attributes within our dataset.

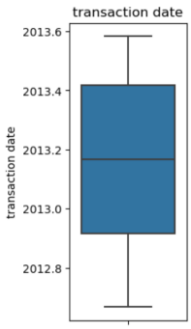

Figure 11

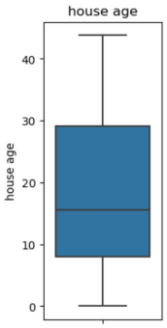

Figure 12

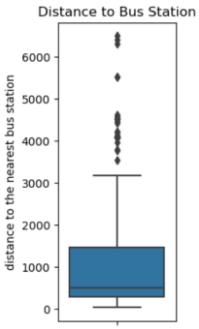

Figure 13

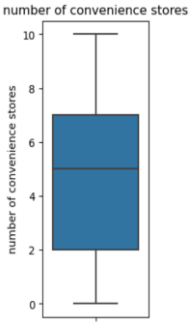

Figure 14

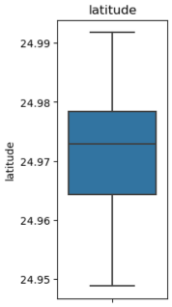

Figure 15

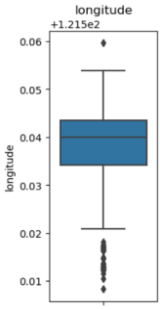

Figure 16

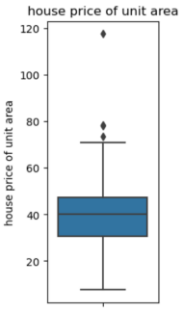

Figure 17

As we can see from the boxplots, we have three attributes which have outliers. In order to improve the prediction results of our Machine Learning models, we decided to remove those datapoints from the dataset.

```
#Drop the outliers
z = np.abs(stats.zscore(house_data['distance to the nearest bus station']))
threshold = 1.3
outliers = house_data[z > threshold]

# drop rows containing outliers
house_data = house_data.drop(outliers.index)
```
Figure 18

```
#Drop the outliers
z = np.abs(stats.zscore(house_data['house price of unit area']))
threshold = 1.3
outliers = house_data[z > threshold]

# drop rows containing outliers
house_data = house_data.drop(outliers.index)
```
Figure 19

```
#Drop the outliers
z = np.abs(stats.zscore(house_data['longitude']))
threshold = 0.04
outliers = house_data[z < threshold]

# drop rows containing outliers
house_data = house_data.drop(outliers.index)
```
Figure 20

Normalization:

The dataset we used did not need any normalization as the various attributes we had within the dataset were all of the same type.

Feature Selection/Extraction:

By looking at figures 2 to 6, and as discussed in Part 3, we decided to remove some features from the dataset and see how our ML model behaves in terms of accuracy improvement. The results are discussed more in depth in section 3.2 where we retrained and tested the model 3 more times.

a. By excluding the age from the dataset.
b. By excluding the number of nearby stores from the dataset.
c. By excluding the latitude and longitude from the dataset.

Data Splitting:
A good and proven way to use any given dataset in an ML solution is to split the dataset into two parts. The training set and the testing set.

The former will be used to feed the ML algorithm data from which will be trained, while the latter will be used to test and evaluate the solution.

A good way to split the data is to use 70% of the dataset as the training set while using the remaining 30% for testing and evaluating the solution.

Moreover, we constructed two datasets, one called X, which consists of all the features of the initial dataset, and another one, called y, which consists of the target label results of the initial dataset. Both sets will be used to train and test our ML models.

```
# b. Design ML Pipeline
# Split the dataset into features (X) and target label (y)
X = house_data.drop('house price of unit area', axis=1)
y = house_data['house price of unit area']
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
y_test = y_test.to_numpy()
```

Figure 21

2.2 Machine Learning Models Used/Developed:

Different Machine Learning models yield different results with various accuracy levels. So, for this project we decided to use five ML models in order to train them and evaluate their performance in predicting house prices using a predefined set of attributes.

The five ML models we decided to use are Linear Regression, Random Forest, K-Nearest Neighbors, Decision Tree Regression, and a Neural Network.

The aforementioned models represent supervised learning solutions which need an initial dataset from which they will learn to make predictions.

2.3 Well-designed ML Pipeline:

Model Initialization and Training:

Using ML models to perform certain tasks, like in our case making house price predictions/estimations, usually the model must first undergo from a series of steps like model training, evaluation, optimization, etc. Some of those steps are repeated many times until the ML model is trained well enough to make the required predictions.

After choosing the models, we used the training dataset, *X_train and y_train*, to initialize and train the models.

```
# Initialize and train the Linear Regression model
linear_reg_model = LinearRegression()
linear_reg_model.fit(X_train, y_train)
```

Figure 22

```
# Initialize and train the Random Forest Regressor model
rf_model = RandomForestRegressor()
rf_model.fit(X_train, y_train)
```

Figure 23

```
# Initialize and train the K-Nearest Neighbors model
knn_model = KNeighborsRegressor()
knn_model.fit(X_train, y_train)
```

Figure 24

```
# Initialize and train the Decision Tree Regression model
dt_model = DecisionTreeRegressor()
dt_model.fit(X_train, y_train)
```

Figure 25

```
# Initialize and train the Neural Network model
def build_nn_model():
    model = keras.Sequential([
        layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
        layers.Dense(32, activation='relu'),
        layers.Dense(1)  # Output layer with 1 neuron for regression
    ])

    model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
    return model

nn_model = build_nn_model()
nn_model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.3)
```

Figure 26

Using the trained ML models to make house price predictions:

After training the ML models with the X_train dataset we used them for making house price predictions using the X_test dataset.

```
# 4.1 Linear Regression Prediction

y_pred_linear_reg_predictions = linear_reg_model.predict(X_test)
```

Figure 27

```
# 4.2 Random Forest Prediction

y_pred_rf_predictions = rf_model.predict(X_test)
```

Figure 28

```
# 4.3 K-Nearest Neighbors Prediction

y_pred_knn_predictions = knn_model.predict(X_test)
```

Figure 29

```
# 4.4 Decision Tree Regression Prediction

y_pred_dt_predictions = dt_model.predict(X_test)
```

Figure 30

```
# 4.5 Neural Network Prediction

nn_model_predictions = nn_model.predict(X_test)
```

Figure 31

Part 3: Experiments, Results, and Evaluation

3.1 Evaluation:

For evaluating and improving our ML models, we used various techniques and methodologies which will be presented here.

The basic objective of this step was to see and analyse the initial performance of our five ML models, and figure our ways of how to improve them.

For evaluating the performance of the ML models we used the *Mean Squared Error* and the *R-Squared* techniques to compare the y_test and the y_predicted data, as well as some graphs to help us visualize the performance of each one of the models.

Linear Regression Evaluation:

The blue line in the graph shows the house prices as taken from the initial dataset (y_test - 30%) while the red dots represent the predicted values using the X_test.

By comparing the actual house prices against the predicted ones, we can see the accuracy of our model.

Using the mean squared error and the r-squared we can also quantify the model's accuracy.

```
Mean Squared Error: 26.299307332963192
R-squared: 0.6357150437536578
```

Figure 32

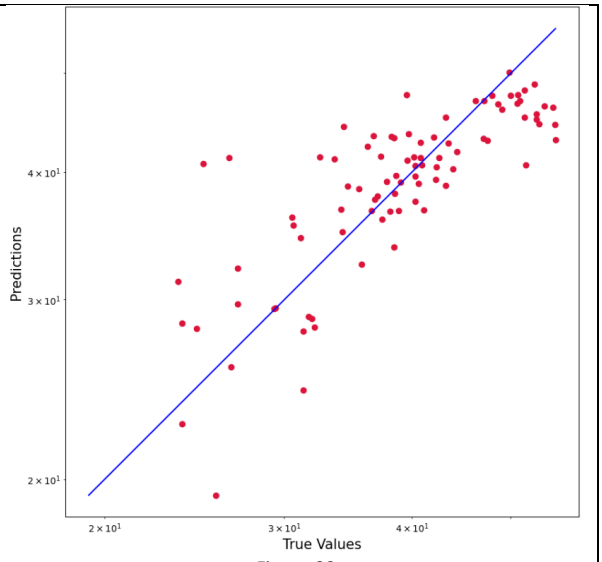

Figure 33

Random Forest Evaluation:

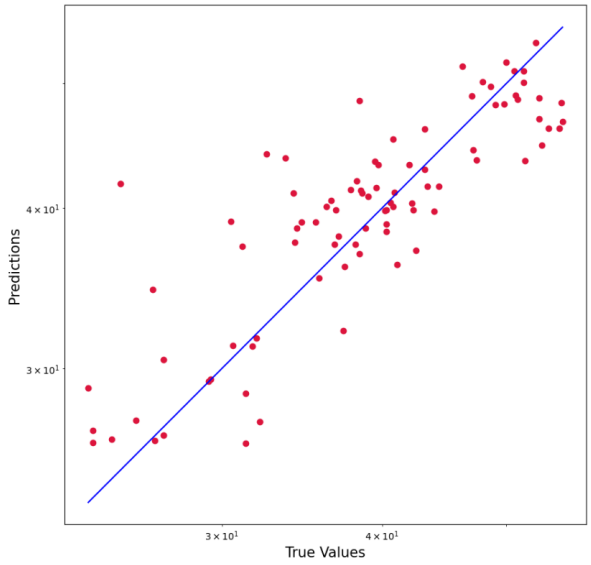

Figure 34

```
Mean Squared Error: 20.914366454545466
R-squared: 0.7103045729548636
```

Figure 35

K-Nearest Neighbors Evaluation:

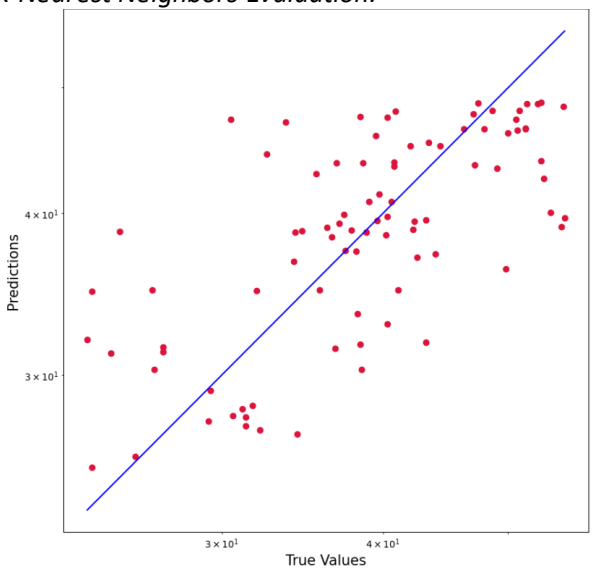

Figure 36

```
Mean Squared Error: 40.622718181818186
R-squared: 0.43731426352346514
```

Figure 37

Decision Tree Regression Evaluation

Neural Network Evaluation

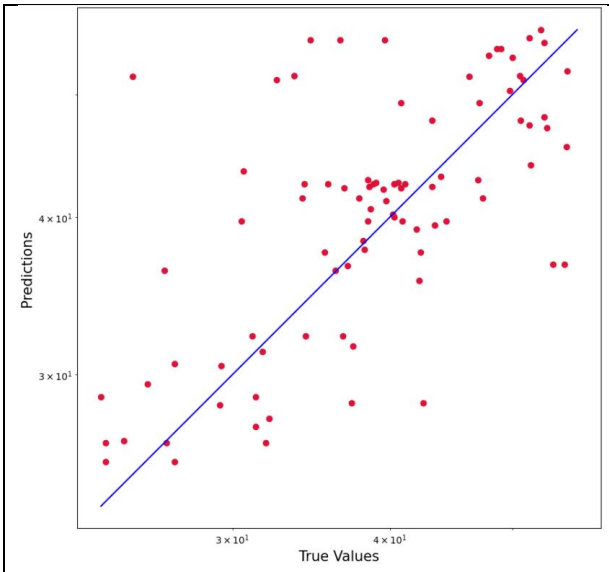

Figure 38

```
Mean Squared Error: 55.518977272727255
R-squared: 0.2309786736745072
```

Figure 39

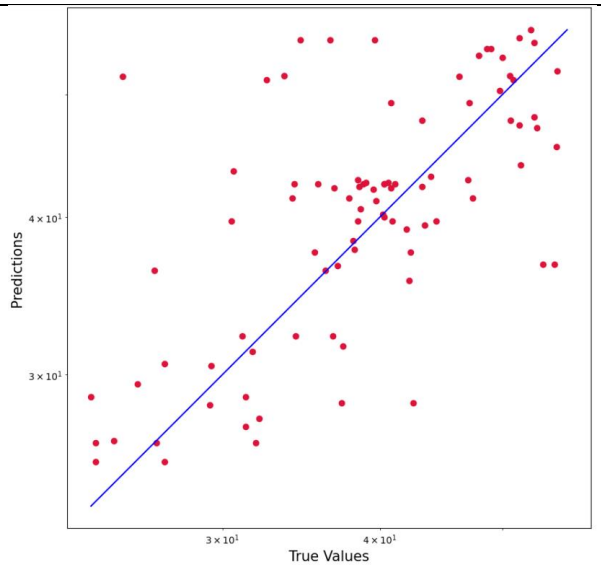

Figure 40

```
Mean Squared Error: 68.80439516214005
R-squared: 0.046955656897440634
```

Figure 41

3.2 Experiments and Results

From the initial predictions of the five models, we can see that the one performing the best is the *Random Forest* which has the least *Mean Squared Error, 20.9* and the highest *R-Squared, 0.71*.

As briefly discussed in the *Feature Selection/Extraction* section, we will try to improve our models by removing some of the attributes from the dataset.

The attributes will be removed one by one from the dataset, and by using our best performing ML model, the random forest, we will see how their removal from the dataset affect the model's performance.

Each time we remove one of the parameters, we calculate new training and testing dataset, using as always, the same 70/30 ratio.

Also, after we use the X_test to make predictions, we evaluate the model again using the mean squared error and the r-squared so that we can compare it with the previously calculated values and see whether our model has benefited from the attribute removal.

Random Forest - Excluding the house age:

```
X_new_plot = house_data_without_outliers.drop(['house age', 'house price of unit area'], axis=1)
y_new_plot = house_data_without_outliers['house price of unit area']

# Split the data into training and testing sets
X_train_new_plot, X_test_new_plot, y_train_new_plot, y_test_new_plot = train_test_split(
    X_new_plot, y_new_plot, test_size=0.3, random_state=42)

# Initialize and train the Linear Regression model
random_forest_model_new_plot = LinearRegression()
random_forest_model_new_plot.fit(X_train_new_plot, y_train_new_plot)

# Make predictions
y_pred_new_plot = random_forest_model_new_plot.predict(X_test_new_plot)
```

Figure 42

Random Forest - Excluding the nearest bus station:

```
X_new_plot = house_data_without_outliers.drop(['distance to the nearest bus station', 'house price of unit area'], axis=1
y_new_plot = house_data_without_outliers['house price of unit area']

# Split the data into training and testing sets
X_train_new_plot, X_test_new_plot, y_train_new_plot, y_test_new_plot = train_test_split(
    X_new_plot, y_new_plot, test_size=0.3, random_state=42)

# Initialize and train the Linear Regression model
random_forest_model_new_plot = RandomForestRegressor()
random_forest_model_new_plot.fit(X_train_new_plot, y_train_new_plot)

# Make predictions
y_pred_new_plot = random_forest_model_new_plot.predict(X_test_new_plot)
```

Figure 44

```
Mean Squared Error: 40.6983999040589
R-squared: 0.43626595785799427
```

Figure 43

```
Mean Squared Error: 25.05643522727274
R-squared: 0.652930691485709
```

Figure 45

Random Forest - Excluding the convenience stores:

```
X_new_plot = house_data_without_outliers.drop(['number of convenience stores', 'house price of unit area'], axis=1)
y_new_plot = house_data_without_outliers['house price of unit area']

# Split the data into training and testing sets
X_train_new_plot, X_test_new_plot, y_train_new_plot, y_test_new_plot = train_test_split(
    X_new_plot, y_new_plot, test_size=0.3, random_state=42)

# Initialize and train the linear Regression model
random_forest_model_new_plot = RandomForestRegressor()
random_forest_model_new_plot.fit(X_train_new_plot, y_train_new_plot)

# Make predictions
y_pred_new_plot = random_forest_model_new_plot.predict(X_test_new_plot)
```

Figure 46

```
Mean Squared Error: 23.445501125000018
R-squared: 0.6752445513730614
```

Figure 47

By comparing the mean squared errors and the r-squared of the initially trained random forest with the random forests for which were trained with datasets that were missing one attribute at a time, we can see that the initial model performed better than the others.

Improving performance using Hyperparameters:

Now we will again try to improve our models by using Hyperparameters by controlling how the models are trained.

Random Forest:

For trying to improve the random forest model using Hyperparameters, we used the following:

- 'n_estimators': [50, 100, 200],
- 'max_depth': [None, 10, 20],
- 'min_samples_split': [2, 5, 10],
- 'min_samples_leaf': [1, 2, 4]

After running the model we found that the best parameters for our model were the following:

- N_estimators = 50
- Max_depth = none
- Min_samples_split = 2
- Min_samples_leaf = 4

Using the initial X_test we got the following results:

- Mean squared error: 22.48
- R-squared: 0.68

Comparing the initial mean squared error (20.91) and the initial r-squared (0.71) with the ones which we got using hyperparameters for the random forest, we conclude that using hyperparameters did not improve our model.

Performing the same kind of optimization for the KNN and the decision tree, here are the results:

K-Nearest Neighbors:

```
param_grid_knn = {
   'n_neighbors': [3, 5, 7, 10],
   'weights': ['uniform', 'distance'],
   'p': [1, 2]
}
```

Best Hyperparameters:

- N_neighbors: 10
- Weights: distance
- P: 1

Evaluation Results:

- Mean squared error: 33.12
- R-squared: 0.54

Decision Tree Regression:

```
param_grid_dt = {
   'max_depth': [None, 10, 20, 30],
   'min_samples_split': [2, 5, 10],
   'min_samples_leaf': [1, 2, 4]
}
```

Best Hyperparameters:

- Max_depth: None
- Min_samples_split: 10
- Min_samples_leaf: 4

Evaluation Results:

- Mean squared error: 35.75
- R-squared: 0.50

Comparing the prediction results of the initially trained KNN and DTR models, we can clearly see that by utilizing Hyperparameters we managed to significantly improve the prediction performance of those models.

Part 4: Conclusion

In this project we have built a house price predicting Machine Learning solution in which, by using a predefined dataset, we utilised and trained five ML models to make predictions on house prices given certain parameters.

The models were evaluated using the mean-squared error and the r-squared techniques.

Some models initially fared better than others, with the best performing one being the solution used incorporated the Random Forest ML model.

Upon finishing and evaluating the initial models, we tried various methodologies to try and improve the performance of the models, with the two most important ones being the feature selection/extraction and the hyperparameters.

From the results, we can see that by utilizing the feature reduction method we did not see an improvement in the overall performance of the models, while by using the hyperparameters method we could see a significant improvement in some of the models.

The overall performance of the models though, either by optimizing them or not, although good, it was not great. The reason I believe we were not able to get excellent results on the house prices' prediction was that the dataset we used was too small so the datapoints actually used for training the model were not adequate enough.

Part 5: Future Work

Improving ML models' performance is an almost never-ending task.

In our case, there are numerous things that we can do to improve our models' accuracy and overall performance.

First, we can try to find a larger dataset consisting of thousands of records and attributes instead of one consisting of records which are less than 500 and just 5 attributes.

That way, our model would benefit from having a larger training sample from which it can learn how to make predictions and also, we could use attribute reduction techniques so that we can make our models stronger in correctly predicting house prices.

Moreover, we incorporate cost functions into the overall performance analysis and improvement of the models.

Finally, after making our models accurate, we could use them and try to solve other real-life problems and see how they could perform into undertaking and solving different tasks.

Part 6: References:

1. Alexisbcook (2023) *Scatter plots*, *Kaggle*. Available at: https://www.kaggle.com/code/alexisbcook/scatter-plots.
2. Georgezoto (2020) *Data Visualization - scatter plots*, *Kaggle*. Available at: https://www.kaggle.com/code/georgezoto/data-visualization-scatter-plots.
3. kaushiksuresh147 (2021) *Box plot python implemenation*, *Kaggle*. Available at: https://www.kaggle.com/code/kaushiksuresh147/box-plot-python-implemenation/notebook.
4. Lusfernandotorres (2023) *The abcs of machine learning: 4 essential models*, *Kaggle*. Available at: https://www.kaggle.com/code/lusfernandotorres/the-abcs-of-machine-learning-4-essential-models#kmeans.
5. sudhirnl7 (2020) *Linear regression tutorial*, *Kaggle*. Available at: https://www.kaggle.com/code/sudhirnl7/linear-regression-tutorial.
6. prashant111 (2020) *Random forest classifier tutorial*, *Kaggle*. Available at: https://www.kaggle.com/code/prashant111/random-forest-classifier-tutorial.
7. Marcinrutecki (2022) *Regression models evaluation metrics*, *Kaggle*. Available at: https://www.kaggle.com/code/marcinrutecki/regression-models-evaluation-metrics.
8. pateljay731 (2021) *Prediction with the KNN classifier*, *Kaggle*. Available at: https://www.kaggle.com/code/pateljay731/prediction-with-the-knn-classifier.
9. prashant111 (2020) *Decision-tree classifier tutorial*, *Kaggle*. Available at: https://www.kaggle.com/code/prashant111/decision-tree-classifier-tutorial.
10. Ryanholbrook (2023) *Deep Neural Networks*, *Kaggle*. Available at: https://www.kaggle.com/code/ryanholbrook/deep-neural-networks.
11. shreayan98c (2020) *Hyperparameter tuning tutorial*, *Kaggle*. Available at: https://www.kaggle.com/code/shreayan98c/hyperparameter-tuning-tutorial.
12. Viveknimsarkar (2021) *Machine Learning Model Evaluation and error metric*, *Kaggle*. Available at: https://www.kaggle.com/code/viveknimsarkar/machine-learning-model-evaluation-and-error-metric.