SUNWAY UNIVERSITY
Sunway BUSINESS SCHOOL

ASSIGNMENT COVER SHEET

PROGRAMME	:	Master's in Business Analytics (MsBA)
SUBJECT CODE AND TITLE	:	BAA5053 – Machine Learning for Business Decisions
ASSIGNMENT TITLE	:	Machine Learning for Business Decisions – Individual Assignment
LECTURER	:	Dr. Aaron Aw Teik Hong ASSIGNMENT DUE DATE: 13/12/2024

STUDENT'S DECLARATION

1. I hereby declare that this assignment is based on my own work except where acknowledgement of sources is made.
2. I also declare that this work has not been previously submitted or concurrently submitted for any other courses in Sunway University/College or other institutions.
   [Submit "Turn-it-in" report (please tick √): Yes __√__ No _____]

NO.	NAME	STUDENT ID NO.	SIGNATURE	DATE
1.	Harresh A/L Ragunathan	19076090	*Harresh*	13/12/2024

E-mail Address / Addresses (according to the order of names above):

1. 19076090@imail.sunway.edu.my	

APPROVAL FOR LATE SUBMISSION OF ASSIGNMENT (If applicable)
IF extension is granted, what is the revised due date? ______________________________

Signature of Lecturer: ____________________________ Date: ____________________

Marker's Comments:

Marks and / or Grade Awarded: ________________________ Date:________________________

ADDENDUM

USE OF ARTIFICAL INTELLIGENCE (A.I.) DECLARATION

Students are allowed to use AI to support completion of assessments. However, students are reminded to do so ethically and transparently. This is so that (a) submissions can be fairly and accurately marked; and (b) feedback can be provided on the content that reflects student ability, in order to help with future submissions. Students are also reminded that in accordance with the University's Academic Malpractice Policy, Item 4.11.2, "*… the representation of work: written, visual, practical or otherwise, of any other person, including another student or **anonymous web-based material** [emphasis added], or any institution, as the candidate's own*" is considered malpractice.

Declaration

[√] I / We used the following A.I. tools to produce content in this submission:

Tool	**Purpose**	**Prompts**	**Sections where AI output was used / Outcome(s) in the submission**
e.g. ChatGPT	*e.g. Generating points for the essay* *Structuring the essay*	*e.g. "Give me 5 key talking points for an essay on…"* *"Show me a structure for an essay on…"*	*e.g. The main point for Section 1.2 and 1.3 were generated by AI, but the discussion was not.* *The organization / structure of the essay was suggested by AI*
e.g. Grammarly	*e.g. Correcting grammar and spelling, improving sentence structure*	*N/A*	*e.g. Grammarly suggestions were used for all sections of the essay*

Note: Add additional rows if necessary.

OR

[ ] I / We did not use any A.I. tools to produce any of the content in this submission.

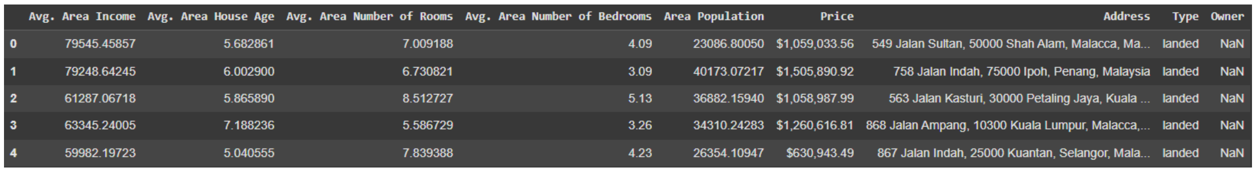
NO.	NAME	STUDENT ID NO.	SIGNATURE	DATE
1.	Harresh A/L Ragunathan	19076090	*Harresh*	13/12/2024

E-mail Address / Addresses (according to the order of names above):

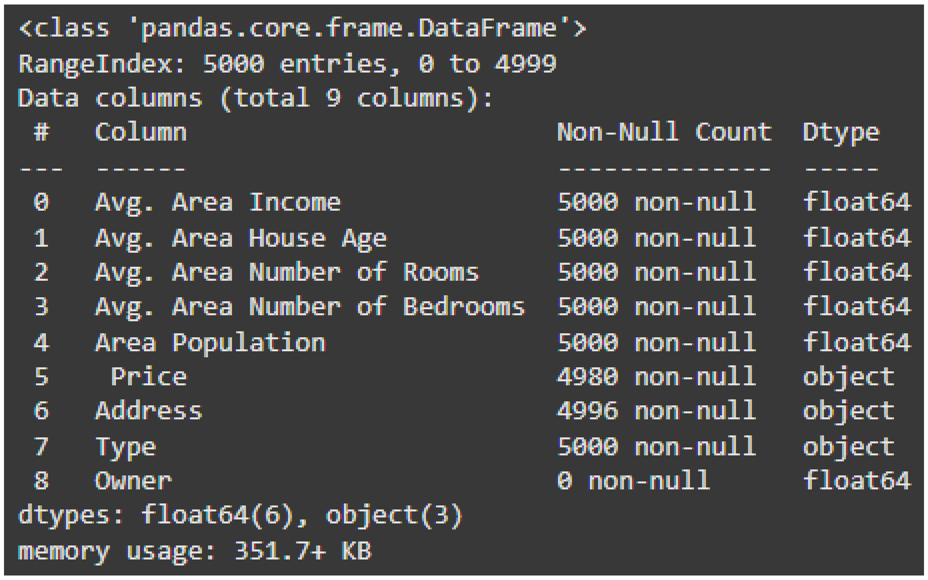
1. 19076090@imail.sunway.edu.my	

Table of Contents

Task 1 (i)

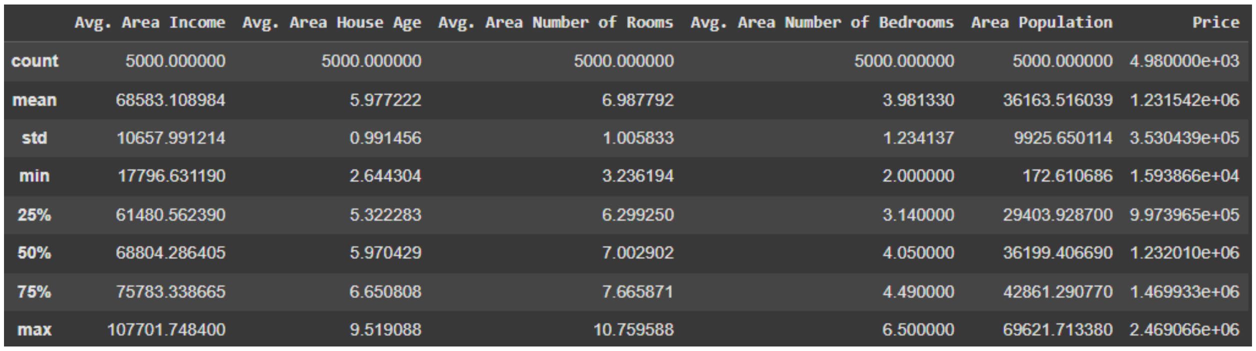


	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address	Type	Owner
0	79545.45857	5.682861	7.009188	4.09	23086.80050	$1,059,033.56	549 Jalan Sultan, 50000 Shah Alam, Malacca, Ma...	landed	NaN
1	79248.64245	6.002900	6.730821	3.09	40173.07217	$1,505,890.92	758 Jalan Indah, 75000 Ipoh, Penang, Malaysia	landed	NaN
2	61287.06718	5.865890	8.512727	5.13	36882.15940	$1,058,987.99	563 Jalan Kasturi, 30000 Petaling Jaya, Kuala ...	landed	NaN
3	63345.24005	7.188236	5.586729	3.26	34310.24283	$1,260,616.81	868 Jalan Ampang, 10300 Kuala Lumpur, Malacca,...	landed	NaN
4	59982.19723	5.040555	7.839388	4.23	26354.10947	$630,943.49	867 Jalan Indah, 25000 Kuantan, Selangor, Mala...	landed	NaN



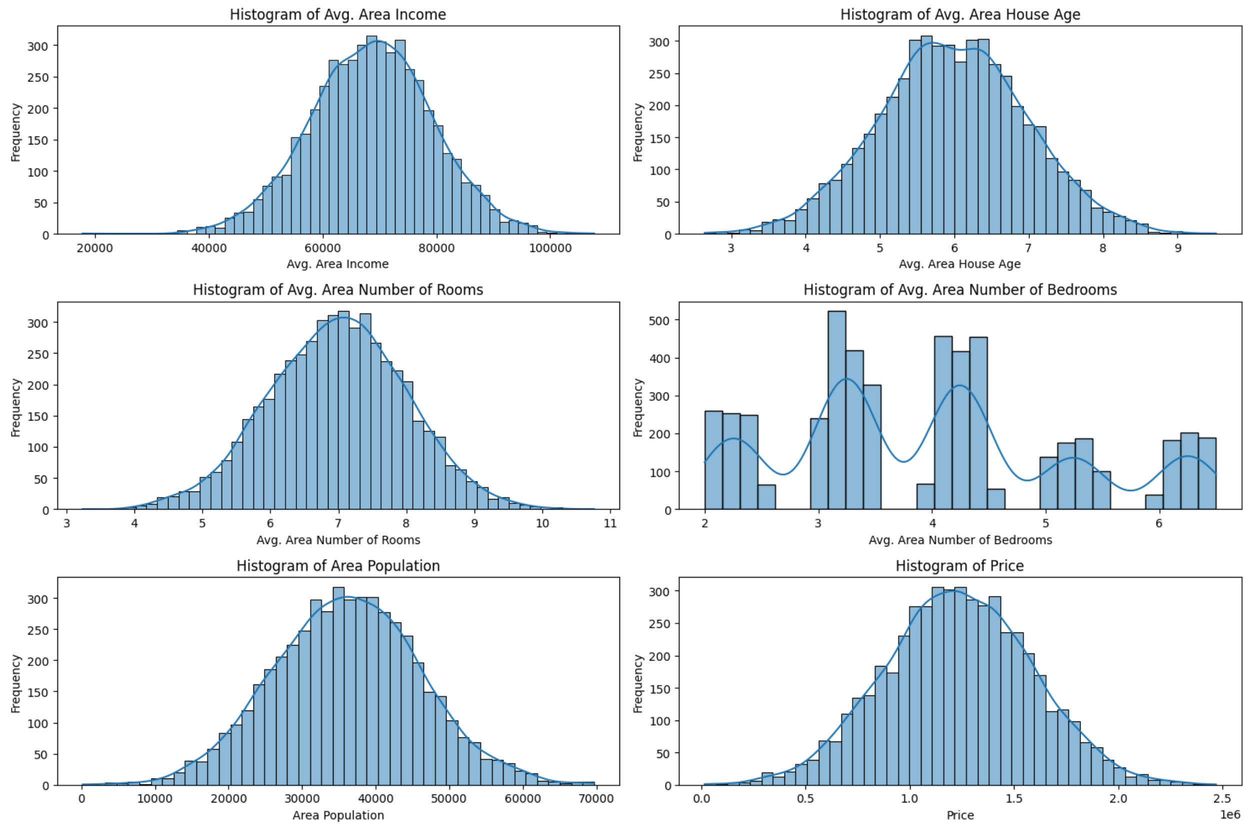
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 9 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5    Price                        4980 non-null   object 
 6   Address                       4996 non-null   object 
 7   Type                          5000 non-null   object 
 8   Owner                         0 non-null      float64
dtypes: float64(6), object(3)
memory usage: 351.7+ KB
```

Using Python in Google Colab, the data was loaded into a dataframe. By observing the first 5 rows and the information about the dataframe, it may be seen that this dataset has 9 columns and 5000 rows. Based on the information output, there are 6 numerical columns and 3 categorical columns. It may be noticed that the Dtype for "Price" as "object". This is because in the original dataset, there are dollar signs ($) and commas (,) present. Therefore, these symbols must be removed to be considered a numerical variable.



	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	4.980000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.231542e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.530439e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562390	5.322283	6.299250	3.140000	29403.928700	9.973965e+05
50%	68804.286405	5.970429	7.002902	4.050000	36199.406690	1.232010e+06
75%	75783.338665	6.650808	7.665871	4.490000	42861.290770	1.469933e+06
max	107701.748400	9.519088	10.759588	6.500000	69621.713380	2.469066e+06

Once the symbols have been removed from the values of "Price", the summary statistics may now be displayed for all the numerical variables. Based on the summary statistics output, it may be concluded that each variable’s distribution is relatively normal, as their means and medians are relatively equal, with minimal deviation. Of all the variables, only “Avg. Area Number of Bedrooms” demonstrates slight skewness to the left, as its median, 4.05, is higher than its mean, 3.98. This variable has the highest difference in mean and median among the rest of the variables.

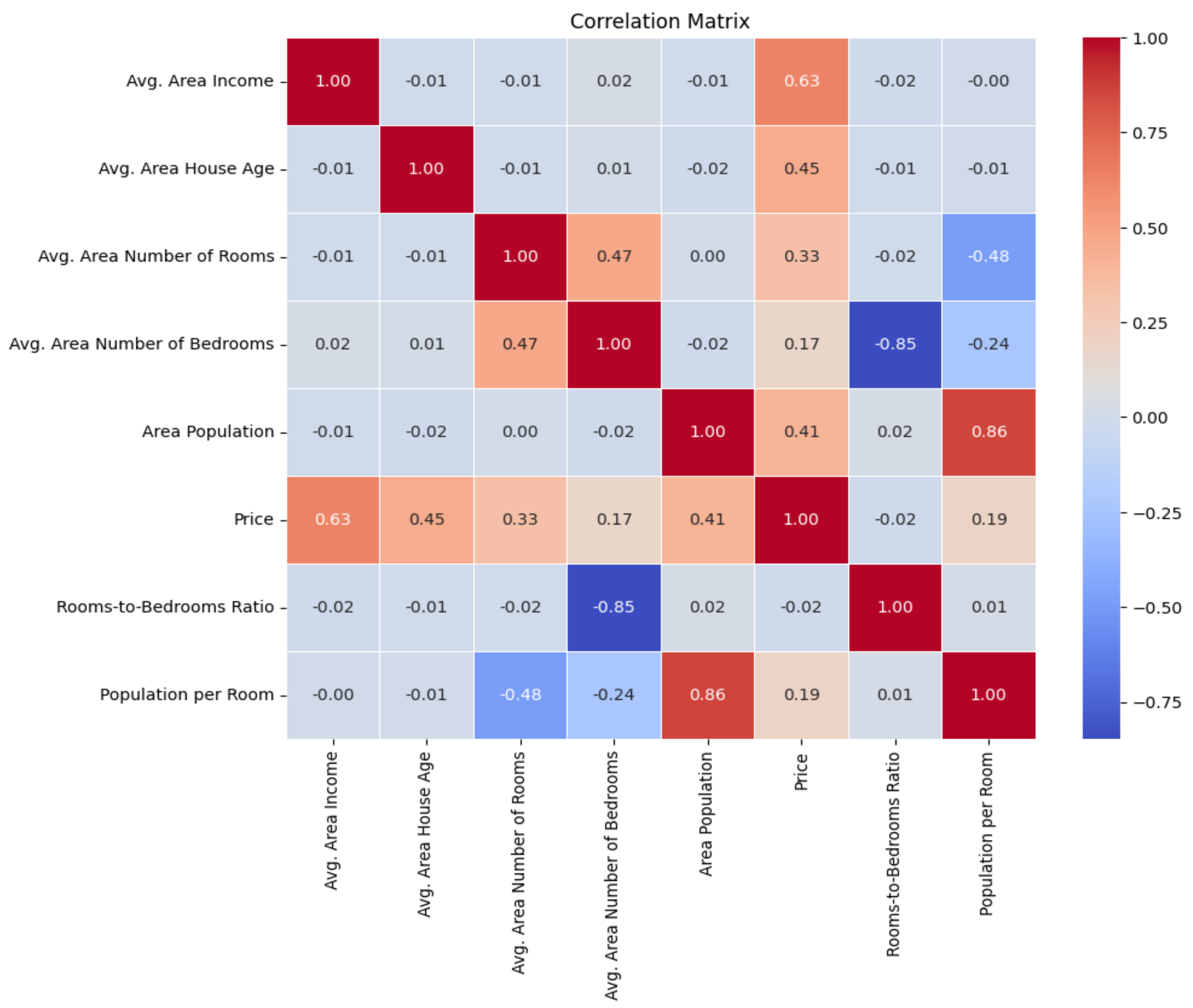


The histograms above illustrate the distribution of each variable. Based on the histograms, it may be observed that the variables “Avg. Area Income”, “Avg. Area House Age”, “Avg. Area Number of Rooms”, “Area Population”, and “Price” portray a normal distribution, as it illustrates a bell-shaped curve.

Task 1 (ii)

Feature engineering may be defined as creating new features through transformation of existing features in a dataset. Feature engineering can improve model performance by creating more meaningful and interpretable features in terms of the representations of the relationships in the data. The first newly created feature is “Rooms-to-Bedrooms Ratio”, which is the ratio of “Avg. Area Number of Rooms” and “Avg. Area Number of Bedrooms”. This ratio may give an idea of the size of the property, as a higher ratio could represent a larger property, while a smaller ratio could represent a smaller property. The other newly created feature is “Population per Room”, which is computed by dividing “Area Population” by “Avg. Area Number of Rooms”. This feature measures the overall housing density, as a higher value may indicate a relatively urban area, while a lower value may indicate a relatively rural area.

Task 1 (iii)

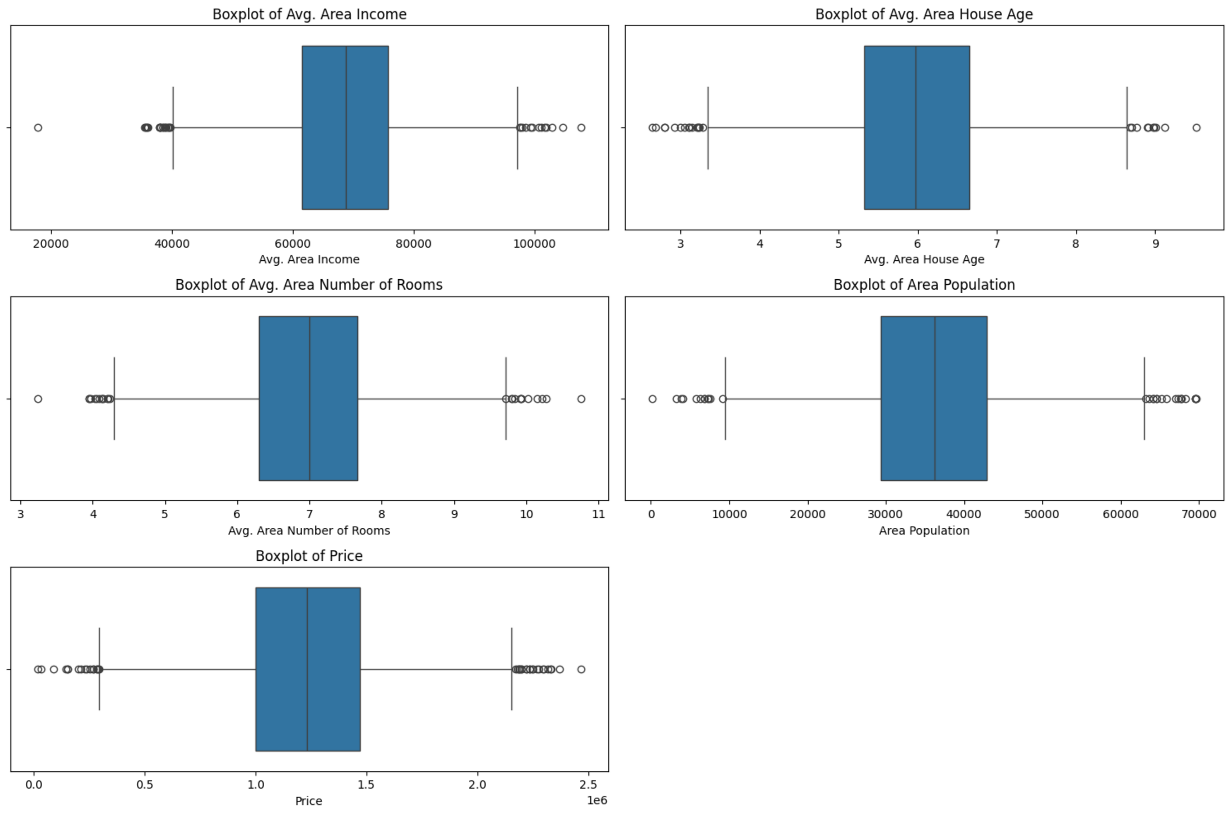


	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Rooms-to-Bedrooms Ratio	Population per Room
Avg. Area Income	1.00	-0.01	-0.01	0.02	-0.01	0.63	-0.02	-0.00
Avg. Area House Age	-0.01	1.00	-0.01	0.01	-0.02	0.45	-0.01	-0.01
Avg. Area Number of Rooms	-0.01	-0.01	1.00	0.47	0.00	0.33	-0.02	-0.48
Avg. Area Number of Bedrooms	0.02	0.01	0.47	1.00	-0.02	0.17	-0.85	-0.24
Area Population	-0.01	-0.02	0.00	-0.02	1.00	0.41	0.02	0.86
Price	0.63	0.45	0.33	0.17	0.41	1.00	-0.02	0.19
Rooms-to-Bedrooms Ratio	-0.02	-0.01	-0.02	-0.85	0.02	-0.02	1.00	0.01
Population per Room	-0.00	-0.01	-0.48	-0.24	0.86	0.19	0.01	1.00

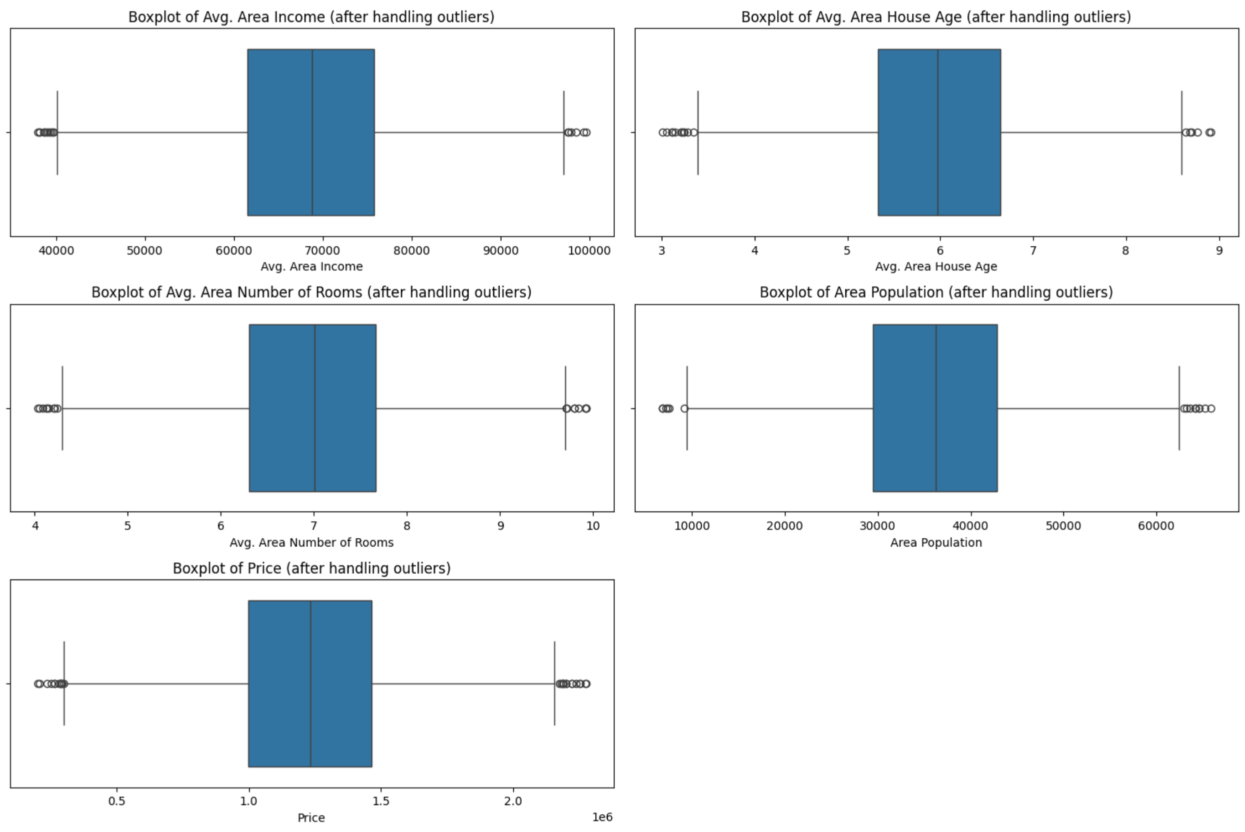
A correlation heatmap was plotted to explore the relationship between each variable. The correlation heatmap above was plotted after the data has been cleaned (handled missing values and outliers) and feature engineering was performed. Based on the correlation heatmap, it may be noticed that “Price” has the strongest relationship with “Avg. Area Income”. This implies that if the average area income were to increase the price of the property will have the tendency to also increase. This relationship reflects the real world, as properties of higher value tend to be located in higher income areas. The next strongest relationship is between “Price” and “Avg. House Age”. This implies that an older property would lead to an increase in price. In terms of weak relationships between “Price” and the other variables, the weakest relationship is with “Population per Room. This implies that more people per room would lead to a slight increase in price. This is a reflection of housing demand where property values remain high in crowded areas.

Task 1 (iv)

In order for the data to be used in machine learning models, the values for each feature must be numerical. This means that any categorical column must be converted to numerical using various methods. The only categorical variable in the dataset is “Address”. In this case, the “Address” column was split into “Street Name”, “Postal Code”, “City”, and “State”. This is due to the intention of performing one-hot encoding to convert these columns to numeric. One-hot encoding was applied to these columns as these newly split columns are nominal categorical variables, meaning that they have no intrinsic order or ranking. The “Address” column was split this way because upon initial inspection of the data, the “Address” column has 4989 unique values. If one-hot encoding was applied to the “Address” column alone, it would result in 4988 columns, each column representing one unique address. This issue was solved by splitting the “Address” into “Street Name”, “Postal Code”, “City”, and “State” columns. Checking the number of unique values for each column, “Street Name”, “Postal Code”, and “City” have 8 unique values, while “State” has 7 unique values. Now that the “Address” column has been split, one-hot encoding may be applied. As a result, the dataset now has 33 columns.



Observing the box plot for the numerical variables, it may be seen that there are outliers for all these variables. It may be argued that these outliers are reasonable as they do not deviate too far from the range of the data. However, the extreme values may affect the performance of the machine learning models, which is why the outliers were handled. By inspecting the distributions and skewness of each variable, it may be concluded that they all demonstrate a relatively normal distribution. Therefore, the outliers were handled utilizing the z-score method.



The plots above are the box plots for each variable after handling the outliers using the z-score method. As a result of handling the outliers, the extreme values have been eliminated (from 4996 rows to 4939 rows), but some outliers are still present. However, as seen in the box plots, these outliers are all within the range of the data, therefore, these outliers will be kept.

Task 2 (i)

After the data has gone through pre-processing, it is now ready to be used to conduct machine learning. Since our dependent variable is "Price", regression-based models will be used and tested. The test models include Linear Regression, Decision Tree, Random Forest (RF), K-Nearest Neighbours (KNN), XGBoost (XGB), and Support Vector Machine (SVM).

Linear Regression is a procedure that estimates the coefficients of the linear equation, with at least one independent variable that best predicts the value of a dependent variable. It is generally used when the distribution of the target variable is continuous (Zhang, 2021). Linear Regression is a suitable machine learning model to predict property prices as price is a continuous variable. Furthermore, it will provide the coefficients of each variable, allowing them to be interpreted based on how they influence the price. Lastly, Linear Regression is computationally efficient, meaning it will be able to make predictions quickly and efficiently.

Decision Tree is a classical machine learning model with the goal of classifying or regressing data through a set of rules. This rule-like approach will decide what values will be obtained under what conditions. The type of decision tree that will be utilized is regression trees, since the target variable is continuous (Z. Zhang, 2021). Decision Tree is a suitable machine learning model for predicting property prices as it is able to handle complex relationships between the features of properties and the price. However, it should be noted that Decision Tree is prone to overfitting, meaning it may not perform well on unseen data.

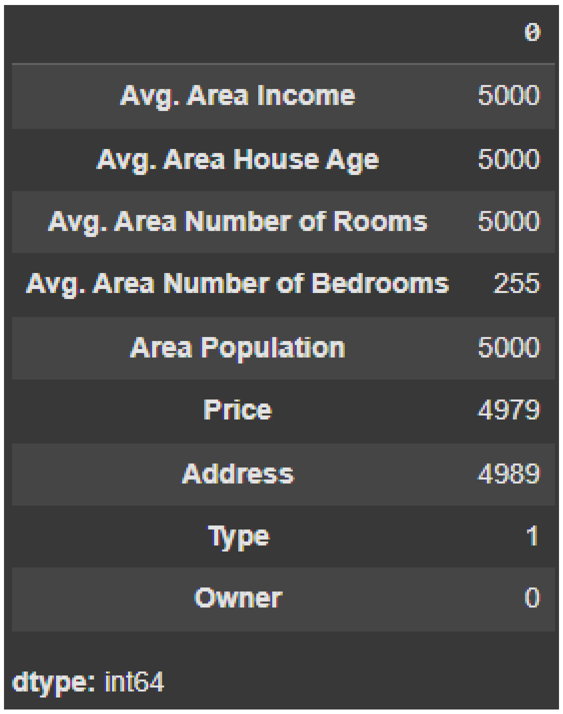
Random Forest (RF) is an ensemble machine learning model that combines the prediction of multiple decision trees in order to produce a more accurate prediction (Truong, Nguyen, Dang, & Mei, 2020). RF is a suitable machine learning model because it is able to handle large datasets and is robust enough to handle outliers and noise. However, it is computationally intensive, especially with larger datasets, since it runs multiple decision trees.

K-Nearest Neighbours (KNN) is an algorithm that finds the nearest neighbour to an item that has some attributes to be predicted, in this case property price. A prediction is made by searching through the training set for the K most similar instances and summarizing the output variable for those K instances (Zhao, 2018). KNN is a suitable algorithm as it captures local patterns and is able to adapt to non-linear relationships. However, it is relatively sensitive to the choice made regarding the number of neighbours (k), meaning it will need to be optimally tuned. Furthermore, it is effective for smaller datasets but may be computationally expensive for larger datasets.

XGBoost (XGB), also known as Extreme Gradient Boosting, is a scalable and efficient implementation of gradient tree boosting and may be used to solve real-world scale problems with minimal resources (Sharma, Harsora, & Ogunleye, 2024). XGB is a suitable machine learning model as it thrives in handling complex relationships. Furthermore, its optimization techniques make it very efficient with larger datasets. However, XGB models need careful hyperparameter tuning in order to prevent overfitting. Additionally, it is computationally expensive compared to other models.

Support Vector Machine (SVM) is a machine learning algorithm based on the structural risk minimization principle in order to minimize the number of expected errors (Gu, Zhu, & Jiang, 2011). SVM is a suitable model because it is efficient with both linear and non-linear relationships. Furthermore, it leads to better generalizations due to its optimal decision boundaries. However, SVM is computationally intensive with larger datasets and requires careful hyperparameter tuning.

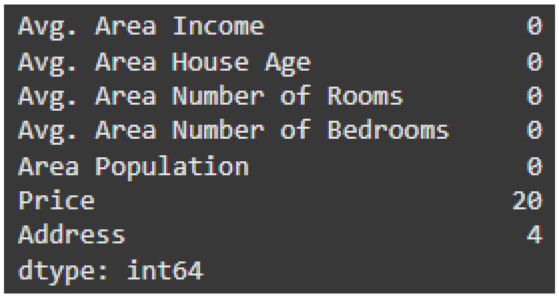
Task 2 (ii)



	0
Avg. Area Income	5000
Avg. Area House Age	5000
Avg. Area Number of Rooms	5000
Avg. Area Number of Bedrooms	255
Area Population	5000
Price	4979
Address	4989
Type	1
Owner	0

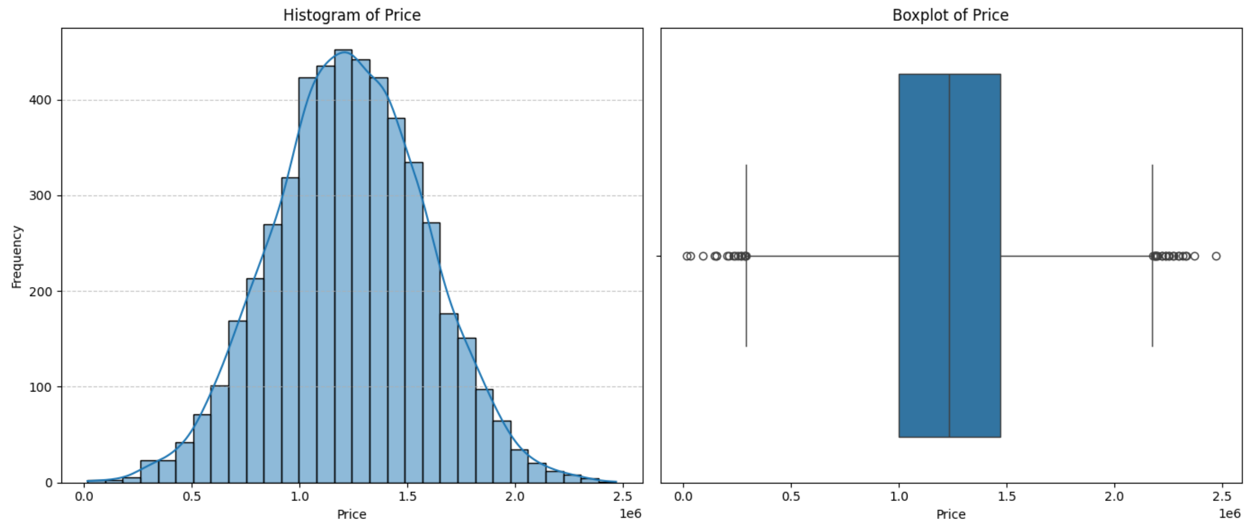
dtype: int64

Before running the machine learning models, the cleanliness of the data must first be assessed and addressed. The table above displays the number of unique values in each column. Based on this table, it may be noticed that the "Type" column has 1 unique value while the "Owner" column has 0 unique values. These columns do not provide any value to machine learning; therefore, they were dropped from the dataset entirely.

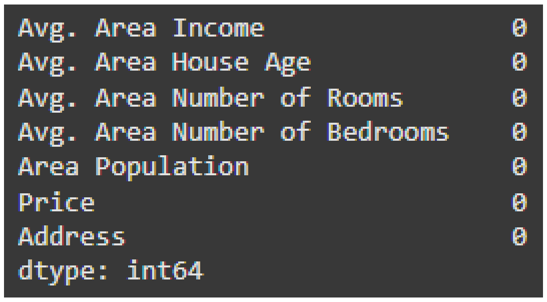


```
Avg. Area Income                0
Avg. Area House Age             0
Avg. Area Number of Rooms       0
Avg. Area Number of Bedrooms    0
Area Population                 0
Price                          20
Address                         4
dtype: int64
```

After the "Type" and "Owner" columns were dropped, the missing values were checked. As seen above, there are 4 missing values in the "Address" column and 20 missing values in the "Price" column. Since addresses are difficult to impute, as they are a unique value, those rows were dropped. In terms of replacing the values for "Price", the distribution needs to be inspected to determine if mean or median imputation should be performed.



Based on the histogram and box plot above, it may be concluded that the distribution of “Price” is relatively normal. Therefore, either the mean or median could be used to replace the missing values for “Price”. In this case, the mean was used to replace those values.

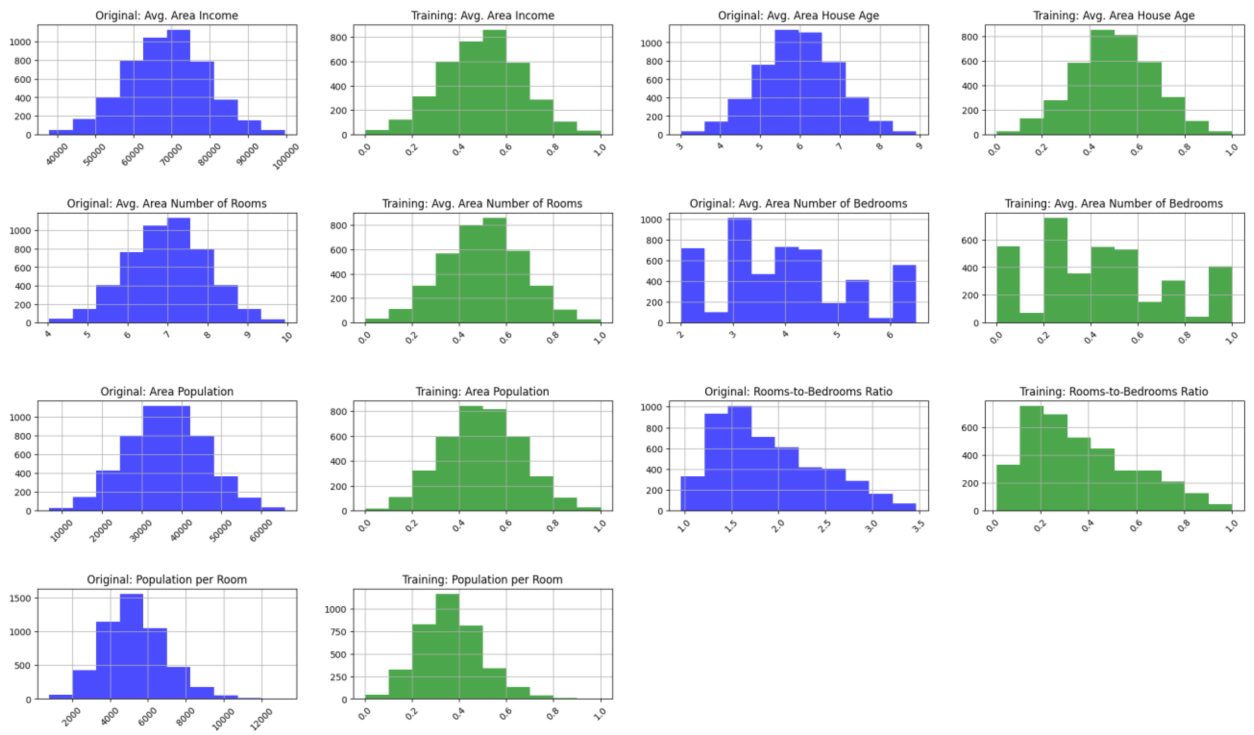


```
Avg. Area Income                0
Avg. Area House Age             0
Avg. Area Number of Rooms       0
Avg. Area Number of Bedrooms    0
Area Population                 0
Price                           0
Address                         0
dtype: int64
```

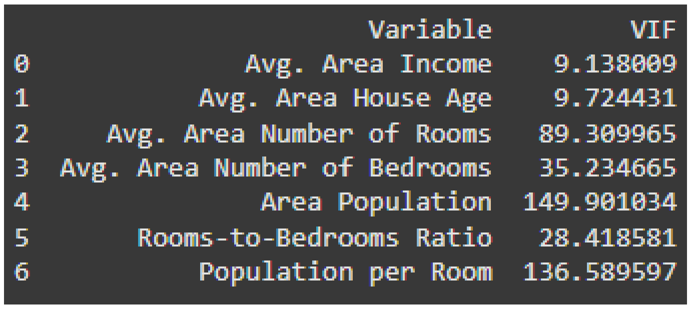
As seen above, the missing values have been handled accordingly, resulting in zero missing values.

Task 2 (iii)

In order to begin testing the machine learning models, the data must first be split into training and testing sets. This was done by splitting 75% of the data and using it as the training set to train the model, while the remaining 25% was used as a testing set to evaluate the model.

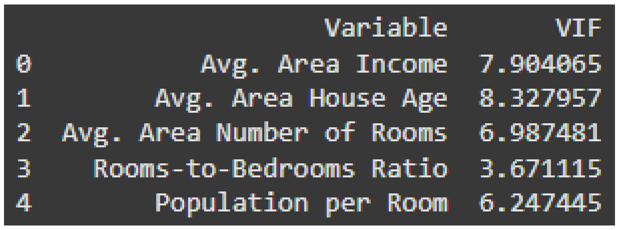


The histograms above compare the distributions of each variable in the training set with the variables in the original data. It may be observed that the distributions of the variables from the training set strongly reflect the distributions of the variables from the original dataset, with a few minor, but negligible, variations.



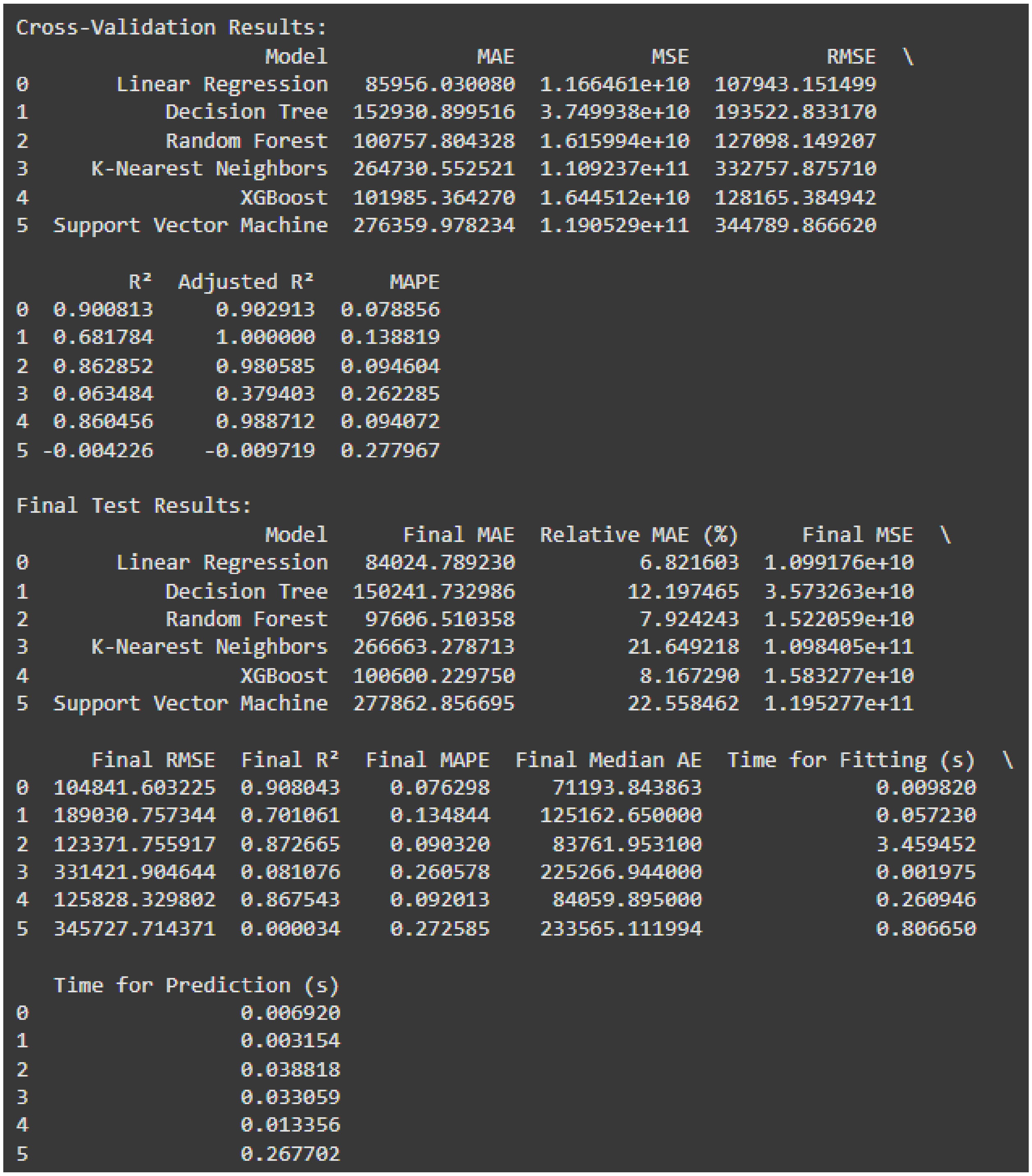
```
                         Variable         VIF
0                Avg. Area Income    9.138009
1             Avg. Area House Age    9.724431
2       Avg. Area Number of Rooms   89.309965
3    Avg. Area Number of Bedrooms   35.234665
4                 Area Population  149.901034
5         Rooms-to-Bedrooms Ratio   28.418581
6             Population per Room  136.589597
```

Additionally, the variables must be checked for multicollinearity. As seen above, the VIFs are relatively high, meaning there are occurrences of multicollinearity, especially between “Area Population”, “Population per room”, “Avg. Area Number of Rooms”, and “Avg. Area Number of Bedrooms”. In order to resolve this issue, some variables will need to be removed to prevent conflicts between variables.



```
                    Variable       VIF
0           Avg. Area Income  7.904065
1        Avg. Area House Age  8.327957
2  Avg. Area Number of Rooms  6.987481
3    Rooms-to-Bedrooms Ratio  3.671115
4        Population per Room  6.247445
```

After removing the “Area Population” and “Avg. Area Number of Bedrooms” columns, the VIFs are much lower, as seen above.



```
Cross-Validation Results:
                    Model            MAE           MSE           RMSE  \
0       Linear Regression   85956.030080  1.166461e+10  107943.151499
1           Decision Tree  152930.899516  3.749938e+10  193522.833170
2           Random Forest  100757.804328  1.615994e+10  127098.149207
3     K-Nearest Neighbors  264730.552521  1.109237e+11  332757.875710
4                 XGBoost  101985.364270  1.644512e+10  128165.384942
5  Support Vector Machine  276359.978234  1.190529e+11  344789.866620

         R²  Adjusted R²      MAPE
0  0.900813     0.902913  0.078856
1  0.681784     1.000000  0.138819
2  0.862852     0.980585  0.094604
3  0.063484     0.379403  0.262285
4  0.860456     0.988712  0.094072
5 -0.004226    -0.009719  0.277967

Final Test Results:
                    Model      Final MAE  Relative MAE (%)     Final MSE  \
0       Linear Regression   84024.789230          6.821603  1.099176e+10
1           Decision Tree  150241.732986         12.197465  3.573263e+10
2           Random Forest   97606.510358          7.924243  1.522059e+10
3     K-Nearest Neighbors  266663.278713         21.649218  1.098405e+11
4                 XGBoost  100600.229750          8.167290  1.583277e+10
5  Support Vector Machine  277862.856695         22.558462  1.195277e+11

      Final RMSE  Final R²  Final MAPE  Final Median AE  Time for Fitting (s)  \
0  104841.603225  0.908043    0.076298     71193.843863              0.009820
1  189030.757344  0.701061    0.134844    125162.650000              0.057230
2  123371.755917  0.872665    0.090320     83761.953100              3.459452
3  331421.904644  0.081076    0.260578    225266.944000              0.001975
4  125828.329802  0.867543    0.092013     84059.895000              0.260946
5  345727.714371  0.000034    0.272585    233565.111994              0.806650

   Time for Prediction (s)
0                 0.006920
1                 0.003154
2                 0.038818
3                 0.033059
4                 0.013356
5                 0.267702
```

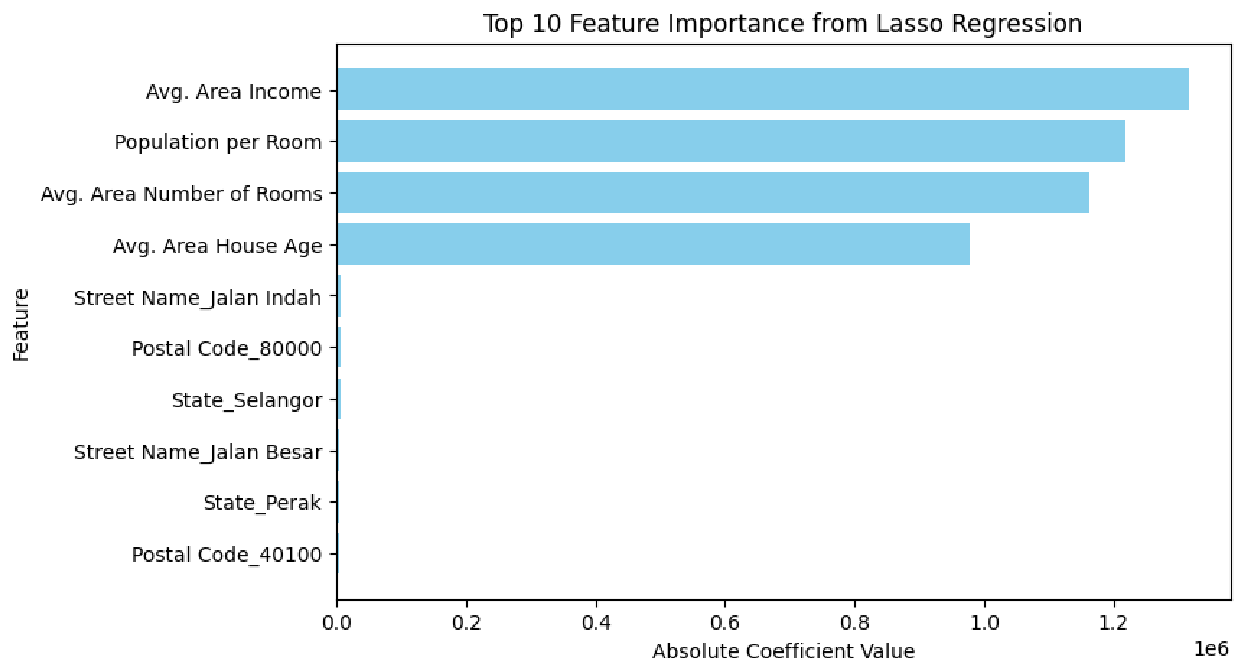
The results above display all 6 regression model's performance metrics using both k-fold cross-validation (10-folds) and using the test-set for evaluation. Both evaluation methods are utilized so that their performance metrics may be compared to check for overfitting. Focusing on the cross-validation results, Linear Regression thrives in all performance metrics compared to the other models due to its desirable performance metrics. Notably, Linear Regression produces the lowest error metrics, specifically the mean absolute error (MAE) and root mean squared error (RMSE). Furthermore, it produces the highest R^2 value of 0.9. Looking at the results from the test-set results, Linear Regression once again thrives as the best performing model due to its desirable performance metrics. Notably, it produces the lowest MAE, Relative MAE, and RMSE. Additionally, it produces the highest R^2 value of 0.9.

As demonstrated by the performance metrics based on the cross-validation results and test-set results, Linear Regression produces desirably low error metrics (MAE, MSE, RMSE, etc.), meaning that its predictions are very close to the actual value in the dataset. Addressing its high R^2 value, it implies that the Linear Regression model closely fits the data. Furthermore, it implies that there is a strong relationship between the predictors and the target variable, which is Price.

Comparing the performance metrics based on the cross-validation results and test-set results, it may be observed that the performance metrics are relatively consistent across the board, meaning the difference between metrics are very minimal. Due to this consistency, it may be concluded that this Linear Regression model does not show signs of overfitting.

Based on the results produced from the base model testing, Linear Regression is the best model for this dataset as it produces desirable and consistent performance metrics using cross-validation and the test set. Exploring further, real estate companies can benefit from utilizing this Linear Regression model to predict property prices based on the predictors used. By being able to predict the prices of properties, real estate companies have a competitive advantage as they are able to gauge an understanding of how they should price their properties based on the predictors. Furthermore, real estate companies are able to predict market trends in the future, gaining a competitive advantage over other real estate companies.

Using an optimized and tuned version of a Linear Regression model will not only allow them to predict property prices in the future, but also identify the significant variables that affect property prices through feature importance. This gives real estate companies a better understanding of what factors significantly affect property prices. If some factor is more important than others, real estate companies can focus more on those factors over others.



The barchart above plots the top 10 features from Lasso Regression (an optimized version of the Linear Regression model) that significantly affect a property's predicted price. It may be observed that the most significant feature is "Avg. Area Income", which has an absolute coefficient value of 1.314925e+06. This implies that the average income of an area has a large influence on a property's price. Real estate companies can use this information to their advantage by focusing on different marketing strategies that cater toward different areas with different income levels. For example, higher priced properties may be marketed to higher-income areas, while lower and more affordable properties may be marketed to lower-income areas.

The next significant variable is "Population per Room", which has an absolute coefficient value of 1.217127e+06. This implies that the density of an area significantly influences a property's price. Real estate companies could use this information to adjust their pricing strategies. For example, in low-density areas, real estate companies may impose premium prices for these properties, as it will appeal to buyers who value space. In terms of high-density areas, real-estate companies may offer more affordable prices.

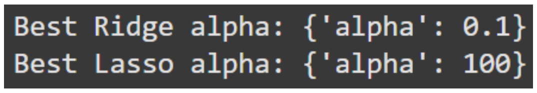
Following that, the next significant variable is "Avg. Area Number of Rooms", which has an absolute coefficient value of 1.162943e+06. This implies that the number of rooms heavily influence the price of a property (eg. a property with more rooms is priced higher). Real-estate companies can benefit from this information, as they may want to optimize their renovations in the future. For example, during renovation, they could split a large room into two smaller sized rooms, which in turn would lead to an increase in the property's price. Furthermore, real-estate companies may want to develop new properties based on market demand. If an area has a higher average number of rooms, they should develop new properties that cater to this demand for more rooms.

Finally, the next most significant variable is "Avg. Area House Age", which has an absolute coefficient value of 9.767884e+05. This implies that the age of a property strongly influences its price. The age of a property is a reliable indicator of the overall condition of the property. In general, if a property is relatively old, it indicates that it may require more maintenance in the future. Furthermore, the overall age of the house may strongly influence the buyer's decision, as an older house may be more affordable, but will require more maintenance than a newer house. Real-estate companies should use this information to adjust their marketing techniques based on the age of the house. For example, they may need to emphasize any recent maintenance/renovation done on older houses. In terms of newer houses, they should emphasize how they are low maintenance, meaning they do not require frequent maintenance in the future.

In conclusion, real-estate companies should use Linear Regression models to gain a competitive advantage as they are able to predict future prices of properties and are able to identify what aspects they should allocate their resources to.

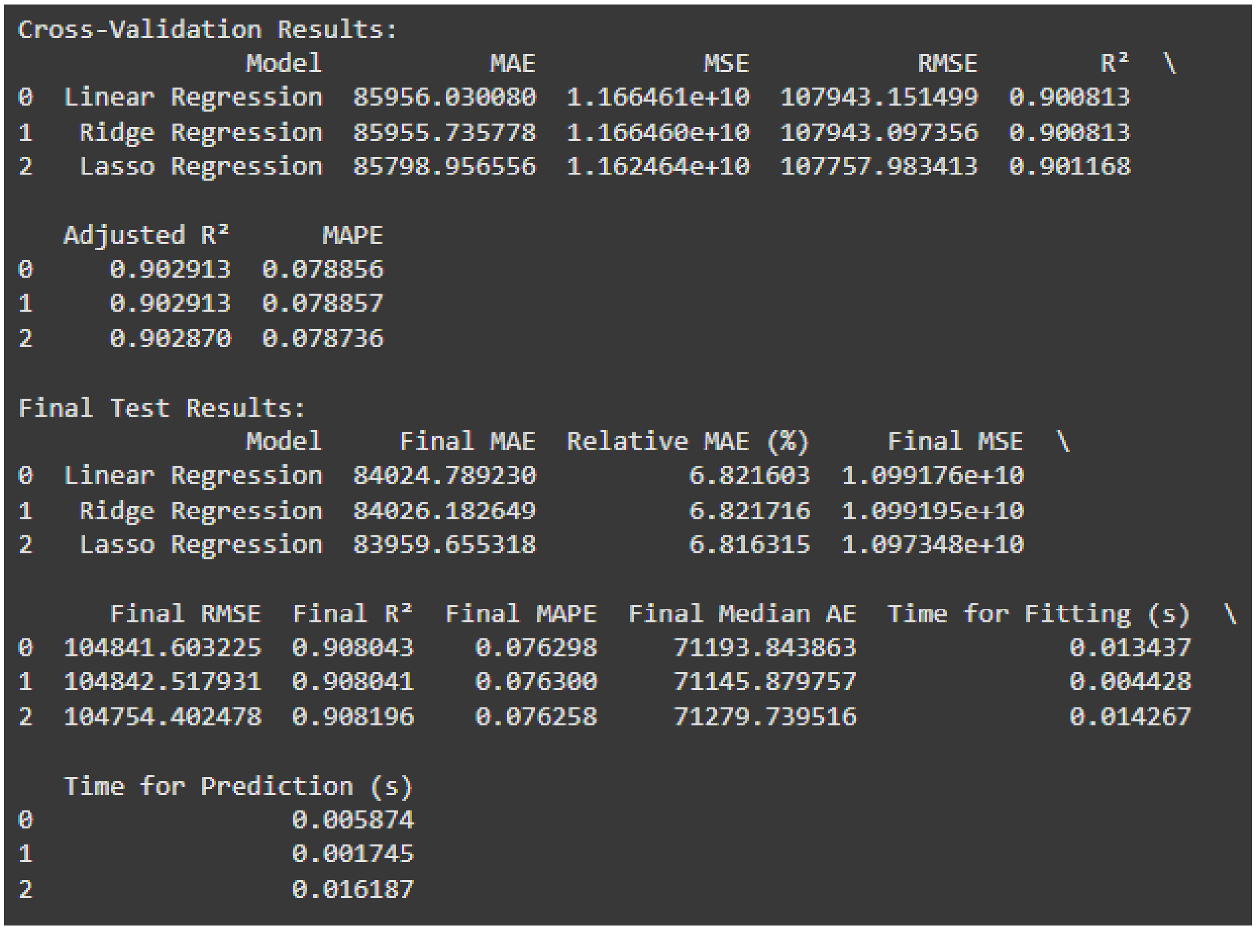
Task 2 (iv)

After Linear Regression has been selected as the best performing model, the model must now be optimized through hyperparameter tuning. However, Linear Regression does not have any parameters that may be tuned, which is why the Ridge and Lasso regression will be used, as they are both regularization techniques for Linear Regression.



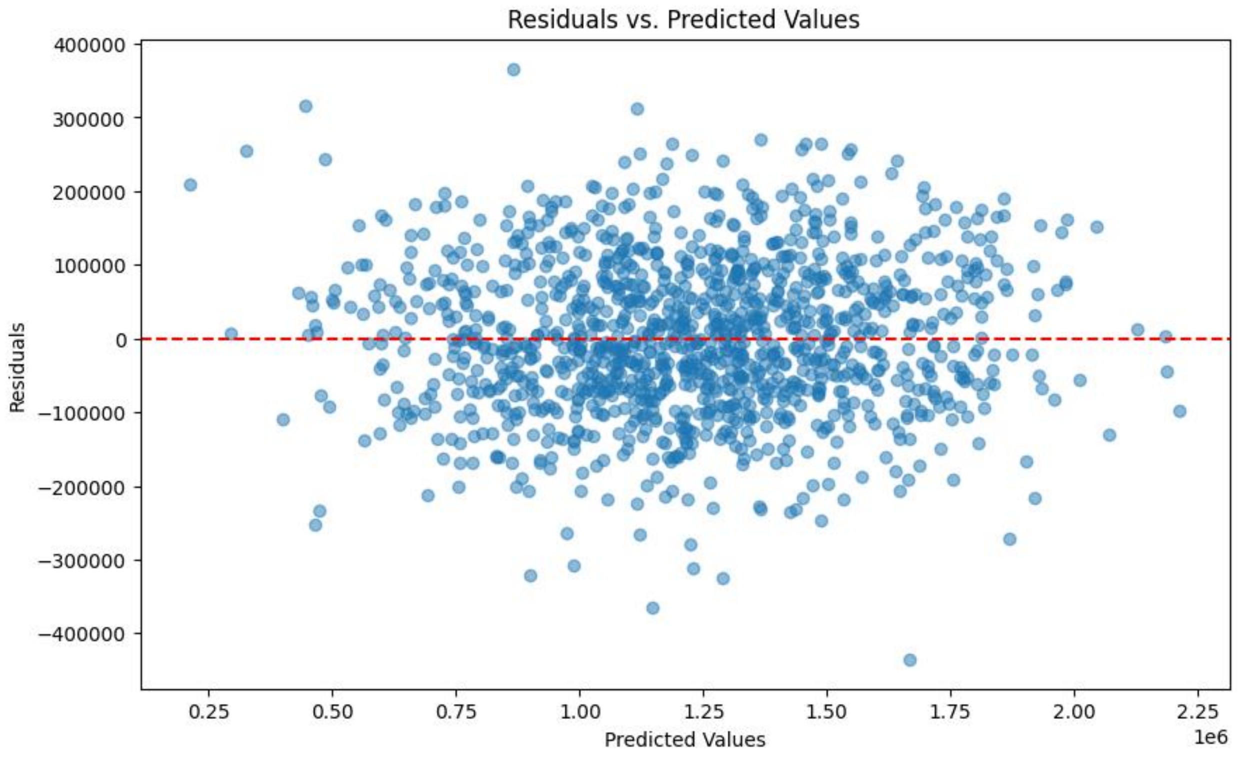
```
Best Ridge alpha: {'alpha': 0.1}
Best Lasso alpha: {'alpha': 100}
```

The initial step of hyperparameter tuning is to find the best parameters for both Ridge and Lasso, which was done using GridSearchCV. As seen above, the best alpha value for Ridge is 0.1 and the best alpha value for Lasso is 100. Now that the parameters have been found, the Ridge and Lasso models can go through cross validation and test evaluation using the alpha values.

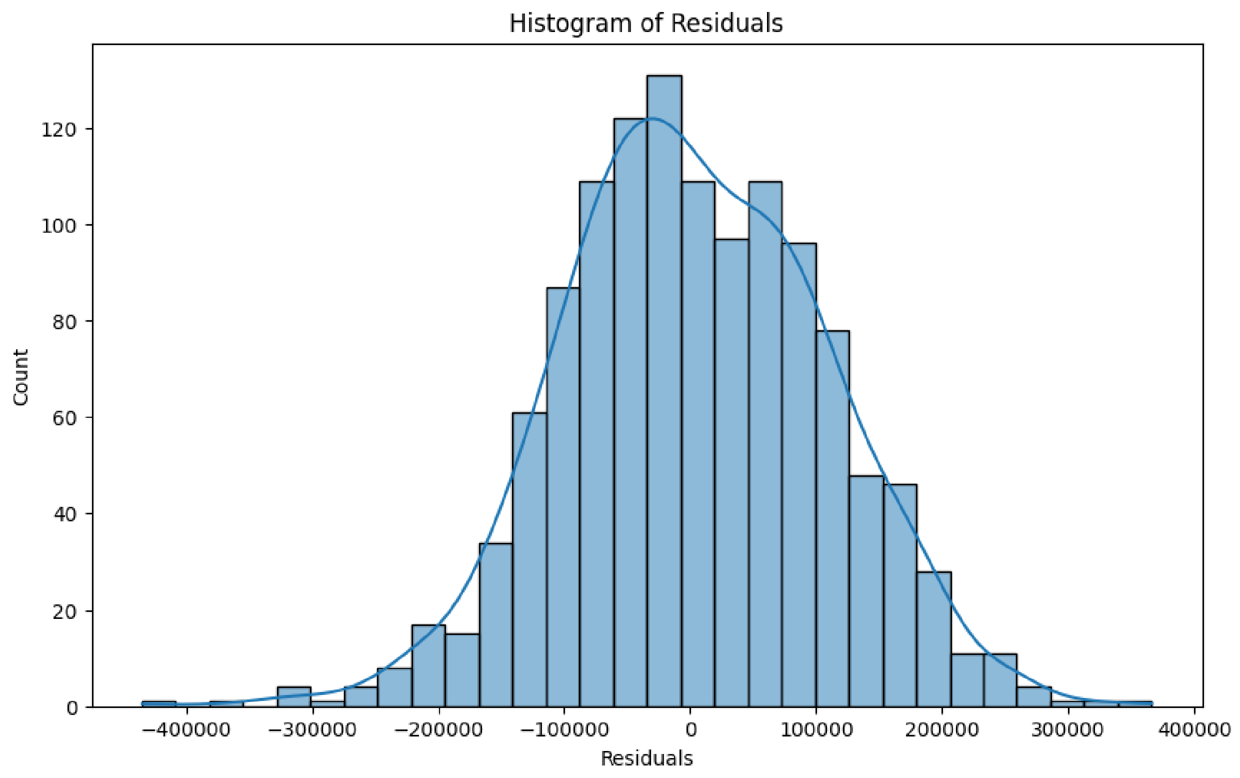


```
Cross-Validation Results:
               Model          MAE           MSE           RMSE        R²  \
0  Linear Regression  85956.030080  1.166461e+10  107943.151499  0.900813
1   Ridge Regression  85955.735778  1.166460e+10  107943.097356  0.900813
2   Lasso Regression  85798.956556  1.162464e+10  107757.983413  0.901168

   Adjusted R²      MAPE
0     0.902913  0.078856
1     0.902913  0.078857
2     0.902870  0.078736

Final Test Results:
               Model     Final MAE  Relative MAE (%)     Final MSE  \
0  Linear Regression  84024.789230          6.821603  1.099176e+10
1   Ridge Regression  84026.182649          6.821716  1.099195e+10
2   Lasso Regression  83959.655318          6.816315  1.097348e+10

      Final RMSE  Final R²  Final MAPE  Final Median AE  Time for Fitting (s)  \
0  104841.603225  0.908043    0.076298     71193.843863              0.013437
1  104842.517931  0.908041    0.076300     71145.879757              0.004428
2  104754.402478  0.908196    0.076258     71279.739516              0.014267

   Time for Prediction (s)
0                 0.005874
1                 0.001745
2                 0.016187
```

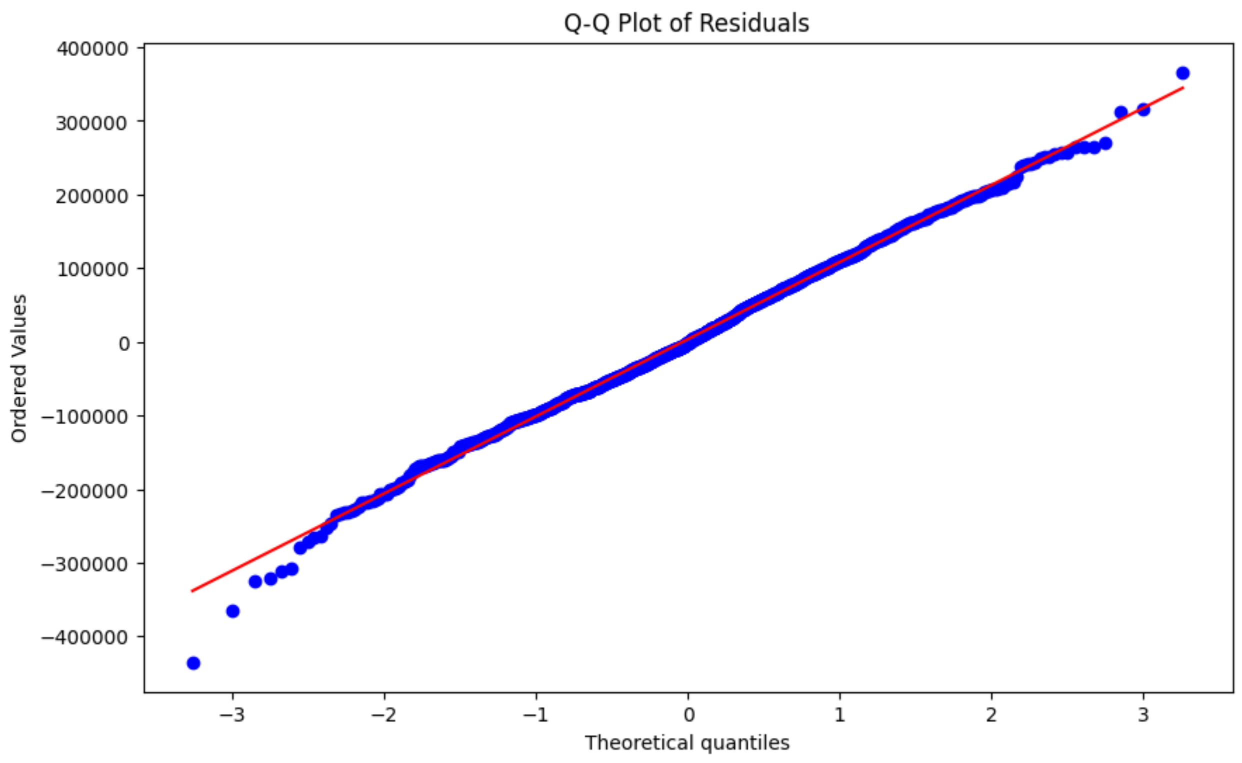
The results above display the performance metrics of Linear Regression, Ridge Regression, and Lasso Regression models, displaying both the cross-validation results and test-set results. Looking at the cross-validation results, it may be noticed that Lasso Regression overall has lower error metrics than both Linear Regression and Ridge Regression. Furthermore, it produces the highest R^2 value compared to the other models. Focusing on the test-set results, a similar outcome may be observed. Lasso Regression continues to produce the lowest error metrics compared to Linear and Ridge Regression, along with producing the highest R^2 value. By comparing the results of all three models between both cross-validation and test-set results, it may be concluded that Lasso Regression, with an alpha value of 100, is the best performing model. Furthermore, it does not show any signs of overfitting due to its consistent results between cross-validation and test-set results.



The Lasso Regression model can further be analyzed by inspecting its residuals. The scatterplot above illustrates the residuals against the predicted values. Based on this scatterplot, it may be seen that the points are scattered relatively randomly, indicating that there are no inherent patterns. The lack of patterns is desirable as the constant and consistent spread of the points indicate homoscedasticity, meaning the variance of the residuals is constant.



The histogram above illustrates the distribution of the residuals. It may be observed that the distribution of the residuals is relatively normal, as it portrays a bell-shaped curve. This normal distribution implies that there is no systematic bias in terms of how the errors are distributed.



The Q-Q plot above compares the quantiles of the residuals produced by the Lasso Regression model to the theoretical quantiles of a normal distribution. It may be observed that the points fall along the diagonal line with minimal deviation. This implies that the distribution of the residuals is relatively normal.

In conclusion, the performance metrics of Lasso Regression and the visualizations of its residuals all justify why it is the best performing model for predicting property prices based on the provided predictors.

References

Gu, J., Zhu, M., & Jiang, L. (2011). Housing price forecasting based on genetic algorithm and support vector machine. Expert Systems with Applications, 38(4), 3383–3386. https://doi.org/10.1016/j.eswa.2010.08.123

Sharma, H., Harsora, H., & Ogunleye, B. (2024). An Optimal House Price Prediction Algorithm: XGBoost. Analytics, 3(1), 30-45. https://doi.org/10.3390/analytics3010003

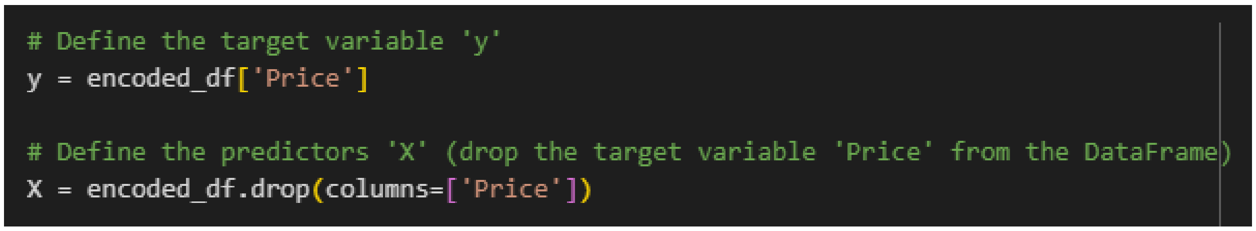
Truong, Q., Nguyen, M., Dang, H., & Mei, B. (2020). Housing price prediction via improved machine learning techniques. *Procedia Computer Science, 174*, 433–442. https://doi.org/10.1016/j.procs.2020.06.111

Zhao, S. (2018). Implementation and Study of K-Nearest Neighbour and Regression Algorithm for Real-time Housing Market Recommendation Application.

Zhang, Qingqi, Housing Price Prediction Based on Multiple Linear Regression, Scientific Programming, 2021, 7678931, 9 pages, 2021. https://doi.org/10.1155/2021/7678931

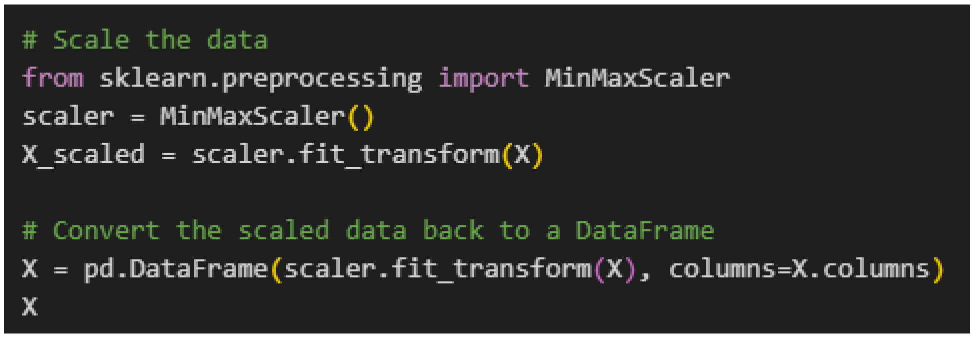
Z. Zhang, "Decision Trees for Objective House Price Prediction," 2021 3rd International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI), Taiyuan, China, 2021, pp. 280-283, doi: 10.1109/MLBDBI54094.2021.00059.

Appendix



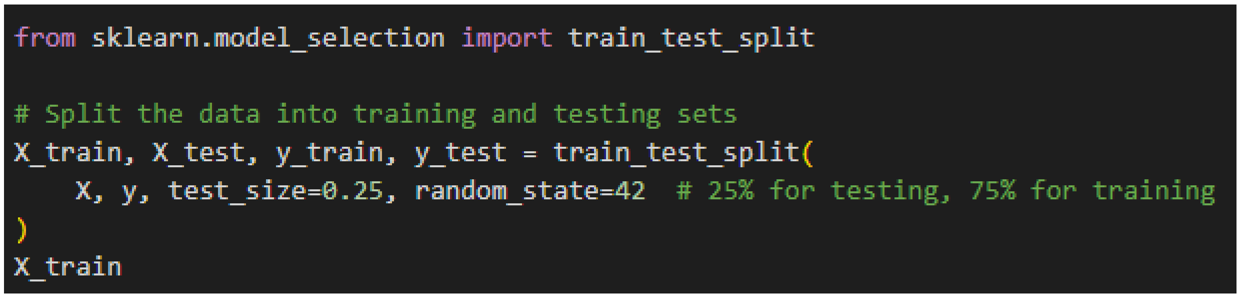
```python
# Define the target variable 'y'
y = encoded_df['Price']

# Define the predictors 'X' (drop the target variable 'Price' from the DataFrame)
X = encoded_df.drop(columns=['Price'])
```

Figure A1 – Defining the Target Variable and Predictors



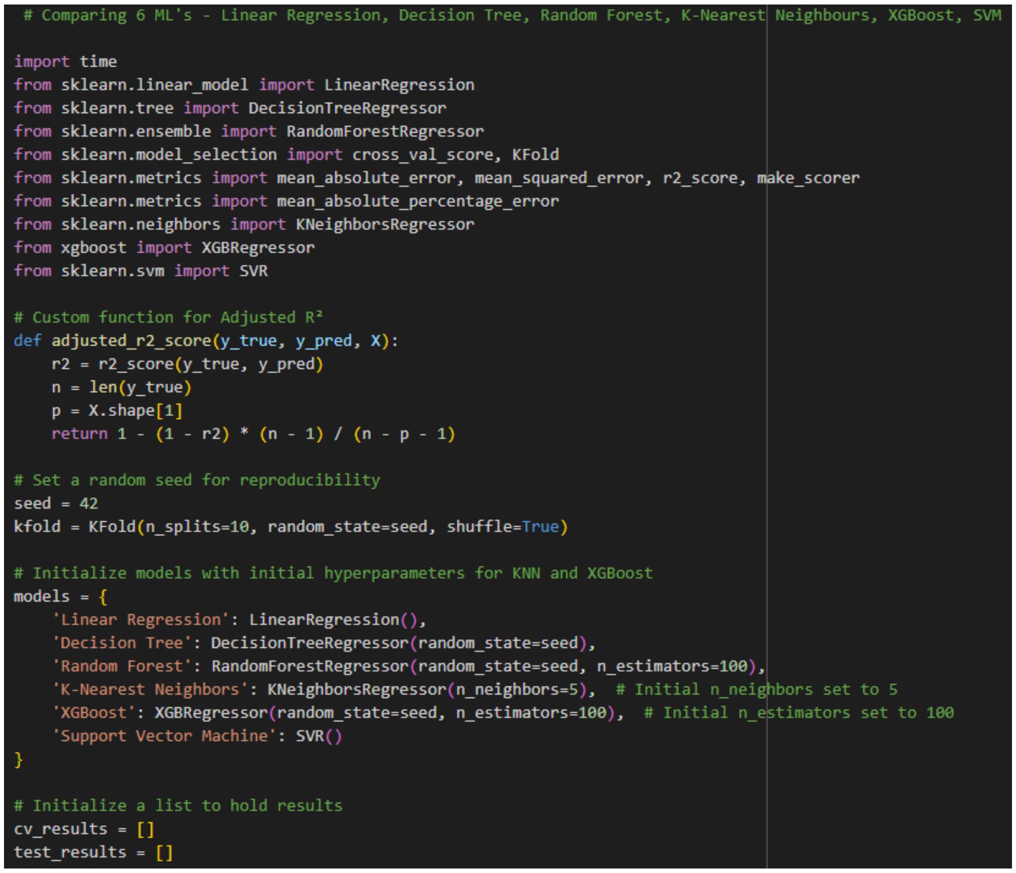
```python
# Scale the data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Convert the scaled data back to a DataFrame
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
X
```

Figure A2 – Scaling the predictors



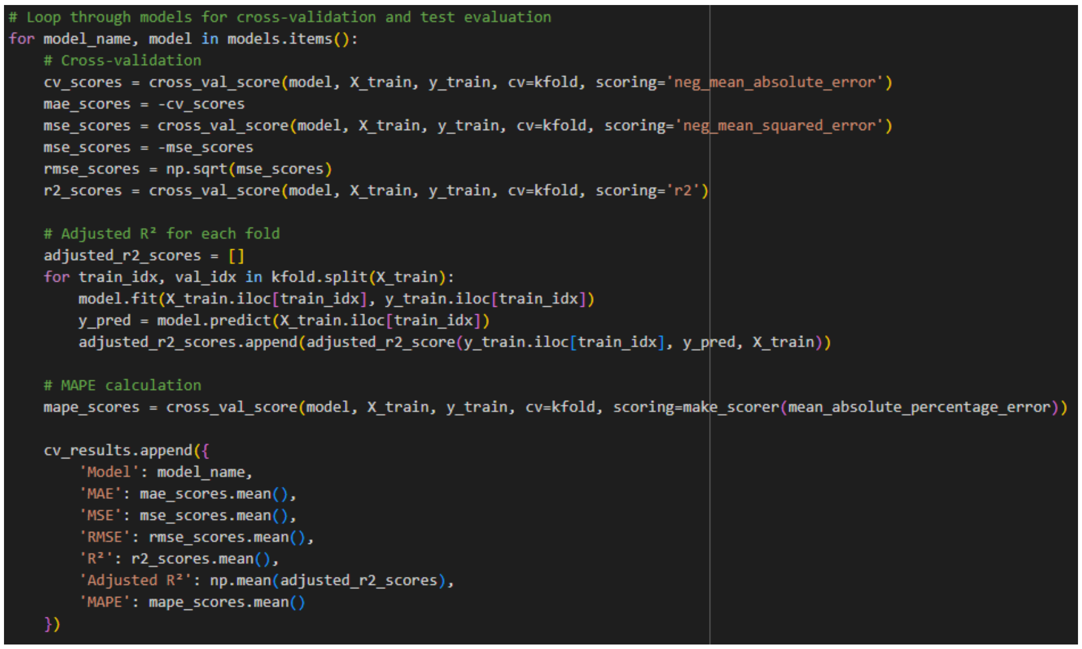
```python
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42  # 25% for testing, 75% for training
)
X_train
```

Figure A3 – Splitting the data into training and testing sets



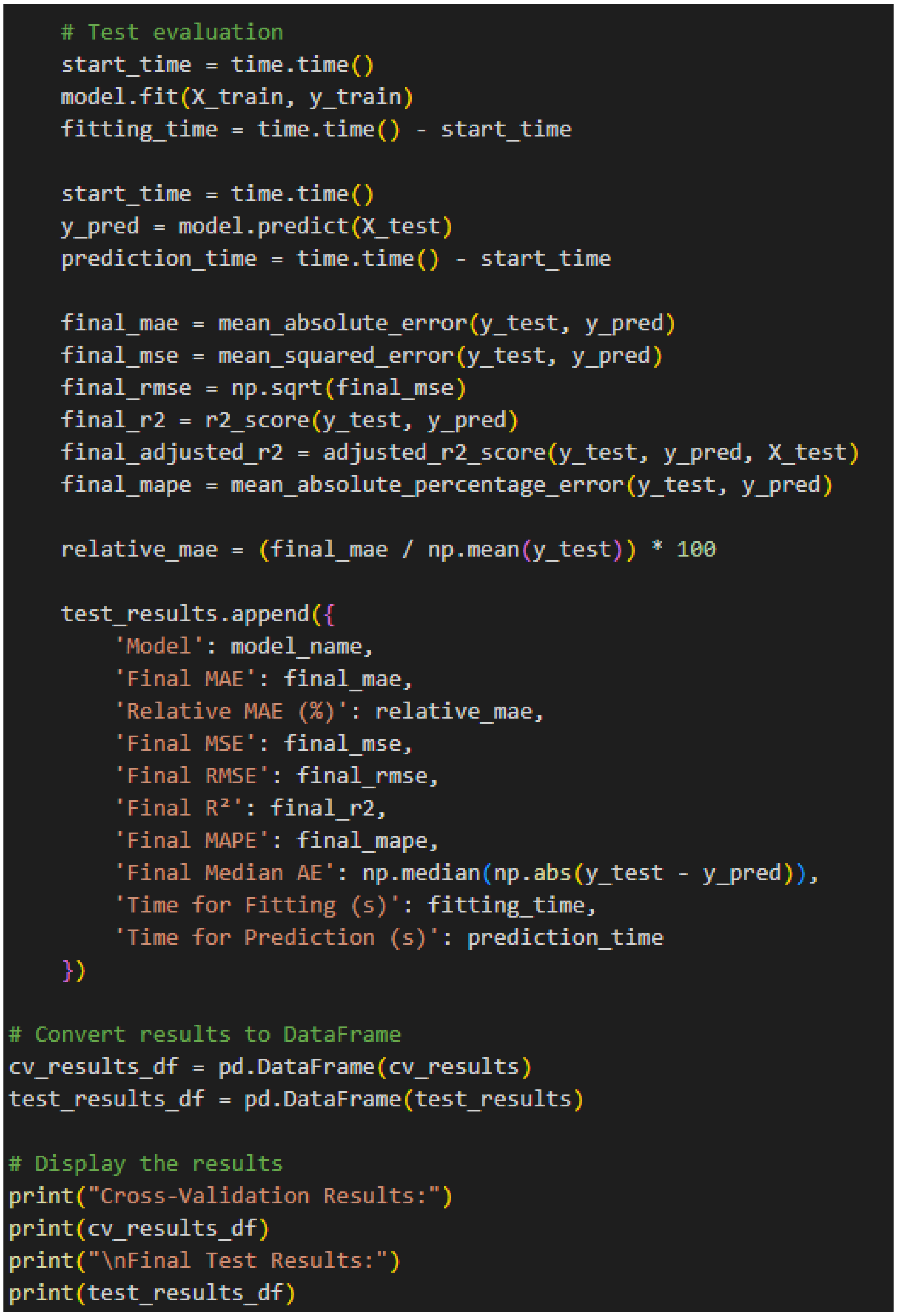
```
 # Comparing 6 ML's - Linear Regression, Decision Tree, Random Forest, K-Nearest Neighbours, XGBoost, SVM

import time
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score, make_scorer
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor
from sklearn.svm import SVR

# Custom function for Adjusted R²
def adjusted_r2_score(y_true, y_pred, X):
    r2 = r2_score(y_true, y_pred)
    n = len(y_true)
    p = X.shape[1]
    return 1 - (1 - r2) * (n - 1) / (n - p - 1)

# Set a random seed for reproducibility
seed = 42
kfold = KFold(n_splits=10, random_state=seed, shuffle=True)

# Initialize models with initial hyperparameters for KNN and XGBoost
models = {
    'Linear Regression': LinearRegression(),
    'Decision Tree': DecisionTreeRegressor(random_state=seed),
    'Random Forest': RandomForestRegressor(random_state=seed, n_estimators=100),
    'K-Nearest Neighbors': KNeighborsRegressor(n_neighbors=5),  # Initial n_neighbors set to 5
    'XGBoost': XGBRegressor(random_state=seed, n_estimators=100),  # Initial n_estimators set to 100
    'Support Vector Machine': SVR()
}

# Initialize a list to hold results
cv_results = []
test_results = []
```

Figure A4 – Base Model Testing (part 1)



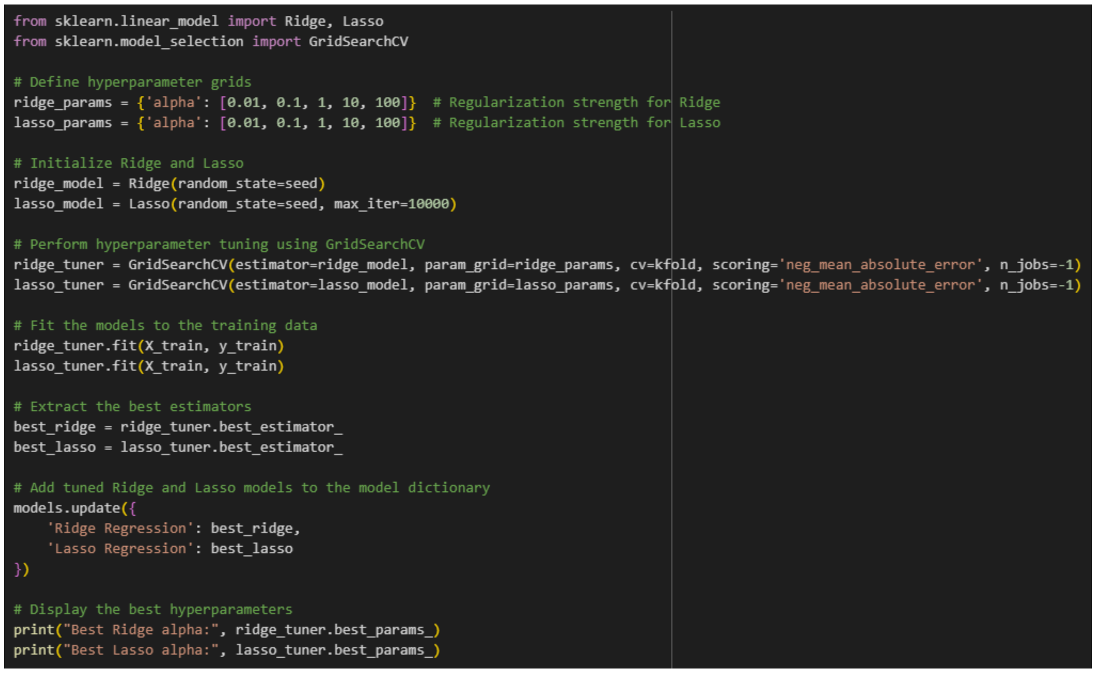
```
# Loop through models for cross-validation and test evaluation
for model_name, model in models.items():
    # Cross-validation
    cv_scores = cross_val_score(model, X_train, y_train, cv=kfold, scoring='neg_mean_absolute_error')
    mae_scores = -cv_scores
    mse_scores = cross_val_score(model, X_train, y_train, cv=kfold, scoring='neg_mean_squared_error')
    mse_scores = -mse_scores
    rmse_scores = np.sqrt(mse_scores)
    r2_scores = cross_val_score(model, X_train, y_train, cv=kfold, scoring='r2')

    # Adjusted R² for each fold
    adjusted_r2_scores = []
    for train_idx, val_idx in kfold.split(X_train):
        model.fit(X_train.iloc[train_idx], y_train.iloc[train_idx])
        y_pred = model.predict(X_train.iloc[train_idx])
        adjusted_r2_scores.append(adjusted_r2_score(y_train.iloc[train_idx], y_pred, X_train))

    # MAPE calculation
    mape_scores = cross_val_score(model, X_train, y_train, cv=kfold, scoring=make_scorer(mean_absolute_percentage_error))

    cv_results.append({
        'Model': model_name,
        'MAE': mae_scores.mean(),
        'MSE': mse_scores.mean(),
        'RMSE': rmse_scores.mean(),
        'R²': r2_scores.mean(),
        'Adjusted R²': np.mean(adjusted_r2_scores),
        'MAPE': mape_scores.mean()
    })
```

Figure A5 – Base Model Testing (Part 2)



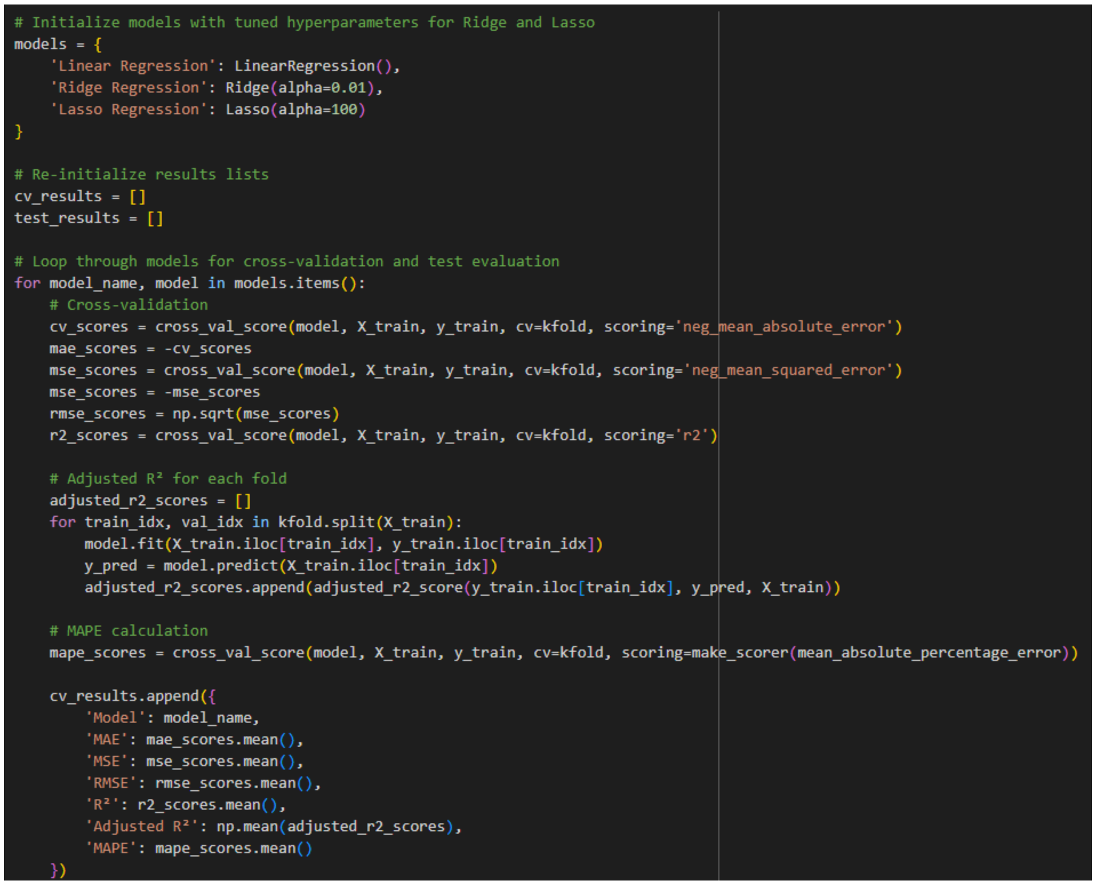
```
    # Test evaluation
    start_time = time.time()
    model.fit(X_train, y_train)
    fitting_time = time.time() - start_time

    start_time = time.time()
    y_pred = model.predict(X_test)
    prediction_time = time.time() - start_time

    final_mae = mean_absolute_error(y_test, y_pred)
    final_mse = mean_squared_error(y_test, y_pred)
    final_rmse = np.sqrt(final_mse)
    final_r2 = r2_score(y_test, y_pred)
    final_adjusted_r2 = adjusted_r2_score(y_test, y_pred, X_test)
    final_mape = mean_absolute_percentage_error(y_test, y_pred)

    relative_mae = (final_mae / np.mean(y_test)) * 100

    test_results.append({
        'Model': model_name,
        'Final MAE': final_mae,
        'Relative MAE (%)': relative_mae,
        'Final MSE': final_mse,
        'Final RMSE': final_rmse,
        'Final R²': final_r2,
        'Final MAPE': final_mape,
        'Final Median AE': np.median(np.abs(y_test - y_pred)),
        'Time for Fitting (s)': fitting_time,
        'Time for Prediction (s)': prediction_time
    })

# Convert results to DataFrame
cv_results_df = pd.DataFrame(cv_results)
test_results_df = pd.DataFrame(test_results)

# Display the results
print("Cross-Validation Results:")
print(cv_results_df)
print("\nFinal Test Results:")
print(test_results_df)
```

Figure A6 – Base Model Testing (Part 3)



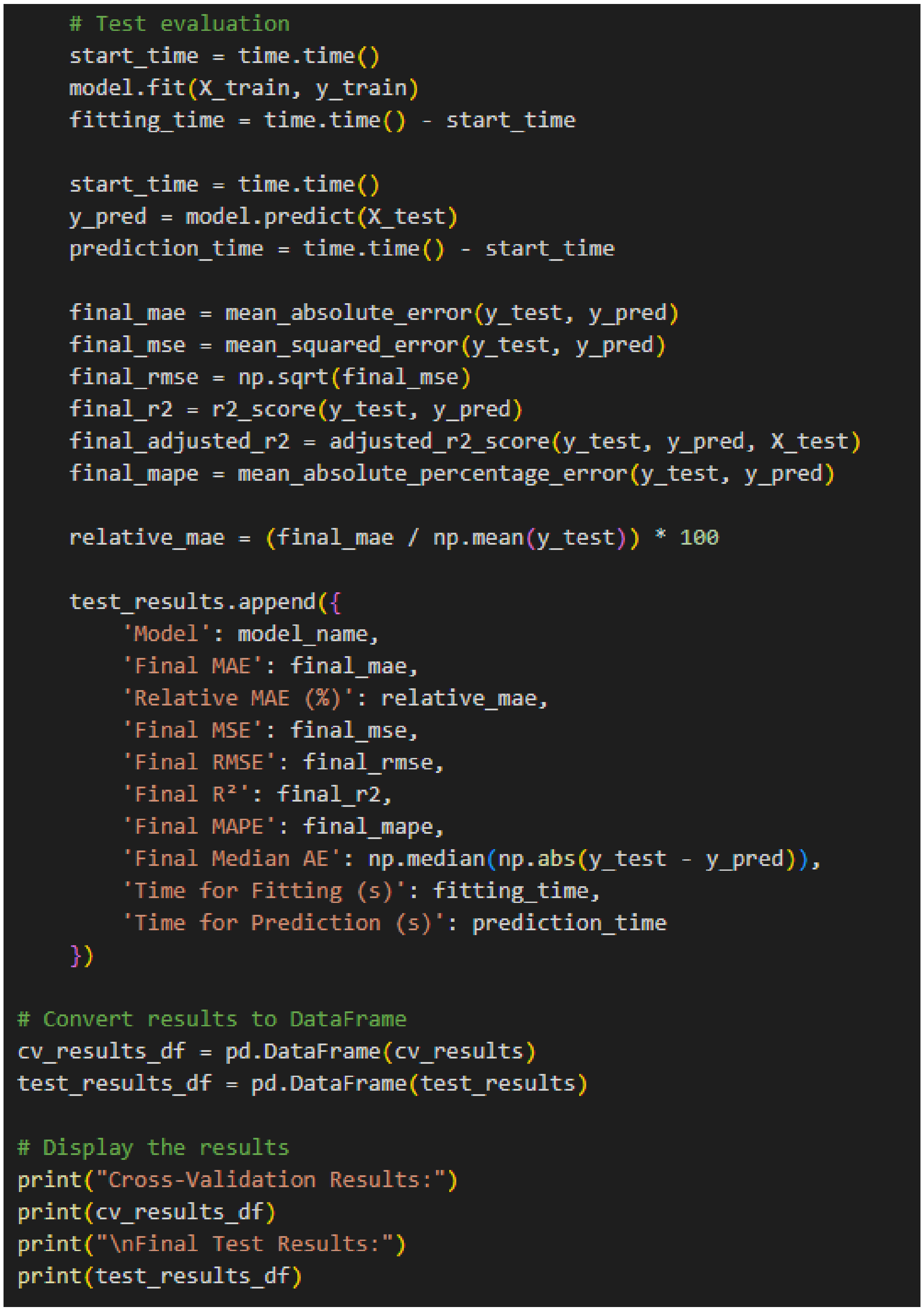
```python
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import GridSearchCV

# Define hyperparameter grids
ridge_params = {'alpha': [0.01, 0.1, 1, 10, 100]}  # Regularization strength for Ridge
lasso_params = {'alpha': [0.01, 0.1, 1, 10, 100]}  # Regularization strength for Lasso

# Initialize Ridge and Lasso
ridge_model = Ridge(random_state=seed)
lasso_model = Lasso(random_state=seed, max_iter=10000)

# Perform hyperparameter tuning using GridSearchCV
ridge_tuner = GridSearchCV(estimator=ridge_model, param_grid=ridge_params, cv=kfold, scoring='neg_mean_absolute_error', n_jobs=-1)
lasso_tuner = GridSearchCV(estimator=lasso_model, param_grid=lasso_params, cv=kfold, scoring='neg_mean_absolute_error', n_jobs=-1)

# Fit the models to the training data
ridge_tuner.fit(X_train, y_train)
lasso_tuner.fit(X_train, y_train)

# Extract the best estimators
best_ridge = ridge_tuner.best_estimator_
best_lasso = lasso_tuner.best_estimator_

# Add tuned Ridge and Lasso models to the model dictionary
models.update({
    'Ridge Regression': best_ridge,
    'Lasso Regression': best_lasso
})

# Display the best hyperparameters
print("Best Ridge alpha:", ridge_tuner.best_params_)
print("Best Lasso alpha:", lasso_tuner.best_params_)
```

Figure A7 – Hyperparameter Tuning for Ridge and Lasso Regression



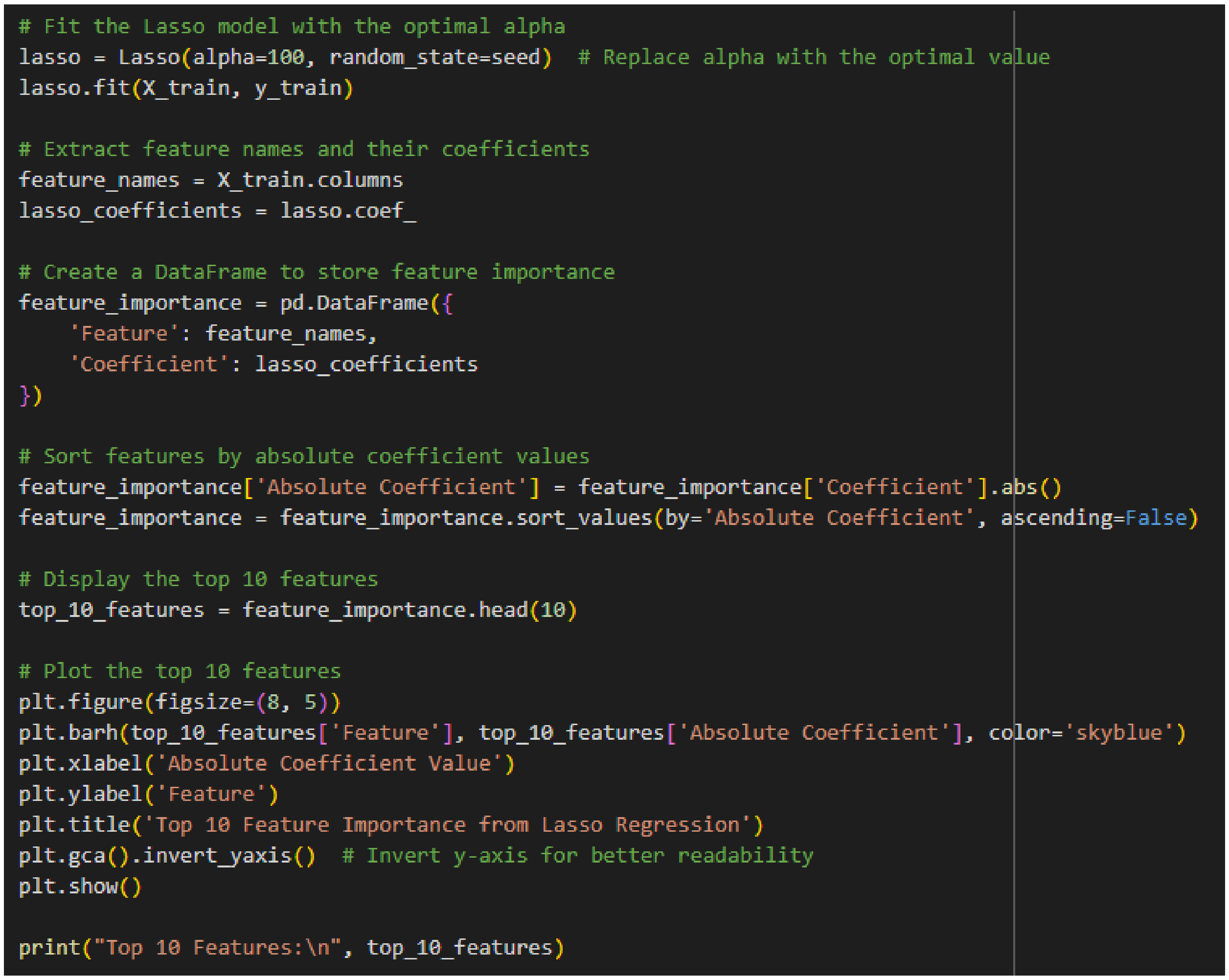
```python
# Initialize models with tuned hyperparameters for Ridge and Lasso
models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(alpha=0.01),
    'Lasso Regression': Lasso(alpha=100)
}

# Re-initialize results lists
cv_results = []
test_results = []

# Loop through models for cross-validation and test evaluation
for model_name, model in models.items():
    # Cross-validation
    cv_scores = cross_val_score(model, X_train, y_train, cv=kfold, scoring='neg_mean_absolute_error')
    mae_scores = -cv_scores
    mse_scores = cross_val_score(model, X_train, y_train, cv=kfold, scoring='neg_mean_squared_error')
    mse_scores = -mse_scores
    rmse_scores = np.sqrt(mse_scores)
    r2_scores = cross_val_score(model, X_train, y_train, cv=kfold, scoring='r2')

    # Adjusted R² for each fold
    adjusted_r2_scores = []
    for train_idx, val_idx in kfold.split(X_train):
        model.fit(X_train.iloc[train_idx], y_train.iloc[train_idx])
        y_pred = model.predict(X_train.iloc[train_idx])
        adjusted_r2_scores.append(adjusted_r2_score(y_train.iloc[train_idx], y_pred, X_train))

    # MAPE calculation
    mape_scores = cross_val_score(model, X_train, y_train, cv=kfold, scoring=make_scorer(mean_absolute_percentage_error))

    cv_results.append({
        'Model': model_name,
        'MAE': mae_scores.mean(),
        'MSE': mse_scores.mean(),
        'RMSE': rmse_scores.mean(),
        'R²': r2_scores.mean(),
        'Adjusted R²': np.mean(adjusted_r2_scores),
        'MAPE': mape_scores.mean()
    })
```

Figure A8 – Testing the tuned models (part 1)



```
    # Test evaluation
    start_time = time.time()
    model.fit(X_train, y_train)
    fitting_time = time.time() - start_time

    start_time = time.time()
    y_pred = model.predict(X_test)
    prediction_time = time.time() - start_time

    final_mae = mean_absolute_error(y_test, y_pred)
    final_mse = mean_squared_error(y_test, y_pred)
    final_rmse = np.sqrt(final_mse)
    final_r2 = r2_score(y_test, y_pred)
    final_adjusted_r2 = adjusted_r2_score(y_test, y_pred, X_test)
    final_mape = mean_absolute_percentage_error(y_test, y_pred)

    relative_mae = (final_mae / np.mean(y_test)) * 100

    test_results.append({
        'Model': model_name,
        'Final MAE': final_mae,
        'Relative MAE (%)': relative_mae,
        'Final MSE': final_mse,
        'Final RMSE': final_rmse,
        'Final R²': final_r2,
        'Final MAPE': final_mape,
        'Final Median AE': np.median(np.abs(y_test - y_pred)),
        'Time for Fitting (s)': fitting_time,
        'Time for Prediction (s)': prediction_time
    })

# Convert results to DataFrame
cv_results_df = pd.DataFrame(cv_results)
test_results_df = pd.DataFrame(test_results)

# Display the results
print("Cross-Validation Results:")
print(cv_results_df)
print("\nFinal Test Results:")
print(test_results_df)
```

Figure A9 – Testing the tuned models (part 2)



```python
# Fit the Lasso model with the optimal alpha
lasso = Lasso(alpha=100, random_state=seed)  # Replace alpha with the optimal value
lasso.fit(X_train, y_train)

# Extract feature names and their coefficients
feature_names = X_train.columns
lasso_coefficients = lasso.coef_

# Create a DataFrame to store feature importance
feature_importance = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': lasso_coefficients
})

# Sort features by absolute coefficient values
feature_importance['Absolute Coefficient'] = feature_importance['Coefficient'].abs()
feature_importance = feature_importance.sort_values(by='Absolute Coefficient', ascending=False)

# Display the top 10 features
top_10_features = feature_importance.head(10)

# Plot the top 10 features
plt.figure(figsize=(8, 5))
plt.barh(top_10_features['Feature'], top_10_features['Absolute Coefficient'], color='skyblue')
plt.xlabel('Absolute Coefficient Value')
plt.ylabel('Feature')
plt.title('Top 10 Feature Importance from Lasso Regression')
plt.gca().invert_yaxis()  # Invert y-axis for better readability
plt.show()

print("Top 10 Features:\n", top_10_features)
```

Figure A10 – Feature Importance using Lasso Regression