Analysis of London Housing Market

Name	**Tonny K Podiyan**
Student ID	**21052135**
Email ID	**Tok0165@my.londonmet.ac.uk**
Module Code	**CC7184**
Module Title	**Data Mining & Machine Learning**
Date	**29th April 2023**

Analysis of London Housing Market

Tonny Kattitharayil Podiyan

Abstract: This report presents an analysis of the London housing market based on data from 2002 to 2019. The aim of the project was to identify trends and patterns in the market and to develop models that can be used to predict future prices. The main findings suggest that the London housing market has been experiencing a steady increase in prices over the last couple of years, with certain areas showing more significant growth than others. The report also highlights the importance of factors such as population , earnings, and other amenities in determining the price of a property.

Key Words: London, Housing Prices, Regression , Decision Tree, Predictive Analysis

1. INTRODUCTION

London's housing market is a challenging problem due to the high demand for housing and limited supply. Several studies have been conducted to identify the factors that contribute to the soaring prices of houses in London. One approach has been to analyse large datasets of property transactions to identify the factors that contribute to the high prices. For instance, a study utilised machine learning algorithms to analyse property data from the Land Registry and identified various factors that influence house prices in London, including location, property size, and age [1].Another technique that has been used is sentiment analysis, which involves analysing social media data to gain insights into people's perceptions of the housing market. For example, another study analysed Twitter data to determine people's attitudes towards housing in London and identified that affordability was a significant concern among social media users [2]. Furthermore, natural language processing techniques have been used to analyse text data from news articles and other sources to identify emerging trends and sentiments around the housing market. For instance, another study by Li and Li (2020) utilised topic modelling to analyse news articles related to London's housing market and identified emerging trends, such as the rise of co-living spaces and the impact of Brexit on the housing market. [3]

2. OBJECTIVES

London housing prices are a complex and dynamic phenomenon that can benefit from both supervised and unsupervised machine learning methods. Objectives of this report is to:

1. Explore the relationship between London house prices and other variables in the dataset.
2. Identify variables that have the strongest correlation with London house prices and quantify their impact.
3. Build a regression model to predict London house prices based on the other variables in the dataset and evaluate the accuracy of the model.
4. Identify and handle any outliers or influential data points that may affect the accuracy of the regression analysis.
5. Use data visualisation techniques to effectively communicate findings and identify key features that influence housing prices.
6. To identify clusters of similar properties based on their characteristics and location.

3. METHODOLOGY

The data used for this analysis was obtained from data.london.gov.uk. We obtained multiple data files from this website, including data on people's earnings, job density, and number of houses/dwellings in each borough. We filtered the data to include only information between 2002 and 2019 and removed any duplicates, missing values, or outliers.

To analyse the data, we selected four models: three models for supervised learning i.e., multiple linear regression ,decision tree and random forest regression and one unsupervised model k-means clustering. We chose these models based on their ability to capture linear and non-linear relationships between the independent and dependent variables and to identify patterns or groups in the data. We evaluated the models using R-squared ,mean squared error and few other metrics for supervised models and silhouette score and Calinski-Harabasz Index for clustering models. To ensure the models' validity and prevent overfitting, we used cross-validation during model validation.

3.1 Dataset Description

To conduct our analysis, we used five datasets, each serving as an attribute for our final dataset. These datasets included UK housing prices, earnings, density of dwellings, job density, and population. We selected specific sheets from each dataset, such as Average Price, Total Weekly, Dwellings per hectare, Job Density, and Custom Age range tool, and transformed and extracted relevant data points for analysis. The data ranged from 1995 to 2022 for UK housing prices, 2002 to 2021 for earnings, 2001 to 2019 for density of dwellings, 2000 to 2021 for job density, and 1999 to 2020 for population datasets respectively.

3.2 Dataset Pre-processing

Data pre-processing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviours or trends, and is likely to contain many errors. Data pre-processing is a proven method of resolving such issues [4]. In our analysis, we used Python on Google Colab Notebook to pre-process the dataset and the following libraries were used: Pandas, NumPy, Seaborn, Matplotlib, and Scikit-learn. We imported multiple data files, dropped unnecessary rows and columns, renamed columns, and created new columns where needed. The pre-processing steps included:

3.2.1 Data Import:

We imported the following data files using the read_excel() function:

1. UK House price index.xlsx
2. earnings-residence-borough.xlsx
3. Number_and_density_of_dwellings_by_borough.xlsx
4. jobs-and-job-density.xlsx
5. ons-mye-custom-age-tool-2020.xlsx

We specified the sheet names for each file using the sheet_name parameter in the read_excel() function.Each of these files contain valuable data for our research, such as house prices, earnings, number of dwellings and job densities in different boroughs.

3.2.2 Pre-processing of Dataset:

After importing the datasets, we dropped unnecessary rows and columns, renamed columns, and created new columns where needed. For example, in the UK House Price Index dataset, we dropped the first row, which contained irrelevant information, renamed the first column to "Time," and melted the dataset to make it more manageable. We also extracted the year from the "Time" column and dropped the "month" and "day" columns. In the Earnings Residence Borough dataset, we dropped the first three rows and selected columns from 2002 to 2021. We then melted the dataset to create a new dataframe that had columns for year, borough, and earnings. To make the earnings data consistent with the housing prices data, we multiplied the earnings by 48 to reflect 48 working weeks in a year.

We repeated similar cleaning and transformation processes for rest of the data sets. By pre-processing the datasets in this way, we were able to create clean and manageable dataframe that we could use for analysis. Now we created a dictionary to normalize the borough names as they were in different formats in different datasets. We then combined the processed data files into one file using the 'pd.merge' function. We merged the datasets based on the 'Year' and 'Borough' columns.

The merged dataframe, 'merged_df', provided us with a comprehensive dataset that included information on house prices, earnings, job density, density of dwellings and population for different boroughs of London for different years. We checked the information of the dataframe using the 'info' function and found that there were some missing values. We dropped the missing values using the 'dropna' function and reset the index of the dataframe using the 'reset_index' function. The resulting dataframe contained 576 rows and 7 columns, with each row representing a unique combination of 'Year' and 'Borough'.

3.3 Exploratory Data Analysis (EDA)

In this section, we present the results of our exploratory data analysis (EDA), where we used various data visualisation techniques to identify trends and patterns in the data.

3.3.1 London House Prices Distribution

We also used a line graph to display the trend in housing prices across the different boroughs in London from 1995 to 2021. Fig 1 shows that there has been a general increase in housing prices across all boroughs over time, with some boroughs experiencing steeper increases than others. Figure 2 presents a histogram illustrating the distribution of house prices in London. We found that the prices are not normally distributed and are skewed towards higher prices. Most houses are priced between £200,000 and £500,000, with a few houses priced above £1,000,000.

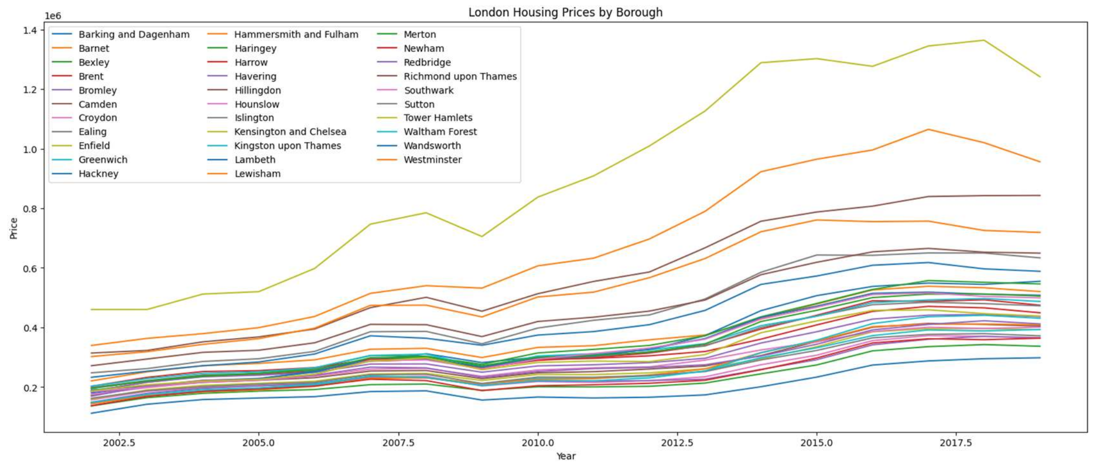


Figure 1: Line Graph showing upward trend of Housing Prices

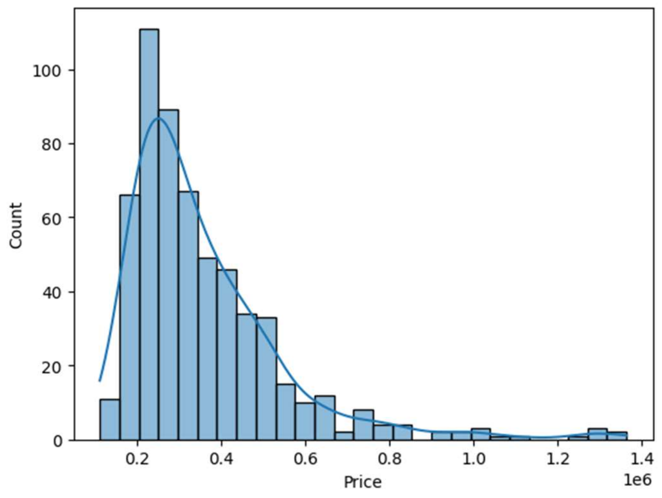


Figure 2: Histogram Depicting Housing Prices

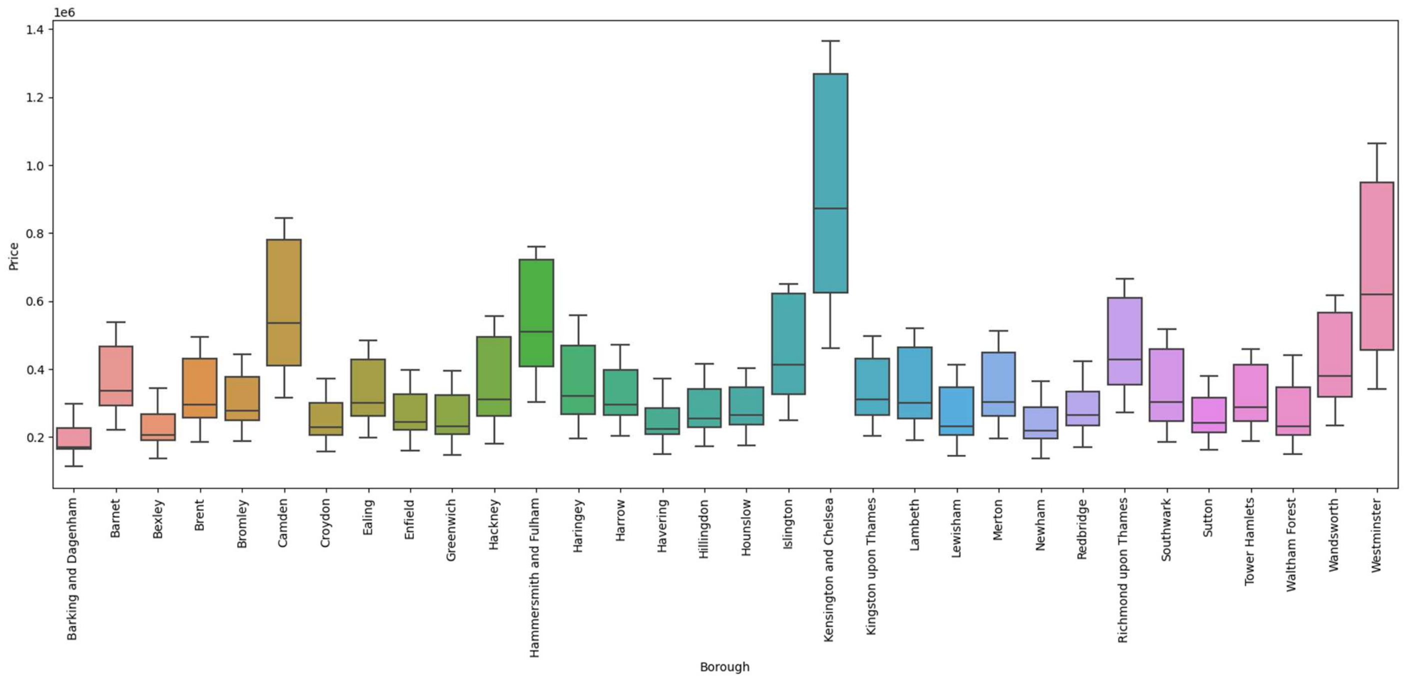


Figure 3: Box Plot Indicating Housing Prices of different London Boroughs

We utilized a boxplot to present the distribution of housing prices among various boroughs in London, depicted in Fig 3. Each box in the plot represents the spread of housing prices for a particular borough. Our analysis shows that there are substantial variations in housing prices across different boroughs, with certain boroughs having notably higher median housing prices than others. Kensington and Chelsea have the highest median price of 872940, while Barking and Dagenham has the lowest median price of 170793.

3.3.2 Correlation Analysis

The results of our analysis indicate that there is a strong positive correlation between house prices and earnings, density of dwellings, year, and job density. Among these variables, earnings have the strongest positive correlation with house prices, followed by density of dwellings, year, and job density. We also found that there is a weak negative correlation between house prices and population, indicating that people prefer to live in areas with lower house prices. We calculated the Pearson correlation coefficient between each variable and the house prices and visualised the results using a heatmap as shown in Fig 4

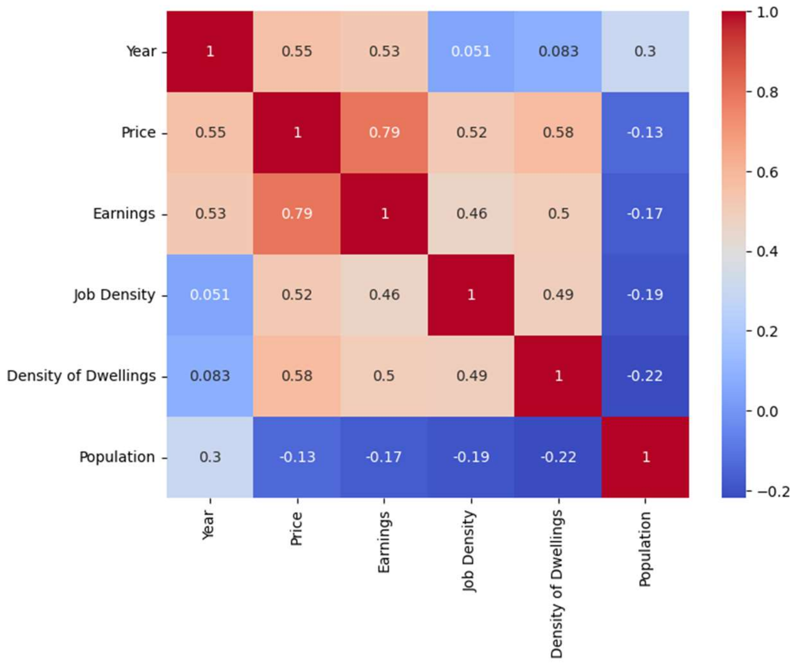


Figure 4: Correlation Matrix

3.3.3 Outlier Analysis

To identify any outliers or influential data points that may affect the analysis, we plotted pair plots between each independent variable and the dependent variable (house prices) as shown in Fig 5.From the pair plot, we can observe that there is a positive correlation between earnings and price, job density and price, and density of dwellings and price. However, the correlation between population and price appears to be weak. The regression lines in the scatterplots show an overall increasing trend between the predictor variables and price, indicating that the predictor variables are likely to have a positive impact on price. Overall, our EDA allowed us to gain insights into the distribution of London house prices, the correlations between various variables and house prices.

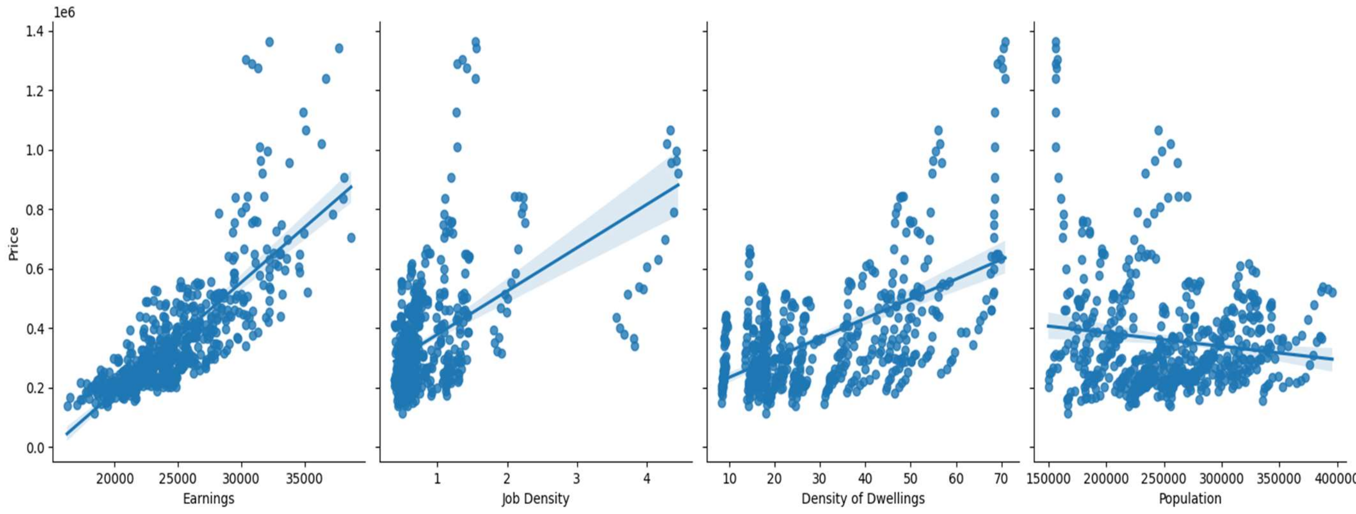


Figure 5: Pair plot of Price against predictor variables

3.4 Modelling Methods

3.4.1 Supervised Modelling

When it comes to machine learning, one popular approach is supervised modelling, which involves training an algorithm on labelled data to make predictions or decisions about new, unseen data [5]. In our analysis of London housing prices, we used three different supervised modelling techniques:

1. Multiple Linear Regression.
2. Decision tree.
3. Random Forest Regression.

1.Multiple Linear Regression

As previously mentioned, we have Price as our dependent variable and Earnings, Density Dwellings , Year, Job Density and Population as our independent variables. To handle categorical data, we converted borough names into dummy variables using the Pandas get_dummies() function. After splitting the data into training and testing sets, we trained a linear regression model on the training set using the scikit-learn LinearRegression() function [6].

The fitted model for the dataset is represented using the equation :

y = -5.48e+07 + 2.75e+04 * Year - 12.23 * Earnings + 4.10e+05 * Job Density + 1.71e+04 * Density of Dwellings - 2.89 * Population + (coefficients for each borough)

2. Decision Tree

After splitting our data into training and testing sets, we wanted to see if we could improve our model's performance by trying a different algorithm. So, we decided to use a Decision Tree Regression. To do this, we first created a DecisionTreeRegressor object in Python, which we imported from the scikit-learn library. We then trained the model on the training set using the fit() function, which builds a tree based on the data's features and labels. The algorithm used for decision tree regression builds a tree structure with each internal node representing a test on an attribute, each branch represents the outcome of the test, and each leaf node represents a prediction. The algorithm tries to find the best split at each node by minimizing the impurity of the child nodes. Once the tree has been built, it can be used to make predictions on new data [7].Finally, we used the predict() function to make predictions on the test set and stored the results in a variable called y_pred1. This allowed us to see how well the decision tree model performed in comparison to our original linear regression model.

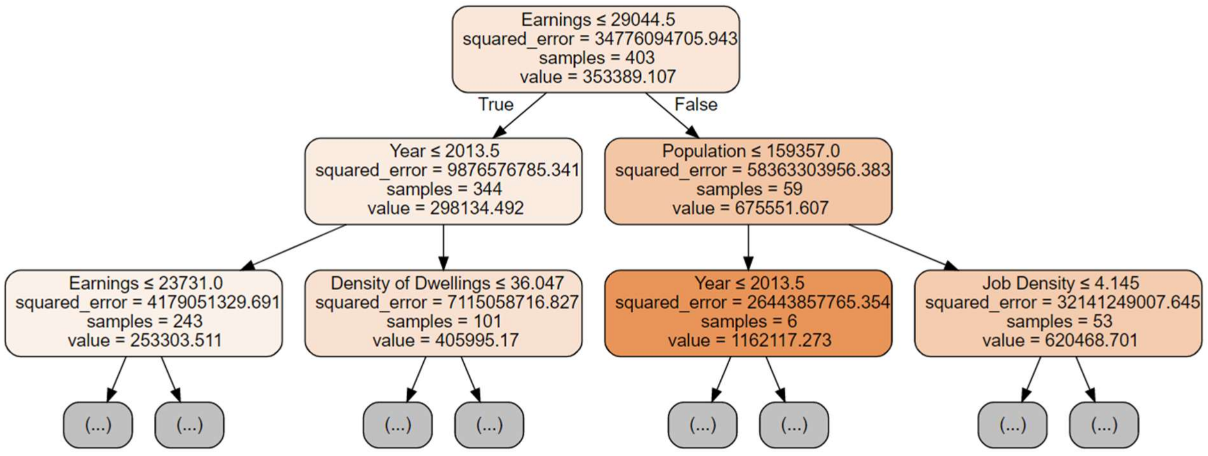


Figure 6: Visual Representation of Decision Tree Model

3. Random Forest Regression

After training a linear regression model and decision tree model we decided to try a Random Forest Regression to see if we could get even better results. To create the Random Forest model, we imported the RandomForestRegressor module from the scikit-learn library and instantiated a RandomForestRegressor object with 100 trees and a random_state parameter set to 42. We trained the model on the training set using the fit() function, which builds multiple decision trees using a random subset of the training data and a random subset of the independent variables at each split. This process creates an ensemble of decision trees that work together to make predictions. We then used the predict() function to make predictions on the test set and stored those predictions in a variable called y_pred2. The model combines the predictions of all the trees in the forest to generate a single prediction for each test sample.

Model Evaluation :

The models were evaluated using several metrics, including mean cross-validation score, mean absolute error, root mean squared error, and R-squared score. The random forest model performed the best, with the highest mean cross-validation score of 0.794, as well as the lowest mean absolute error and root mean squared error, indicating superior accuracy and precision. However, the regression model achieved the highest R-squared score of 0.950, meaning it accounted for the most variability in the data. Despite this, the random forest model is the most suitable model overall based on the evaluation metrics.

Metrics	**Model**		
	Regression	**Decision Tree**	**Random Forest**
CV mean	0.758	0.77	0.794
CV sd	0.286	0.111	0.09
MAE	34064.407	40878.306	31659.994
MSE	1.89e+09	3.71e+09	2.38e+09
RMSE	43517.49	60870.621	48798.896
R2 Score	0.95	0.902	0.937

Table 1: Evaluation Metrics of Supervised Models

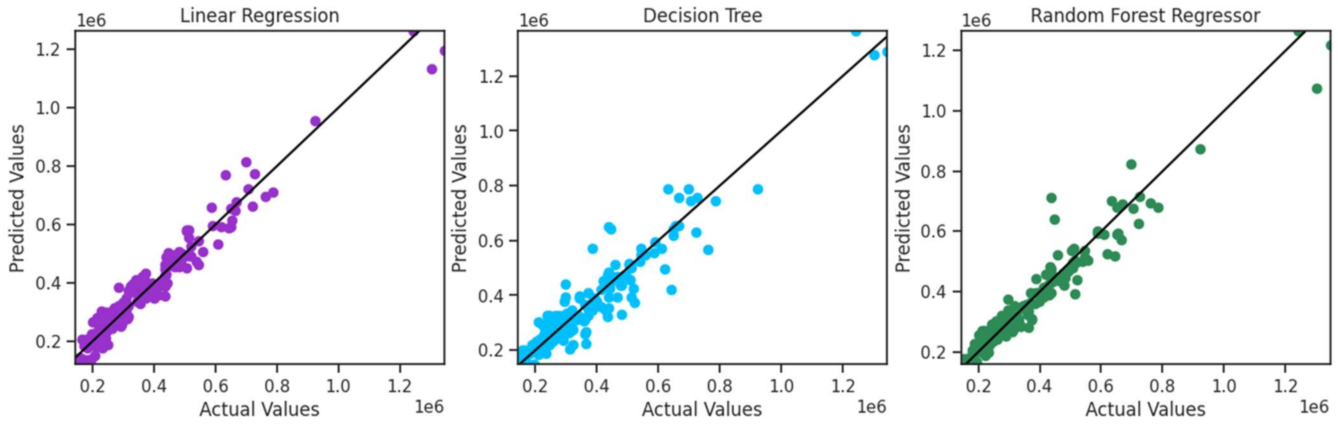


Figure 7: Side by Side Comparison of Supervised Learning Models

3.4.2 Unsupervised Modelling

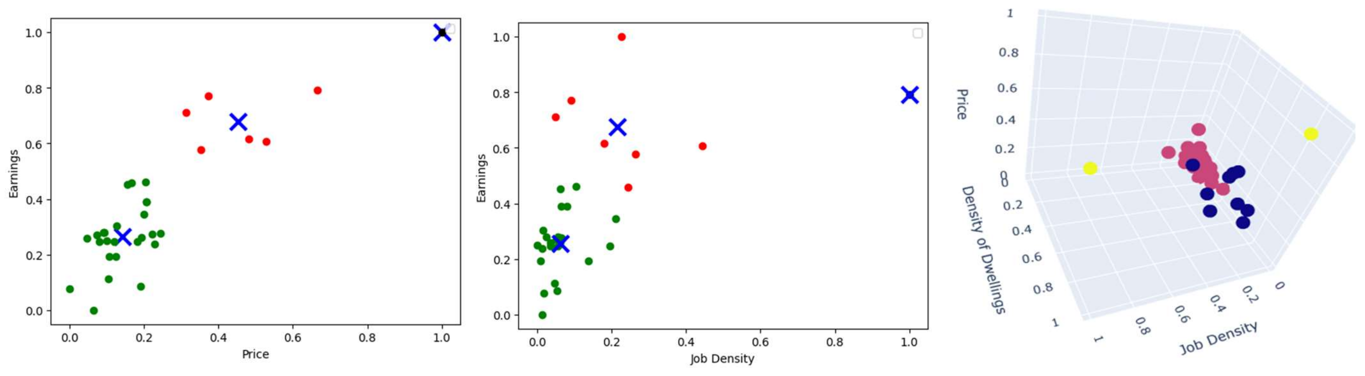
Unsupervised modelling is a type of machine learning where an algorithm learns patterns in data without prior knowledge of labelled outputs. The model searches for similarities or structure in the data, often clustering data points into groups or finding latent variables. For this report we would be using K means clustering for further analysis.

1.K Means Clustering

First, we scaled the data using MinMaxScaler from the Scikit-learn library. We then created three subsets of data based on the variables of interest:

1. Housing Price and Earnings
2. Job Density and earnings
3. Job density, Density of Dwellings and Housing Price.

For the first subset of data, we assumed that there are three clusters based on the visualisation of the scatter plot. We then fitted the K-means model to the data and identified three clusters. As in Fig 8 (a) Cluster 0 consisted of boroughs with relatively low housing prices and earnings. They include Barking and Dagenham, Brent, Enfield, etc. Cluster 1 had higher housing prices and earnings. They included Camden, Islington etc and cluster 2 included only Kensington and Chelsea borough, which had the highest housing prices and earnings among all boroughs. For the second subset of data, we used the elbow method as the optimal number of clusters could not be visually determined using a scatter plot and decided on three clusters.



(a) *(b)* *(c)*

Figure 8: 2-Dimensional(a,b) and 3-Dimensional(c) Data Clustering

As shown in fig 8(b) we found that cluster 0 including Ealing , Merton and 22 other Boroughs had a relatively lower job density and earnings, cluster 1 comprising of 7 clusters like Camden , Wandsworth etc had a higher job density and earnings than cluster 0 but lower than cluster 2, and cluster 2 which had only Westminster had the highest job density and earnings compared to the other two clusters.

Finally, for the third subset of data, we applied the K-means algorithm to identify clusters based on job density, density of dwellings, and price as shown in fig 9. We determined the optimal number of clusters to be three and found that cluster 0 had a relatively low job density, density of dwellings, and housing price, cluster 1 had a higher job density and density of dwellings than cluster 0 but lower than cluster 2, and cluster 2 had the highest job density, density of dwellings, and housing price compared to the other two clusters. We used plotly.express to visualise our cluster.

Model Evaluation :

Based on the clustering analysis of the dataset, we evaluate the performance of the models using the Silhouette Coefficient [8] and Calinski-Harabasz Index scores [9].

Data Subset no.	Silhouette Coefficient	Calinski-Harabasz Index
1	0.632	71.8
2	0.586	47.47
3	0.539	41.66

Table 2: Evaluation Metrics of Clustering

We obtained the following Silhouette Coefficient and Calinski-Harabasz Index scores for each data subset.Based on these scores, we can see that Clustering of 1st data set has the best performance with the highest Silhouette Coefficient and Calinski-Harabasz Index scores, indicating well-separated and well-defined clusters. 2nd dataset and 3rd dataset have lower scores, indicating that the clusters may not be as well-defined and may have some overlap between them. To improve the clustering performance, we can try different hyperparameters for the clustering algorithm, such as the number of clusters or the distance metric, to find the best values that give the highest Silhouette Coefficient and Calinski-Harabasz Index. We can also try different pre-processing techniques, such as feature scaling or dimensionality reduction, to improve the clustering performance.

4. KEY INSIGHTS FROM ANALYSIS:

1. Kensington and Chelsea, Richmond upon Thames, and Wandsworth have the highest average earnings, while Newham and Barking and Dagenham have the lowest.
2. Westminster, Kensington, and Chelsea have the highest job densities, while Newham and Haringey have the lowest.
3. Kensington, Chelsea, and Islington have the highest density of dwellings, while Havering and Bexley have the lowest.
4. Wandsworth, Merton, and Bromley offer more affordable housing options and moderate average incomes and job opportunities.
5. Barking and Dagenham and Newham offer the cheapest housing options but with lower average incomes and job densities.
6. Boroughs such as Wandsworth, Merton, Lambeth, and Hammersmith and Fulham could be attractive options for middle-class people, as they have average prices, moderate to high average earnings, an average density of dwellings, and a moderate to high job density index. These boroughs could provide a balance between affordability, job opportunities, and access to amenities.

5.CONCLUSION

Based on the results of the analysis, it is evident that there exists a relationship between the predictor variables (earnings, job density, density of dwellings, and population) and the target variable (price). Three supervised models were constructed, namely multiple linear regression, decision tree, and random forest regression, to predict the housing prices based on the predictor variables. Additionally, k-means clustering was used to group the data based on their similarity. The multiple linear regression model was able to explain the relationship between the predictor variables and the target variable. The model showed that all four predictor variables had a significant impact on the target variable. The decision tree and random forest models provided similar results, with high accuracy in predicting the target variable based on the predictor variables.

Furthermore, the k-means clustering analysis revealed distinct clusters of data. This finding suggests that the housing market is driven by multiple factors, and a one-size-fits-all approach to pricing would not be appropriate. In conclusion, the analysis suggests that the housing prices are influenced by multiple factors, including earnings, job density, density of dwellings, and population. The models constructed in this study can be used as tools to predict the housing prices. This knowledge can help policymakers and normal people to make informed decisions about pricing and investment in the housing market.

References

[1] A. A. AWONAIKE, "ESTIMATING UK HOUSE PRICES USING MACHINE LEARNING," p. 224, 2022.

[2] X. C. H. C. Y. &. Q. J. Han, "Housing affordability on Twitter: A sentiment analysis of London and New York City," vol. 89, pp. 117-126, 2019.

[3] Y. &. L. C. Li, "Trend Analysis of London Housing Market: An Investigation Based on Topic Modeling," Journal of Applied Mathematics, Statistics and Informatics, vol. 16, no. 1, pp. 1-9, 2020.

[4] M. Sharma, "Data Preprocessing: 6 Necessary Steps for Data Scientists," 27 October 2020. [Online]. Available: https://hackernoon.com/what-steps-should-one-take-while-doing-data-preprocessing-502c993e1caa. [Accessed 29 April 2023].

[5] T. T. R. &. F. J. Hastie, The elements of statistical learning: data mining, inference, and prediction, Springer Science & Business Media., 2009.

[6] D. Hudson, A beginner's guide to machine learning with scikit-learn, Sebastopol: O'Reilly Media, Inc., 2017.

[7] F. V. G. G. A. M. V. T. B. G. O. .. &. V. J. Pedregosa, "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, p. 2830, 2011.

[8] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," Journal of computational and applied mathematics, vol. 20, 1987.

[9] T. a. H. J. Author: Calinski, "A dendrite method for cluster analysis," Communications in Statistics-theory and Methods, vol. 3, no. 1, 1974.

APPENDIX

Code Repository : To view the Google Colab code used in this analysis, please visit the below link. The code is publicly available and can be downloaded and used for further research or analysis.

Link: https://colab.research.google.com/drive/11FgQpRuvqUmdlyiTBxmZjmA0ChmVnDZ1?usp=sharing

Supervised Learning :

```
# @title Importing necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.cluster import KMeans
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

```
# @title **Supervised Learning Models**
# Specifying target and  independent variables
X = merged_df1.drop(['Price'], axis=1)
y = merged_df1['Price']
# Splitting into training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

```
# @title Trying Regression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

scores_rg = cross_val_score(regressor, X, y, cv=5)

# Print the mean and standard deviation of the cross-validation scores of Regression
print('Cross-validation scores of Regression Model:', scores_rg)
print('Mean of Cross-validation scores of Regression Model:', scores_rg.mean())
print('Standard deviation of Cross-validation scores of Regression Model:', scores_rg.std())

from sklearn.metrics import mean_absolute_error,mean_squared_error,explained_variance_score

# Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error (MAE):", mae)

# Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)

# Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)

# R-squared score
r2 = r2_score(y_test, y_pred)
print("R-squared score:", r2)
```

```
# Get the coefficients and intercept
coefficients = regressor.coef_
intercept = regressor.intercept_

# Create the equation
equation = 'y = ' + str(intercept)

for i, col in enumerate(X.columns):
    equation += ' + (' + str(coefficients[i]) + ' * ' + col + ')'

print(equation)
```

```
# @title Trying Decision Tree
decisiontree=DecisionTreeRegressor()
decisiontree.fit(X_train,y_train)

y_pred1 = decisiontree.predict(X_test)
```

```
scores_dt = cross_val_score(decisiontree, X, y, cv=5)

# Print the mean and standard deviation of the cross-validation scores of Decision Treee
print('Cross-validation scores of Decision Tree:', scores_dt)
print('Mean:', scores_dt.mean())
print('Standard deviation:', scores_dt.std())

# Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred1)
print("Mean Absolute Error (MAE):", mae)

# Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred1)
print("Mean Squared Error (MSE):", mse)

# Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)

# R-squared score
r2 = r2_score(y_test, y_pred1)
print("R-squared score:", r2)
```

```
# Visualising the Decision Tree
import graphviz

# Export decision tree to DOT format
dot_data = export_graphviz(decisiontree, out_file=None,
                           feature_names=X.columns,
                           filled=True, rounded=True,
                           special_characters=True, max_depth=2)

# Display the decision tree using Graphviz
graph = graphviz.Source(dot_data)
graph.render('decision_tree')  # Save the decision tree as a PDF file
graph
```

```
# @title Trying Random Forest Regression
from sklearn.ensemble import RandomForestRegressor

# Create a Random Forest model with 100 trees
rf = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model on the training set
rf.fit(X_train, y_train)

# Make predictions on the test set
y_pred2 = rf.predict(X_test)

# Evaluate the model performance
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Cross Validation Scores
scores_rf = cross_val_score(rf, X, y, cv=5)
print('Cross-validation scores of Random Forest:', scores_rf)
print('Mean:', scores_rf.mean())
print('Standard deviation:', scores_rf.std())

print('Mean Absolute Error (MAE):', mean_absolute_error(y_test, y_pred2))
print('Mean Squared Error (MSE):', mean_squared_error(y_test, y_pred2))
print('Root Mean Squared Error (RMSE):', np.sqrt(mean_squared_error(y_test, y_pred2)))
print('R-squared score:', r2_score(y_test, y_pred2))
```

```
# @title **Comparing the 3 Models**

# Create scatter plot for Linear Regression model
fig, axs = plt.subplots(1, 3, figsize=(15, 4))
axs[0].scatter(y_test, y_pred, color='darkorchid')
axs[0].set_title('Linear Regression')
axs[0].set_xlabel('Actual Values')
axs[0].set_ylabel('Predicted Values')
axs[0].set_xlim([np.min(y_test), np.max(y_test)])
axs[0].set_ylim([np.min(y_pred), np.max(y_pred)])
axs[0].plot([np.min(y_test), np.max(y_test)], [np.min(y_test), np.max(y_test)], color='black')
```

```
# Create scatter plot for Decision Tree model
axs[1].scatter(y_test, y_pred1, color='deepskyblue')
axs[1].set_title('Decision Tree')
axs[1].set_xlabel('Actual Values')
axs[1].set_ylabel('Predicted Values')
axs[1].set_xlim([np.min(y_test), np.max(y_test)])
axs[1].set_ylim([np.min(y_pred1), np.max(y_pred1)])
axs[1].plot([np.min(y_test), np.max(y_test)], [np.min(y_test), np.max(y_test)], color='black')

# Create scatter plot for Random Forest Regressor model
axs[2].scatter(y_test, y_pred2, color='seagreen')
axs[2].set_title('Random Forest Regressor')
axs[2].set_xlabel('Actual Values')
axs[2].set_ylabel('Predicted Values')
axs[2].set_xlim([np.min(y_test), np.max(y_test)])
axs[2].set_ylim([np.min(y_pred2), np.max(y_pred2)])
axs[2].plot([np.min(y_test), np.max(y_test)], [np.min(y_test), np.max(y_test)], color='black')

plt.show()
```

Unsupervised Learning :

```
# @title **Unsupervised Learning Model- Kmeans Clustering**
df=merged_df.groupby(['Borough']).mean()

# Scaling Data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

scaler.fit(df[['Price']])
df['Price'] = scaler.transform(df[['Price']])

scaler.fit(df[['Earnings']])
df['Earnings'] = scaler.transform(df[['Earnings']])

scaler.fit(df[['Job Density']])
df['Job Density'] = scaler.transform(df[['Job Density']])

scaler.fit(df[['Density of Dwellings']])
df['Density of Dwellings'] = scaler.transform(df[['Density of Dwellings']])
```

```
# @title Clustering For Housing Price and Earnings - 1st subset of Data
df1=df[['Price','Earnings']]
plt.scatter(df1['Price'],df1['Earnings'])
plt.xlabel('Price')
plt.ylabel('Earnings')
# Asumming cluster count =3 from visualisation
km=KMeans(n_clusters=3, random_state= 42)
y_predicted1=km.fit_predict(df1[['Price','Earnings']])
df1['Cluster']=y_predicted1

dfc1 = df1[df1.Cluster==0]
dfc2 = df1[df1.Cluster==1]
dfc3 = df1[df1.Cluster==2]

centroids = km.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=200, linewidths=3, color='blue')

plt.scatter(dfc1.Price,dfc1['Earnings'],color='green')
plt.scatter(dfc2.Price,dfc2['Earnings'],color='red')
plt.scatter(dfc3.Price,dfc3['Earnings'],color='black')
plt.xlabel('Price')
plt.ylabel('Earnings')
plt.legend()
```

```
from sklearn.metrics import silhouette_score, calinski_harabasz_score

# Calculate Silhouette Coefficient for cluster 1
score1 = silhouette_score(df1[['Price','Earnings']], df1['Cluster'])

# Calculate Calinski-Harabasz Index for cluster 1
score2 = calinski_harabasz_score(df1[['Price','Earnings']], df1['Cluster'])

print('Silhouette Coefficient:', score1)
print('Calinski-Harabasz Index:', score2)
```

```
# @title Clustering for Job Density And Earnings  - 2nd subset of Data
df2=df[['Job Density','Earnings']]
plt.scatter(df2['Job Density'], df2['Earnings'])
```

```

# Specify the range of clusters to test
n_clusters_range = range(1, 10)

# Create an empty list to store the WCSS values
wcss0 = []

# Loop through the range of clusters and calculate the WCSS for each
for n_clusters in n_clusters_range:
    kmeans = KMeans(n_clusters=n_clusters)
    kmeans.fit(df2)
    wcss0.append(kmeans.inertia_)

# Plot the WCSS values against the number of clusters
plt.plot(n_clusters_range, wcss0)
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
plt.show()
```

```
kmeans = KMeans(n_clusters=3, random_state=42)

# Fit the model to the data
kmeans.fit(df2)

# Get the cluster labels for each data point
labels = kmeans.labels_

# Add the cluster labels to the original dataframe
df2['Cluster'] = labels
```

```
dfcc1 = df2[df2.Cluster==0]
dfcc2 = df2[df2.Cluster==1]
dfcc3 = df2[df2.Cluster==2]

plt.scatter(dfcc1['Job Density'],dfcc1['Earnings'],color='green')
plt.scatter(dfcc2['Job Density'],dfcc2['Earnings'],color='red')
plt.scatter(dfcc3['Job Density'],dfcc3['Earnings'],color='black')
plt.xlabel('Job Density')
plt.ylabel('Earnings')

centroids = kmeans.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=200, linewidths=3, color='blue')

plt.legend()

# Calculate Silhouette Coefficient for cluster 2
score1 = silhouette_score(df2[['Job Density','Earnings']], df2['Cluster'])

# Calculate Calinski-Harabasz Index for cluster 2
score2 = calinski_harabasz_score(df2[['Job Density','Earnings']], df2['Cluster'])

print('Silhouette Coefficient:', score1)
print('Calinski-Harabasz Index:', score2)
```

```
# @title Clustering For Job Density, Density of Dwellings and Price - 3rd subset of Data
df3=df[['Job Density','Density of Dwellings','Price']]
```

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Specify the range of clusters to test
n_clusters_range = range(1, 10)

# Create an empty list to store the WCSS values
wcss = []

# Loop through the range of clusters and calculate the WCSS for each
for n_clusters in n_clusters_range:
    kmeans = KMeans(n_clusters=n_clusters,)
    kmeans.fit(df3)
    wcss.append(kmeans.inertia_)

# Plot the WCSS values against the number of clusters
plt.plot(n_clusters_range, wcss)
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
plt.show()
```

```
kmeans = KMeans(n_clusters=3, random_state=42)

# Fit the model to the data
kmeans.fit(df3)

# Get the cluster labels for each data point
labels = kmeans.labels_

# Add the cluster labels to the original dataframe
df3['Cluster'] = labels
```

```
# Calculate Silhouette Coefficient for cluster 3
score1 = silhouette_score(df3[['Job Density','Density of Dwellings','Price']], df3['Cluster'])

# Calculate Calinski-Harabasz Index for cluster 3
score2 = calinski_harabasz_score(df3[['Job Density','Density of Dwellings','Price']], df3['Cluster'])

print('Silhouette Coefficient:', score1)
print('Calinski-Harabasz Index:', score2)
```

```
# Visualising 3d Cluster

import plotly.express as px

fig = px.scatter_3d(df3, x='Job Density', y='Density of Dwellings', z='Price', color='Cluster')
fig.show()
```