**UCD Professional Academy**

**Specialist Certificate in Data Analytics Essentials (June 2023) Project Report**

**Average House price prediction in London Borough**



**GitHub URL**

<https://github.com/kavyakkk/UCDPA_Kavyashree>

Note: This project is done in Google collab

**Abstract**

Predictive modeling project aimed at predicting average House prices in London Borough. Several key libraries like pandas, numpy, matplotlib, seaborn, and scikit-learn were used for data preprocessing, data manipulation, model implementation, performance evaluation for insightful visualizations. Supervised machine learning method used to predict average house price. Collected dataset from Kaggle, focused on 33 London boroughs region. It has property sold from 1995-2020. The scikit-learn library played a pivotal role, providing tools for model training, implementation and evaluation. 4 models were implemented to predict average house price, such as Linear Regression, Decision Tree Regressor, Lasso Regression, and K-Nearest Neighbors models.

The Decision Tree Regressor stood out as the best performer with a remarkable R-squared of 0.991 and low RMSE of 18,120. When we compare model, we have to choose high R-squared with Low RMSE and MSE. The high R-squared value of the Decision Tree Regressor underscores the model's proficiency in capturing detailed patterns, precision, variations, and its effectiveness in predicting average house prices.

Actual and predicted house prices were plotted in table and bar graph, enhancing the clarity and interpretability of the results. The key insights from the data are that Westminster, Lambeth, and Camden are the areas with the highest number of crimes. Trends show us that there is a consistent increase in average house prices over the years. Kensington and Chelsea emerge as notably expensive areas, emphasizing the importance of location in influencing property values. Highest number of houses sold was Wandsworth, with a total of 164,967 transactions. City of London had the fewest houses sold among the London boroughs, with a total of 7,636 transactions. Additionally, temporal trends, correlations with crime rates, borough area analyses, and insights into houses sold over time enriched the understanding of real estate dynamics in the London market. Borough (Location) and crime rates are the important factors in predicting property prices.

**1.Introduction**

The rich and diverse culture of the UK has always fascinated me, particularly the dynamic tapestry of London. As a thriving and cosmopolitan city, London serves as a hub of cultural, economic, and social activities. In particular, the UK's reputation as a hub for innovation and creativity, especially in areas such as technology and the arts, adds to its allure.

The dataset consists of average house prices in 33 London boroughs and 12 areas outside of London. The decision to focus on the London boroughs, known for their distinct personalities and roles within the greater metropolitan subject, aligns with my fascination for the multifaceted nature of the UK.

The London boroughs are 32 local government districts that, along with the City of London, form Greater London's administrative area. Governed by individual councils, these boroughs have populations ranging from 150,000 to 400,000. Borough councils primarily handle local services such as schools, waste management, and social services, while the Greater London Authority oversees the broader strategic aspects of Greater London

**Relevance to Real-World Challenges**: This Housing prices are a critical aspect of urban life and can be indicative of economic trends, and overall living standards. Predicting house prices is a valuable and practical challenge with real-world applications.

**Focused on Urban area (London Boroughs):** Project is focused only on London Boroughs excluded rural region. it gives us more understanding of the factors influencing property values in urban region. Generally urban area houses have more demand and populated than rural area.

**Supervised Learning Objective:** The primary goal of predicting average house prices aligns with a common objective in supervised learning—to understand and predict trends based on historical data.

**Machine Learning Potential**: London housing dataset enabled me to try various machine learning algorithms. This diversity allowed explore different models, such as regression algorithms, to predict house prices accurately.

**2.Dataset**

Data source: <https://www.kaggle.com/datasets/justinas/housing-in-london/data>

The dataset used in this project was obtained from Kaggle, and the provided link directs to the specific webpage. This dataset is particularly intriguing as it encompasses average house prices in 33 London boroughs and 12 areas outside of London. Due to its comprehensive nature, it serves as an ideal candidate for developing a supervised learning machine learning model. The primary objective of this model is to predict the target feature, which, in this case, is the average house price.

There are 2 datasets in folder, Project is based on housing\_in\_london\_monthly\_variables.csv

The dataset contains the following feature variables,

**1.Date**: Property sold date

**2.Area**: Name of London boroughs and broader geographical regions across the UK, like England, the North East, and others

**3. Average price**: Monthly average house prices

**4.Code**: Code for each Area names. Code starts with E and followed by 8 digits.

**5.Houses\_sold**: Monthly number of houses sold

**6.No\_of\_crimes**: Monthly number of crimes committed

**7.Borough\_flag**: Dataset is split by areas of London called boroughs and broader geographical regions across the UK, like England, the North East, and others. It has binary distinction 1 meaning London Borough and 0 other regions.

3.**Data Preparation**

Data preparation is crucial when we fit any machine learning models. Model will not perform well, if data preparation is not done correctly.

**Step 3.1: Real-world scenario**

Dataset: housing\_in\_london\_monthly\_variables.csv

**Step 3.2: Import dataset in CSV form downloaded from Kaggle website**

The Project has a comprehensive set of Python libraries for house price prediction. Leveraging 'sklearn' for pre-processing and regression models, including 'LightGBM' for advanced prediction, 'tabulate' for organized tabular output.

In the Google Colab notebook, the Kaggle package is installed, and the Kaggle API key is uploaded to facilitate dataset access. (Create Kaggle account and Kaggle API can be downloaded from Account tab of user profile). The key is securely moved and permissions are set. Subsequently, the "housing-in-london" dataset is downloaded from Kaggle, and its contents are extracted using the unzip command & load dataset.

!pip install Kaggle

from google.colab import files

uploaded = files.upload()

!mkdir -p ~/.kaggle && mv kaggle.json ~/.kaggle/ && chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d justinas/housing-in-london

!unzip housing-in-london.zip

df = pd.read\_csv("housing\_in\_london\_monthly\_variables.csv")

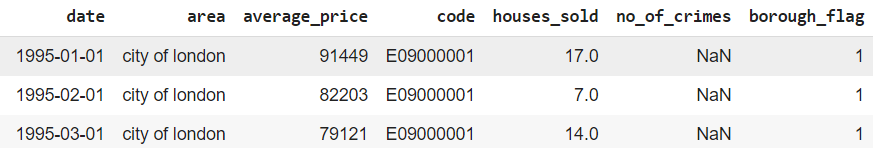
Or

Load dataset into Pandas DataFrame using the 'read\_csv' function from the 'pandas' library. The dataset, obtained in CSV format from the Kaggle website, is named "housing\_in\_london\_monthly\_variables.csv."

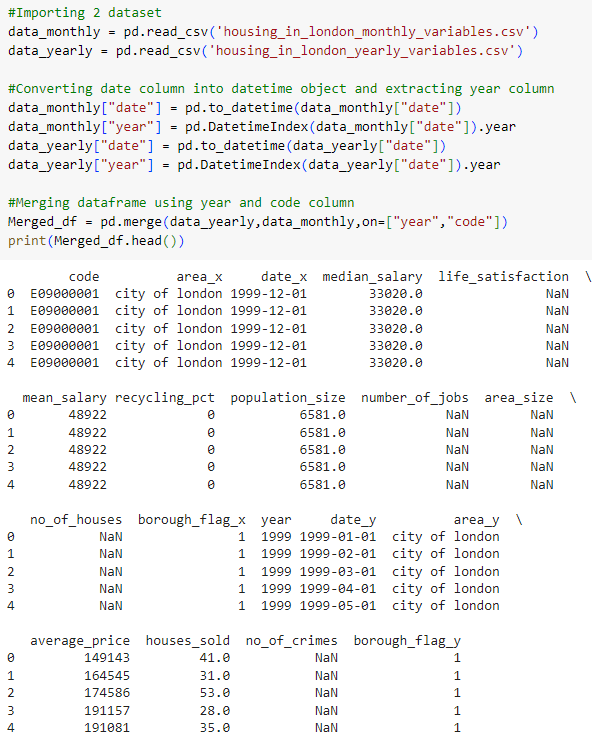
df = pd.read\_csv("housing\_in\_london\_monthly\_variables.csv")

**Step 3.3: Analysing dataframe**

Dataset has 13,549 rows and 7 Columns.

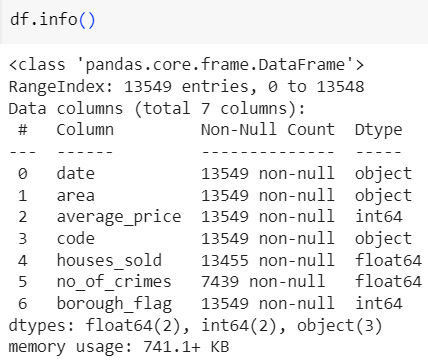


**Merge Dataframes:** Merging Monthly and Yearly dataset (Project analysis is performed on monthly dataset(df), For demonstration purpose I am merging monthly and yearly dataset).



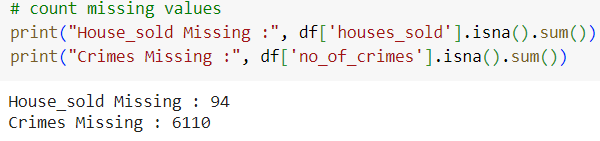
**3.4 Understanding Dataset:**

* no\_of\_crimes and houses\_sold have missing values. no\_of\_crimes have only 7439 non-null values, meaning that 6110 dates are missing crime features.
* average\_price, houses\_sold,and no\_of\_crimes are numerical/float values
* area and borough\_flag are categorical values even though borough\_flag is currently an integer data type



**3.5 Checking for Missing Values**

* houses\_sold has 94 NAN values, meaning no houses were sold during these months.
* no\_of\_crimes have only 7439 non-null values, meaning that 6110 dates are missing crime features.



**3.6 Handling missing Values**

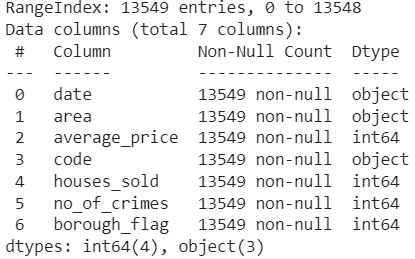
df['no\_of\_crimes'].fillna(df.groupby('area')['no\_of\_crimes'].transform('mean'), inplace = True)

df['houses\_sold'].fillna(df.groupby('area')['houses\_sold'].transform('mean'), inplace = True)

df['houses\_sold'] = df['houses\_sold'].fillna(0).astype(int)

df['no\_of\_crimes'] = df['no\_of\_crimes'].fillna(0).astype(int)

df.info()



* We have replaced NULL data with 'mean' based on area (groupby object).
* Some area may not have crime data. If all values within a specific 'area' are NaN.calculating the mean for that 'area' will result in NaN.so we are filling 'houses\_sold' and 'no\_of\_crimes' by 0 , using the fillna method. then converting the columns to integer datatype by using the astype(int) method.

**3.7 Checking for currency symbol #$£€ in House price column.**

We are creating clean\_price\_and\_count Function to remove currency symbols from an average price and count occurrences of currency symbols using re.findall. Then we have to Initializes a dictionary (total\_counts) to store the total counts of each currency symbol.

The map function applies the clean\_price\_and\_count function to each element in the 'average\_price' column of the DataFrame.This creates an iterator (cleaned\_prices\_and\_counts) containing tuples with the original price and the result of the clean\_price\_and\_count function.

The zip function is used to iterate over pairs of elements from df.average\_price and cleaned\_prices\_and\_counts in parallel.



Result shows total counts for each currency symbol, Its zero. average\_price datatype is int64, meaning it contains only integers, there is no need for additional cleaning or counting based on currency patterns. I usually use this function to any dataset which has values like Price or salary.

# Define the pattern to match currency symbol #$£€, we can use any special characters.

currency\_pattern = re.compile(r'[#$£€#]')

def clean\_price\_and\_count(price):

# Convert to string before applying the regular expression

cleaned\_price = re.sub(currency\_pattern, '', str(price))

# Count occurrences of the currency pattern

count = len(re.findall(currency\_pattern, str(price)))

return cleaned\_price, count

# Initializes a dictionary (total\_counts) to store the total counts of each currency symbol.

total\_counts = {'$': 0, '£': 0, '€': 0, '#': 0}

cleaned\_prices\_and\_counts = map(clean\_price\_and\_count, df.average\_price)

#Result will be displayed by each rows by iteration:- Original: 208743, Cleaned: ('208743', 0).

for original, cleaned in zip(df.average\_price, cleaned\_prices\_and\_counts):

print(f"Original: {original}, Cleaned: {cleaned}")

# Update total counts based on individual Currency counts

for \_, (\_, count) in zip(df.average\_price, cleaned\_prices\_and\_counts):

total\_counts['$'] += count

total\_counts['£'] += count

total\_counts['€'] += count

total\_counts['#'] += count

# Display the total counts for each currency symbol

for symbol, count in total\_counts.items():

print(f"{symbol}: {count}")

**3.8 Transformation of Columns:**

* Removing E from Code variable : Code is Object datatype, Code begins with E for each area,we need to remove E. Converting to numeric values in float format, with any non-numeric entries replaced by 'NaN'.

df['code'] = df['code'].str.replace('E', '')

df['code'] = pd.to\_numeric(df['code'], errors='coerce') # 'coerce' will replace non-numeric values with NaN

* Creating New columns Month and Year: Year and Month columns are created. Year column is very helpful to show insights.

df["date"] = pd.to\_datetime(df["date"])

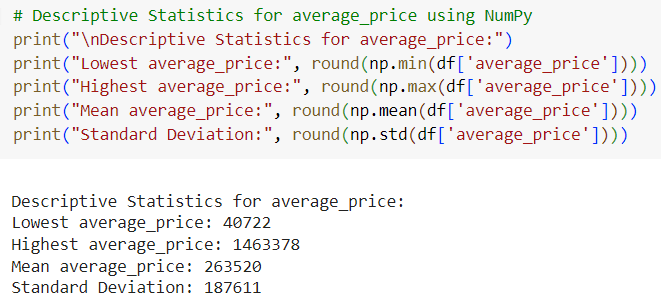
df["Month"] = df["date"].dt.month

df["Year"] = df["date"].dt.year

**4. Statistics and Explanatory Data Analysis**

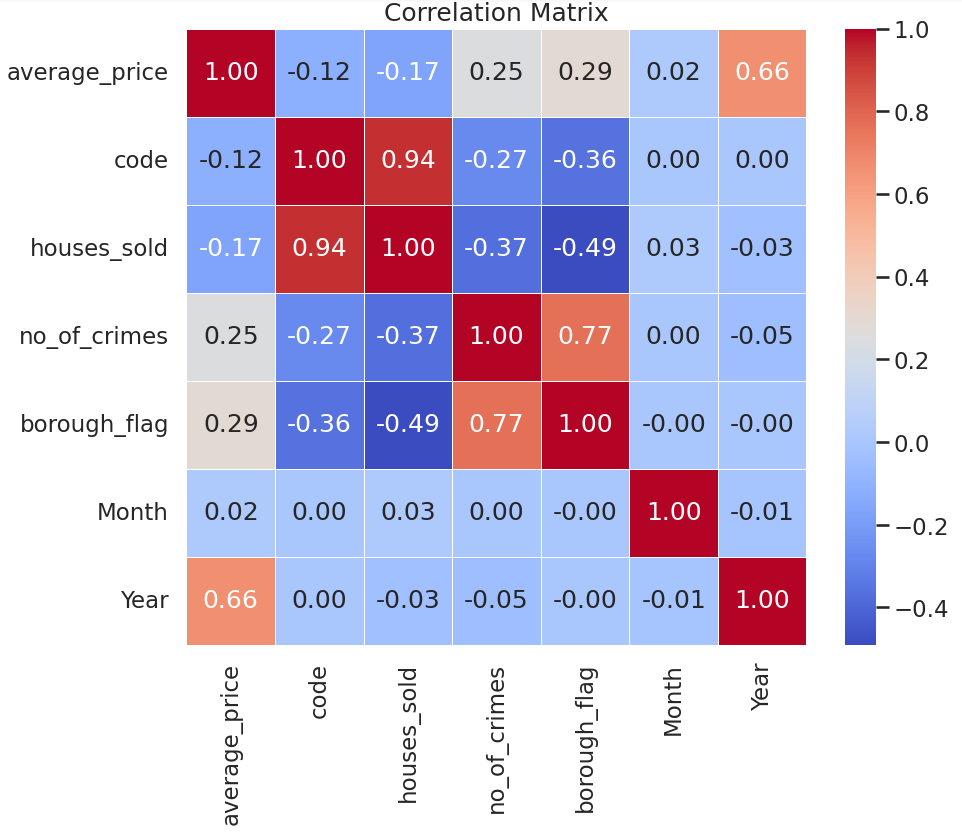
**4.1 Descriptive Statistics for average\_price using NumPy :**

Aggregated functions explains distribution of average prices in your dataset. The range between the lowest and highest average prices is quite large, indicating a variation in housing prices. Higher standard deviation suggests a more dispersed distribution of prices.



**4.2 Correlation Matrix:**

* Heatmap provides a visual summary of the relationships between different variables in the housing dataset. I have used correlation to check the other variables influencing on average\_price. If correlation coefficient close to 1, it is positive linear relationship, as one variable increases, the other also increases proportionally.
* matplotlib and seaborn library are used. seaborn library, which is built on top of matplotlib and provides a high-level interface for attractive and informative statistical graphics.
* correlation\_matrix = df.corr(): This calculates the correlation matrix for the DataFrame 'df' using the corr() function. The correlation matrix shows the pairwise correlation coefficients between different variables.



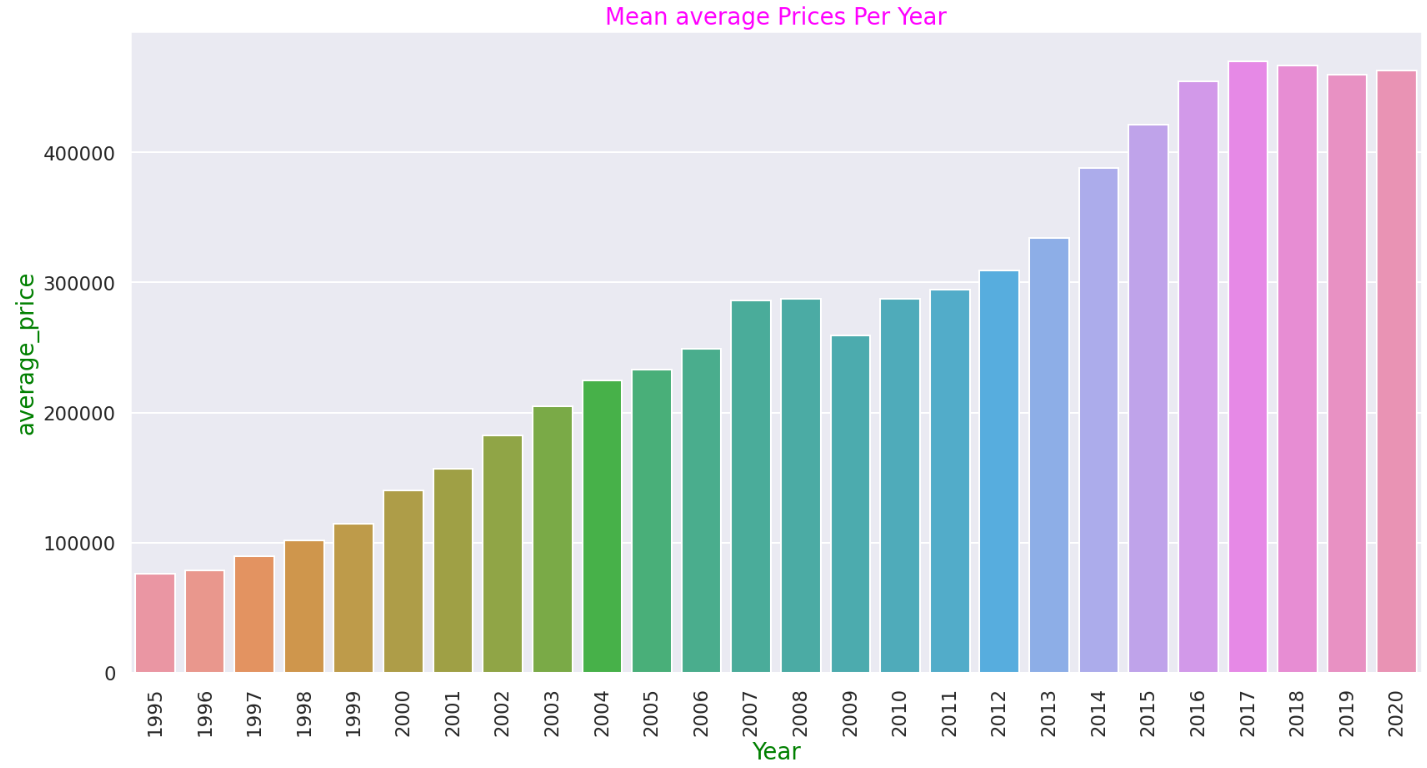
* average\_price vs Year: 0.66 indicating a strong positive relationship.There is a tendency for average prices to increase as the years go by. Indicates a consistent increase in average prices over the years, reflecting the dynamic nature of the real estate market.
* average\_price vs. no\_of\_crimes:0.25, indicating a weak positive relationship, average prices and the number of crimes suggests a nuanced interaction between housing dynamics and the safety perception of an area.
* average\_price vs. borough\_flag:0.29, indicating a weak positive relationship.
* houses\_sold vs. no\_of\_crimes: -0.37, indicating a moderate negative relationship.
* houses\_sold vs. borough\_flag: -0.49, indicating a moderate negative relationship.
* no\_of\_crimes vs. borough\_flag:0.77, indicating a strong positive relationship.
* average\_price vs. houses\_sold:-0.17, indicating a weak negative relationship.

**4.3 Exploratory data analysis (EDA) :**

* **Mean average Prices per Year**:

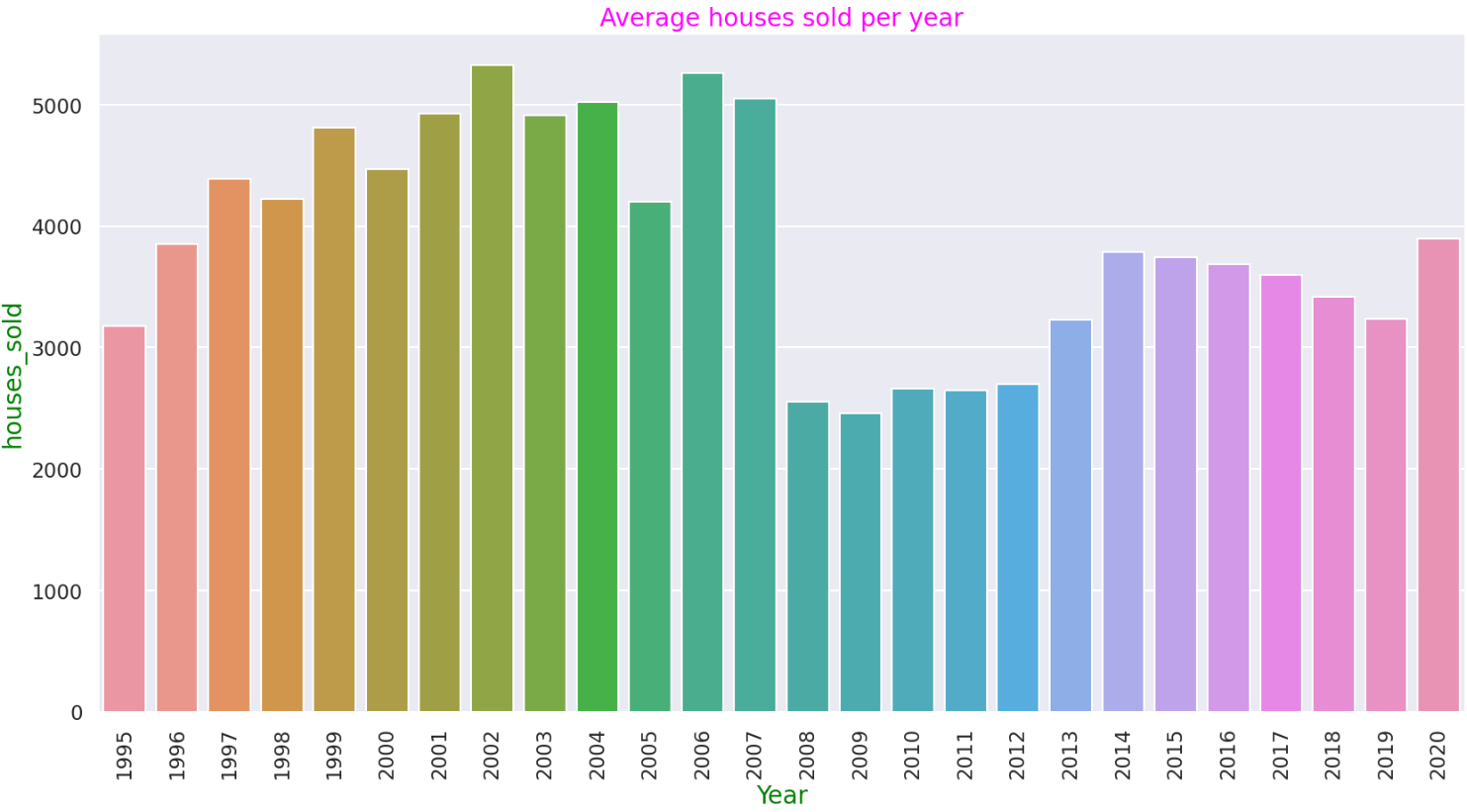
The bar graph depicts how the mean average price increases by year. Positive trend of increasing mean average prices over the years.

A new DataFrame is constructed by grouping the original dataset ('df') by unique years and calculating the mean of average prices for each year. The resulting DataFrame is then visualized using a Seaborn bar plot, illustrating the mean average prices per year.



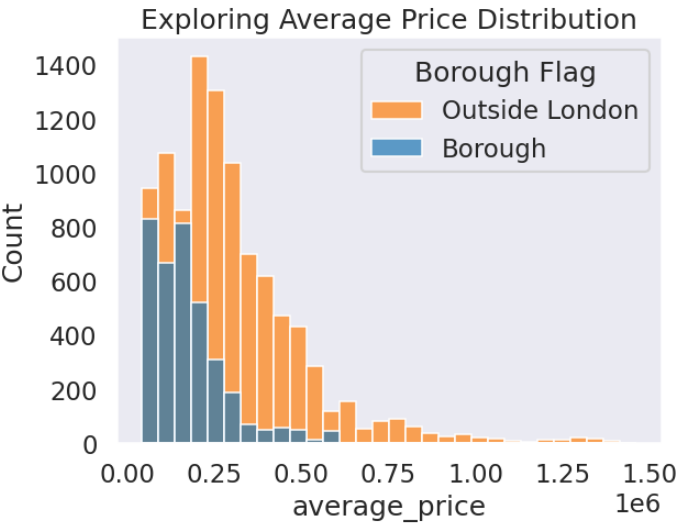
* **Average houses sold per year**: The below Bar graph shows the average houses sold per year. It reveals in decreasing trend in houses sold after the year 2007.

no\_house\_sold = df.groupby("Year")["houses\_sold"].mean().reset\_index() a new DataFrame is constructed by grouping the original dataset ('df') by unique years and calculating the mean of houses sold for each year. The resulting 'no\_house\_sold' DataFrame is then visualized using a Seaborn bar plot, illustrating the average houses sold per year.



* **Exploring Average Price Distribution**: Price Histogram shows the distribution Double-Peaked or Bimodal with Right-skewed to the area of borough flag 1 & 0.

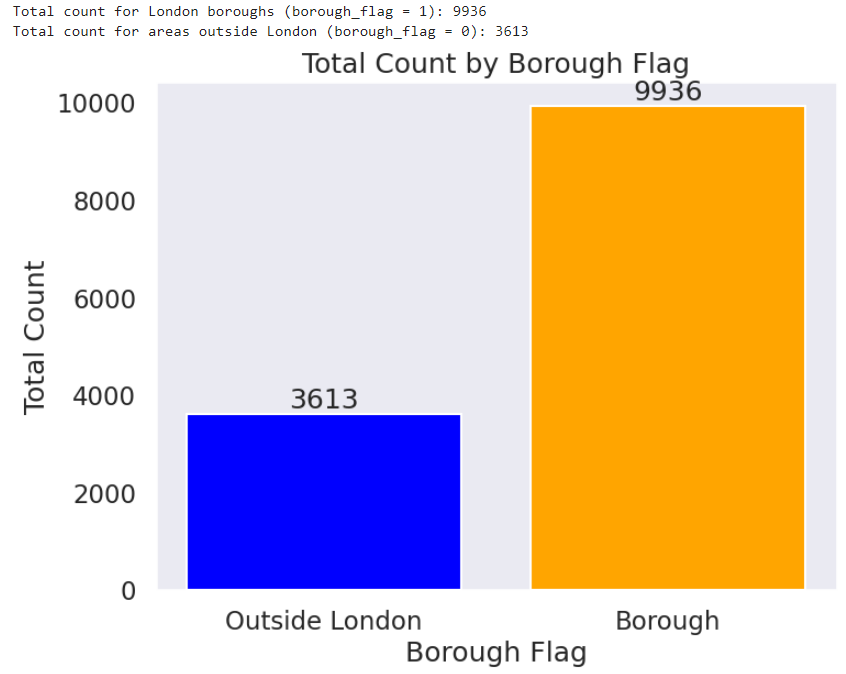
This visualization provides a clear overview of the distribution of average prices in the dataset. The color distinction based on 'borough\_flag' adds an additional layer of information, allowing for a visual comparison between properties located within London boroughs and those outside London.



* **Total count of rows by Borough Flag**: Borough\_flag has 1 and 0 value,Creating new dataframe that has total count of rows for each unique 'borough\_flag' value.

1.Total count for London boroughs (borough\_flag = 1): 9936

2. Total count for areas outside London (borough\_flag = 0): 3613

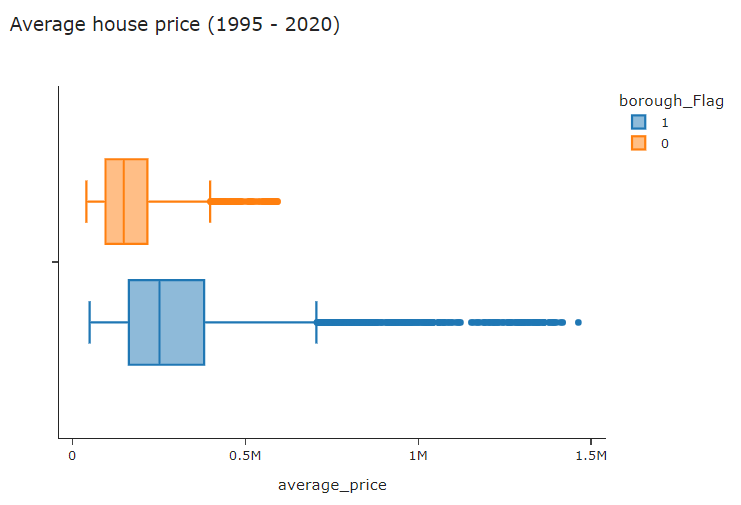


* **London boroughs vs Outside London** : London boroughs are the 32 local authority districts that together with the City of London (Total 33). Outside London there are 12 areas.

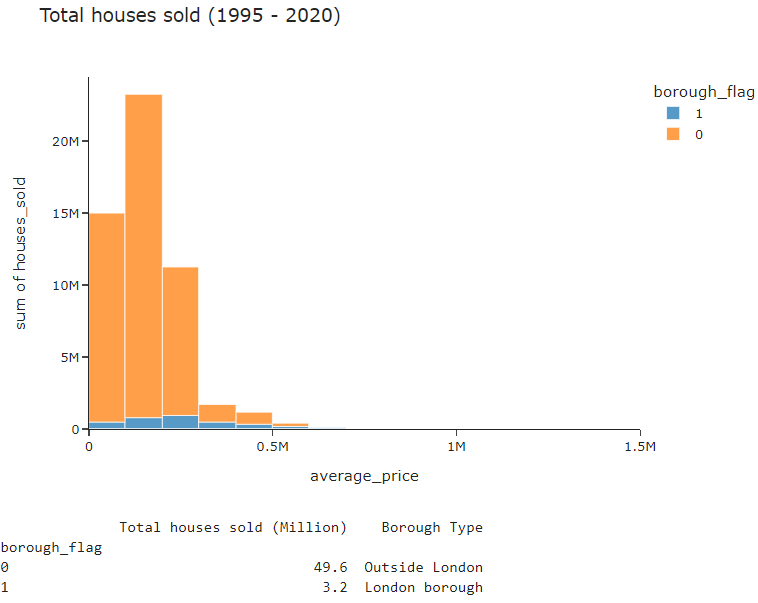


Above code prints Borough names. Loop prints the index (incremented by 1 to start from 1) and the name of each borough. During each iteration, enumerate function to gets both the index (i) and the borough name (name).

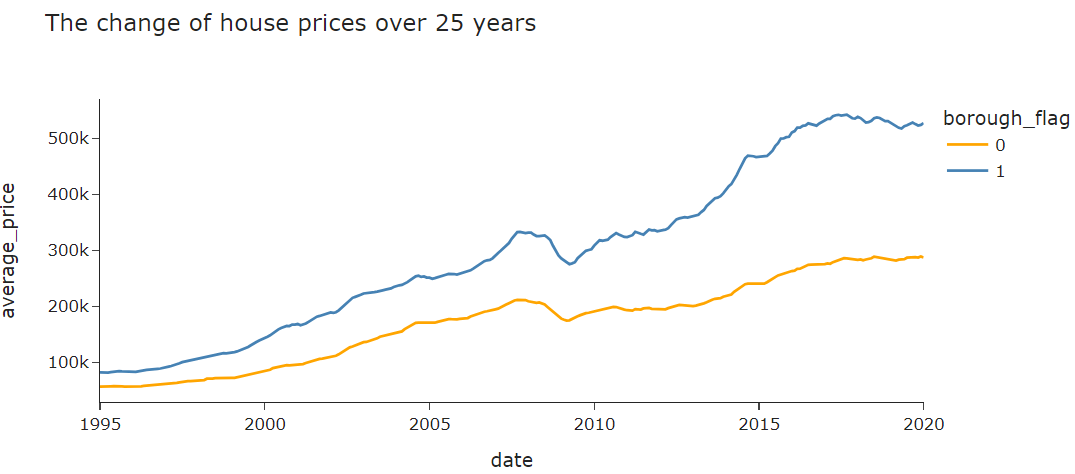
* **Average House prices (1995-2020)**: Boxplot shows the distribution of average house prices by 'borough\_flag'.It shows more dispersed distribution of house prices. Points beyond the whiskers are considered as outliers. Some areas in boroughs area have very high prices.



* **Total Houses sold (1995-2020)**: Histogram shows distribution of houses sold in average price by each category of borough\_flag. Table provide information about the total houses sold in millions for each 'borough\_flag' category. In Outside London, the total house sold is 49.6 million, and London borough ('borough\_flag' 1), the total house sold is 3.2 million. The houses sold of outside London are outnumbered London boroughs.

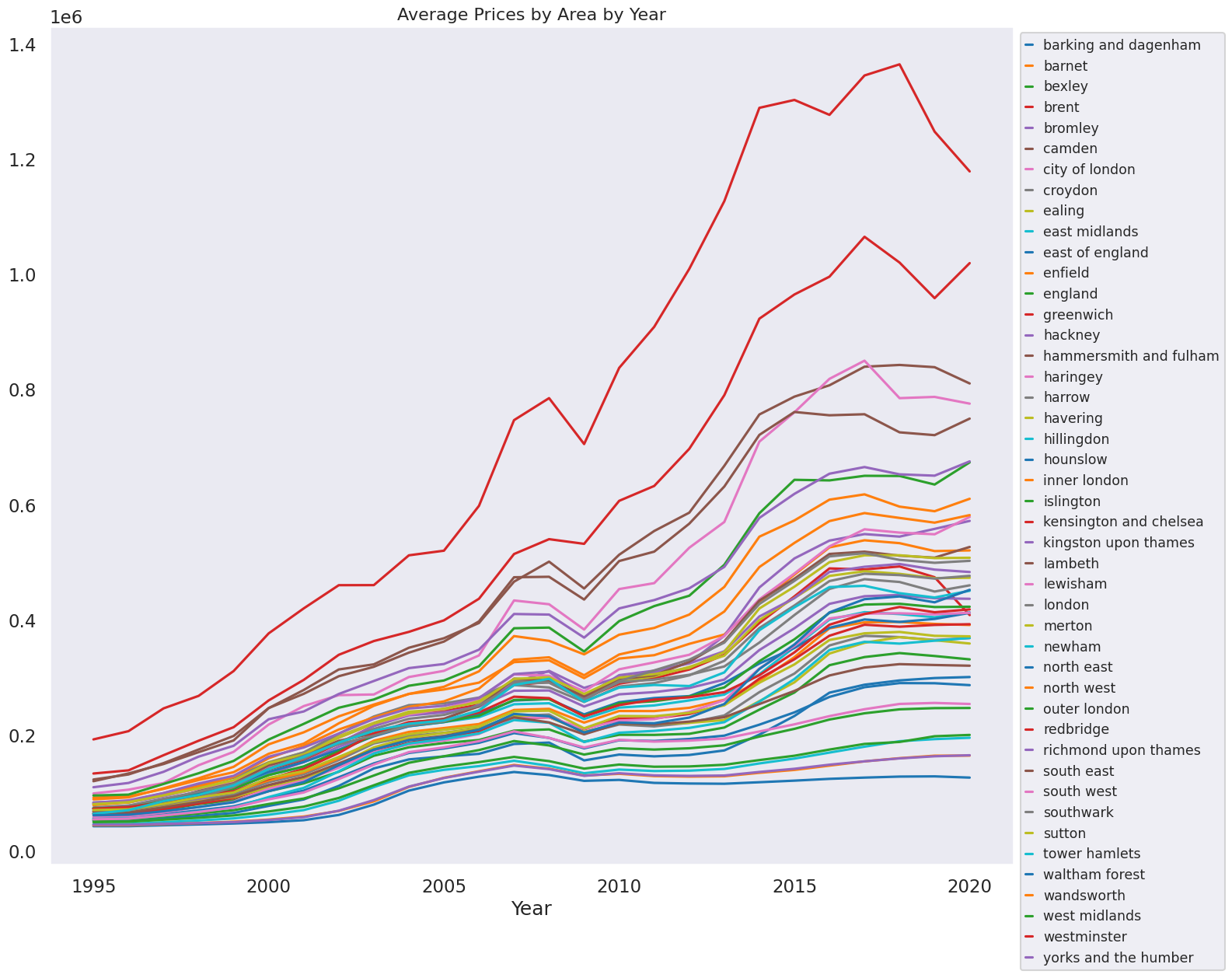


* **The change of house prices over 25 years**:



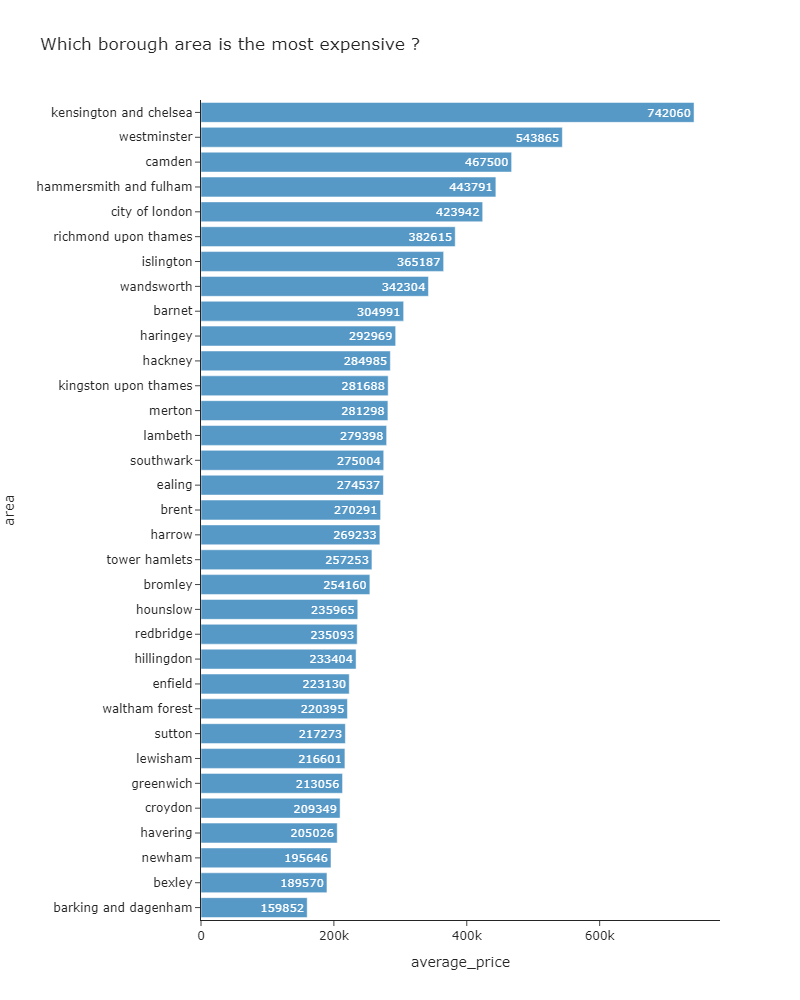
* + Line graph shows us the remarkable increase in average house prices as year goes on. From 1995 to 2020, house prices in London boroughs went up a lot, from about £82,000 to around £528,000 in 25 years. Prices vary a lot among boroughs, starting at £50,000 and going up to about £1.5 million.
  + Outside of London, house prices went up more slowly, from £57,000 to about £290,000, which is almost 1.8 times less than in London boroughs.
  + In 2008-2009, there was a big drop in house prices for both London boroughs and outside London. This happened because of the financial crash in 2007-2009, which was a tough time for the economy.
* **Average prices by Area**:

Line graph shows us the remarkable increase in average house prices as year goes on by area. Kensington and Chelsea, Westminter and Camden area are expensive to buy houses.



**4.4 London Borough area Analysis (Project is based on Borough area):**

Horizontal Bar Graph below shows expensive area in London Borough. The bar graph visualizes the mean average prices of houses by 33 areas in London Boroughs. Kensington and Chelsea, westminter and Camden are expensive areas. Houses are sold in very high prices. The mean average price for houses in the Kensington area is £742,060. However, if I were to consider purchasing a house in Kensington, I would be looking at a price range around £700,000.

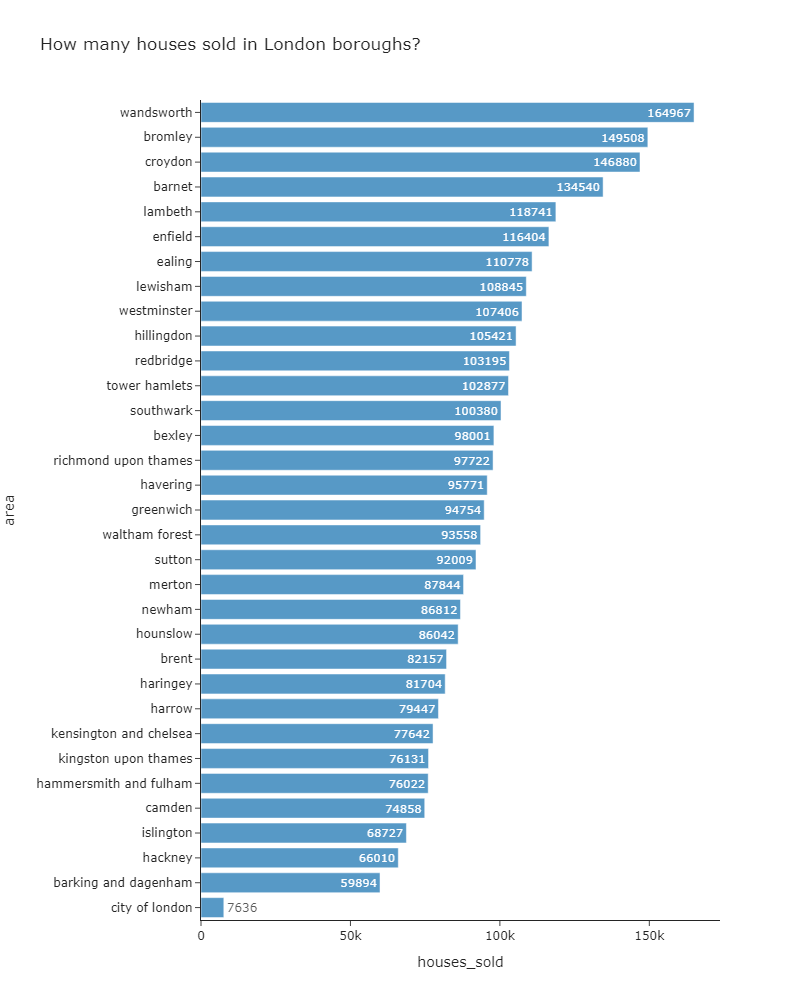


* 1. **Houses sold in London Boroughs:**

A horizontal bar graph below depicts the total number of houses sold across 33 areas in London Boroughs.

Notably, Wandsworth, Bromley, and Croydon are standout areas with particularly high numbers of houses sold.In the case of Wandsworth, a substantial 164,967 houses were sold from 1995 to 2020, showcasing a noteworthy level of real estate activity in that area.

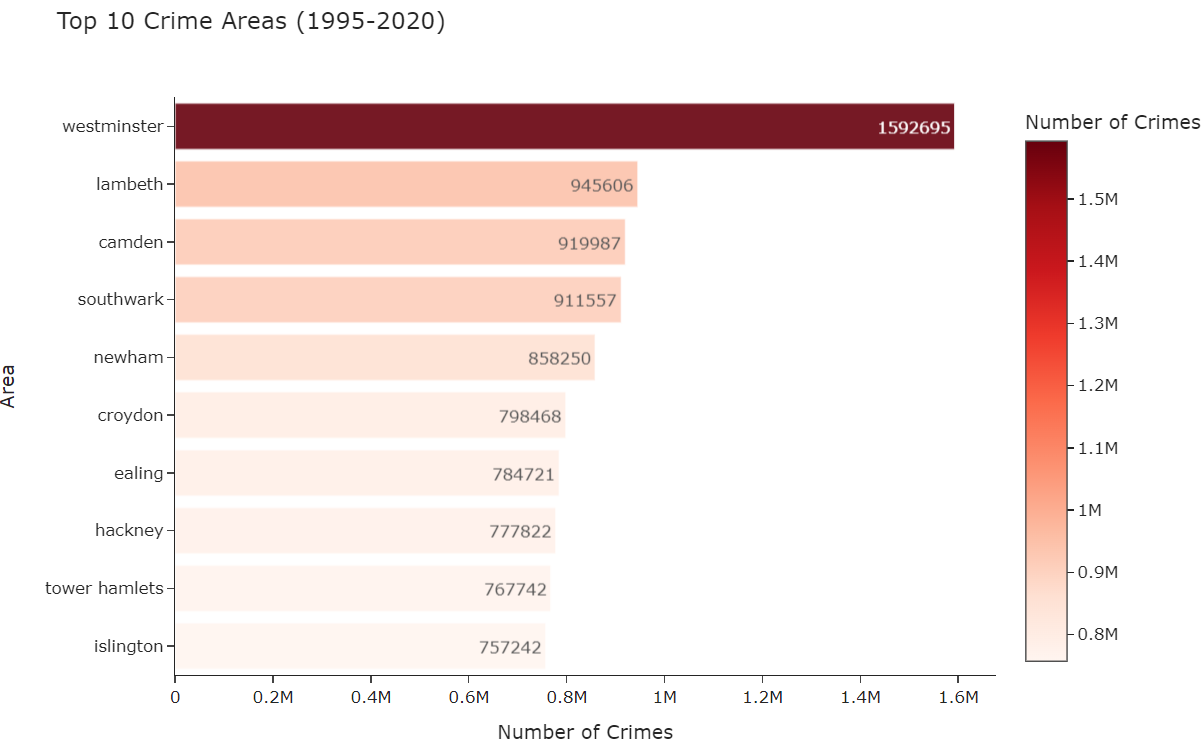
It's interesting to observe that the City of London has a comparatively lower total of 7,636 houses sold from 1995 to 2020. City of London had the fewest houses sold among the London boroughs, with a total of 7,636 transactions. It might be of small geographic area and a higher concentration of commercial properties.



* **Top 10 Crime Areas(1995-2020) :**

Horizontal Bar graph shows top 10 areas with the highest number of crimes from 1995 to 2020.Graph is generated using Plotly Express.

The df.groupby('area')['no\_of\_crimes'].sum().nlargest(10).reset\_index() part of the code groups the data by area, sums up the number of crimes for each area, selects the top 10 areas with the largest sum, and then resets the index for better plotting.



Analysis of crime data spanning from 1995 to 2020 reveals that Westminster, Lambeth, and Camden are among the top areas in London with the highest incidences of reported crimes. Westminster, in particular, stands out with a staggering total of 1,592,695 crimes over this period. This data raises concerns about the safety and security of these areas, with Westminster being notably identified as a location with a substantial number of reported crimes, suggesting potential challenges related to public safety.

**5. Machine Learning: Model Implementation**

**5.1 Pre-processing data to predict average\_price**

df\_boroughs = df[df['borough\_flag'] == 1]

def preprocessing\_data(df=df\_boroughs, training\_size=0.8):

# Drop unnecessary features

df\_predict = df.drop(columns =['code','borough\_flag','date'])

ohe = pd.get\_dummies(df\_predict['area'], drop\_first=True)

df\_predict = pd.concat([df\_predict, ohe], axis=1)

df\_predict = df\_predict.drop(columns=['area'], axis=1)

# Given x, y

x = df\_predict.drop(columns=['average\_price'])

y = df\_predict['average\_price']

# Train-test split (train data 80%)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, train\_size=training\_size, shuffle=True, random\_state=42)

scaler = StandardScaler()

scaler.fit(x\_train)

x\_train = pd.DataFrame(scaler.transform(x\_train), index=x\_train.index, columns=x\_train.columns)

x\_test = pd.DataFrame(scaler.transform(x\_test), index=x\_test.index, columns=x\_test.columns)

return x\_train, x\_test, y\_train, y\_test

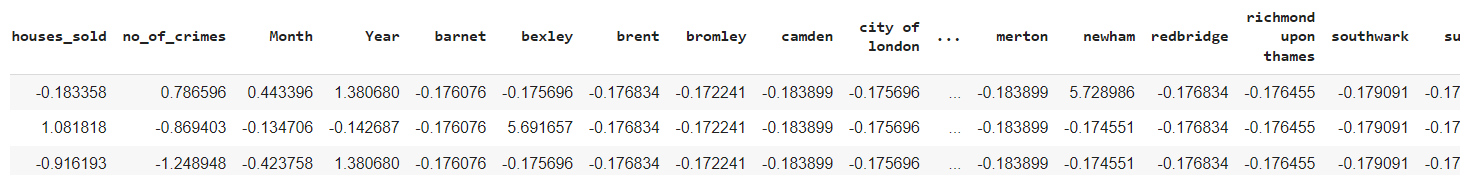
x\_train, x\_test, y\_train, y\_test = preprocessing\_data()

Data preparation is crucial when we fit any machine learning models. The initial step involves filtering the dataset to focus specifically on borough-related information. Subsequently, irrelevant features such as 'code,' 'borough\_flag,' and 'date' are excluded, streamlining the dataset for analysis. Month and Year is extracted from date column, so we are excluding date feature.

The categorical variable 'area' is then subjected to one-hot encoding to facilitate machine learning algorithms' , to avoid multicollinearity. Following this, the dataset is divided into training and testing sets, constituting 80% and 20% of the data, respectively, to evaluate model performance effectively. The features undergo standard scaling, a critical step in ensuring that numerical variables are on a comparable scale, preventing any particular feature from dominating the model training process.

The resulting preprocessed data, comprised of scaled features and the target variable 'average\_price,' is fundamental for constructing and assessing machine learning models aimed at predicting average house prices in London boroughs. This robust preprocessing methodology enhances the model's interpretability and generalization ability, contributing to the accuracy of predictions and the robustness of subsequent analyses.

X\_Train and x\_test dataframe after scaled and transformation feature will look like below.



**5.2 Implementing model**

# Initialize models

tree = DecisionTreeRegressor()

la = Lasso()

reg = LinearRegression()

knn = KNeighborsRegressor()

# Fit models

reg.fit(x\_train, y\_train)

tree.fit(x\_train, y\_train)

la.fit(x\_train, y\_train)

knn.fit(x\_train, y\_train)

# Make predictions

pd1 = reg.predict(x\_test)

pd3 = tree.predict(x\_test)

pd4 = la.predict(x\_test)

pd5 = knn.predict(x\_test)

# Calculate R-squared scores

s1 = r2\_score(y\_test, pd1)

s3 = r2\_score(y\_test, pd3)

s4 = r2\_score(y\_test, pd4)

s5 = r2\_score(y\_test, pd5)

# Create a DataFrame for visualization

Models = ["LinearRegression", "DecisionTreeRegressor", "Lasso", "KNeighborsRegressor"]

Scores = [s1, s3, s4, s5]

df = pd.DataFrame({"Models": Models, "Scores": Scores})

# Calculate MSE for each model

mse1 = mean\_squared\_error(y\_test, pd1)

mse3 = mean\_squared\_error(y\_test, pd3)

mse4 = mean\_squared\_error(y\_test, pd4)

mse5 = mean\_squared\_error(y\_test, pd5)

# Calculate RMSE for each model

rmse1 = np.sqrt(mse1)

rmse3 = np.sqrt(mse3)

rmse4 = np.sqrt(mse4)

rmse5 = np.sqrt(mse5)

# Create a DataFrame for the table

data = {

"Models": Models,

"R-squared": Scores,

"MSE": [mse1, mse3, mse4, mse5],

"RMSE": [rmse1, rmse3, rmse4, rmse5]

}

table\_df = pd.DataFrame(data)

To predict average house price in London Borough, diverse regression models has been employed, encompassing linear, tree-based, and nearest neighbors algorithms.

Linear Patterns Captured by Linear Regression:The Linear Regression model, known for its simplicity and interpretability, seeks to establish a linear relationship between the input features and the target variable.

Non-Linear Patterns Captured by Tree-Based and KNN Models:Decision Tree Regressor, Lasso Regression, and K-Nearest Neighbors Regressor explore more complex relationships within the data. Regressor offer alternatives capable of capturing non-linear relationships.

Following the fitting of these models to the pre-processed training data, predictions were generated for the test set.

Model Evaluation Metrics:Evaluation metrics, such as R-squared scores, mean squared error (MSE), and root mean squared error (RMSE), were employed to quantify the models' predictive performance. R-squared scores quantify the proportion of variance explained, while MSE and RMSE gauge prediction errors.

**5.3 Comparative model evaluation** : In 4 model, we have to select model with higher R-squared scores and lower RMSE values.

# Create a DataFrame for the table

data = {

"Models": Models,

"R-squared": Scores,

"MSE": [mse1, mse3, mse4, mse5],

"RMSE": [rmse1, rmse3, rmse4, rmse5]

}

table\_df = pd.DataFrame(data)

# Sort the DataFrame by R-squared in descending order

table\_df = table\_df.sort\_values(by="R-squared", ascending=False)

# Format the table using tabulate

table\_str = tabulate(table\_df, headers='keys', tablefmt='pretty', showindex=False)

# Display the formatted table

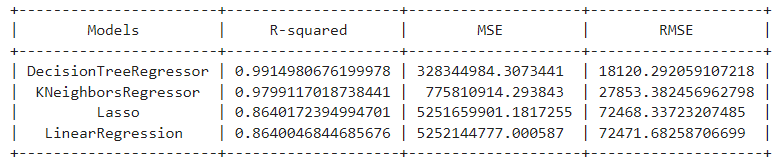
print(table\_str)

# Plot R-squared and RMSE

plt.figure(figsize=[15, 10])

The table is structured to showcase key performance metrics—R-squared scores, mean squared error (MSE), and root mean squared error (RMSE)—across Linear Regression, Decision Tree Regressor, Lasso Regression, and K-Nearest Neighbors Regressor. Sorted in descending order by R-squared scores, the table highlights the effectiveness of each model in capturing the variance in the target variable. R-squared scores provide insight into the proportion of variance explained by the models, while MSE and RMSE quantify the prediction errors.

**Model evaluation Table**



* Decision Tree Regressor Dominance: The Decision Tree Regressor is the the top-performing model, achieving an exceptional R-squared score of 0.991, demonstrating its capability to explain nearly 99.15% of the variance in average property prices in London boroughs.
* Low Prediction Error: The Decision Tree Regressor's impressive performance is further underscored by a relatively low Mean Squared Error (RMSE) of 328,344,984 and a corresponding Root Mean Squared Error (RMSE) of 18,120.29, indicating accurate predictions with minimal error.
* K-Nearest Neighbors (KNN) Robustness: The K-Nearest Neighbors Regressor closely follows, showcasing robust performance with an impressive R-squared score of 0.9799. Despite higher MSE and RMSE values compared to the Decision Tree Regressor, the KNN model performs exceptionally well in capturing the variance in average property prices.
* Lasso and Linear Regression Performance: Lasso Regression and Linear Regression models exhibit slightly lower R-squared scores, suggesting a relatively lower ability to explain variance in comparison to the tree-based models.

Higher Prediction Errors for Lasso and Linear Regression: Lasso Regression and Linear Regression models show substantially higher MSE and RMSE values, indicating a higher level of prediction error compared to the Decision Tree and K-Nearest Neighbors models.

**5.4 Perform hyper parameter tuning** : Hyperparameter tuning is a powerful tool for improving the performance of machine learning models. Hyperparameter tuning is used on our best model Decision Tree Regressor.

# hyperparameter grid

param\_grid = {

'max\_depth': [None, 10, 20, 30],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4]

}

# Create the grid search model

grid\_search = GridSearchCV(DecisionTreeRegressor(), param\_grid, cv=5, scoring='r2')

grid\_search.fit(x\_train, y\_train)

# Get the best parameters

best\_params = grid\_search.best\_params\_

# Fit the model with the best parameters

best\_model = DecisionTreeRegressor(\*\*best\_params)

best\_model.fit(x\_train, y\_train)

The hyperparameter grid spans values for 'max\_depth,' 'min\_samples\_split,' and 'min\_samples\_leaf,' providing a comprehensive exploration of the Decision Tree's parameter space. The GridSearchCV function, coupled with a 5-fold cross-validation scheme and R-squared scoring, facilitates the identification of optimal hyperparameters.

* **Result of Hyper parameter tuning**:



hyperparameter tuning process yielded a set of optimal parameters: {'max\_depth': None, 'min\_samples\_leaf': 2, 'min\_samples\_split': 2}. R-squared value of 0.9911. Factors contribute to the observed performance:

**Unconstrained Tree Depth (max\_depth = None)**: In scikit-learn's DecisionTreeRegressor, the 'max\_depth' parameter controls the maximum depth of the tree. When 'max\_depth' is set to None, it means there is no maximum depth, and the tree will expand until each leaf node contains fewer samples than 'min\_samples\_split' or until all leaves are pure. While this might result in a model that performs well on the training data, it could also lead to overfitting, where the model may not generalize well to new, unseen data.

**Minimum Samples for Split (min\_samples\_split = 2):** The tuning process identified a minimal requirement of only two samples for a node to split further.It creates a detailed and sensitive model, it could also be indicative of an increased risk of overfitting, especially when combined with an unconstrained tree depth.

Minimum Samples per Leaf (min\_samples\_leaf = 2): The optimal choice of 'min\_samples\_leaf' at 2 indicates a preference for smaller leaf sizes. Enhances the model's ability to capture fine-grained patterns but may also make it more susceptible to noise.

**5.5 Comparison of Result with DecisionTreeRegressor (Before Tuning) and Tuned DecisionTreeRegressor :**

R-squared for DecisionTreeRegressor (Before Tuning): **0.9915**

R-squared for Tuned DecisionTreeRegressor: **0.9911**

The decrease in R-squared might be due to various reasons:

**Overfitting**: The tuned model might have learned to fit the training data too closely, leading to a decrease in performance on the test data.

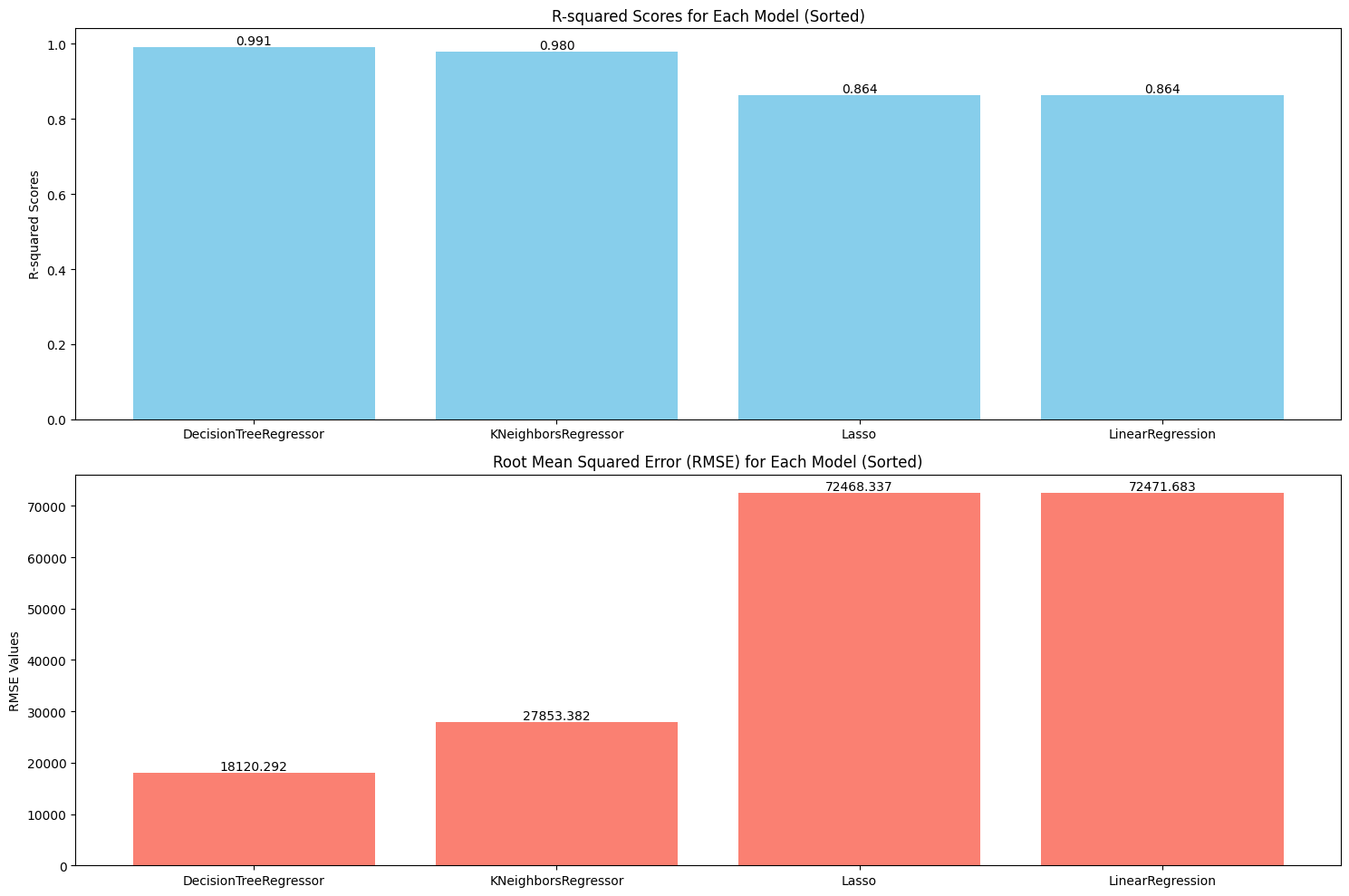
**Noise in Data**: The tuning might have exposed the model to noise in the data, affecting its generalization.

**Complexity**: Increasing the complexity of the model (e.g., increasing max\_depth) could lead to overfitting.

**6: Result (Visualize):**

**6.1 Which is the best model to predict property price in London Borough?**

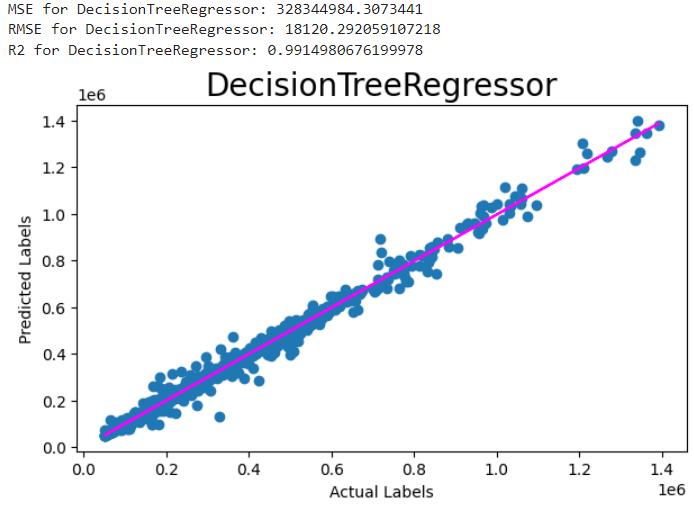
Decision Tree Regressor is the best model to predict average house price in London Borough. Decision Tree Regressor and K-Nearest Neighbors Regressor outperform Lasso and Linear Regression models. Decision Tree Regressor identified as the model of choice for its superior performance.



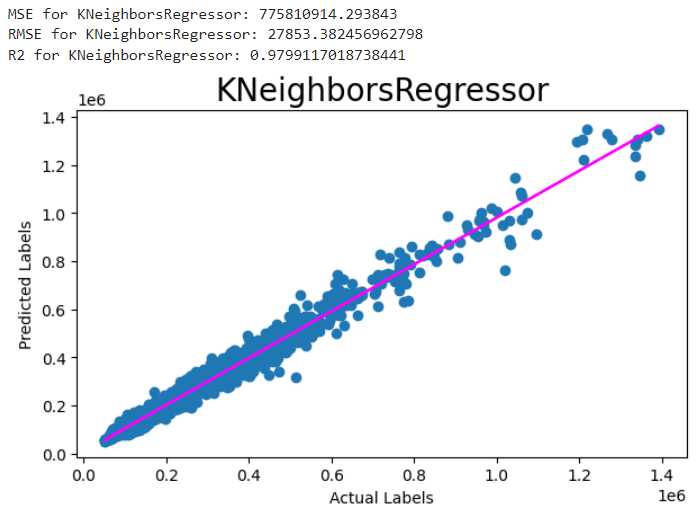
**6.2 Scatter plot for actual and predicted values:**

Decision Tree Regressor Result : Very accurate average property price prediction .The scatter plot for the Decision Tree Regressor reveals a highly clustered pattern of points tightly aligned along the magenta regression line. Distribution signifies a strong correspondence between the actual and predicted values. The minimal dispersion of points around the regression line indicates low variability in the model's predictions.

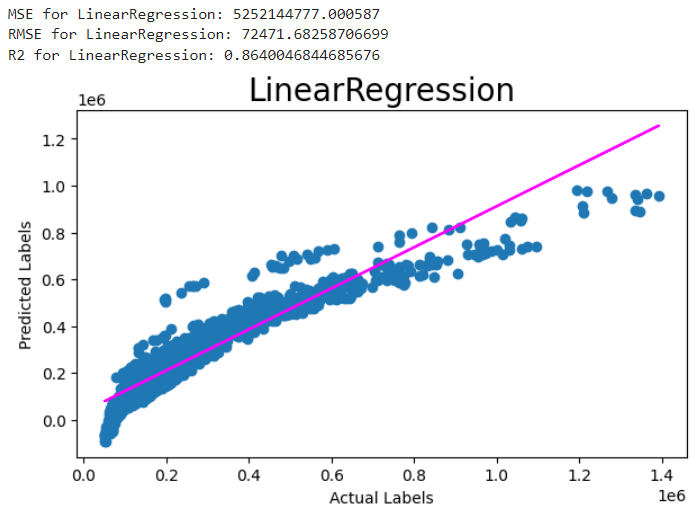
scikit-learn, NumPy, Pandas, and Matplotlib, facilitated the implementation of the model and the generation of insightful visualizations, Scikit-learn provided tools for modeling, while NumPy and Pandas eased data handling, especially for the target variable representing average property prices. Matplotlib enabled the creation of clear visualizations, helping to grasp how well the model predicted property prices, making the entire analysis more accessible and insightful.



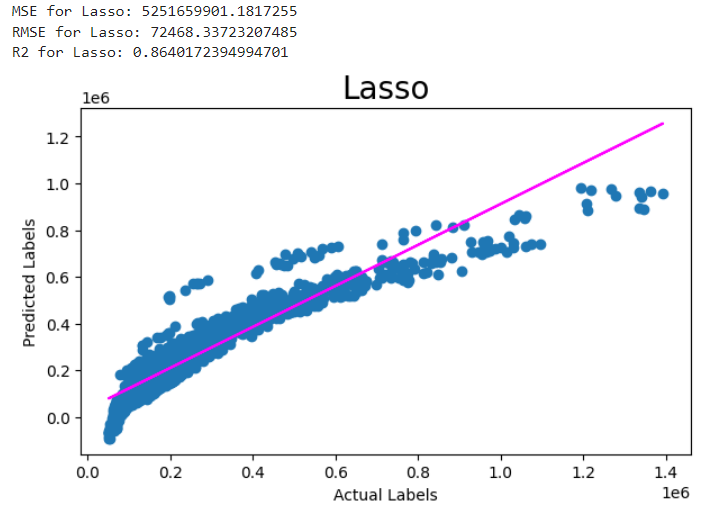
**K-Nearest Neighbors Regression**: The plot suggests that the K-Nearest Neighbors model performs well in predicting the House price due to its non-linear relationships and clustering method.



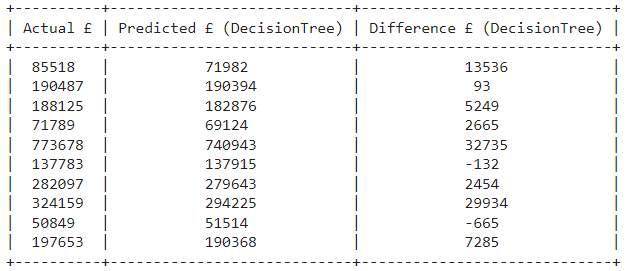
**Linear Regression**: There is a variability between actual and predicted price. It has not predicted accurately when prices were high



**Lasso Regression**: There is a variability between actual and predicted price. Similar to the Linear Regression plot, the Lasso Regression scatter plot exhibits a linear relationship between actual and predicted values.



**6.3** **Table for Actual and predicted house prices.**

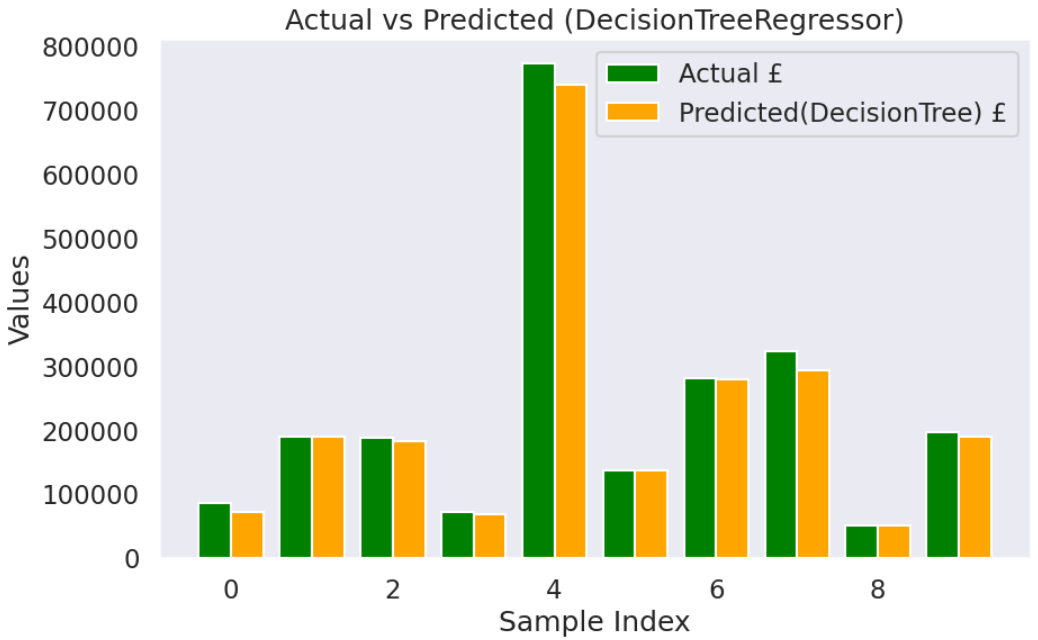


Quick assessment of the predictive accuracy and understanding of the disparities between predicted prices and actual property prices in the London Borough dataset.

Table represents the Actual house price in first column, model predicted values in 2nd column. The "Difference £ (DecisionTree)" column represents the disparity between actual and predicted prices for the DecisionTreeRegressor.

**6.4 Comparison of Actual and Predicted Property Prices (DecisionTreeRegressor):**

Bar plot visually compares the actual and predicted values for the DecisionTreeRegressor for the first 10 samples in your dataset. The green bars represent the actual values, and the orange bars represent the predicted values. Predicted values aligns with the actual values for each sample.



**7.Insights**

1. **Borough Area Analysis**: The exploration of specific borough urban areas, particularly the Horizontal Bar Graph showing expensive areas, sheds light on the substantial variations in average house prices across different regions. Kensington and Chelsea emerge as notably expensive areas, emphasizing the importance of location in influencing property values.
2. **Model Performance and Selection**: The comparative model evaluation section underscores the dominance of the Decision Tree Regressor in predicting average house prices, outperforming other models. The emphasis on R-squared scores and low prediction errors contributes to the informed selection of the Decision Tree Regressor as the model of choice.
3. **High Accuracy and Precision**: The integration of scikit-learn, NumPy, Pandas, and Matplotlib played a crucial role in implementing the Decision Tree Regressor model, enabling insightful visualizations. The scatter plot for the Decision Tree Regressor illustrates a strong correspondence between actual and predicted values. This alignment signifies a high level of accuracy, showcasing the model's proficiency in predicting average property prices.
4. **Consistency and Capacity to Capture details**: Both the scatter plot and the graph depicting actual and predicted values highlight the model's consistency and capacity to capture detailed relationships within the dataset. The close alignment of points in these visualizations indicates a consistent pattern in the model's predictions, showcasing its ability to navigate the complexity of the data and provide accurate forecasts across a spectrum of property prices.
5. **Temporal Trends**: The analysis explores temporal trends by examining the relationship between average property prices and the variable 'Year.' The identified positive correlation (R-squared of 0.66) indicates a consistent increase in average prices over the years, reflecting the dynamic nature of the real estate market. Line graph showed us the remarkable increase in average house prices as year goes on. From 1995 to 2020, house prices in London boroughs went up a lot, from about £82,000 to around £528,000 in 25 years. Prices vary a lot among boroughs, starting at £50,000 and going up to about £1.5 million.

Outside of London, house prices went up more slowly, from £57,000 to about £290,000, which is almost 1.8 times less than in London boroughs

1. **Crime Analysis**: The identification of the top 10 crime areas using a horizontal bar graph, highlighting areas with higher crime rates. The key insights from the data are that Westminster, Lambeth, and Camden are the areas with the highest number of crimes. Specifically, Westminster has 1,592,695 reported crimes over the specified period. This information could be valuable for individuals considering property purchases who also prioritize safety.
2. **Crime Rates and Housing**: The correlation analysis reveals interesting relationships between average property prices and other variables. For instance, the positive correlation (0.25) between average prices and the number of crimes suggests a nuanced interaction between housing dynamics and the safety perception of an area.
3. **Temporal Changes in Houses Sold**: The analysis of houses sold over time, visualized through a horizontal bar graph, provides insights into real estate activity. Notably, Wandsworth, Bromley, and Croydon emerge as areas with consistently high numbers of houses sold, reflecting vibrant real estate markets. City of London has a comparatively lower total of 7,636 houses sold from 1995 to 2020. City of London had the fewest houses sold among the London boroughs. It might be of small geographic area and a higher concentration of commercial properties

**References**

Cirtautas, J. 2020, 'Housing in London', Kaggle, Available at: https://www.kaggle.com/datasets/justinas/housing-in-london/data (Accessed: 29 April 2020).

Wikipedia contributors. (2023, October 20, at 17:27 UTC). "London boroughs." Wikipedia, The Free Encyclopedia. Retrieved from https://en.wikipedia.org/wiki/London\_boroughs on October 20, 2023.

Name: Kavyashree

Email:kavy012@gmail.com