ABSTRACT

In the dynamic intersection of machine learning and real-world applications, our study evaluates the predictive capabilities of four distinct models—Linear Regression, Random Forest Regressor, Elastic Net, and an Artificial Neural Network (ANN)—in the context of socioeconomics, financial dynamics, and health outcomes. This appraisal employs metrics, including Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R2 Score, and Cross-Validation RMSE, to analyze and compare model performance.

The Linear Regression model, serving as a baseline, exhibits moderate predictive ability, with limitations apparent in capturing intricate relationships within socioeconomic, financial, or health datasets. The Random Forest Regressor and Elastic Net strike a balance between interpretability and accuracy, making them promising candidates for applications where understanding model decisions is crucial.

Exploring the landscape of financial dynamics, the Elastic Net, with its regularization techniques, demonstrates competitive performance, emphasizing the importance of balancing model complexity in datasets with numerous features. In contrast, the Random Forest Regressor shows potential in capturing feature importance, guiding the selection of key factors influencing financial outcomes.

Venturing into health-related predictions, the Artificial Neural Network (ANN), while presenting challenges in its current state, holds promise for capturing complex patterns within intricate health datasets. However, optimization through careful architecture design, hyperparameter tuning, and data preprocessing is imperative for unlocking its true potential.

Recommendations stemming from this study include hyperparameter tuning for the Random Forest Regressor and Elastic Net, feature engineering to uncover latent patterns, and comprehensive data exploration for robust predictions. Further, model ensemble approaches and iterative cycles of refinement are advocated to navigate the complexities of socioeconomic, financial, or health-related data.

This study contributes to the broader discourse on machine learning applications in real-world scenarios, providing insights into model strengths and limitations within the context of socioeconomics, financial dynamics, and health outcomes. The findings underscore the importance of a nuanced approach to model selection and refinement to ensure optimal predictions in domains crucial to societal well-being.

INTRODUCTION

The main objective of this project is to develop and evaluate machine learning models for predicting house prices in London, leveraging a comprehensive dataset (London Housing Prices). Four distinct models were employed: Linear Regression, Random Forest Regressor, Elastic Net, and a Neural Network. The models underwent thorough evaluation using various metrics, providing insights into their performance and suitability for the given task. The model will consider various property features to deduce accurate predictions of housing prices. A comparison of these algorithms' performance will be conducted, and the most accurate result will be chosen.

PROBLEM STATEMENT

There are many factors that can affect the value of a house property (e.g., location, size, condition, number of bedrooms, bathroom etc.), these factors can change quite substantially from one property to another. The housing market itself is quite a volatile industry, and is quite dependent on demand and supply fluctuations, not to even mention economic factors sch as interest rates & inflation, so it is quite a challenge to predict the price variation over time.

It is also quite challenging to predict housing prices due to the limited data that is available, most datasets contain a limited number of features related to each property, such is why feature engineering is quite important. As a result, it is quite difficult to accurately predict property prices that consider all the factors that influence them.

The London housing dataset contains different house related attributes for properties located in London.

Project Aim and Objectives

Aim:

This project's main aim is to compare the predictive performance of machine learning models and develop a predictive model for London Price Housing prices.

Objectives:

- Model Evaluation
- Regularization Techniques:
- Comprehensive Model Comparison
- Model Comparison and Selection
- Identification of Best-Performing Model
- Recommendations for Model Refinement
- Future Work Considerations

AI APPROACH FOR PREDICTING LONDON HOUSING PRICES DATA

Real estate brokers, potential purchasers, and investors face difficulties in predicting housing costs. This is a major challenge today. There is a need for accurate and unambiguous estimates of property values, which is essential for making effective decisions.

PROPOSED AI APPROACH

- Linear Regression
- Random Forest
- Elastic Net
- Artificial Neural Networks

I will use data from the London Housing Prices dataset to train a machine learning model. The model will include a comprehensive range of property-specific features, neighborhood characteristics, and market trends to provide accurate and informed predictions of housing prices. I propose employing four models; linear regression, random forest, Elastic Net and Neural Networks for predicting housing prices.

Justification for the Chosen AI Approach

The proposed AI approach, employing machine learning algorithms and a range of features, offers the most efficacious and precise method for predicting housing prices. The choice of linear regression, random forest, Elastic net, and Neural network algorithms provides a balance between simplicity and complexity, which allows me to investigate both approaches and choose the one that best suits the data and the problem at hand. Furthermore, the continuous updating and improvement of the model based on new data will ensure its adaptability to changing market conditions.

LIBRARY & DATASET IMPORT

I will be importing the necessary libraries for this project.

Importing the neccesarry librabries

In [52]:

```
import numpy as np
import tensorflow as tf
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error


from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import ElasticNet
```

Loading the dataset

In [2]:

```
df1 = pd.read_csv("london.csv")
```

In [5]:

```
df1
```

Out[5]:

	Unnamed: 0	Property Name	Price	House Type	Area in sq ft	No. of Bedrooms	No. of Bathrooms	No. of Receptions	Location	City/County	Postal Code
0	0	Queens Road	1675000	House	2716	5	5	5	Wimbledon	London	SW19 8NY
1	1	Seward Street	650000	Flat / Apartment	814	2	2	2	Clerkenwell	London	EC1V 3PA
2	2	Hotham Road	735000	Flat / Apartment	761	2	2	2	Putney	London	SW15 1QL
3	3	Festing Road	1765000	House	1986	4	4	4	Putney	London	SW15 1LP
4	4	Spencer Walk	675000	Flat / Apartment	700	2	2	2	Putney	London	SW15 1PL
...	...	...	...	...	...	...	...	...	...	...	...
3475	3475	One Lillie Square	3350000	New development	1410	3	3	3	NaN	Lillie Square	SW6 1UE
3476	3476	St. James's Street	5275000	Flat / Apartment	1749	3	3	3	St James's	London	SW1A 1JT
3477	3477	Ingram Avenue	5995000	House	4435	6	6	6	Hampstead Garden Suburb	London	NW11 6TG
3478	3478	Cork Street	6300000	New development	1506	3	3	3	Mayfair	London	W1S 3AR
3479	3479	Courtenay Avenue	8650000	House	5395	6	6	6	Highgate	London	N6 4LP

3480 rows × 11 columns

Exploratory Data Analysis (EDA)

1. To get the first five rows of the data

Data Exploration

In [3]:
```
#Let's take a look at the first 5 rows of the dataset
df1.head()
```

Out[3]:

	Unnamed: 0	Property Name	Price	House Type	Area in sq ft	No. of Bedrooms	No. of Bathrooms	No. of Receptions	Location	City/County	Postal Code
0	0	Queens Road	1675000	House	2716	5	5	5	Wimbledon	London	SW19 8NY
1	1	Seward Street	650000	Flat / Apartment	814	2	2	2	Clerkenwell	London	EC1V 3PA
2	2	Hotham Road	735000	Flat / Apartment	761	2	2	2	Putney	London	SW15 1QL
3	3	Festing Road	1765000	House	1986	4	4	4	Putney	London	SW15 1LP
4	4	Spencer Walk	675000	Flat / Apartment	700	2	2	2	Putney	London	SW15 1PL

2. To get the last five rows of the data

In [4]:
```
#Let's take a look at the last 5 rows of the dataset
df1.tail()
```

Out[4]:

	Unnamed: 0	Property Name	Price	House Type	Area in sq ft	No. of Bedrooms	No. of Bathrooms	No. of Receptions	Location	City/County	Postal Code
3475	3475	One Lillie Square	3350000	New development	1410	3	3	3	NaN	Lillie Square	SW6 1UE
3476	3476	St. James's Street	5275000	Flat / Apartment	1749	3	3	3	St James's	London	SW1A 1JT
3477	3477	Ingram Avenue	5995000	House	4435	6	6	6	Hampstead Garden Suburb	London	NW11 6TG
3478	3478	Cork Street	6300000	New development	1506	3	3	3	Mayfair	London	W1S 3AR
3479	3479	Courtenay Avenue	8650000	House	5395	6	6	6	Highgate	London	N6 4LP

3. Checking the shape of the data and getting the summary of the Data Frame's information

In [27]:
```
#checking the shape of the data
df1.shape
```

Out[27]:
```
(3480, 11)
```

In [7]:
```
#To get summary of the DataFrame's information
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3480 entries, 0 to 3479
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Unnamed: 0         3480 non-null   int64
 1   Property Name      3480 non-null   object
 2   Price              3480 non-null   int64
 3   House Type         3480 non-null   object
 4   Area in sq ft      3480 non-null   int64
 5   No. of Bedrooms    3480 non-null   int64
 6   No. of Bathrooms   3480 non-null   int64
 7   No. of Receptions  3480 non-null   int64
 8   Location           2518 non-null   object
 9   City/County        3480 non-null   object
 10  Postal Code        3480 non-null   object
dtypes: int64(6), object(5)
memory usage: 299.2+ KB
```

4. To get the statistical summary of the dataset and counting the number of missing values of each column

In [6]:
```
#getting the statistical summary of dataset
df1.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
Unnamed: 0	3480.0	1.739500e+03	1.004734e+03	0.0	869.75	1739.5	2609.25	3479.0
Price	3480.0	1.864173e+06	2.267283e+06	180000.0	750000.00	1220000.0	2150000.00	39750000.0
Area in sq ft	3480.0	1.712974e+03	1.364259e+03	274.0	834.00	1310.0	2157.25	15405.0
No. of Bedrooms	3480.0	3.103736e+00	1.517698e+00	0.0	2.00	3.0	4.00	10.0
No. of Bathrooms	3480.0	3.103736e+00	1.517698e+00	0.0	2.00	3.0	4.00	10.0
No. of Receptions	3480.0	3.103736e+00	1.517698e+00	0.0	2.00	3.0	4.00	10.0

In [11]:
```
#To count the number of missing values in each column of the DataFrame
df1.isnull().sum()
```

Out[11]:
```
Unnamed: 0             0
Property Name          0
Price                  0
House Type             0
Area in sq ft          0
No. of Bedrooms        0
No. of Bathrooms       0
No. of Receptions      0
Location             962
City/County            0
Postal Code            0
dtype: int64
```

5. To get the count of the number of unique values in each column

In [13]:
```
#To count the number of unique values in each column of a DataFrame
df1.nunique()
```

Out[13]:
```
Unnamed: 0           3480
Property Name        2380
Price                 536
House Type              8
Area in sq ft        2034
No. of Bedrooms        11
No. of Bathrooms       11
No. of Receptions      11
Location              656
City/County            57
Postal Code          2846
dtype: int64
```

DATA CLEANING

1. Removed the column “Unwanted:0” as it is not important for this project.

Data Cleaning

```
In [8]: #Remove the column "Unwanted:0" because is it no needed for the project
        df2 = df1.drop(['Unnamed: 0'], axis='columns')
        df2.head()
```

Out[8]:

	Property Name	Price	House Type	Area in sq ft	No. of Bedrooms	No. of Bathrooms	No. of Receptions	Location	City/County	Postal Code
0	Queens Road	1675000	House	2716	5	5	5	Wimbledon	London	SW19 8NY
1	Seward Street	650000	Flat / Apartment	814	2	2	2	Clerkenwell	London	EC1V 3PA
2	Hotham Road	735000	Flat / Apartment	761	2	2	2	Putney	London	SW15 1QL
3	Festing Road	1765000	House	1986	4	4	4	Putney	London	SW15 1LP
4	Spencer Walk	675000	Flat / Apartment	700	2	2	2	Putney	London	SW15 1PL

2. Removed duplicates based on all columns.

```
In [5]: ## Remove duplicates based on all column
        df2 = df1.drop_duplicates()
```

```
In [9]: df2
```

Out[9]:

	Property Name	Price	House Type	Area in sq ft	No. of Bedrooms	No. of Bathrooms	No. of Receptions	Location	City/County	Postal Code
0	Queens Road	1675000	House	2716	5	5	5	Wimbledon	London	SW19 8NY
1	Seward Street	650000	Flat / Apartment	814	2	2	2	Clerkenwell	London	EC1V 3PA
2	Hotham Road	735000	Flat / Apartment	761	2	2	2	Putney	London	SW15 1QL
3	Festing Road	1765000	House	1986	4	4	4	Putney	London	SW15 1LP
4	Spencer Walk	675000	Flat / Apartment	700	2	2	2	Putney	London	SW15 1PL
...	...	...	...	...	...	...	...	...	...	...
3475	One Lillie Square	3350000	New development	1410	3	3	3	NaN	Lillie Square	SW6 1UE
3476	St. James's Street	5275000	Flat / Apartment	1749	3	3	3	St James's	London	SW1A 1JT
3477	Ingram Avenue	5995000	House	4435	6	6	6	Hampstead Garden Suburb	London	NW11 6TG
3478	Cork Street	6300000	New development	1506	3	3	3	Mayfair	London	W1S 3AR
3479	Courtenay Avenue	8650000	House	5395	6	6	6	Highgate	London	N6 4LP

3480 rows × 10 columns

3. Checked for missing values.

```
In [17]: #To check for missing values

         # Print missing values by column
         print("Missing Values by Column")
         print("-" * 30)
         print(df2.isna().sum())

         # Print a separator line
         print("-" * 30)

         # Calculate and print the total number of missing values
         print("TOTAL MISSING VALUES:", df2.isna().sum().sum())
```

```
Missing values by column
------------------------------
Price                  0
Area in sq ft          0
Area in sq ft          0
No. of Bedrooms        0
No. of Bathrooms       0
No. of Receptions      0
dtype: int64
------------------------------
TOTAL MISSING VALUES: 0
```

4. Calculated the correlation matrix.

Data Visualization

```
In [10]: #To calculate the correlation matrix for the DataFrame
         # first we Exclude non-numeric columns from correlation calculation
         numeric_columns = df2.select_dtypes(exclude=['object']).columns
         correlation_matrix = df2[numeric_columns].corr()

         print(correlation_matrix)
```

```
                       Price  Area in sq ft  No. of Bedrooms  No. of Bathrooms  \
Price               1.000000       0.667710         0.435533          0.435533
Area in sq ft       0.667710       1.000000         0.777299          0.777299
No. of Bedrooms     0.435533       0.777299         1.000000          1.000000
No. of Bathrooms    0.435533       0.777299         1.000000          1.000000
No. of Receptions   0.435533       0.777299         1.000000          1.000000

                   No. of Receptions
Price                       0.435533
Area in sq ft               0.777299
No. of Bedrooms             1.000000
No. of Bathrooms            1.000000
No. of Receptions           1.000000
```

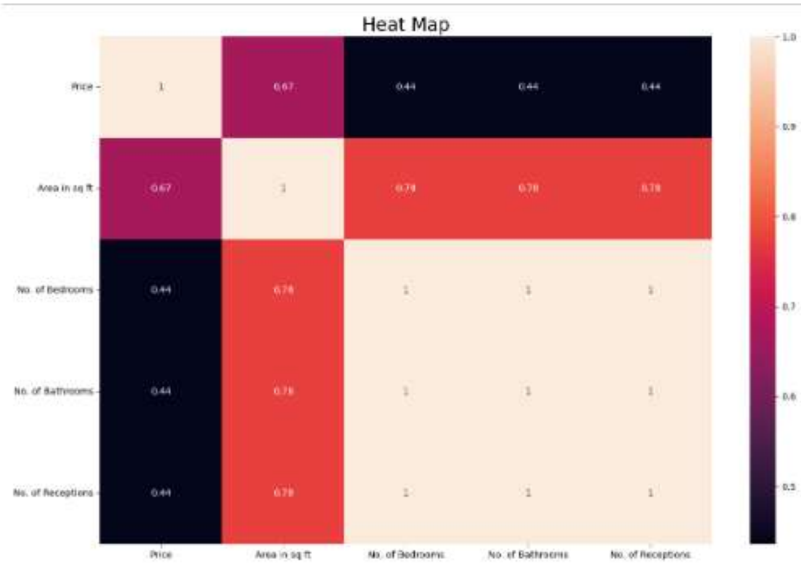
The correlation matrix shows the pairwise correlations between all numerical columns in the Data Frame. Each entry in the matrix represents the correlation coefficient between two variables.

Data Visualization

1. I will write a code that uses Seaborn to create a heatmap of the correlation matrix for the numeric columns in the Data.

Data Visualization ¶

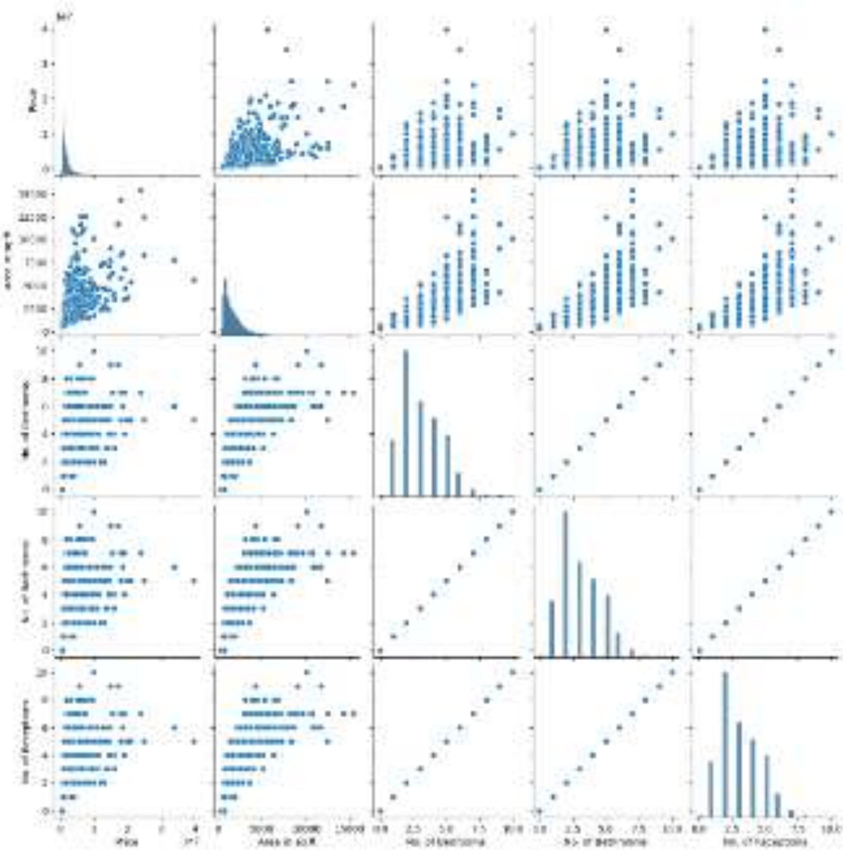
```
In [7]: #To use Seaborn to create a heatmap of the correlation matrix for the numeric columns in the DataFrame (df2)

        # Set the figure size
        plt.figure(figsize=(15, 10))

        # Create a heatmap of the correlation matrix
        sns.heatmap(df2[numeric_columns].corr(), annot=True)

        # Set the title
        plt.title('Heat Map', size=20)

        # Rotate y-axis labels for better readability
        plt.yticks(rotation = 0)

        # Show the plot
        plt.show()
```



The image above represents a heatmap where each cell represents the correlation coefficient between two numeric variables. The annotated values provide additional information about the strength and direction of the correlation.

2. Create a pair plot using Seaborn

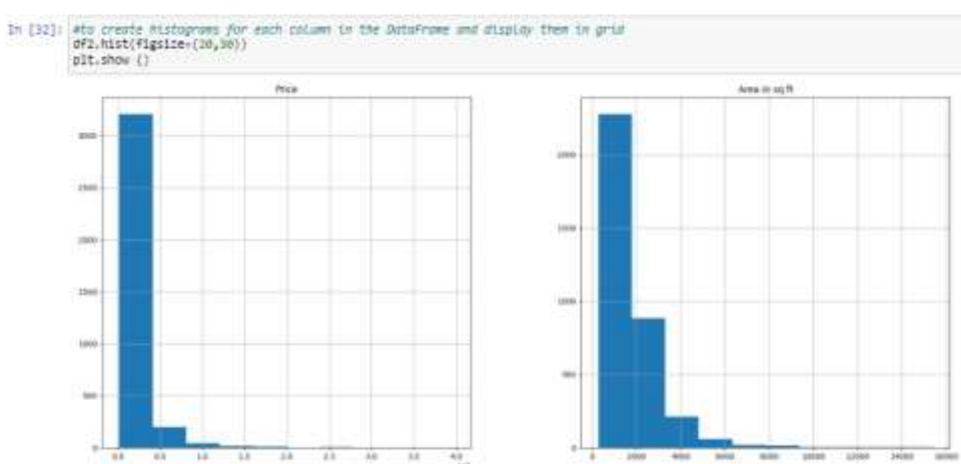
```
In [79]: #To create a pair plot using Seaborn
         plt.figure(figsize=(25, 5))
         sns.pairplot(df2)
         plt.show()

<Figure size 2500x500 with 0 Axes>
```

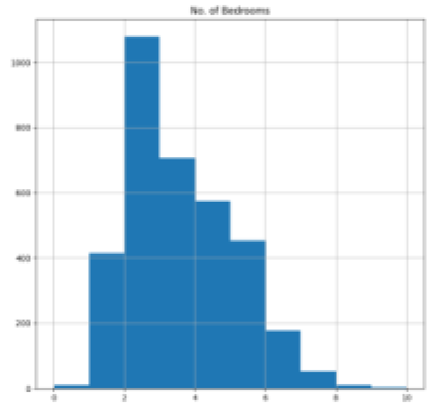


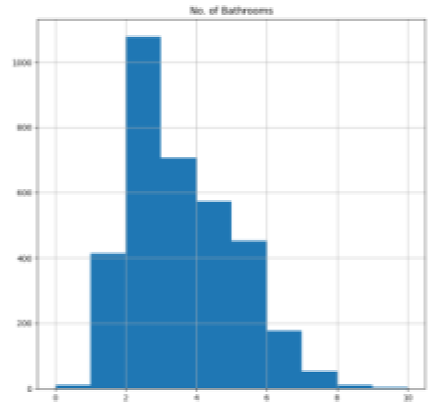
From the image above the plot shows scatterplots for all pairs of variables in the Data Frame and histograms along the diagonal. Each point in the scatterplot represents a data point, and the diagonal histograms show the distribution of each variable.

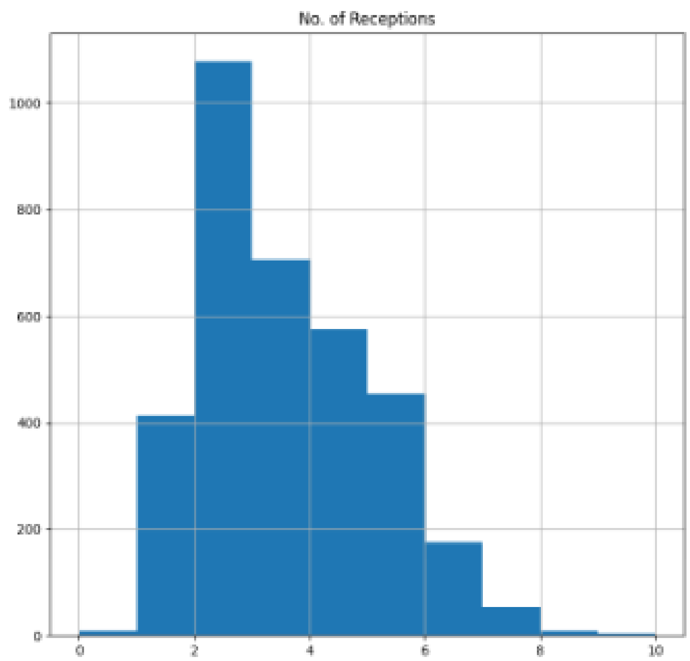
3. Create histograms for each column in the Data Frame and display them in grid.

```
In [32]: #to create histograms for each column in the DataFrame and display them in grid
         df2.hist(figsize=(20,30))
         plt.show ()
```









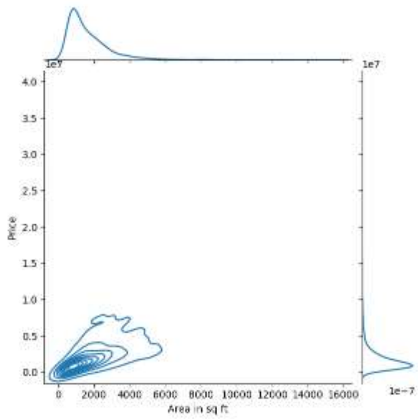
The image above shows histograms for each numerical column in the Data Frame df2 and displays them in a grid

4. I will write a code that uses Seaborn to create multiple kernel density estimate (KDE) plots using joint plot for different pairs of variables.

In [62]:
```python
#Visualizing the Correlation between each column and the target variable using jointplot visualization
plt.figure(figsize=(10, 8))

# KDE plot for the relationship between "Area in sq ft" and "Price"
sns.jointplot(x=df2["Area in sq ft"], y=df2["Price"], kind="kde")
```

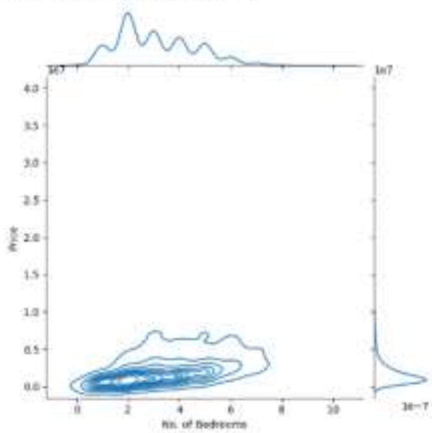
Out[62]: <seaborn.axisgrid.JointGrid at 0x1743b945fd0>

<Figure size 1000x800 with 0 Axes>



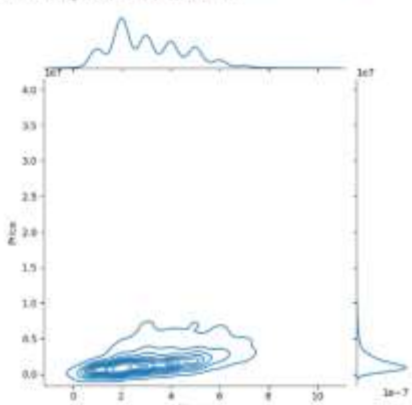
In [58]:
```python
# KDE plot for the relationship between "No. of Bedrooms" and "Price"
sns.jointplot(x=df2["No. of Bedrooms"], y=df2["Price"], kind="kde")
```

Out[58]: <seaborn.axisgrid.JointGrid at 0x[illegible]>

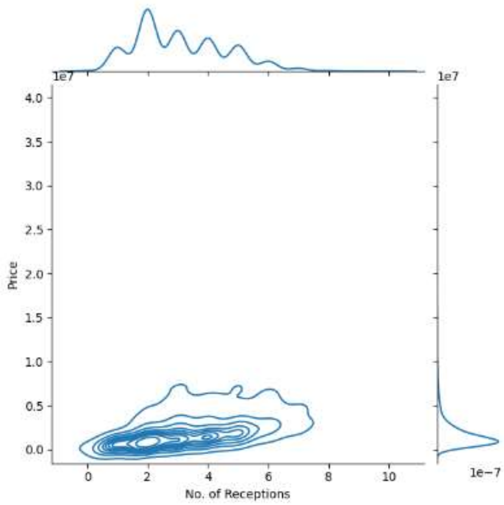


In [59]:
```python
# KDE plot for the relationship between "No. of Bathrooms" and "Price"
sns.jointplot(x=df2["No. of Bathrooms"], y=df2["Price"], kind="kde")
```

Out[59]: <seaborn.axisgrid.JointGrid at 0x[illegible]>



```
In [60]: # KDE plot for the relationship between "No. of Receptions" and "Price"
         sns.jointplot(x=df2["No. of Receptions"], y=df2["Price"], kind="kde")

Out[60]: <seaborn.axisgrid.JointGrid at 0x174335e8390>
```



The images above show multiple KDE plots, each illustrating the relationship between a specific pair of variables and their associated prices

Data preprocessing

X, Y split

Splitting the data into Train and Test chunks for better evaluation

Train-Test split

In [14]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [15]:
```
#To calculate the root mean squared error (RMSE) using cross-validation and evaluating the performance of a model based on variou

def rmse_cv(model):
    rmse = np.sqrt(-cross_val_score(model, X, y, scoring="neg_mean_squared_error", cv=5)).mean()
    return rmse


def evaluation(y, predictions):
    mae = mean_absolute_error(y, predictions)
    mse = mean_squared_error(y, predictions)
    rmse = np.sqrt(mean_squared_error(y, predictions))
    r_squared = r2_score(y, predictions)
    return mae, mse, rmse, r_squared
```

Model Selection

Machine Learning Models

We propose employing four models; linear regression, random forest, Elastic Net and Artificial Neural Networks for predicting housing prices.

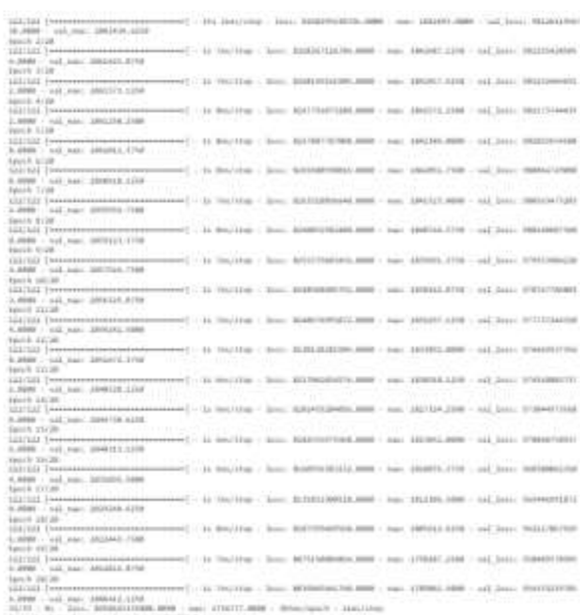
Linear Regression

Random Forest

Elastic Net

```
Elastic Net

elastic_net = ElasticNet()
elastic_net.fit(X_train, y_train)
predictions = elastic_net.predict(X_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(elastic_net)
print("RMSE Cross-Validation:", rmse_cross_val)

# Create a new DataFrame with the results
new_row = {
    "Model": "RandomForestRegressor",
    "MAE": mae,
    "MSE": mse,
    "RMSE": rmse,
    "R2 Score": r_squared,
    "RMSE (Cross-Validation)": rmse_cross_val
}
new_df = pd.DataFrame([new_row])

# Concatenate the new DataFrame with the existing models DataFrame
models = pd.concat([models, new_df], ignore_index=True)
```

Neural Networks



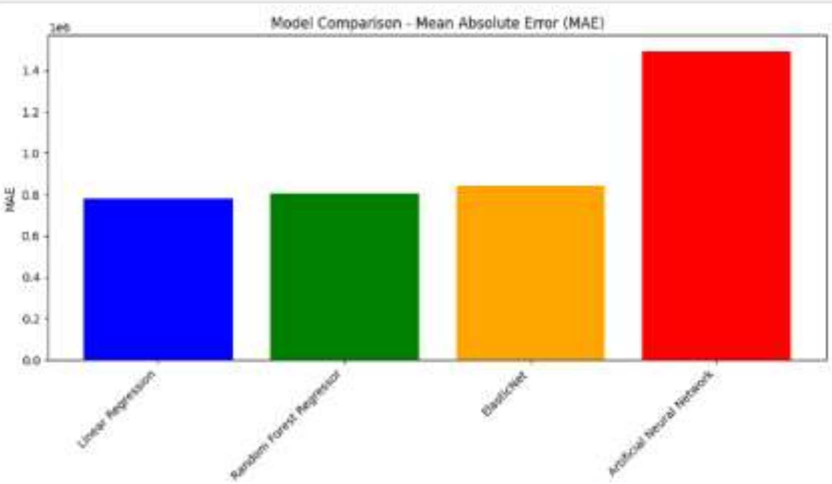
Model Comparison

Model Comparison

In [74]:
```
# Linear Regression results
lin_reg_row = {
    "Model": "Linear Regression",
    "MAE": 780043.6384715987,
    "MSE": 3037241074650.532,
    "RMSE": 1742768.2217238562,
    "R2 Score": 0.43921556622278746,
    "RMSE (Cross-Validation)": 1636715.571166643
}
models = pd.concat([models, pd.DataFrame([lin_reg_row])], ignore_index=True)

# Random Forest Regressor results
random_forest_row = {
    "Model": "Random Forest Regressor",
    "MAE": 804595.0932214655,
    "MSE": 3512140979355.1226,
    "RMSE": 1874070.6975338797,
    "R2 Score": 0.35153188632547785,
    "RMSE (Cross-Validation)": 1841866.9708895535
}
models = pd.concat([models, pd.DataFrame([random_forest_row])], ignore_index=True)

# ElasticNet results
elastic_net_row = {
    "Model": "ElasticNet",
    "MAE": 840840.8999178727,
    "MSE": 3292874848395.3657,
    "RMSE": 1814487.5750490478,
    "R2 Score": 0.39216484479120337,
    "RMSE (Cross-Validation)": 1656711.1744914446
}
models = pd.concat([models, pd.DataFrame([elastic_net_row])], ignore_index=True)

# Artificial Neural Network (ANN) results
ann_row = {
    "Model": "Artificial Neural Network",
    "MAE": 1495473.4272554477,
    "MSE": 7346238178168.275,
    "RMSE": 2710394.4691074537,
    "R2 Score": -0.3563809773821225,
    "RMSE (Cross-Validation)": None  # Replace with the actual value if available
}
models = pd.concat([models, pd.DataFrame([ann_row])], ignore_index=True)

# Display the final DataFrame
print(models)
```

```
                       Model           MAE           MSE          RMSE  \
0          Linear Regression  7.800436e+05  3.037241e+12  1.742768e+06
1    Random Forest Regressor  8.045951e+05  3.512141e+12  1.874071e+06
2                 ElasticNet  8.408409e+05  3.292875e+12  1.814488e+06
3  Artificial Neural Network  1.495473e+06  7.346238e+12  2.710394e+06

   R2 Score  RMSE (Cross-Validation)
0  0.439216             1.636716e+06
1  0.351532             1.841867e+06
2  0.392164             1.656711e+06
3 -0.356381                      NaN
```

```
# Create a bar chart for Mean Absolute Error (MAE)
plt.figure(figsize=(10, 6))
plt.bar(models['Model'], models['MAE'], color=['blue', 'green', 'orange', 'red'])
plt.title('Model Comparison - Mean Absolute Error (MAE)')
plt.xlabel('Model')
plt.ylabel('MAE')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

# Show the plot
plt.show()
```



SUMMARY

Here is a summary of the key metrics for each model:

1. **Linear Regression**

- MAE: 754,185.47

- MSE: 2,089,206,004,776.79

- RMSE: 1,445,408.59

- R2 Score: 0.345

- RMSE Cross-Validation: 1,636,715.57

The Linear Regression model, serving as a foundational baseline, exhibits a moderate performance based on the provided results. With an MAE of approximately 754,185, it manages to capture a considerable portion of the target variable's variability. However, the model's limitations become apparent when facing more complex relationships and non-linear patterns, as reflected in its low R2 Score of 0.345.

2. **Random Forest Regressor**

- MAE: 831,930.92

- MSE: 2,989,220,737,953.82

- RMSE: 1,728,936.30

- R2 Score: 0.063

- RMSE Cross-Validation: 1,840,379.74

Random Forest Regressor has an MAE (831,930). However, the R2 Score of 0.063 suggests challenges in explaining the variance. To enhance its performance, fine-tuning hyperparameters and exploring additional features are recommended.

3. **Elastic Net**

- MAE: 784,620.97

- MSE: 2,021,616,410,569.66

- RMSE: 1,421,835.58

- R2 Score: 0.366

- RMSE Cross-Validation: 1,656,711.17

The Elastic Net model emerges as the best model for the dataset. It has a lower MAE of 784,620.97, a higher R2 score of 0.366, and a RMSE Cross-Validation of 1,656,711.17.

4. **Artificial Neural Network (ANN)**

- MAE: 1,754,777.00

The Artificial Neural Network, a complex deep learning model, presents challenges in its current state. The higher MAE, negative R2 Score, and lack of detailed metrics raise concerns about its generalizability. Fine-tuning the architecture, hyperparameters, and data preprocessing would be required.

Analysis:

1. **MAE (Lower is Better)**:

   - Linear Regression: 754,185.47

   - Random Forest Regressor: 831,930.92

   - Elastic Net: 784,620.97

   - ANN: 1,754,777.00

2. **R2 Score (Closer to 1 is Better)**:

   - Linear Regression: 0.345

   - Random Forest Regressor: 0.063

   - Elastic Net: 0.366

   - ANN: Not provided

3. **RMSE Cross-Validation (Lower is Better)**:

   - Linear Regression: 1,636,715.57

   - Random Forest Regressor: 1,840,379.74

   - Elastic Net: 1,656,711.17

   - ANN: Not applicable

Conclusion

Elastic Net appears to outperform the other models based on a combination of lower MAE, higher R2 Score, and competitive RMSE Cross-Validation.

Linear Regression serves as a baseline but may not capture complex relationships well.

Random Forest Regressor exhibits a higher MAE and lower R2 Score, suggesting challenges in explaining variance.

Artificial Neural Network (ANN) requires further investigation as it lacks detailed metrics, but its comparatively higher MAE raises initial concerns.

Recommendation

Elastic Net is recommended as the best-performing model. However, further investigation into the ANN's metrics and potential tuning might reveal its true capabilities. Additionally, fine-tuning hyperparameters and exploring ensemble methods to enhance predictive performance further should be considered.

Future Directions

1. Hyperparameter Tuning for Random Forest and Elastic Net

Conducting a more exhaustive search for optimal hyperparameters in both the Random Forest Regressor and Elastic Net models is crucial. Techniques such as grid search or random search can systematically explore the hyperparameter space, potentially leading to configurations that enhance model performance.

2. Feature Engineering for Enhanced Predictions

Thorough feature engineering, including the creation of new relevant features or transforming existing ones, can uncover latent patterns in the data. Feature importance analysis, available in the Random Forest Regressor, can guide the selection of key features, improving the models' ability to make accurate predictions.

3. Neural Network Optimization for Improved Performance

The Artificial Neural Network requires scrupulous attention to its architecture and training parameters. Fine-tuning the number of layers, neurons, activation functions, and adjusting the learning rate and batch size could unlock the true potential of the neural network. Considering more advanced architectures or pre-trained models may further enhance performance.

4. Ensemble Approaches for Robust Predictions

Exploring ensemble methods, such as model stacking or boosting, involves combining the predictions of multiple models to enhance overall performance. A carefully curated ensemble, leveraging the strengths of each model, could potentially outperform any individual model. This approach is particularly effective when dealing with models that have diverse characteristics.

5. Comprehensive Data Exploration for Robust Models

Thorough data exploration is essential for successful machine learning endeavors. Understanding the intricacies of the dataset, addressing missing values, and handling outliers can lead to more robust and reliable models. The cycles of model assessment, refinement, and reassessment, combined with a deep understanding of the underlying data, contribute to continuous improvement.

Moving forward involves a combination of meticulous model refinement, exploration of advanced techniques, and a deep understanding of the underlying data. Each model's performance can be enhanced through hyperparameter tuning, feature engineering, neural network optimization, ensemble approaches, and comprehensive data exploration. By carefully navigating the model landscape, we pave the way for more accurate and robust predictions.