SC1015 MINI PROJECT
DATA SCIENCE & ARTIFICIAL INTELLIGENCE

# Singapore HDB Resale Price Prediction

FCSA TEAM 2

AGARWAL ARYAMAN
BATRA SIA
AGARWALA GRISHA

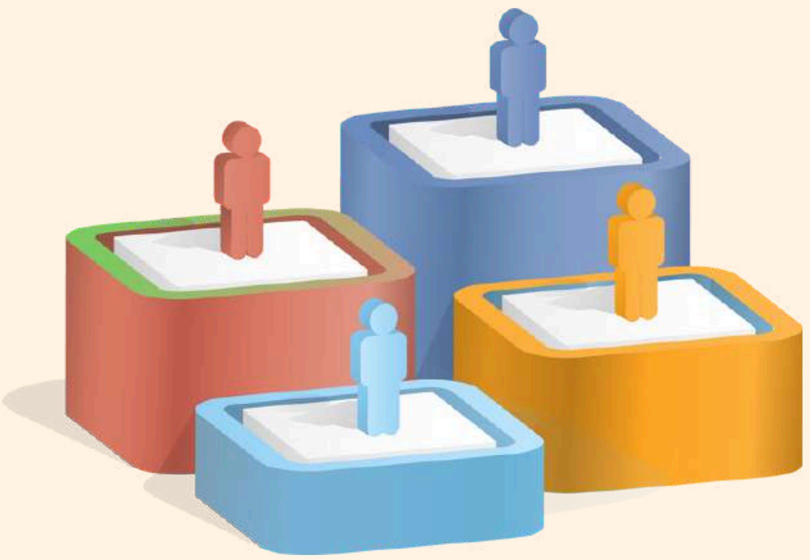
# Flow

What we'll be discussing in this video

# The Resale HDB Market

The HDB market plays a crucial role in Singapore's affordable housing vision, but home affordability remains a challenge despite government efforts, with a significant increase in resale prices.


# Factors Influencing HDB Resale Prices

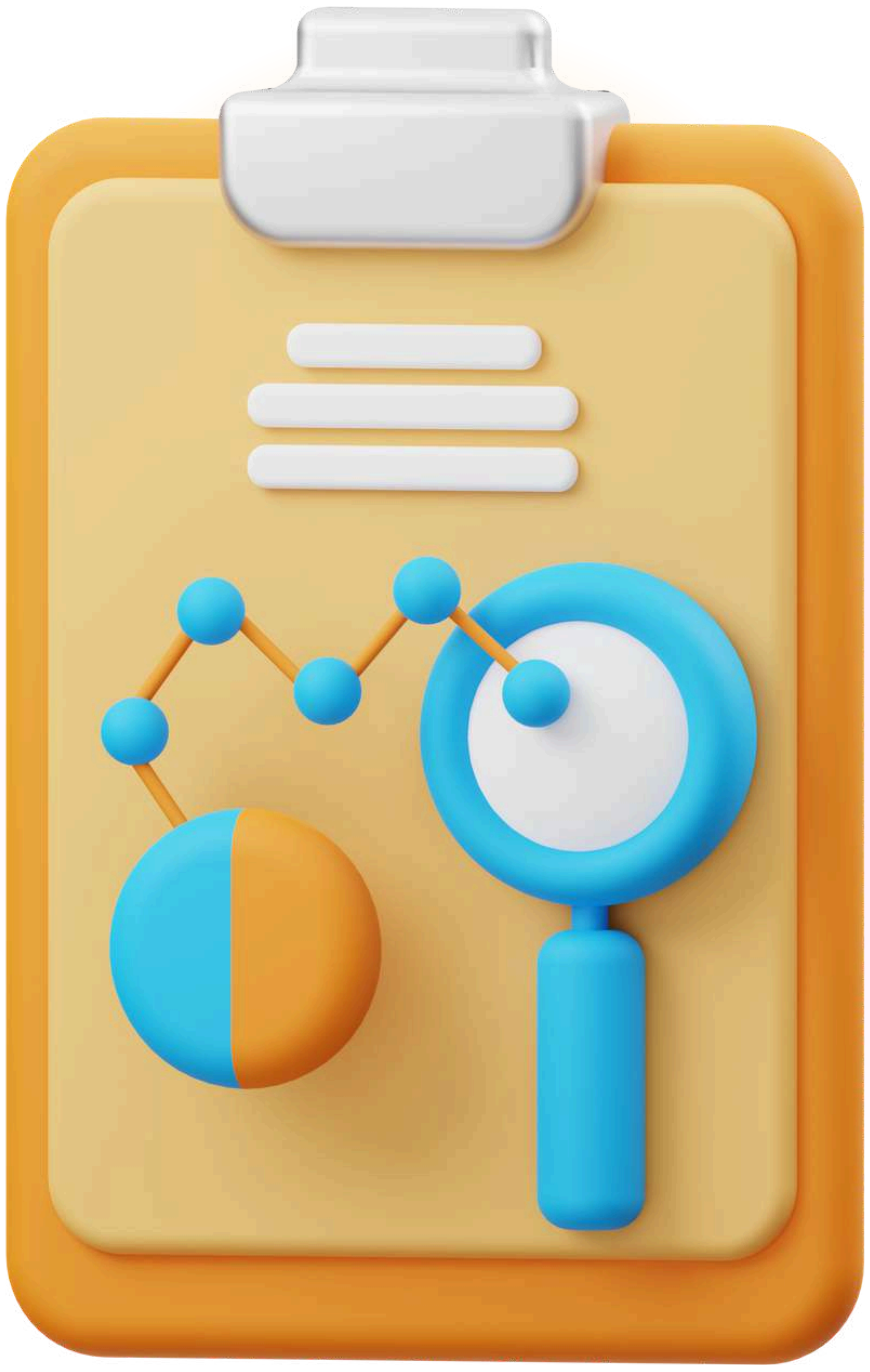
Limited Supply vs High Demand

Population Growth

Goverment Policies

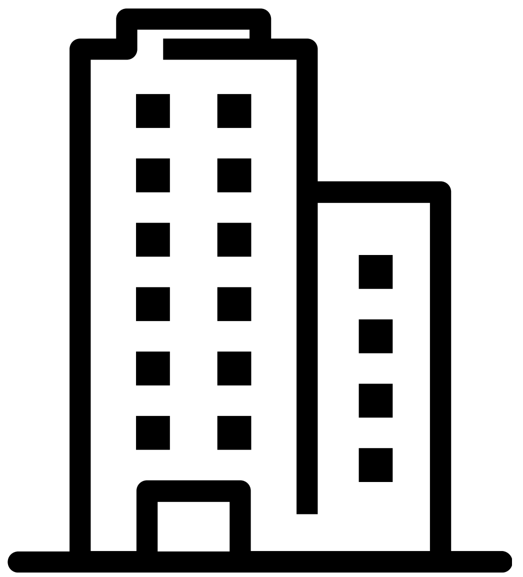
Renovation & Upgrading

# Project Motivation & Objective



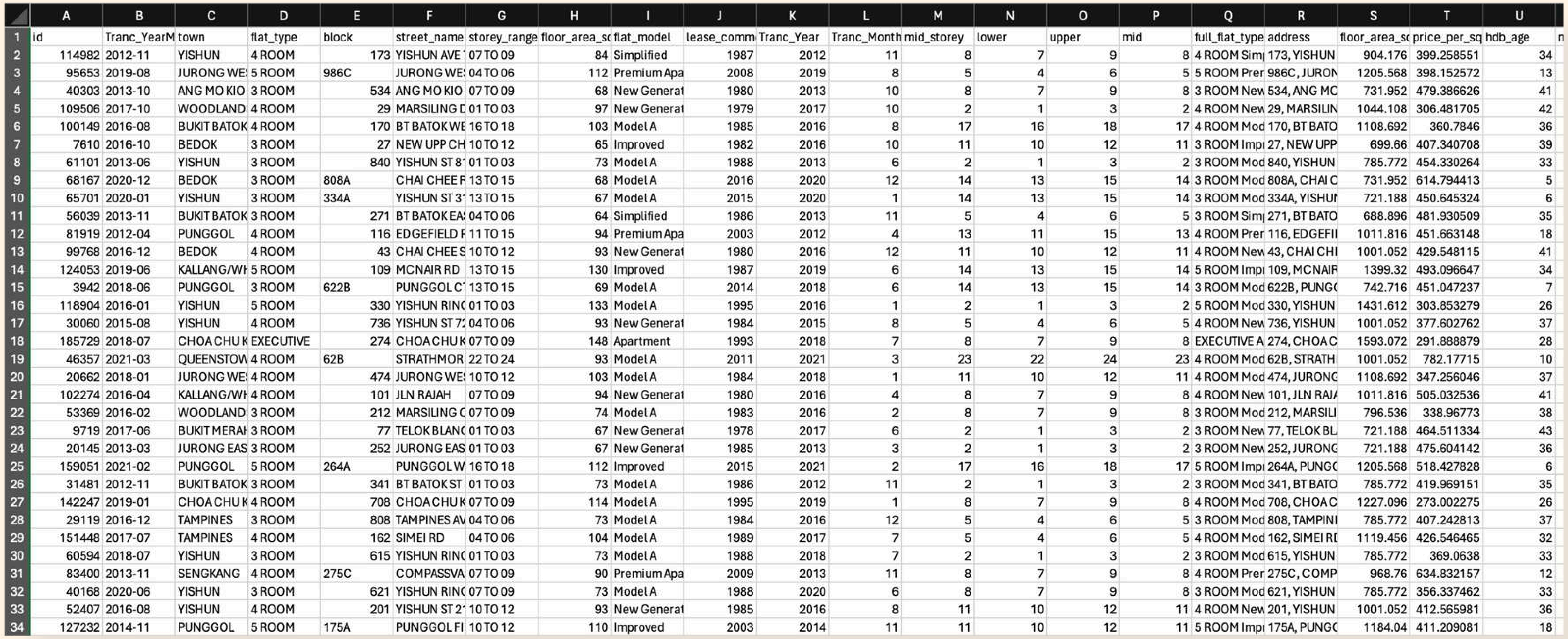
# Problem Statement

How can we predict HDB flat resale prices in Singapore to empower both buyers and sellers and enhance the transparency and efficiency of the public housing marketplace?



# ABOUT OUR DATASET

Singapore Public Housing Dataset from Kaggle



| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | id | Tranc_YearM | town | flat_type | block | street_name | storey_range | floor_area_sc | flat_model | lease_comm | Tranc_Year | Tranc_Month | mid_storey | lower | upper | mid | full_flat_type | address | floor_area_sc | price_per_sq | hdb_age | n |
| 2 | 114982 | 2012-11 | YISHUN | 4 ROOM | 173 | YISHUN AVE | 07 TO 09 | 84 | Simplified | 1987 | 2012 | 11 | 8 | 7 | 9 | 8 | 4 ROOM Sim | 173, YISHUN | 904.176 | 399.258551 | 34 | |
| 3 | 95653 | 2019-08 | JURONG WE | 5 ROOM | 986C | JURONG WE | 04 TO 06 | 112 | Premium Apa | 2008 | 2019 | 8 | 5 | 4 | 6 | 5 | 5 ROOM Prer | 986C, JURON | 1205.568 | 398.152572 | 13 | |
| 4 | 40303 | 2013-10 | ANG MO KIO | 3 ROOM | 534 | ANG MO KIO | 07 TO 09 | 68 | New Generat | 1980 | 2013 | 10 | 8 | 7 | 9 | 8 | 3 ROOM New | 534, ANG MC | 731.952 | 479.386626 | 41 | |
| 5 | 109506 | 2017-10 | WOODLAND | 4 ROOM | 29 | MARSILING D | 01 TO 03 | 97 | New Generat | 1979 | 2017 | 10 | 2 | 1 | 3 | 2 | 4 ROOM New | 29, MARSILIN | 1044.108 | 306.481705 | 42 | |
| 6 | 100149 | 2016-08 | BUKIT BATOK | 4 ROOM | 170 | BT BATOK WE | 16 TO 18 | 103 | Model A | 1985 | 2016 | 8 | 17 | 16 | 18 | 17 | 4 ROOM Mod | 170, BT BATO | 1108.692 | 360.7846 | 36 | |
| 7 | 7610 | 2016-10 | BEDOK | 3 ROOM | 27 | NEW UPP CH | 10 TO 12 | 65 | Improved | 1982 | 2016 | 10 | 11 | 10 | 12 | 11 | 3 ROOM Impi | 27, NEW UPP | 699.66 | 407.340708 | 39 | |
| 8 | 61101 | 2013-06 | YISHUN | 3 ROOM | 840 | YISHUN ST 8 | 01 TO 03 | 73 | Model A | 1988 | 2013 | 6 | 2 | 1 | 3 | 2 | 3 ROOM Mod | 840, YISHUN | 785.772 | 454.330264 | 33 | |
| 9 | 68167 | 2020-12 | BEDOK | 3 ROOM | 808A | CHAI CHEE F | 13 TO 15 | 68 | Model A | 2016 | 2020 | 12 | 14 | 13 | 15 | 14 | 3 ROOM Mod | 808A, CHAI C | 731.952 | 614.794413 | 5 | |
| 10 | 65701 | 2020-01 | YISHUN | 3 ROOM | 334A | YISHUN ST 3 | 13 TO 15 | 67 | Model A | 2015 | 2020 | 1 | 14 | 13 | 15 | 14 | 3 ROOM Mod | 334A, YISHUI | 721.188 | 450.645324 | 6 | |
| 11 | 56039 | 2013-11 | BUKIT BATOK | 3 ROOM | 271 | BT BATOK EA | 04 TO 06 | 64 | Simplified | 1986 | 2013 | 11 | 5 | 4 | 6 | 5 | 3 ROOM Sim | 271, BT BATO | 688.896 | 481.930509 | 35 | |
| 12 | 81919 | 2012-04 | PUNGGOL | 4 ROOM | 116 | EDGEFIELD F | 11 TO 15 | 94 | Premium Apa | 2003 | 2012 | 4 | 13 | 11 | 15 | 13 | 4 ROOM Prer | 116, EDGEFII | 1011.816 | 451.663148 | 18 | |
| 13 | 99768 | 2016-12 | BEDOK | 4 ROOM | 43 | CHAI CHEE S | 10 TO 12 | 93 | New Generat | 1980 | 2016 | 12 | 11 | 10 | 12 | 11 | 4 ROOM New | 43, CHAI CHI | 1001.052 | 429.548115 | 41 | |
| 14 | 124053 | 2019-06 | KALLANG/WH | 5 ROOM | 109 | MCNAIR RD | 13 TO 15 | 130 | Improved | 1987 | 2019 | 6 | 14 | 13 | 15 | 14 | 5 ROOM Impi | 109, MCNAIF | 1399.32 | 493.096647 | 34 | |
| 15 | 3942 | 2018-06 | PUNGGOL | 3 ROOM | 622B | PUNGGOL C | 13 TO 15 | 69 | Model A | 2014 | 2018 | 6 | 14 | 13 | 15 | 14 | 3 ROOM Mod | 622B, PUNG( | 742.716 | 451.047237 | 7 | |
| 16 | 118904 | 2016-01 | YISHUN | 5 ROOM | 330 | YISHUN RIN( | 01 TO 03 | 133 | Model A | 1995 | 2016 | 1 | 2 | 1 | 3 | 2 | 5 ROOM Mod | 330, YISHUN | 1431.612 | 303.853279 | 26 | |
| 17 | 30060 | 2015-08 | YISHUN | 4 ROOM | 736 | YISHUN ST 7 | 04 TO 06 | 93 | New Generat | 1984 | 2015 | 8 | 5 | 4 | 6 | 5 | 4 ROOM New | 736, YISHUN | 1001.052 | 377.602762 | 37 | |
| 18 | 185729 | 2018-07 | CHOA CHU K | EXECUTIVE | 274 | CHOA CHU K | 07 TO 09 | 148 | Apartment | 1993 | 2018 | 7 | 8 | 7 | 9 | 8 | EXECUTIVE A | 274, CHOA C | 1593.072 | 291.888879 | 28 | |
| 19 | 46357 | 2021-03 | QUEENSTOW | 4 ROOM | 62B | STRATHMOR | 22 TO 24 | 93 | Model A | 2011 | 2021 | 3 | 23 | 22 | 24 | 23 | 4 ROOM Mod | 62B, STRATH | 1001.052 | 782.17715 | 10 | |
| 20 | 20662 | 2018-01 | JURONG WE | 4 ROOM | 474 | JURONG WE | 10 TO 12 | 103 | Model A | 1984 | 2018 | 1 | 11 | 10 | 12 | 11 | 4 ROOM Mod | 474, JURONG | 1108.692 | 347.256046 | 37 | |
| 21 | 102274 | 2016-04 | KALLANG/WH | 4 ROOM | 101 | JLN RAJAH | 07 TO 09 | 94 | New Generat | 1980 | 2016 | 4 | 8 | 7 | 9 | 8 | 4 ROOM New | 101, JLN RAJ/ | 1011.816 | 505.032536 | 41 | |
| 22 | 53369 | 2016-02 | WOODLAND | 3 ROOM | 212 | MARSILING C | 07 TO 09 | 74 | Model A | 1983 | 2016 | 2 | 8 | 7 | 9 | 8 | 3 ROOM Mod | 212, MARSILI | 796.536 | 338.96773 | 38 | |
| 23 | 9719 | 2017-06 | BUKIT MERAH | 3 ROOM | 77 | TELOK BLAN( | 01 TO 03 | 67 | New Generat | 1978 | 2017 | 6 | 2 | 1 | 3 | 2 | 3 ROOM New | 77, TELOK BL | 721.188 | 464.511334 | 43 | |
| 24 | 20145 | 2013-03 | JURONG EAS | 3 ROOM | 252 | JURONG EAS | 01 TO 03 | 67 | New Generat | 1985 | 2013 | 3 | 2 | 1 | 3 | 2 | 3 ROOM New | 252, JURONG | 721.188 | 475.604142 | 36 | |
| 25 | 159051 | 2021-02 | PUNGGOL | 5 ROOM | 264A | PUNGGOL W | 16 TO 18 | 112 | Improved | 2015 | 2021 | 2 | 17 | 16 | 18 | 17 | 5 ROOM Impi | 264A, PUNG( | 1205.568 | 518.427828 | 6 | |
| 26 | 31481 | 2012-11 | BUKIT BATOK | 3 ROOM | 341 | BT BATOK ST | 01 TO 03 | 73 | Model A | 1986 | 2012 | 11 | 2 | 1 | 3 | 2 | 3 ROOM Mod | 341, BT BATO | 785.772 | 419.969151 | 35 | |
| 27 | 142247 | 2019-01 | CHOA CHU K | 4 ROOM | 708 | CHOA CHU K | 07 TO 09 | 114 | Model A | 1995 | 2019 | 1 | 8 | 7 | 9 | 8 | 4 ROOM Mod | 708, CHOA C | 1227.096 | 273.002275 | 26 | |
| 28 | 29119 | 2016-12 | TAMPINES | 3 ROOM | 808 | TAMPINES AV | 04 TO 06 | 73 | Model A | 1984 | 2016 | 12 | 5 | 4 | 6 | 5 | 3 ROOM Mod | 808, TAMPINI | 785.772 | 407.242813 | 37 | |
| 29 | 151448 | 2017-07 | TAMPINES | 4 ROOM | 162 | SIMEI RD | 04 TO 06 | 104 | Model A | 1989 | 2017 | 7 | 5 | 4 | 6 | 5 | 4 ROOM Mod | 162, SIMEI RI | 1119.456 | 426.546465 | 32 | |
| 30 | 60594 | 2018-07 | YISHUN | 3 ROOM | 615 | YISHUN RIN( | 01 TO 03 | 73 | Model A | 1988 | 2018 | 7 | 2 | 1 | 3 | 2 | 3 ROOM Mod | 615, YISHUN | 785.772 | 369.0638 | 33 | |
| 31 | 83400 | 2013-11 | SENGKANG | 4 ROOM | 275C | COMPASSVA | 07 TO 09 | 90 | Premium Apa | 2009 | 2013 | 11 | 8 | 7 | 9 | 8 | 4 ROOM Prer | 275C, COMP | 968.76 | 634.832157 | 12 | |
| 32 | 40168 | 2020-06 | YISHUN | 3 ROOM | 621 | YISHUN RIN( | 07 TO 09 | 73 | Model A | 1988 | 2020 | 6 | 8 | 7 | 9 | 8 | 3 ROOM Mod | 621, YISHUN | 785.772 | 356.337462 | 33 | |
| 33 | 52407 | 2016-08 | YISHUN | 4 ROOM | 201 | YISHUN ST 2 | 10 TO 12 | 93 | New Generat | 1985 | 2016 | 8 | 11 | 10 | 12 | 11 | 4 ROOM New | 201, YISHUN | 1001.052 | 412.565981 | 36 | |
| 34 | 127232 | 2014-11 | PUNGGOL | 5 ROOM | 175A | PUNGGOL FI | 10 TO 12 | 110 | Improved | 2003 | 2014 | 11 | 11 | 10 | 12 | 11 | 5 ROOM Impi | 175A, PUNG( | 1184.04 | 411.209081 | 18 | |

- Compiled HDB Resale data from 2012 to 2021
- Information collected on the details of each unit and its sale.
- Describing a total of 78 features of every single unit

# DATA PREPARATION & PREPROCESSING

# INITIAL DATA CLEANING

```
In [216]: # removing columns which are irrelevant
          irrelevant_cols = ['id', 'block', 'street_name', 'address', 'postal', 'bus_stop_name', 'mrt_latitude', 'mrt_longitud
                             'bus_stop_latitude', 'bus_stop_longitude', 'pri_sch_latitude', 'pri_sch_longitude', 'sec_sch_lati
                             'floor_area_sqft', 'lease_commence_date', 'tranc_yearmonth', 'mid_storey', 'full_flat_type', 'blo

          # Filter out columns that do not exist in the DataFrame
          cols_to_drop = [col for col in irrelevant_cols if col in data.columns]

          # Drop these columns from the DataFrame
          data.drop(columns=cols_to_drop, axis=1, inplace=True)

          # Print the remaining columns to verify
          print("Remaining columns after dropping:", data.columns)

          Remaining columns after dropping: Index(['town', 'flat_type', 'storey_range', 'floor_area_sqm', 'flat_model',
                 'resale_price', 'tranc_year', 'tranc_month', 'lower', 'upper', 'mid',
                 'price_per_sqft', 'hdb_age', 'max_floor_lvl', 'year_completed',
                 'residential', 'commercial', 'market_hawker', 'multistorey_carpark',
                 'precinct_pavilion', 'total_dwelling_units', '1room_sold', '2room_sold',
                 '3room_sold', '4room_sold', '5room_sold', 'exec_sold', 'multigen_sold',
                 'studio_apartment_sold', '1room_rental', '2room_rental', '3room_rental',
                 'other_room_rental', 'latitude', 'longitude', 'planning_area',
                 'mall_nearest_distance', 'mall_within_500m', 'mall_within_1km',
                 'mall_within_2km', 'hawker_nearest_distance', 'hawker_within_500m',
                 'hawker_within_1km', 'hawker_within_2km', 'hawker_food_stalls',
                 'hawker_market_stalls', 'mrt_nearest_distance', 'mrt_name',
                 'bus_interchange', 'mrt_interchange', 'bus_stop_nearest_distance',
                 'pri_sch_nearest_distance', 'pri_sch_name', 'vacancy',
                 'pri_sch_affiliation', 'sec_sch_nearest_dist', 'sec_sch_name',
                 'cutoff_point', 'affiliation'],
                dtype='object')
```

- **Standardization**: All column names were converted to lowercase and formatted to snake case for consistency and ease of access, e.g., FloorAreaSqft to floor_area_sqm.
- **Irrelevant Columns**: Removed non-essential columns like IDs, address details, and coordinates that do not influence resale prices, focusing on variables directly affecting housing values.

# DATA IMPUTATION & ENRICHMENT

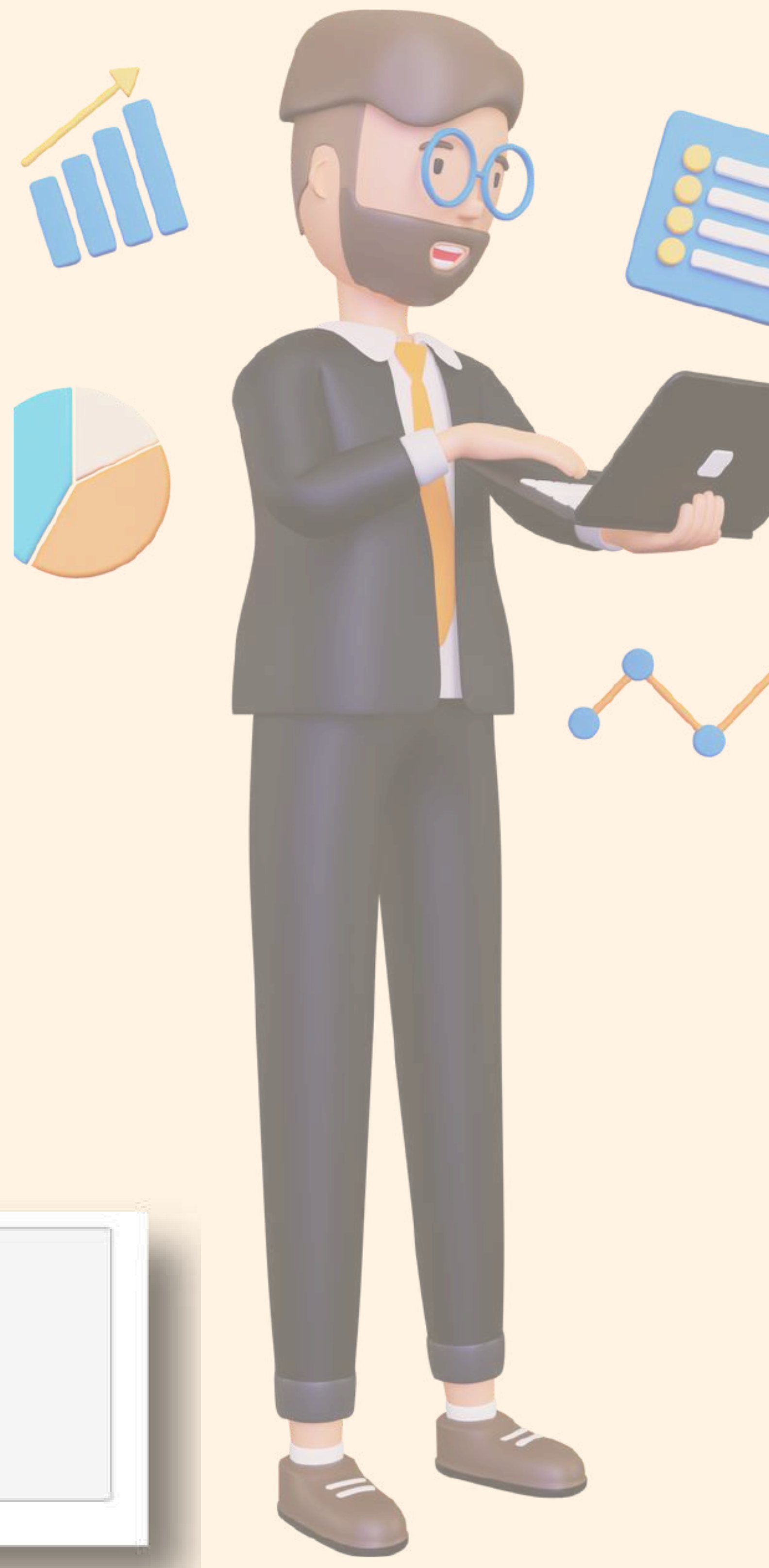
```
In [135]: #finding columns with null values
          data.isnull().sum().sort_values().tail(8)

Out[135]: 4room_sold                  0
          mall_nearest_distance     829
          mall_within_2km          1940
          mall_within_1km         25426
          hawker_within_2km       29202
          hawker_within_1km       60868
          mall_within_500m        92789
          hawker_within_500m      97390
          dtype: int64

In [136]: data['mall_nearest_distance']

Out[136]: 0         1094.090418
          1          866.941448
          2         1459.579948
          3          950.175199
          4          729.771895
                        ...
          150629     585.138715
          150630     250.084466
          150631    1790.053482
          150632     587.244922
          150633     225.435937
          Name: mall_nearest_distance, Length: 150634, dtype: float64
```

- Missing Data: Identified columns with missing values through a systematic review.
- Imputation Strategy: Applied **SimpleImputer** to replace missing values with predetermined constants, ensuring no data loss and maintaining dataset integrity
- Added calculated fields such as mall_nearest_distance to enhance the dataset's utility for spatial analysis.

# PREPARING DATA FOR ANALYSIS

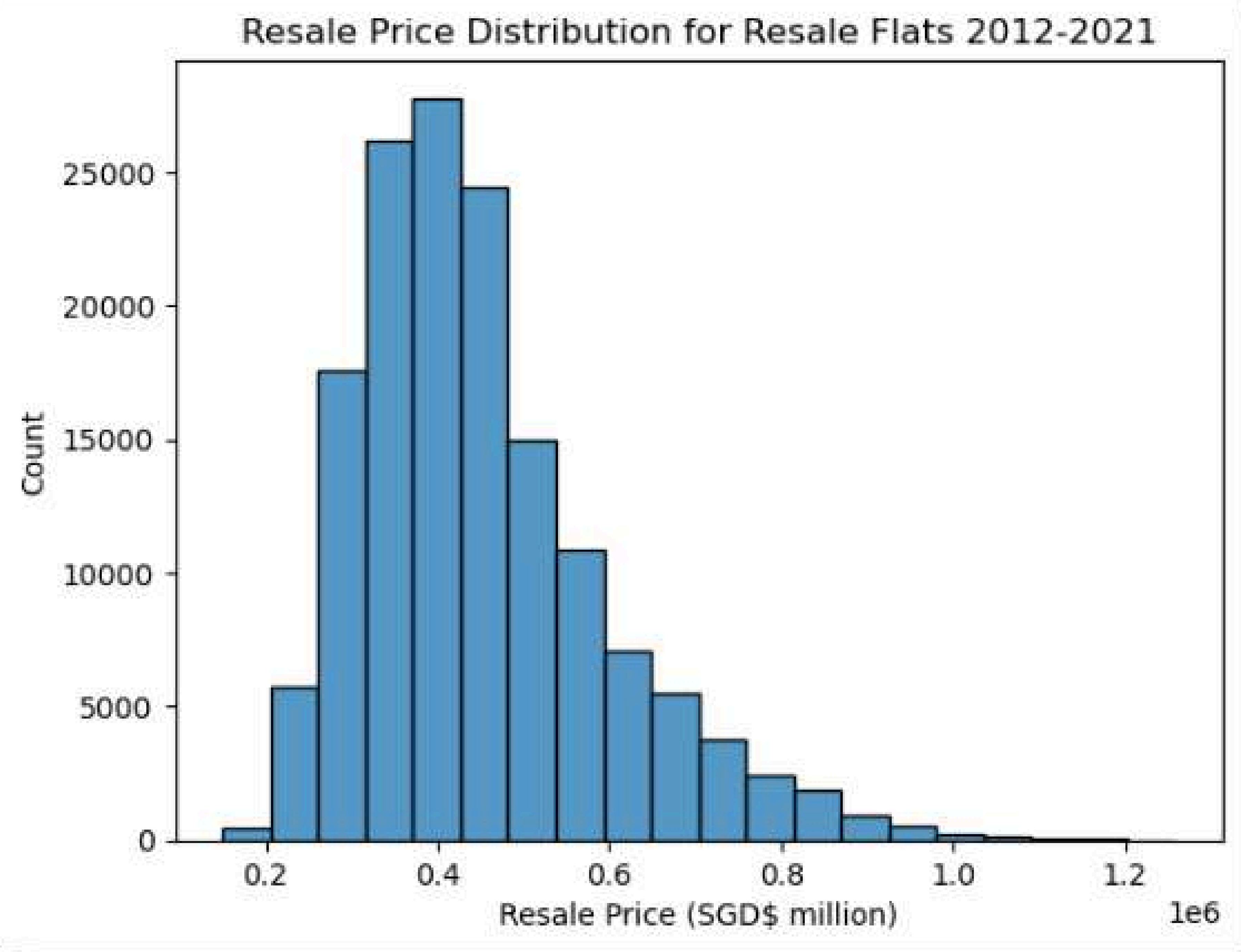
- **Unique Value Analysis**: Removed any columns with only a single unique value, which are statistically irrelevant for modeling.
- **Data Types Correction**: Ensured all data types align with the requirements of the predictive models, such as converting dates and categoricals.

```
In [140]: #filling null values with 0
          col_with_null = data.columns[data.isnull().sum() != 0].to_list()
          imputer=SimpleImputer(missing_values=np.NaN, strategy='constant', fill_value=0)
          for x in col_with_null:
              data[x]=imputer.fit_transform(data[x].values.reshape(-1,1))
```



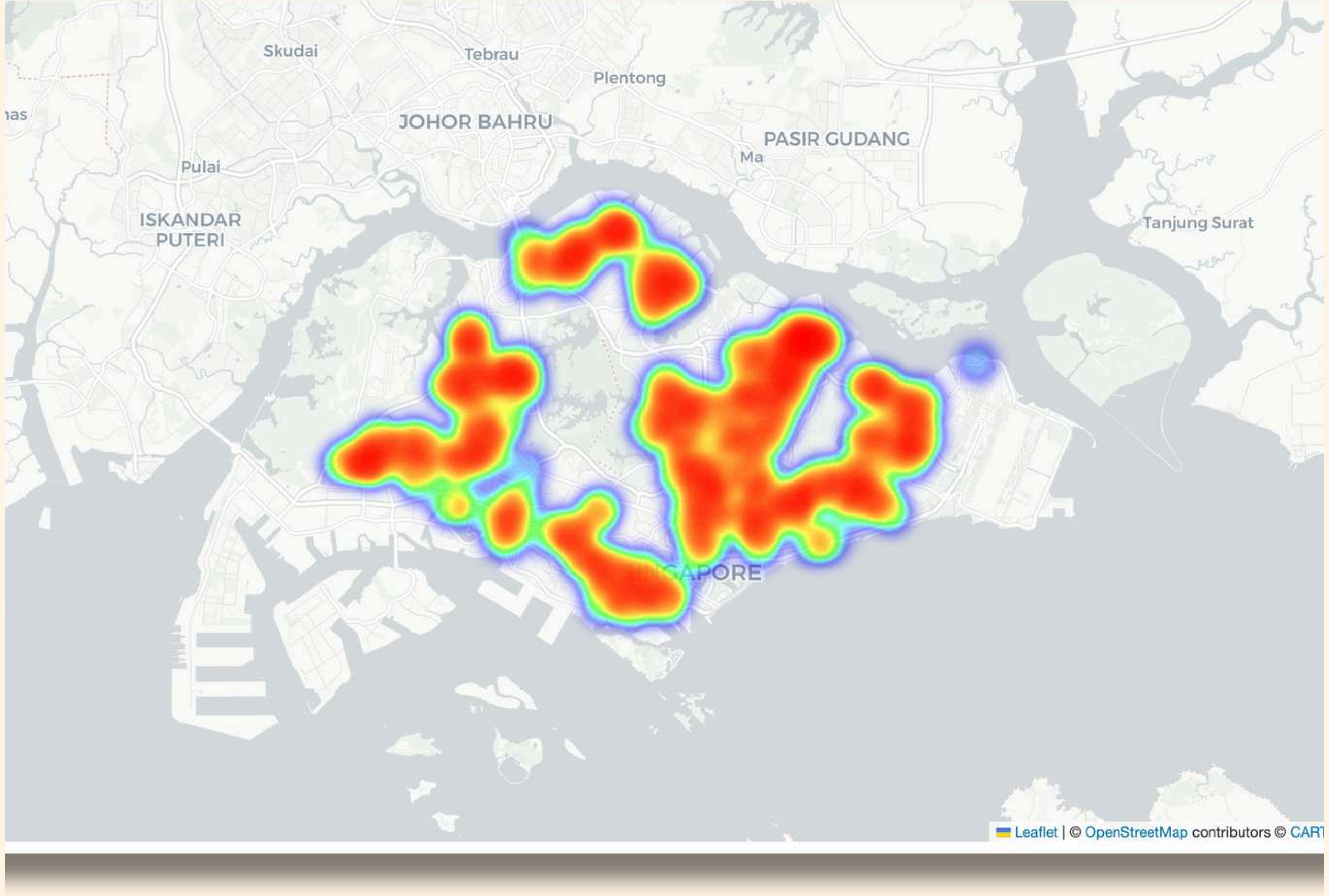
# EXPLORATORY DATA ANALYSIS

# TRENDS IN RESALE FLAT PRICES (2012-2021)



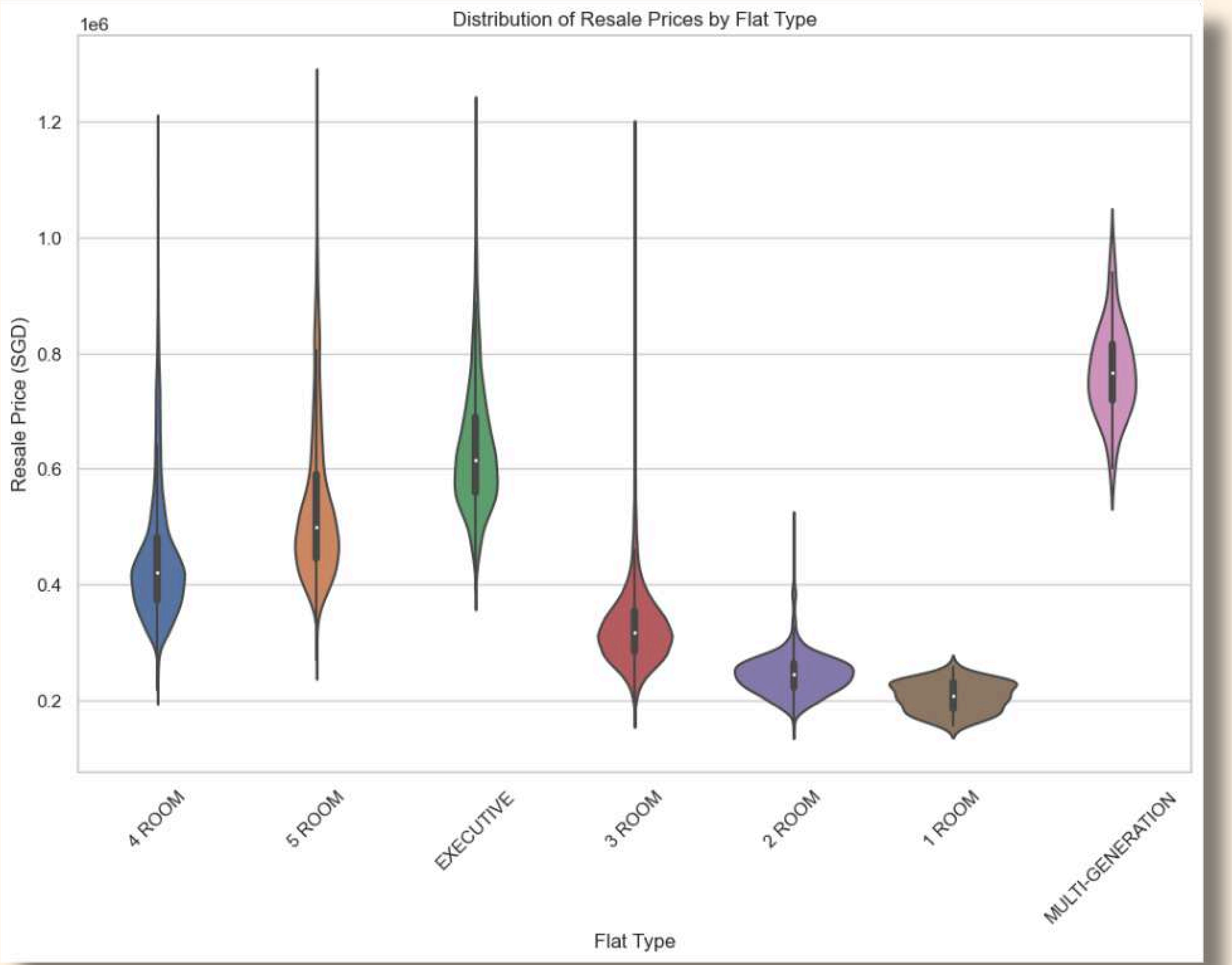

- The majority of resale flats fall within the SGD 200,000 to SGD 400,000 range, revealing a significant concentration of market transactions in this segment.
- The histogram exhibits a right-skewed distribution, indicative of a larger quantity of flats sold at lower to mid-range prices.
- There is a noticeable decline in counts as resale prices increase beyond SGD 400,000, suggesting a smaller market for higher-priced flats.

# GEOSPATIAL HEATMAP OF RESALE PRICES



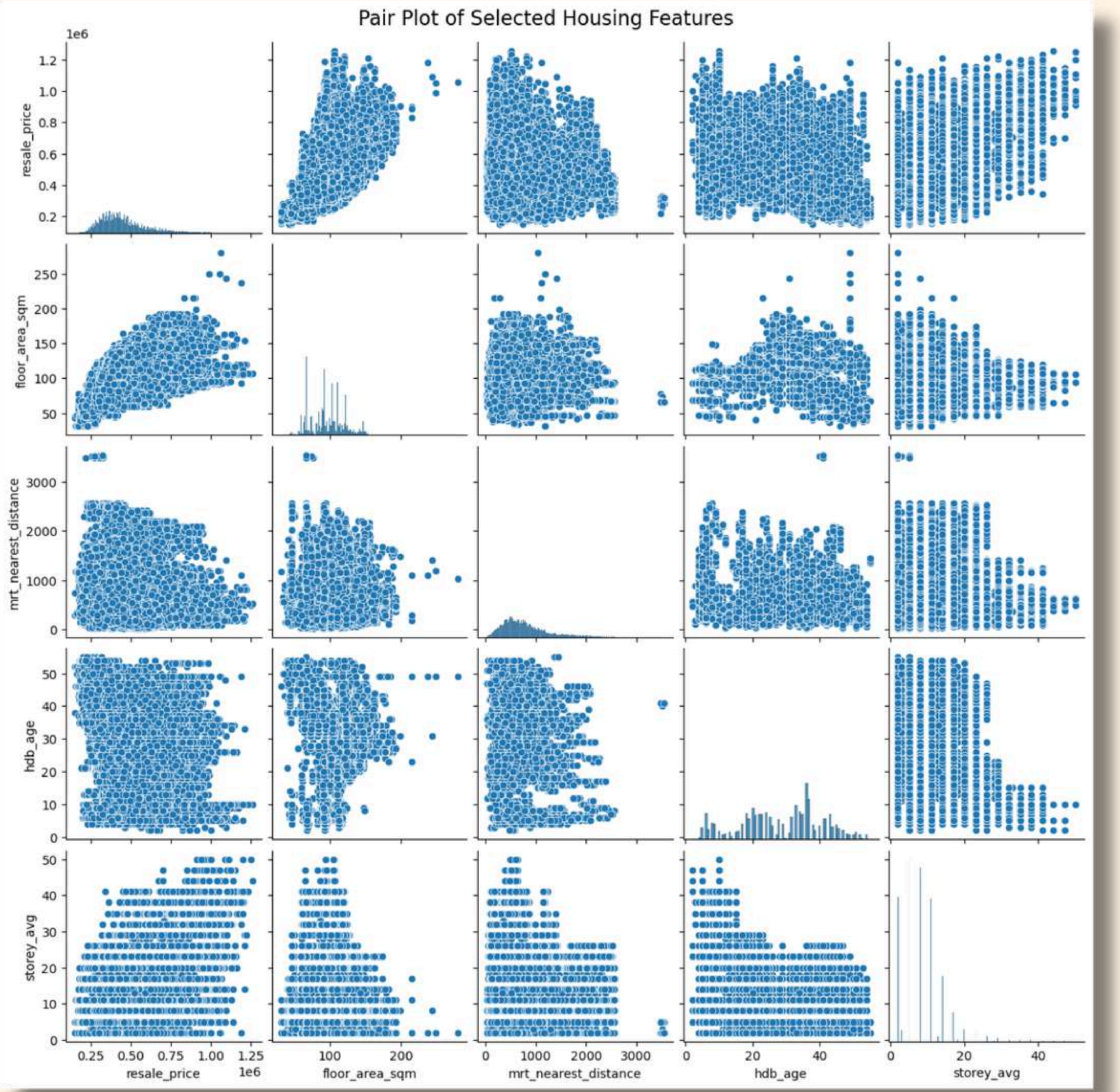

The heatmap reveals high-density areas where resale prices are significantly higher, showcasing key residential zones.

# VIOLIN PLOT OF RESALE PRICE & FLAT TYPE



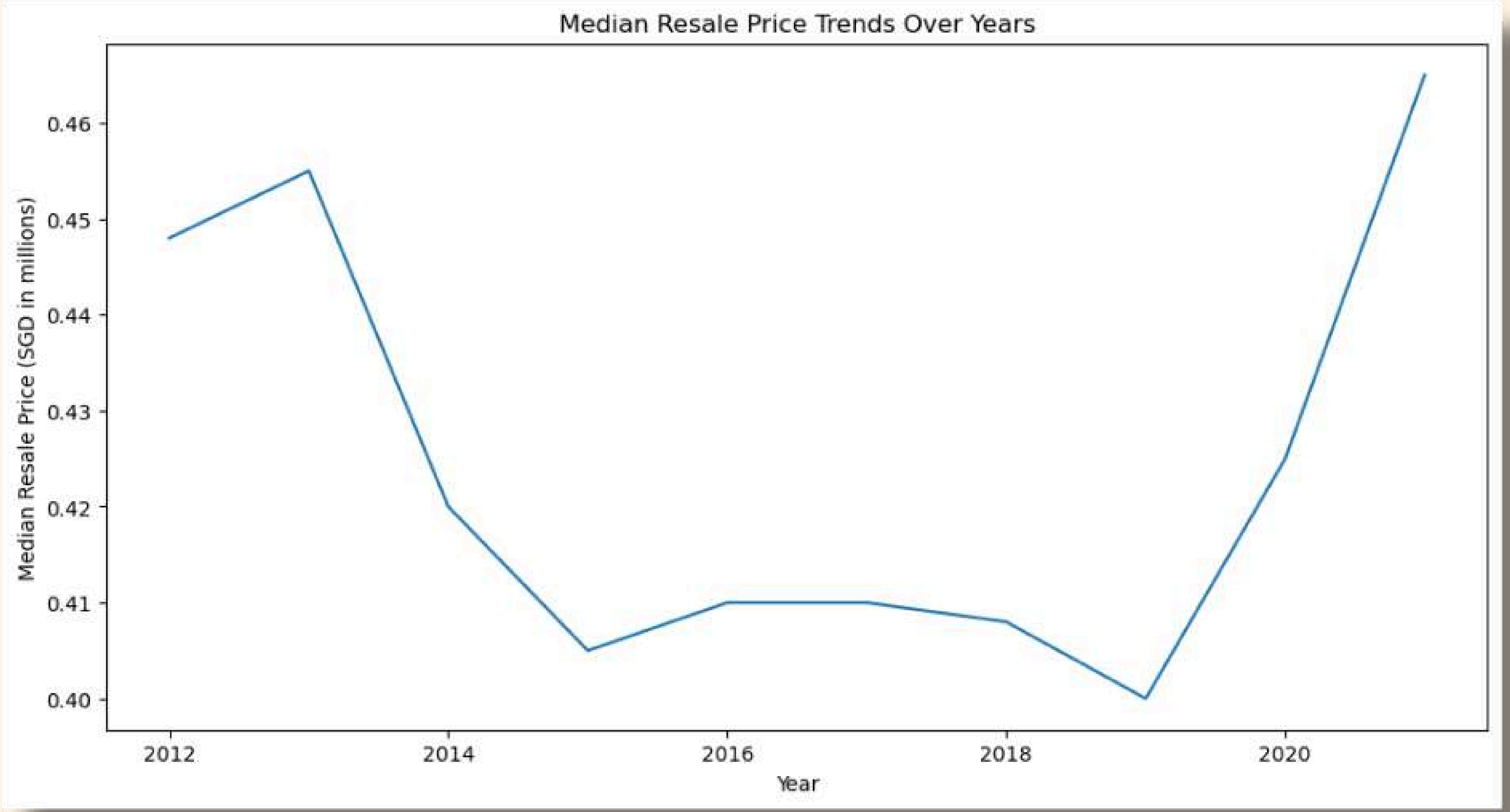

- Violin plots visually compare price variability and medians for different flat types.
- Executive flats have higher median prices due to larger living spaces.
- Multi-generation flats have limited transactions and a focused price range.

# PAIR PLOT OF SELECTED HOUSING FEATURES



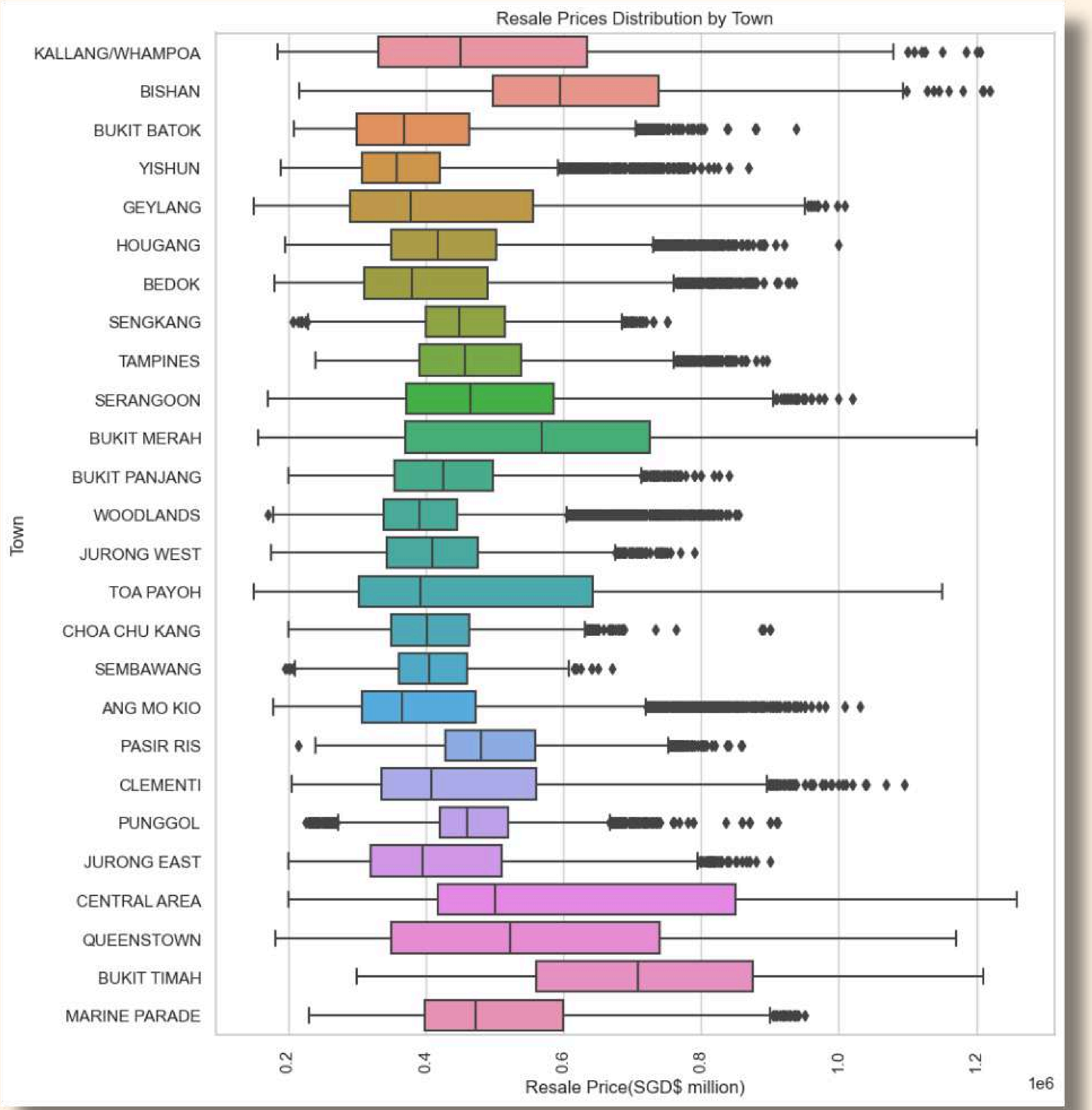

This scatter plot matrix visualizes the relationships between five housing features: resale price, floor area (in square meters), distance to the nearest MRT station, HDB age, and storey average. Each subplot shows the distribution of a pair of features.

# LINEPLOT OF MEDIAN RESALE PRICE & YEAR




- "The line chart presents a clear trajectory of median resale prices for flats from 2012 to 2021."
- "A noticeable decline is observed around 2014, followed by stabilisation until a sharp incline post-2018."
- "This pattern underscores the volatile nature of the resale market over the decade.

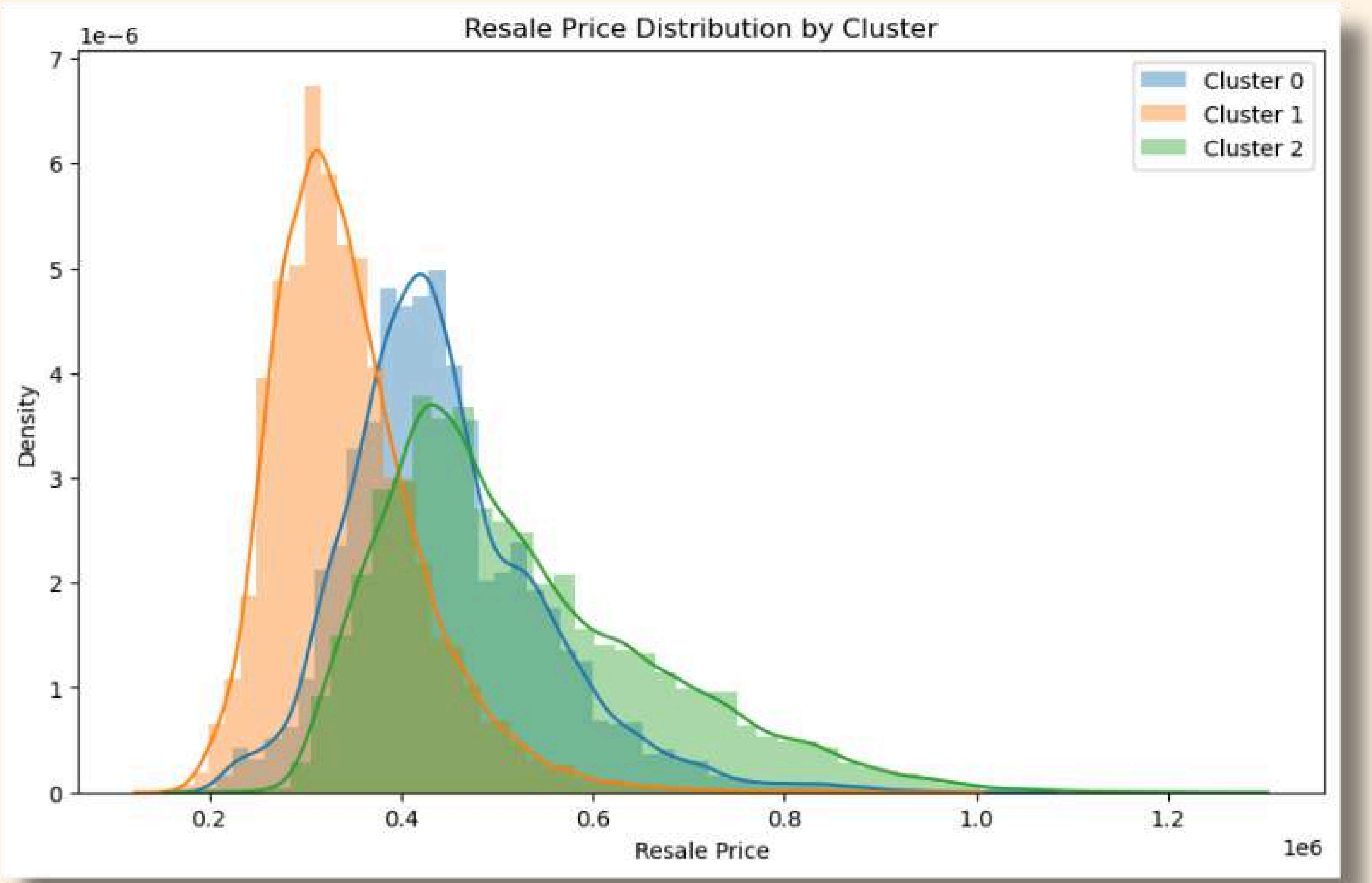
# RESALE PRICES DISTRIBUTION BY TOWN




- This boxplot visualizes the distribution of resale flat prices across towns in Singapore.

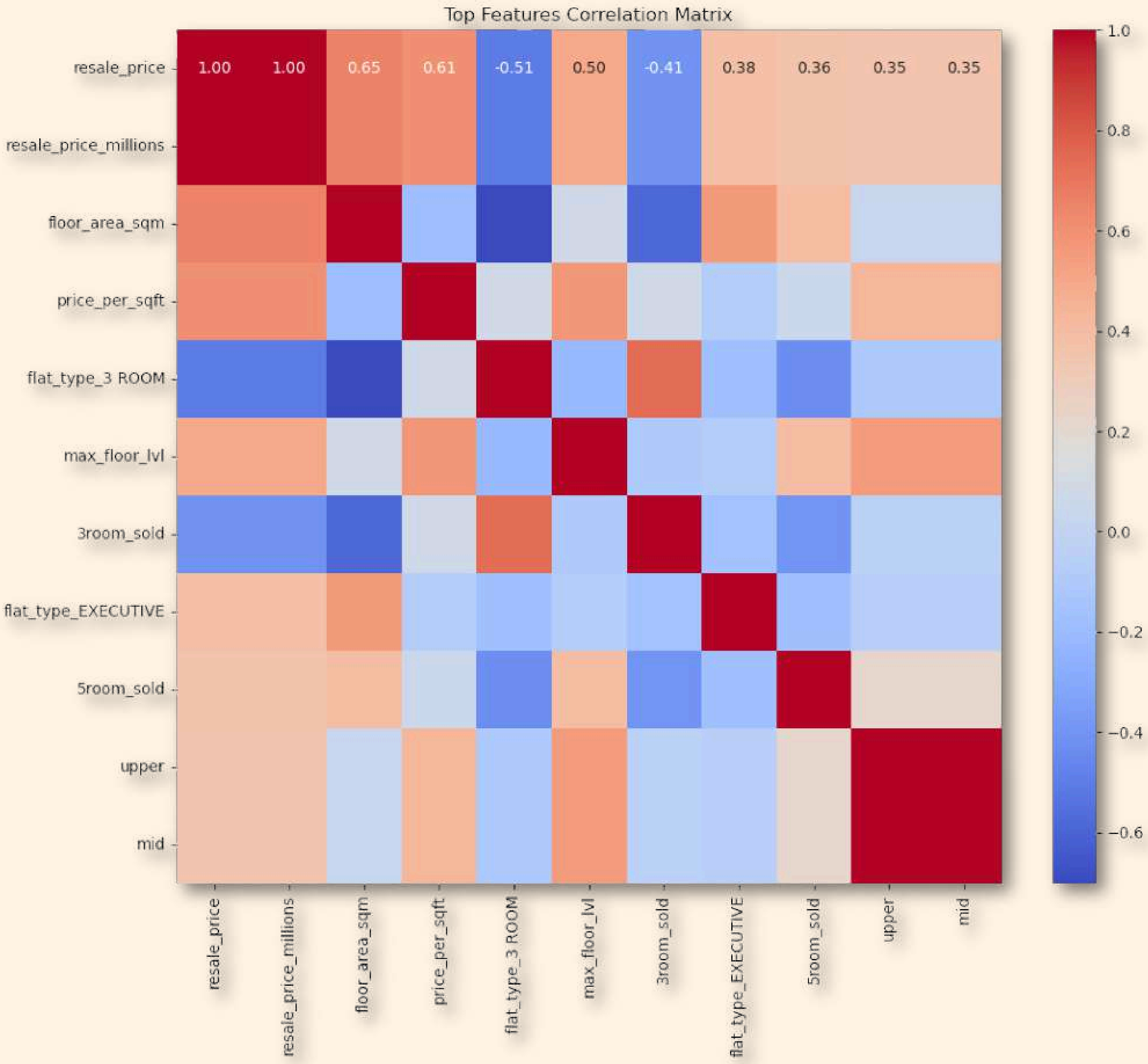
- Notably, towns like 'Marine Parade' and 'Bukit Timah' exhibit higher median prices with wider price ranges, indicating a higher property value in these areas. Conversely, towns like 'Punggol' and 'Sembawang' show more affordable options with a tighter price distribution, suggesting a higher density of lower-priced flats.

# CLUSTER ANALYSIS: RESALE PRICE DISTRIBUTION




- **Cluster 0:** Characterized by a narrower price range, Cluster 0 represents properties that have a high density around the lower to mid-range resale price bracket.
- **Cluster 1:** Exhibiting a broader distribution, Cluster 1 encompasses properties with a more varied price range, stretching towards the higher end of the market.
- **Cluster 2:** Cluster 2 shows properties with a mid to high price distribution, with fewer low-priced options, indicating a premium segment of the resale market.

# TOP FEATURES: CORRELATION MATRIX




Key Relationships:

- Floor Area vs. Price: We observe a strong positive correlation between 'floor_area_sqm' and 'resale_price', validating the premise that larger homes tend to have higher market values.

- Price per Square Foot: Interestingly, 'price_per_sqft' reveals how the value of space varies, providing an essential metric for property valuation.

# MACHINE LEARNING

# LINEAR REGRESSION: MODEL TRAINING

### Model Training

- **Initialization**: A `LinearRegression` model from scikit-learn is instantiated.
- **Fitting the Model**: The model is trained using a dataset of 100 generated points with a random seed set for reproducibility.

In [186]:
```
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Sample data generation
np.random.seed(0)
features = np.random.rand(100, 1)
targets = 2 + 3 * features + np.random.normal(0, 0.1, (100, 1))

# Convert to pandas DataFrame
df = pd.DataFrame(data=np.hstack((features, targets)), columns=['Feature', 'Target'])

# Splitting the data into training and testing sets
features_train, features_test, targets_train, targets_test = train_test_split(df[['Feature']], df['Target'], test_si

# Linear Regression Model
model_lr = LinearRegression()
model_lr.fit(features_train, targets_train)
```

Out[186]:
```
▾ LinearRegression
LinearRegression()
```

- Model initialised with scikit-learn's LinearRegression function, offering ease of use and robustness.
- Trained on a dataset comprising 100 data points, ensuring the model's capacity to predict is well-calibrated.

# LINEAR REGRESSION: MODEL PREDICTIONS

### Model Prediction

- **Prediction Generation**: Predictions are made for both training and test datasets.
- **Performance Metrics**:
  - The **R-squared** value, indicating the proportion of variance explained by the model, is calculated for both sets.
  - The **Mean Squared Error (MSE)**, representing the average squared difference between the predicted and actual values, is also computed.

In [3]:

```python
# Making predictions
predictions_train = model_lr.predict(features_train)
predictions_test = model_lr.predict(features_test)

# Calculate R-squared and MSE
r2_train = r2_score(targets_train, predictions_train)
mse_train = mean_squared_error(targets_train, predictions_train)
r2_test = r2_score(targets_test, predictions_test)
mse_test = mean_squared_error(targets_test, predictions_test)

print(f'R2 score for training set: {r2_train}')
print(f'MSE for training set: {mse_train}')
print(f'R2 score for testing set: {r2_test}')
print(f'MSE for testing set: {mse_test}')
```

```
R2 score for training set: 0.9875738032878963
MSE for training set: 0.010113360910427542
R2 score for testing set: 0.9809551843591459
MSE for testing set: 0.009177532469714303
```

### Model Coefficients

- The model's **intercept** and **coefficient** for the feature are printed out, revealing the slope and the y-intercept of the fitted line.

In [4]:

```python
print(f'Intercept: {model_lr.intercept_}')
print(f'Coefficient: {model_lr.coef_[0]}')
```

```
Intercept: 2.020634018871143
Coefficient: 2.9980518202009776
```

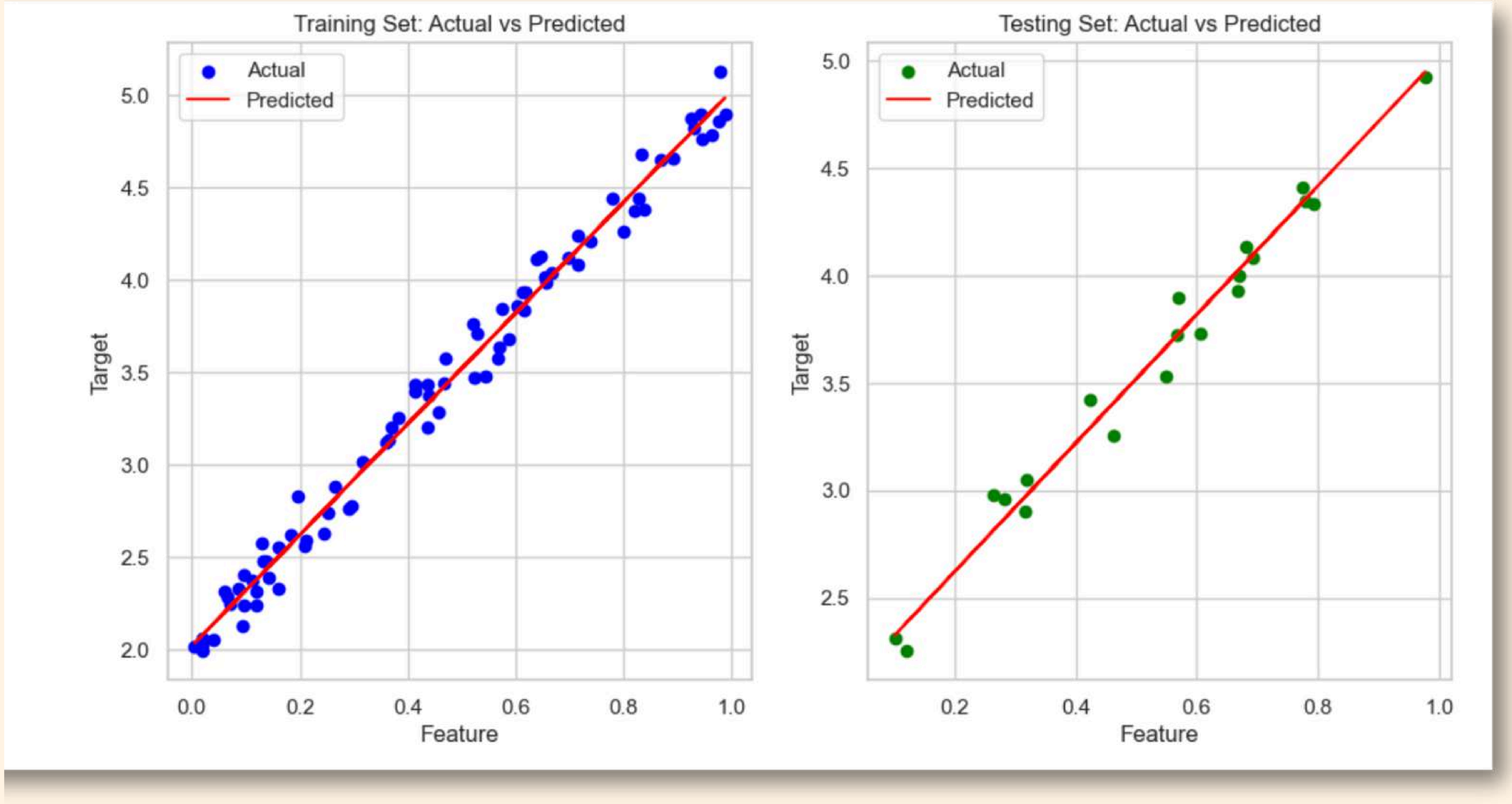
R^2 score for training set: 0.988

MSE for training set: 0.010

R^2 score for testing set: 0.981

MSE for testing set: 0.009

# LINEAR REGRESSION: ACTUAL VS. PREDICTED VALUES



# DECISION TREES: MODEL TRAINING & TRAINING

### Model Training and Prediction

- **Model Initialization**: A `DecisionTreeRegressor` from scikit-learn is used, configured with a maximum depth of 5 and a minimum of 10 samples per leaf.
- **Model Fitting**: The model is trained using the training dataset comprising features and targets derived from a synthetically generated dataset.
- **Prediction Generation**: After training, predictions are made on both the training and testing datasets.

In [187]:
```
from sklearn.tree import DecisionTreeRegressor

# Initialize the Decision Tree Regressor
dt_regressor = DecisionTreeRegressor(max_depth=5, min_samples_leaf=10, random_state=42)
# Fit the model to the training data
dt_regressor.fit(features_train, targets_train)

# Making predictions
predictions_train_dt = dt_regressor.predict(features_train)
predictions_test_dt = dt_regressor.predict(features_test)
```

- Configured with a depth of 5 and at least 10 samples per leaf, our model aims for a balanced approach between precision and generalizability.
- Using scikit-learn's DecisionTreeRegressor, we fit our model to a carefully crafted dataset, prepared to mimic real-world scenarios.

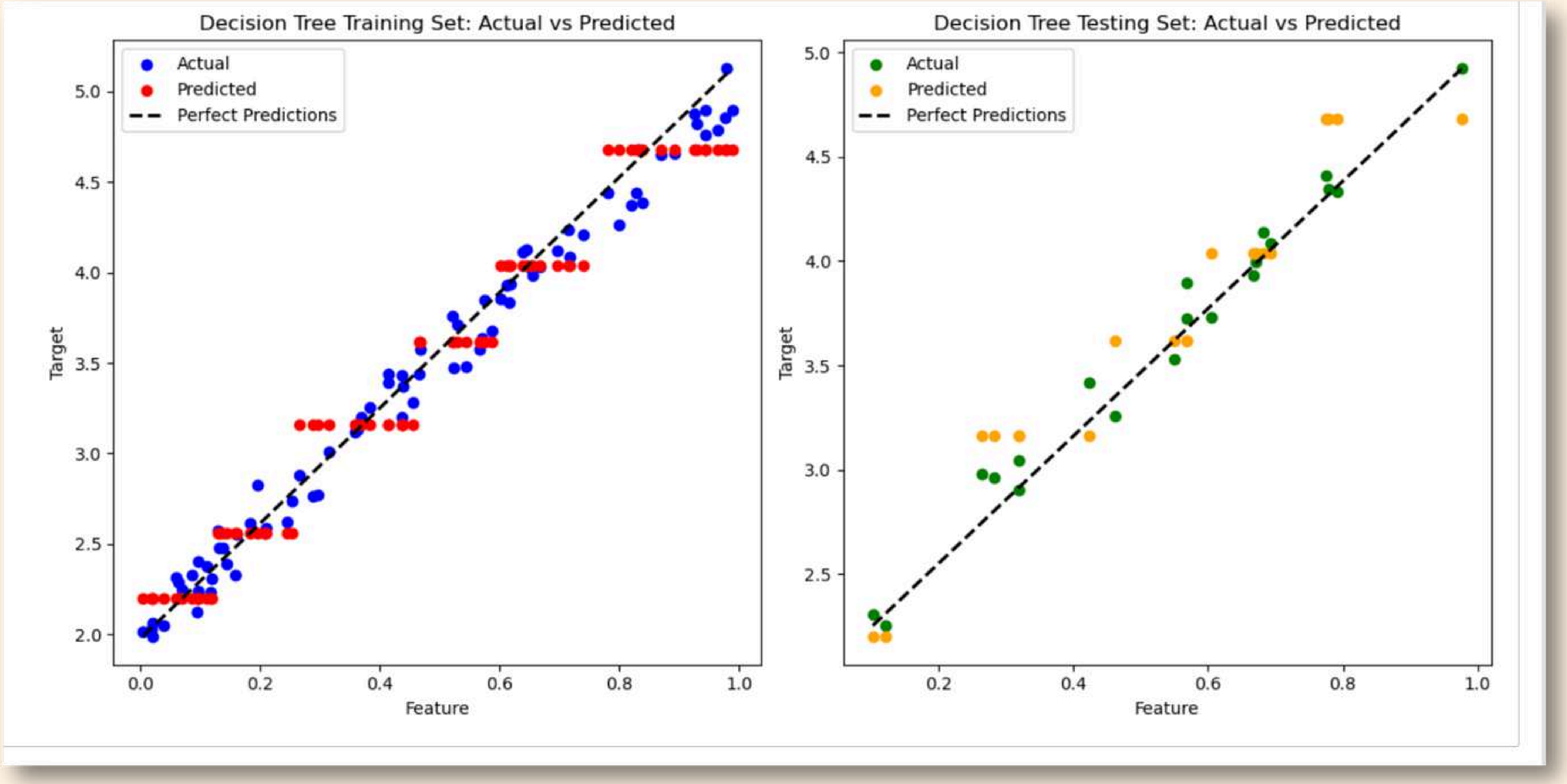
# DECISION TREES: EVALUATING PERFORMANCE

In [6]:
```python
# Evaluate the model
r2_train_dt = r2_score(targets_train, predictions_train_dt)
mse_train_dt = mean_squared_error(targets_train, predictions_train_dt)
r2_test_dt = r2_score(targets_test, predictions_test_dt)
mse_test_dt = mean_squared_error(targets_test, predictions_test_dt)

print("Decision Tree - Training performance:")
print(f"R^2 score for training set: {r2_train_dt}")
print(f"MSE for training set: {mse_train_dt}")

print("Decision Tree - Testing performance:")
print(f"R^2 score for testing set: {r2_test_dt}")
print(f"MSE for testing set: {mse_test_dt}")
```

```
Decision Tree - Training performance:
R^2 score for training set: 0.96324763912056
MSE for training set: 0.02991179831573179
Decision Tree - Testing performance:
R^2 score for testing set: 0.9026961991134502
MSE for testing set: 0.04688986277962592
```

Our model achieves an impressive $R^2$ score, suggesting a strong fit to the data.

# DECISION TREES: VISUALISATION




By comparing actual values (blue for training, green for testing) to predictions (red), we can visually assess the model's performance.

# RANDOM FOREST:
# MODEL TRAINING & PERFOMANCE EVALUATION

### Model Training and Prediction ¶

- **Model Initialization**: A `RandomForestRegressor` with 100 estimators is utilized, providing a robust approach to regression through ensemble learning.
- **Model Fitting**: The model is trained using the features and target values from the training dataset.
- **Prediction Generation**: Predictions are made for both the training set and testing set, allowing for an evaluation of the model's performance on unseen data.

In [188]:
```
from sklearn.ensemble import RandomForestRegressor

# Initialize the Random Forest Regressor
rf = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model to the training data
rf.fit(features_train, targets_train)

# Make predictions on the training and testing sets
predictions_train_rf = rf.predict(features_train)
predictions_test_rf = rf.predict(features_test)
```

### Model Performance Evaluation

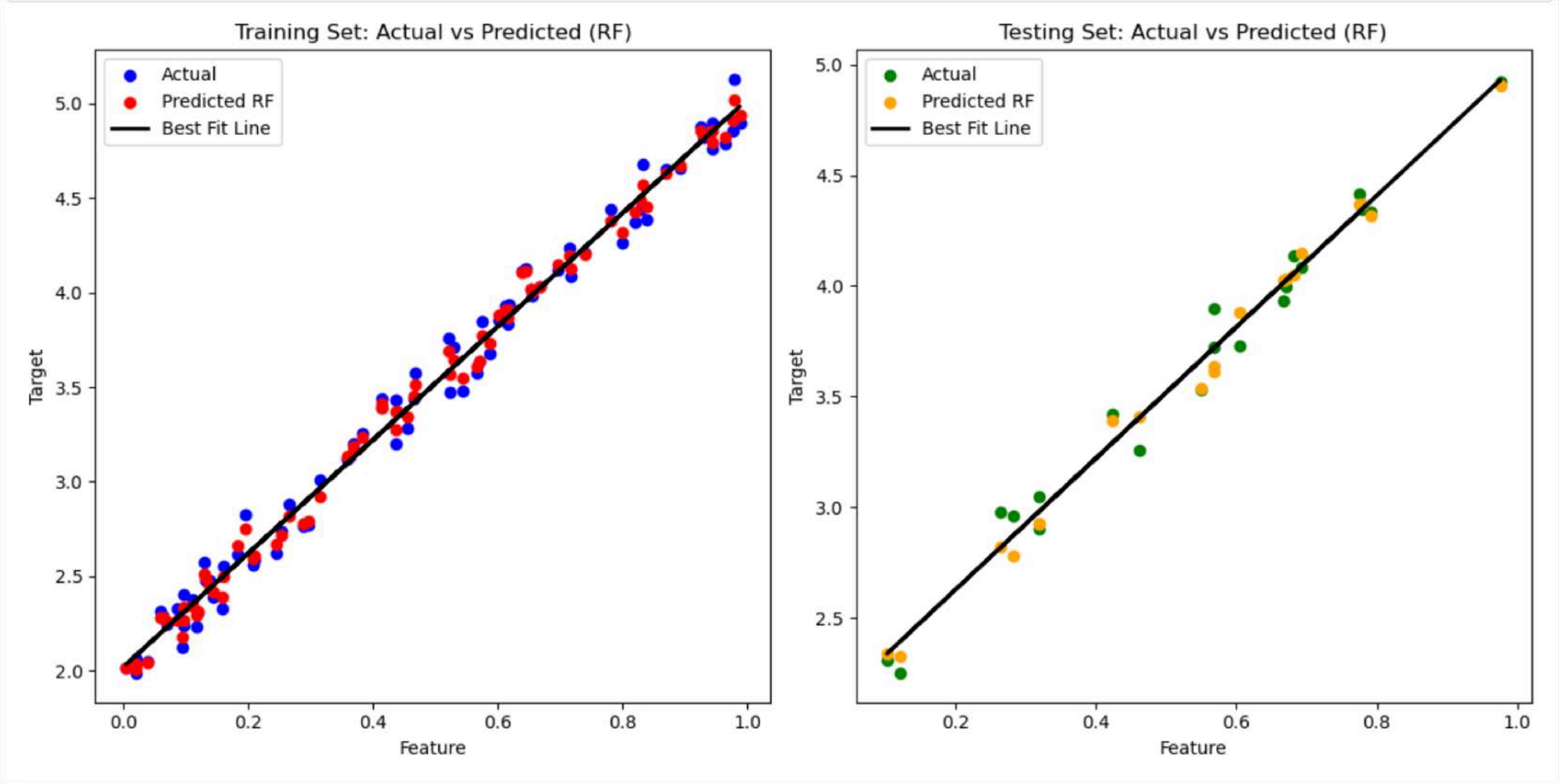
- **Performance Metrics**:
  - **R-squared ($R^2$)**: This metric indicates the proportion of variance in the dependent variable that is predictable from the independent variables, with higher values representing a better fit.
  - **Mean Squared Error (MSE)**: This measures the average of the squares of the errors, providing an insight into the average magnitude of the prediction errors.
  - **Results**:
    - The model achieves an $R^2$ score and MSE for both the training and testing sets, demonstrating its ability to not only fit the training data but also generalize to new data.

In [189]:
```
# Evaluate the model's performance
r2_train_rf = r2_score(targets_train, predictions_train_rf)
mse_train_rf = mean_squared_error(targets_train, predictions_train_rf)
r2_test_rf = r2_score(targets_test, predictions_test_rf)
mse_test_rf = mean_squared_error(targets_test, predictions_test_rf)

print(f'R2 score for training set (RF): {r2_train_rf}')
print(f'MSE for training set (RF): {mse_train_rf}')
print(f'R2 score for testing set (RF): {r2_test_rf}')
print(f'MSE for testing set (RF): {mse_test_rf}')
```

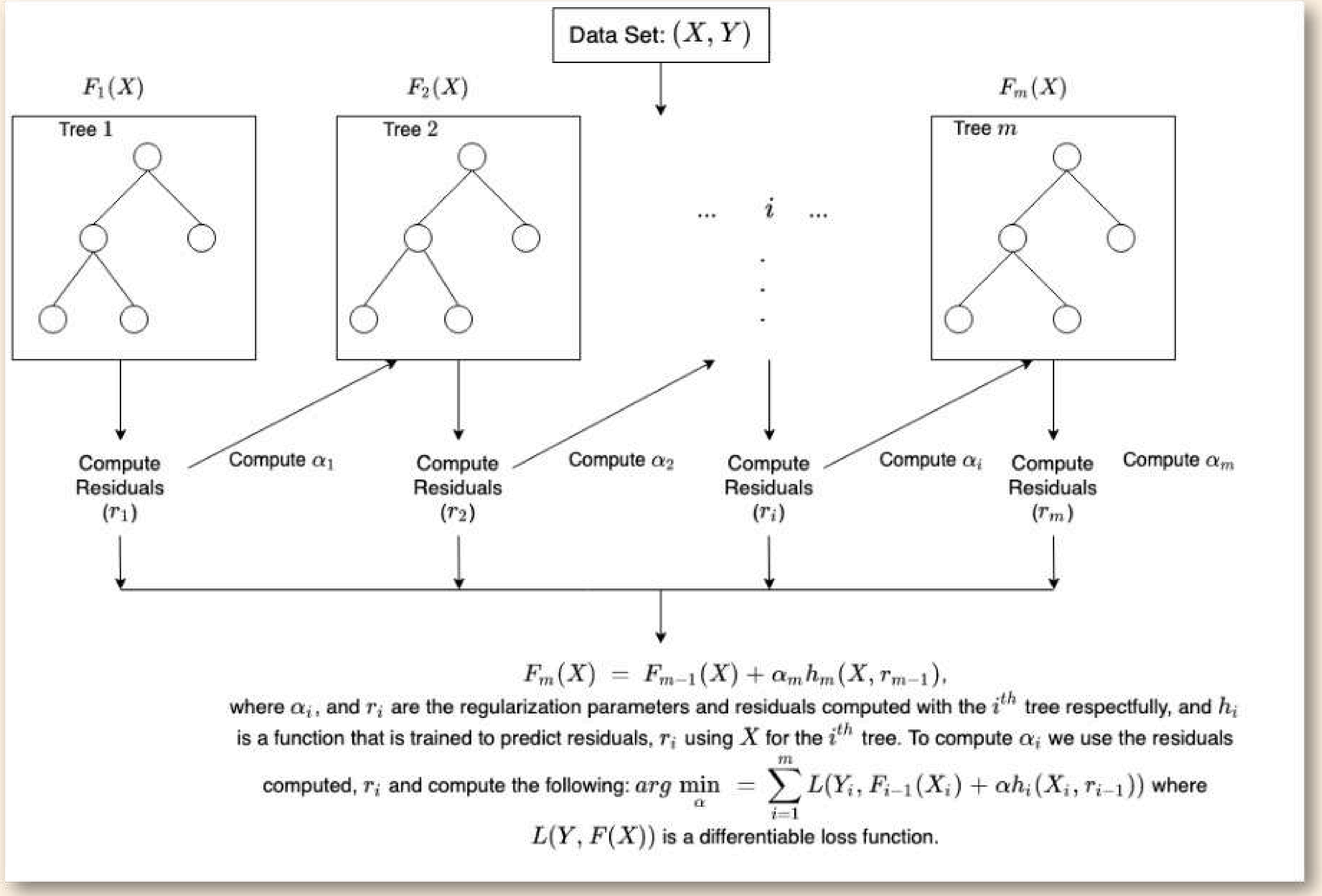
```
R2 score for training set (RF): 0.9975084799893854
MSE for training set (RF): 0.002027783855888463
R2 score for testing set (RF): 0.9759108334091994
MSE for testing set (RF): 0.011608361704545985
```

- Our model is initialized with 100 decision trees, n_estimators=100, allowing it to capture a broad spectrum of data variances without overfitting, thanks to the randomness in tree construction.
- The model's hyperparameters are tuned to optimize both bias and variance. The max_depth is set to control tree growth and min_samples_leaf ensures a sufficient sample size for stability in leaf nodes.
- Features are randomly sampled for each tree's split decision, thus each tree grows differently, increasing the model's generalization capability.

# RANDOM FOREST: ACTUAL VS PREDICTED



# XGBOOST:
## A POWERFUL GRADIENT BOOSTING FRAMEWORK




- XGBoost stands for Extreme Gradient Boosting, known for its performance and speed in machine learning tasks.
- XGBoost's ability to handle large datasets and its usage of advanced regularization techniques reduce overfitting, making it a leading algorithm for regression challenges.

# XGBOOST:
## PREPROCESSING AND DATA PREPARATION

- Prior to model training, categorical variables were encoded into numerical formats using Label Encoding, ensuring that our XGBoost model could process them effectively.

- The dataset is split into a training set (80%) and a test set (20%), a standard practice for evaluating the model's performance on unseen data."

In [191]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import xgboost as xgb
```

- **Data Preparation**:
  - **Encoding Categorical Variables**: All categorical variables are encoded using `LabelEncoder` to transform them into numerical values that can be processed by the model.
    - **Data Splitting**: The dataset is split into training and testing sets, with 80% of the data used for training and 20% for testing to evaluate model performance.

In [192]:
```python
# Encode categorical variables using Label Encoding
categorical_vars = data.select_dtypes(include=['object']).columns
for col in categorical_vars:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])

# Split the data into features and target
X = data.drop('resale_price', axis=1)
y = data['resale_price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

# XGBOOST:

## DMATRIX , PARAMETER TUNING & MODEL TRAINING

- XGBoost utilizes a DMatrix, an optimized data structure that boosts computation speed and enhances performance.
- The model was meticulously tuned with a max_depth of 3 and min_child_weight of 10 to manage complexity and support more robust leaf decisions.
- Training involved 100 rounds of boosting, with early stopping enabled to prevent overfitting and to optimize performance.

- **DMatrix Conversion**: The feature and target sets are converted into `DMatrix` , a data structure optimized for memory efficiency and training speed in XGBoost.

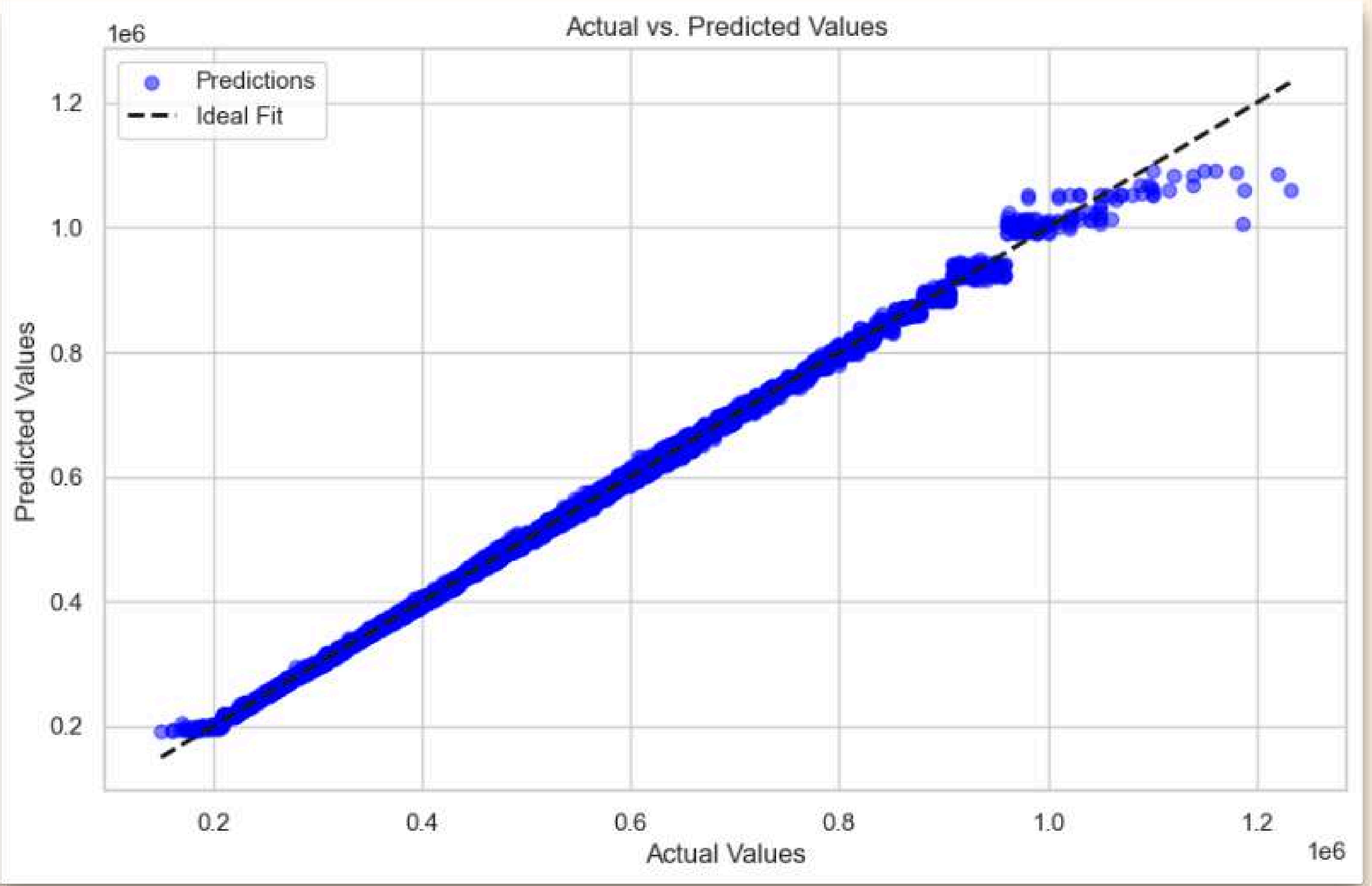
In [193]:
```
# Convert the dataset into an optimized data structure called Dmatrix
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)
```

- **Model Parameters**:
  - Configured with parameters like `max_depth` , `min_child_weight` , and learning rate ( `eta` ) to control the complexity and performance of the model.
- **Model Training**: The model is trained with 100 boosting rounds, allowing for iterative refinement of predictions.

In [200]:
```
# Assuming data is already preprocessed and split into X_train, X_test, y_train, y_test
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

params = {
    'max_depth': 3,  # Reduced from a higher value to limit tree complexity
    'min_child_weight': 10,  # Increased from a lower value to ensure more samples per leaf
    'eta': 0.1,  # Learning rate
    'subsample': 0.8,  # Subsample ratio of the training instances
    'colsample_bytree': 0.8,  # Subsample ratio of columns when constructing each tree
    'objective': 'reg:squarederror',  # Regression with squared error
    'eval_metric': 'rmse'  # Root mean squared error as evaluation metric
}

# Train the model
num_boost_round = 100
early_stopping_rounds = 10
evals = [(dtrain, 'train'), (dtest, 'test')]
model = xgb.train(params, dtrain, num_boost_round=num_boost_round,
                  early_stopping_rounds=early_stopping_rounds, evals=evals)

# Make predictions
predictions = model.predict(dtest)
```

# XGBOOST:

## VISUALIZING PREDICTION ACCURACY

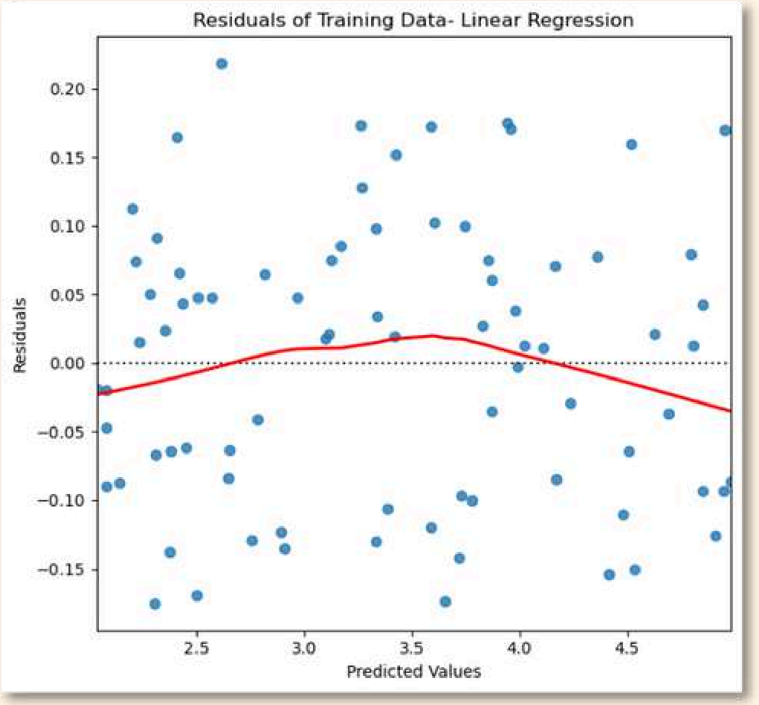



```
# Calculate metrics
mse = mean_squared_error(y_test, predictions)
r2 = r2_score(y_test, predictions)

print(f"MSE: {mse}")
print(f"R2 score: {r2}")
```

```
MSE: 19619115.783947933
R2 score: 0.999037834379737
```

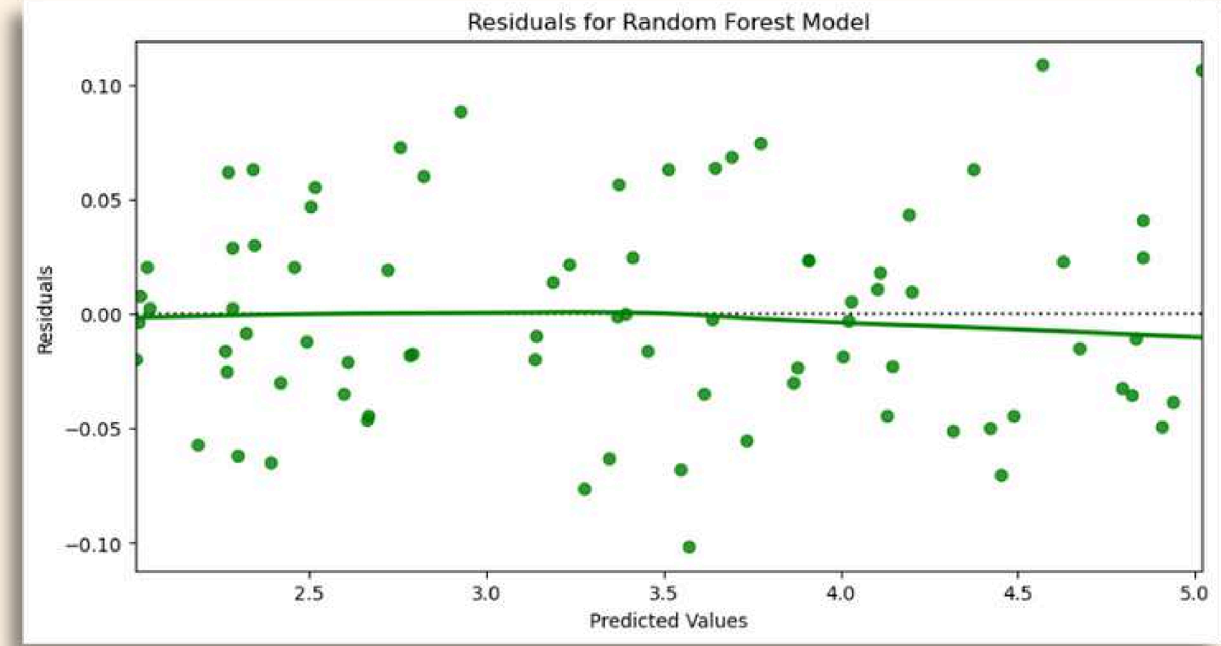
The proximity of data points to the 'ideal fit' line, where actual and predicted values match, highlights the accuracy of the XGBoost model.

# MODEL COMPARISION



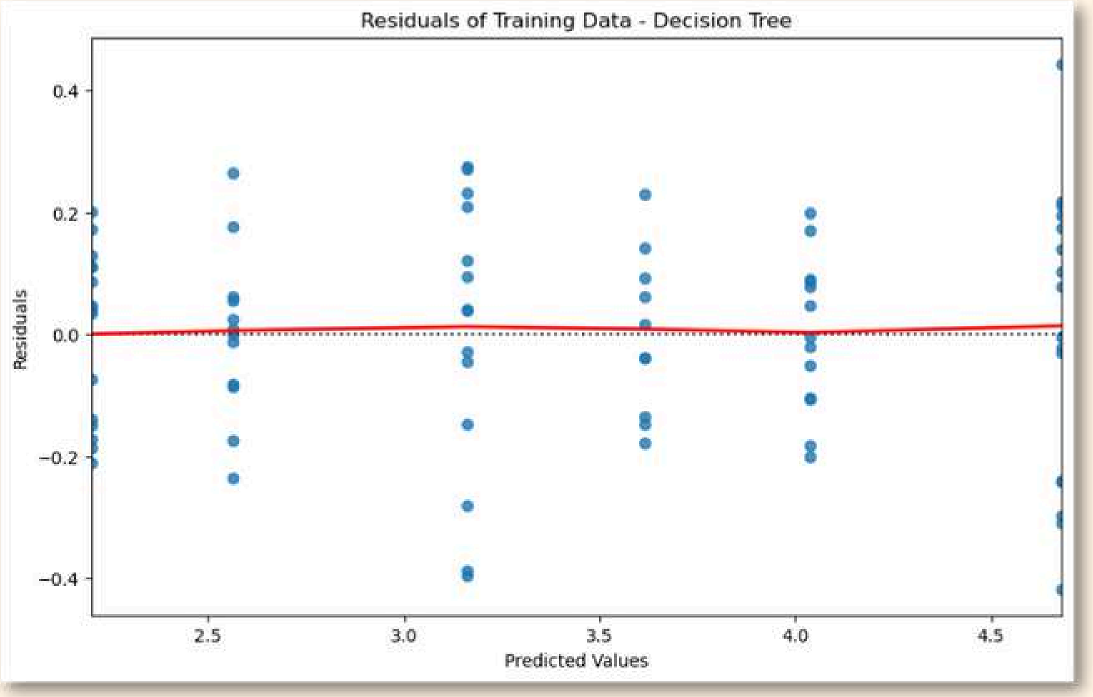

Linear Regression

R^2 score: 0.988



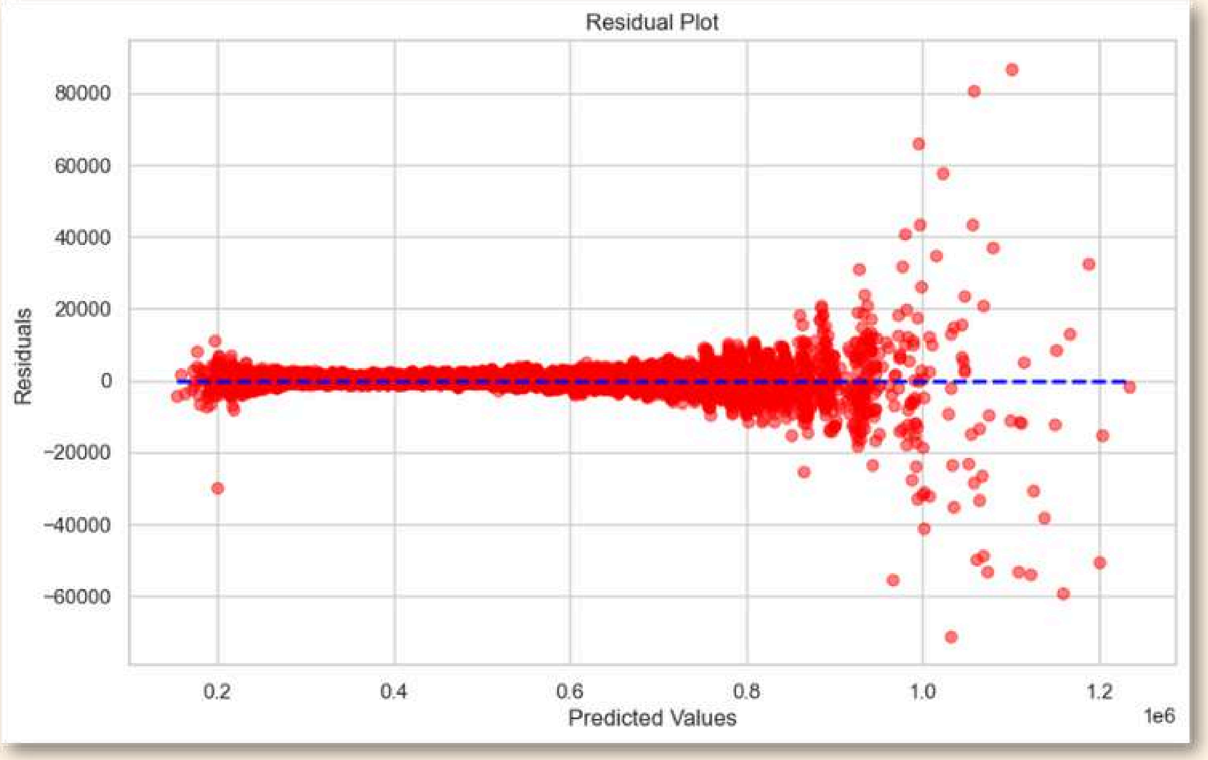

Random Forest

R^2 score: 0.963




Decision Trees

R^2 score: 0.988




**XGBOOST**

best model

R^2 score: 0.999

# BENEFITS

## OF OUR MODEL

**Scalability & Adaptability**

**Increased Accuracy and Predictive Power**

**Time & Cost Efficiency**

**Stay Ahead of Market Fluctuations**

# EFFORTLESS SOLUTIONS

- Data-Driven Roadmap for Homebuyers, Sellers, and Policymakers
- Pricing Strategies and Market Value Assessment
- Informed Investment Decisions

# THANK YOU!