Title: House Price Prediction using Machine Learning Model

Puja Dhungana (pujadhungana@my.unt.edu), Ya Meng (yameng1@my.unt.edu)

1. Abstract

House is one of human life's most needs, demand for houses has grown rapidly over the years as people's living standards have improved. Buying a house is a big decision and investment. A proper analysis of the housing market is always necessary before making this big decision. There are numerous factors influencing the house price, such as location, property size and features including the number of bedrooms, bathrooms, square footage, lot size, and floor plan layout. Therefore, most stakeholders including buyers and developers, house builders and the real estate industry would like to know the exact attributes or the accurate factors influencing the house price to help investors make decisions and help house builders set the house price.

House price prediction can be accomplished through the utilization of various machine learning models. Employing these models offers numerous benefits to home buyers, property investors, and house builders. By leveraging a house-price model, stakeholders can gain valuable insights into market trends and dynamics, enabling them to make informed decisions. Home buyers can benefit from accurate price estimates, aiding them in budgeting and identifying suitable properties. Property investors can leverage the model to evaluate investment opportunities and optimize their portfolios. House builders can utilize the model to set competitive prices and tailor their offerings to meet market demands. Overall, the adoption of house-price prediction models empowers stakeholders to enhance their decision-making processes and maximize their outcomes in the real estate industry.

In this project, our goal is to predict house prices using various regression methods. We will explore multiple machines learning models, including multiple linear regression, K-Nearest Neighbors (KNN), Random forest and Decision Tree. The performance of each model will be compared based on the accuracy metric. By covering various regression models and comparing their performances, we aim to identify the most accurate model for house price prediction. This approach allows us to leverage the strengths of each model and gain insights into their individual performance characteristics.

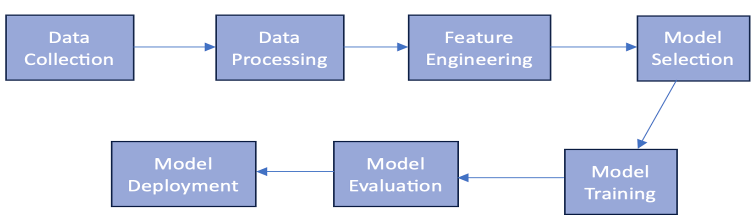


Fig: Block Diagram of machine learning

2. Data specification

Dataset and Feature:

Source of data: https://www.kaggle.com/datasets/ahmedshahriarsakib/usa-real-estate-dataset

The project will utilize the Realtor dataset from the USA Real Estate Dataset available on Kaggle. The dataset contains 306,000 rows and 10 columns, representing the following features:

1. Status - Indicates whether the house is for sale or off the market.
2. Price - House price, which serves as the target variable for prediction.
3. Bed - Number of bedrooms in the property.
4. Bath- Number of bathdrooms in the property.
5. Acre Lot - Size of the property in acres.
6. City - The name of the city where the property is situated.
7. State - The name of the state where the property is located.
8. Zip Code - The zip code corresponding to the property's location.
9. House Size - The size of the property in square feet.
10. Previous Sold Date - The date on which the property was previously sold.

The supervised learning problem that can be solved using this dataset is to predict the house price (target variable) based on the given set of features. Since the dataset includes labeled instances with the target variable present, it is suitable for supervised learning tasks.The task is regression since the goal is to predict a continuous numerical value (house price) based on the given features. Various regression algorithms can be applied to build a predictive model that learns the relationship between the features and the target variable.

3. Project Design

Framwork/Libraries: To implement the project, we will utilize the Pandas and scikit-learn machine learning frameworks. These frameworks offer a wide range of tools and algorithms for data preprocessing, feature engineering, model training, and evaluation.

Tools: Google collabatory/ Jupyter notebook

Data Preprocessing:

(1) Checking and handling missing values:

To check the number of NaN values for each column in the DataFrame, we have used the code: *null_counts = new_dataframe.isnull().sum()*. To drop any rows from the DataFrame that contain at least one NaN value in any column, you used the code: *new_dataframe2 = new_dataframe.dropna(how='any')*.

(2) Handling categorical feature "Status":

Since the "Status" feature has two categories, "for_sale" and "ready_to_built", we have used the code: *new_dataframe3.loc[:, 'status'] = new_dataframe3['status'].apply(lambda x: 1 if x == 'for_sale' else 0)*. This code assigns the value 1 to "for_sale" and 0 to "ready_to_built" in the "status" column of the DataFrame "new_dataframe3".

(3) Scaling numerical features and converting categorical features:

To scale numerical features, we have used the StandardScaler from scikit-learn. To convert the categorical feature "State" into multiple binary features, we used the OneHotEncoder with the arguments sparse=False and handle_unknown='ignore'.

Model implementation:

We have extracted "price" as target variable and the selected features for the models were 'status', 'state', 'bed', 'bath', 'acre_lot', and 'house_size'.We have split the preprocessed data into train and test sets by using "X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y, test_size=0.3, random_state=42)".

(1) Firstly, we implemented and trained a random forest regression model.

To do this, we have initialized a Random Forest Regressor model with the RandomForestRegressor class from scikit-learn. Then we fitted the Random Forest Regressor model to the training data (X_train and y_train), allowing the model to learn the patterns and relationships between the features and the target variable. We have used the trained Random Forest Regressor model to make predictions on the test data (X_test). We then have calculated the Mean Squared Error (MSE) and R-squared score for evaluating the performance of the Random Forest Regressor model. The mean_squared_error function from scikit-learn's metrics module is used to compute the MSE between the actual house prices (y_test) and the predicted house prices (predictions_df). The r2_score function from scikit-learn's metrics module is used to compute the R-squared score. In this case, the R-squared score of 0.9307 and the MSE of 65322711200.162506 suggest that the Random Forest Regressor model has performed well in predicting house prices.

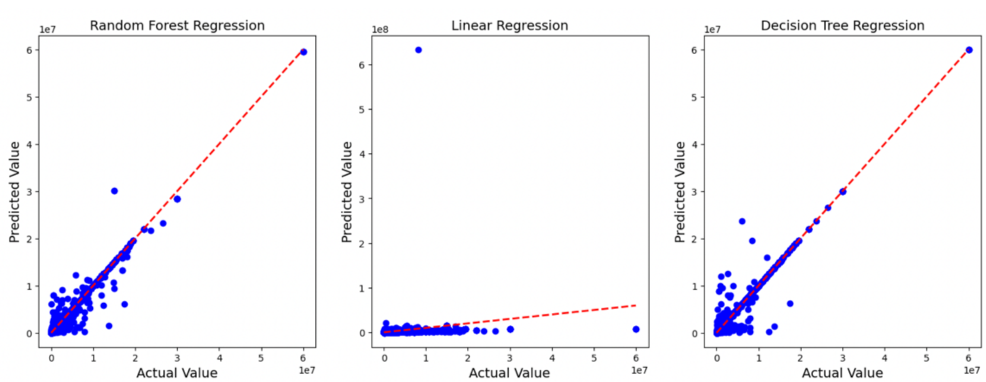
(2) Next, we used a Linear Regression model for predicting house prices.

To do this, we firstly initialized a Linear Regression model with the LinearRegression class from scikit-learn. We fitted the Linear Regression model to the training data (X_train and y_train), allowing the model to learn the linear relationship between the features and the target variable. After that, we have used the trained Linear Regression model to make predictions on the test data (X_test). In this case, the MSE suggests that, on average, the squared difference between the predicted and actual house prices is approximately 751134915107.097. A score of 0.2029850807744169 suggests that the Linear Regression model explains only about 20.30% of the variance in the house prices. The results suggest that the Linear Regression model may not be a good fit for the data, as it has relatively high error and low predictive power.

(3) Then, we implemented and trained a decision tree regression model.

To do this, we firstly initialized a Decision Tree model using the DecisionTreeRegressor class from scikit-learn. We fitted the Decision Tree to the training data, allowing the model to learn the relationship between the features and the target variable. After that, we have used the trained model to make predictions on the test data (X_test). The R-squared score measures the proportion of the variance in the target variable (house prices) that can be explained by the decision tree regression model. In this case, a score of 0.8664297396592684 suggests that the model explains approximately 86.64% of the variance in the house prices.

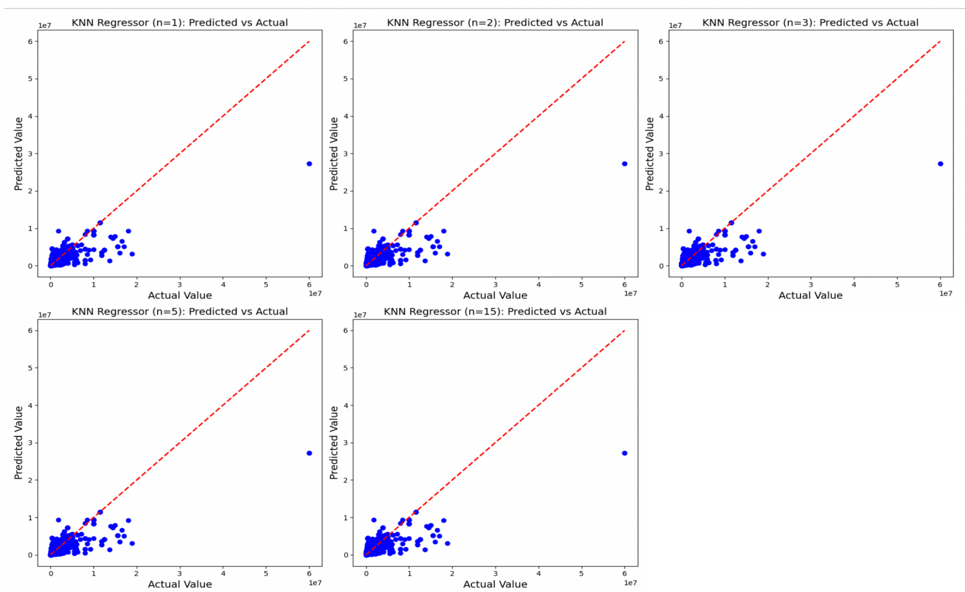
We have used Matplotlib to create a scatter plot comparing the actual house prices (y_test) with the predicted house prices (predictions_df). The scatter plot shows the relationship between the actual and predicted values, while the red dashed line represents a perfect prediction. The following picture is the screenshot the scatter plot about the between the actual and predicted values for random forest regression, linear regression, and decision tree regression.



(4) Last, we implemented and trained a KNN regression model.

Before implementing KNN, we firstly have created a list called neighbor_values that contains different values for the number of neighbors to consider in the KNN algorithm. In this project, we have tried k = 1, 2, 3, 5, 15. We have used a loop to go through neighbor values and fitting models. This loop iterates over each value in neighbor_values. For each iteration, it creates a new instance of the KNeighborsRegressor class with the specified number of neighbors (n) and fits the model to the training data (X_train and y_train). After fitting the model, it uses the trained model to predict house prices using the test data (X_test). Next, we have calculated the MSE between the predicted prices (predictions3_df) and the actual prices (y_test) using the mean_squared_error function. The RMSE is then computed by taking the square root of the MSE. The R-squared score is calculated using the r2_score function, which compares the predicted prices with the actual prices and measures how well the model fits the data.

The following pictures are the model accuracy scores and scatter plots between the actual and predicted values for different n in KNN model.



The Reasult for KNN-model with different values of n:

4. Project Results

The following picture is the screenshot of the performance metrics of different trained t regression models for house price prediction. From the results, we can see Random Forest and KNN models have similar performance with relatively low MSE and RMSE values. Additionally, both models achieve high R-squared values, indicating a good fit to the data. On the other hand, the Linear Regression model seems to perform poorly with a high MSE and RMSE, along with a negative R-squared value, indicating a poor fit.

	Model	MSE	RMSE	R-squared
0	Linear Regression	8.201974e+12	2.863909e+06	-3.218193
1	Random Forest	3.662134e+10	1.913670e+05	0.981166
2	Decision Tree	4.129967e+10	2.032232e+05	0.978760
3	KNN	3.662134e+10	1.913670e+05	0.981166

The following picture is the screenshot of the performance of the KNN model for different values of 'n'. It can be seen as the number of neighbors increases from 1 to 15, the MSE and RMSE values tend to increase, indicating a higher prediction error. On the other hand, the R-squared value decreases as the number of neighbors increases, suggesting a weaker fit to the data. From these results, it seems that using a smaller number of neighbors (e.g., 1 or 2) yields better performance in terms of lower MSE and RMSE, as well as a higher R-squared value.

	n	MSE	RMSE	R-squared
0	1.0	7.537074e+10	274537.328435	0.929075
1	2.0	8.931111e+10	298849.646770	0.915957
2	3.0	9.678103e+10	311096.503603	0.908928
3	5.0	1.256905e+11	354528.625725	0.881724
4	15.0	3.335672e+11	577552.784141	0.686109

5. Project Milestones

(1)Milestone: Data Collection and Preprocessing
Acquire a suitable dataset and preprocess data for house prices.
(2) Milestone: Exploratory Data Analysis and Feature Selection
Explore the dataset and select relevant features for the regression models.
(3) Milestone: Model Selection and Training
Implement and train a random forest model.
Evaluate the model's performance using metrics including mean squared error (MSE) and R-squared.
(4) Milestone: Model Evaluation and Improvement
Implement linear regression and evaluate the model's performance using metrics including MSE and R-squared.
Implement decision tree model and evaluate the model's performance using MSE and R-squared.
Explore KNN regression and select the best n value according to the model's performance .
(5) Milestone: Prediction and Model Deployment
Use the trained regression models to make house price predictions on new data samples.
(6) Milestone: Documentation and Reporting
Compile and organize project documentation, including data preprocessing steps, model architectures, and training details for each regression model.
Create a comprehensive report summarizing the project's goals, methodology, and results.
Prepare a presentation to communicate the project's key findings and insights to stakeholders.

6. Archive

The dataset (realtor_data.csv) and source code(house_price_prediction_group5.ipynb) are submitted with project report.

7. Reference Material

(1) We have referred the following research paper about the background introduction and methodology of house price prediction project.
Zulkifley, Nor Hamizah, et al. "House Price Prediction using a Machine Learning Model: A Survey of Literature." International Journal of Modern Education & Computer Science 12.6 (2020).

(2) For the codes how to implement different machine learning regression models, we will reference the sklearn tutorial: https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

(3) For those of which values are categorical variables, we will use the following references to encode them. https://stackoverflow.com/questions/37292872/how-can-i-one-hot-encode-in-python
https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html

In [3]:
```
import pandas as pd
import numpy as np
```

In [4]:
```
#load data from csv file
dataframe = pd.read_csv('realtor_data.csv')
#dataframe.head(10)
```

In [5]:
```
#new dataframe with only necessary features
new_dataframe = dataframe[['status', 'bed', 'bath', 'acre_lot', 'state', 'house
```

In [6]:
```
# the number of NaN for each column in the DataFrame
null_counts = new_dataframe.isnull().sum()
null_counts
```

Out[6]:
```
status            0
bed           24950
bath          24888
acre_lot      14013
state             0
house_size    24918
price             0
dtype: int64
```

In [7]:
```
#non-null values for each column in the DataFrame
not_null_count=new_dataframe.count()
not_null_count
```

Out[7]:
```
status        100000
bed            75050
bath           75112
acre_lot       85987
state         100000
house_size     75082
price         100000
dtype: int64
```

In [8]:
```
house_count_by_state = new_dataframe['state'].value_counts()
print(house_count_by_state)
```

```
Massachusetts     52694
Puerto Rico       24679
Connecticut       12178
Virgin Islands     2573
Rhode Island       2401
New Hampshire      2232
New York           1874
Vermont            1324
South Carolina       24
Tennessee            16
Virginia              3
New Jersey            2
Name: state, dtype: int64
```

In [9]:
```
#drops any rows from the dataframe that contain at least one NaN in any column
new_dataframe2 = new_dataframe.dropna(how='any')
#new_dataframe2
```

In [10]:
```
# change for_sale = 1 and ready_to_built = 0 i.e. binary data
new_dataframe3 = pd.DataFrame(new_dataframe2)
new_dataframe3.loc[:, 'status'] = new_dataframe3['status'].apply(lambda x: 1 if

#new_dataframe3.loc[73008:73021, 'status'] = new_dataframe3.loc[73008:73021, 's
#dataframe_subset = new_dataframe3.loc[73008:73021]
#print(dataframe_subset)
```

In [11]:
```
# Extract features and target variable
X = new_dataframe3.drop('price', axis=1)
y = new_dataframe3['price'] #target
```

In [12]:
```
# Preprocessing steps

# Define the preprocessing steps for numerical and categorical features
numerical_features = ['bed', 'bath', 'acre_lot', 'house_size', 'status']
categorical_features = ['state']
#categorical_features = ['status',   'state']
```

In [13]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import mean_squared_error,mean_absolute_error, r2_score
from sklearn.ensemble import RandomForestRegressor
```

In [14]:
```
#make numerical features have similar scale values
numerical_transformer = StandardScaler()

#convert each categorical feature into multiple binary features
categorical_transformer = OneHotEncoder(sparse=False, handle_unknown='ignore')
```

In [15]:
```
# Preprocessing class
preprocessor = ColumnTransformer(
    transformers=[
        #numericla transform for column specified in numericla_features
        ('num', numerical_transformer, numerical_features),
        #categotical transform of column specified in categorical_features
        ('cat', categorical_transformer, categorical_features)
    ])
```

In [16]:
```
# Preprocess the data
X_preprocessed = preprocessor.fit_transform(X)
```

In [17]:
```
# Split the preprocessed data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y, test_siz
```

RANDOM FOREST REGRESSION

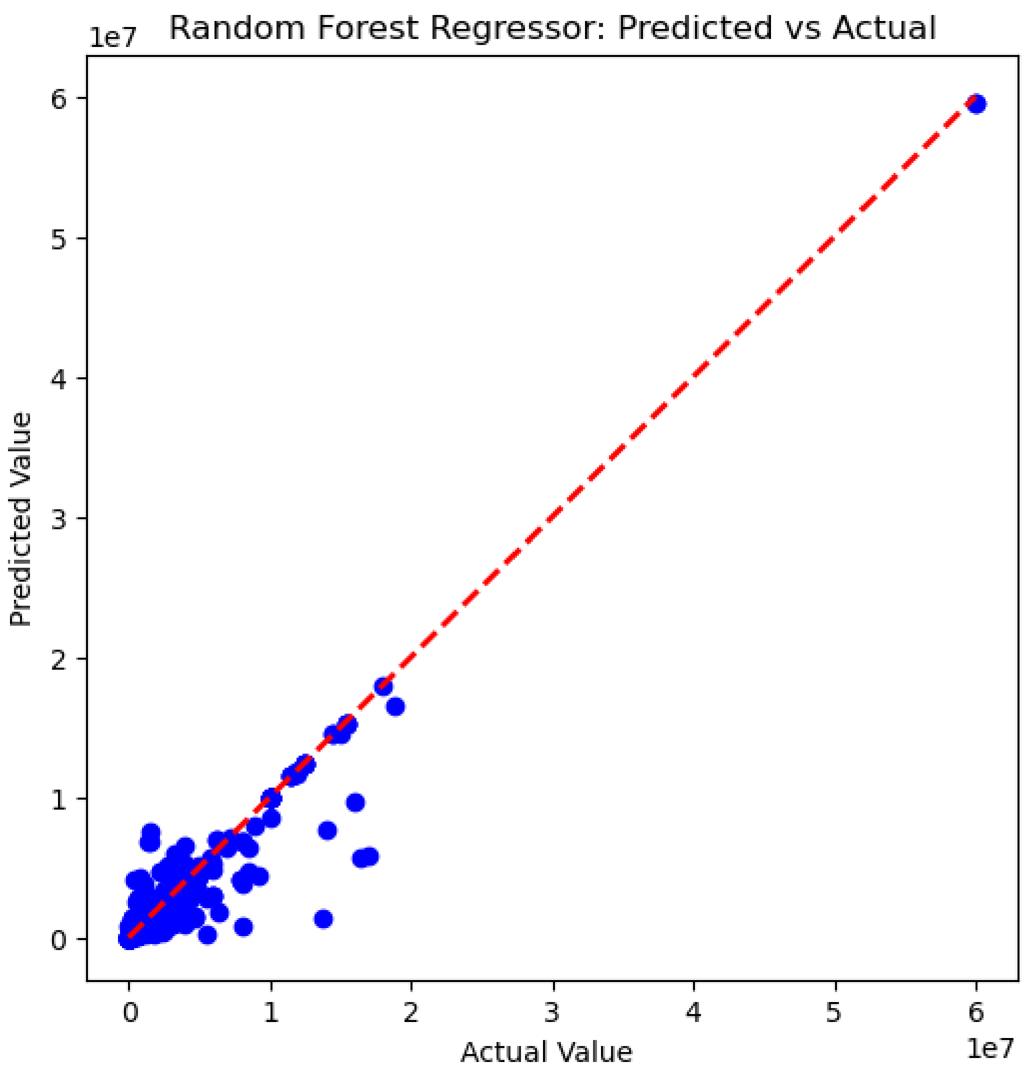
In [18]:
```
# random forest regressor
model = RandomForestRegressor(random_state=42)
```

In [19]:
```
# Fit the model to the training data
model.fit(X_train, y_train)
```

Out[19]:
```
RandomForestRegressor(random_state=42)
```

In [20]:
```
# Predict house prices using the trained model
predictions = model.predict(X_test)
predictions_df = pd.DataFrame(predictions)
#predictions_df
```

In [21]:
```
import matplotlib.pyplot as plt

# Plot of predicted values in  linear plot
plt.figure(figsize=(6, 6))
plt.scatter(y_test, predictions_df, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='rec
plt.xlabel('Actual Value ')
plt.ylabel('Predicted Value')
plt.title('Random Forest Regressor: Predicted vs Actual')
plt.show()
```



In [22]:
```
#mean square error
mse = mean_squared_error(y_test, predictions_df)
print('Mean Squared Error for random forest regression:', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error for linear regrression is:', rmse)
```

```
Mean Squared Error for random forest regression: 60942947872.3715
Root Mean Squared Error for linear regrression is: 246866.25502966478
```

In [23]:
```
#Calculate R-squared score
r2 = r2_score(y_test, predictions_df)
print('R-squared for random forest regression:', r2)
```

```
R-squared for random forest regression: 0.9426519562456948
```

LINEAR REGRESSION MODEL

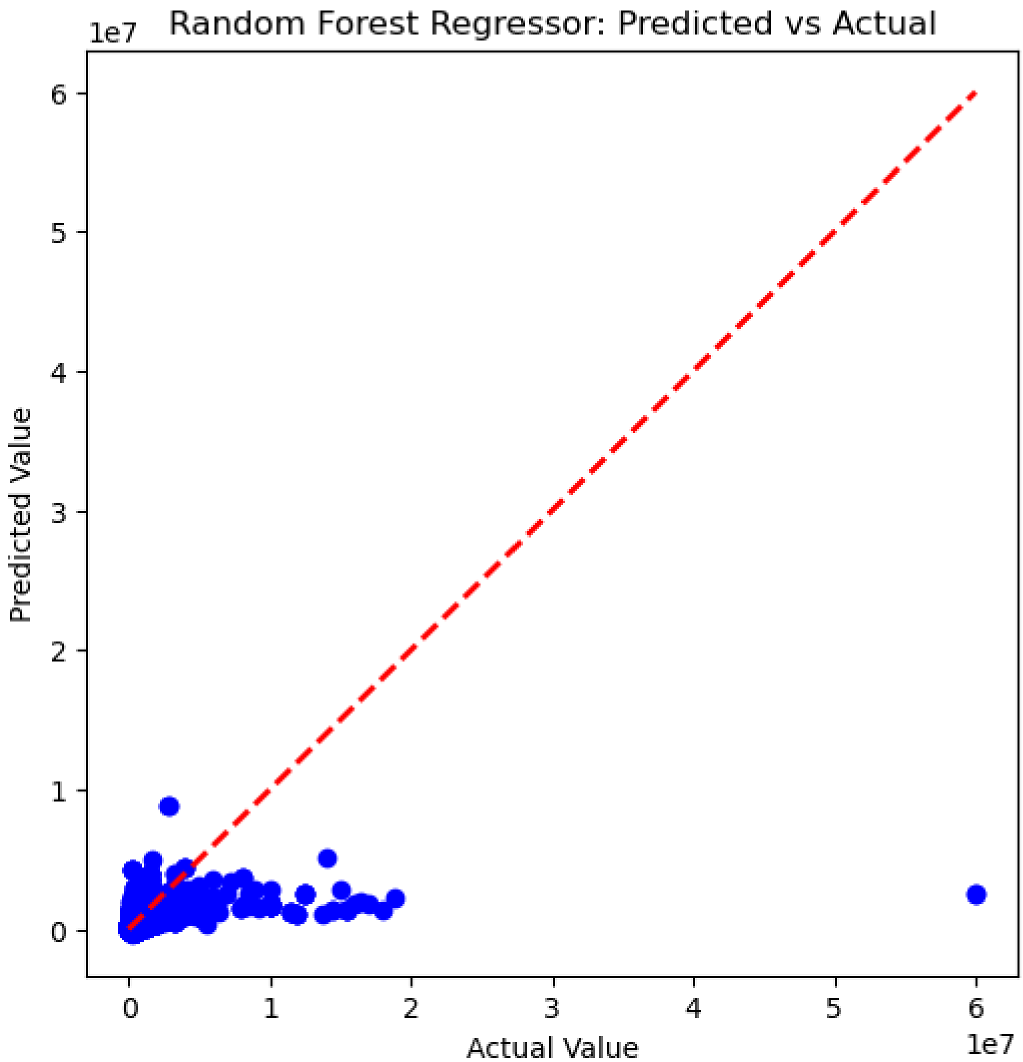
In [24]:
```
# linear regression model
model2 = LinearRegression()
```

In [25]:
```
# Fit the model to the training dataset
model2.fit(X_train, y_train)
```

Out[25]:
```
LinearRegression()
```

In [26]:
```
# Predict house prices using the trained model
predictions2 = model2.predict(X_test)
predictions2_df = pd.DataFrame(predictions2)
#predictions2_df
```

In [27]:
```
#  Plot of predicted values in  linear plot
plt.figure(figsize=(6, 6))
plt.scatter(y_test, predictions2_df, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='rec
plt.xlabel('Actual Value')
plt.ylabel('Predicted Value')
plt.title('Random Forest Regressor: Predicted vs Actual')
plt.show()
```



Here, red dotted line is perfect prediction line and blue dots are predicted values

In [28]:

```
#mean square error
mse2 = mean_squared_error(y_test, predictions2_df)
print('Mean Squared Error for linear regression model :', mse2)
rmse2 = np.sqrt(mse2)
print('Root Mean Squared Error for linear regrression is:', rmse2)
```

```
Mean Squared Error for linear regression model : 870292089655.7306
Root Mean Squared Error for linear regrression is: 932894.4686596285
```

In [29]:

```
# R-squared score
r2_2 = r2_score(y_test, predictions2_df)
print('R-squared for linear regression model:', r2_2)
```

```
R-squared for linear regression model: 0.18104472167764807
```

DECISION TREE

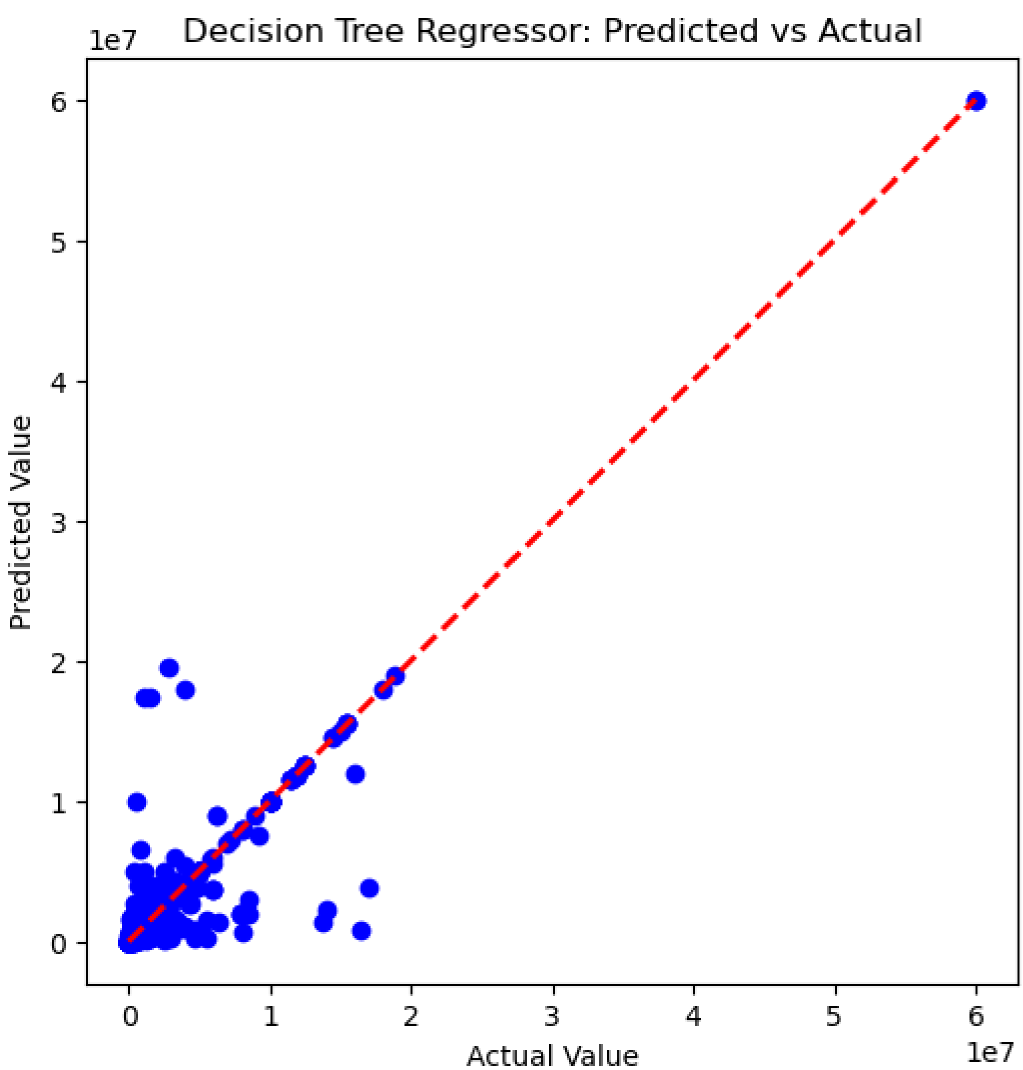
In [30]:
```
from sklearn.tree import DecisionTreeRegressor

# Create the decision tree regressor
model4 = DecisionTreeRegressor(random_state=42)
```

In [31]:
```
# Fit the model to the training data
model4.fit(X_train, y_train)
```

Out[31]:
```
DecisionTreeRegressor(random_state=42)
```

In [32]:
```
# Predict house prices using the trained model
predictions4 = model4.predict(X_test)
predictions4_df = pd.DataFrame(predictions4)
```

In [33]:
```
# Plot of predicted values in  linear plot
plt.figure(figsize=(6, 6))
plt.scatter(y_test, predictions4_df, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='rec
plt.xlabel('Actual Value')
plt.ylabel('Predicted Value')
plt.title('Decision Tree Regressor: Predicted vs Actual')
plt.show()
```



In [34]:
```
mse4 = mean_squared_error(y_test, predictions4_df)
print('Mean Squared Error for decision tree is:', mse4)

rmse4 = np.sqrt(mse4)
print('Root Mean Squared Error for decision tree is:', rmse4)

mae4 = mean_absolute_error(y_test, predictions4_df)
print('Mean Absolute Error for decision tree is:', mae4)

r2_4= r2_score(y_test, predictions4_df)
print('R-squared for decision tree is:', r2_4)
```

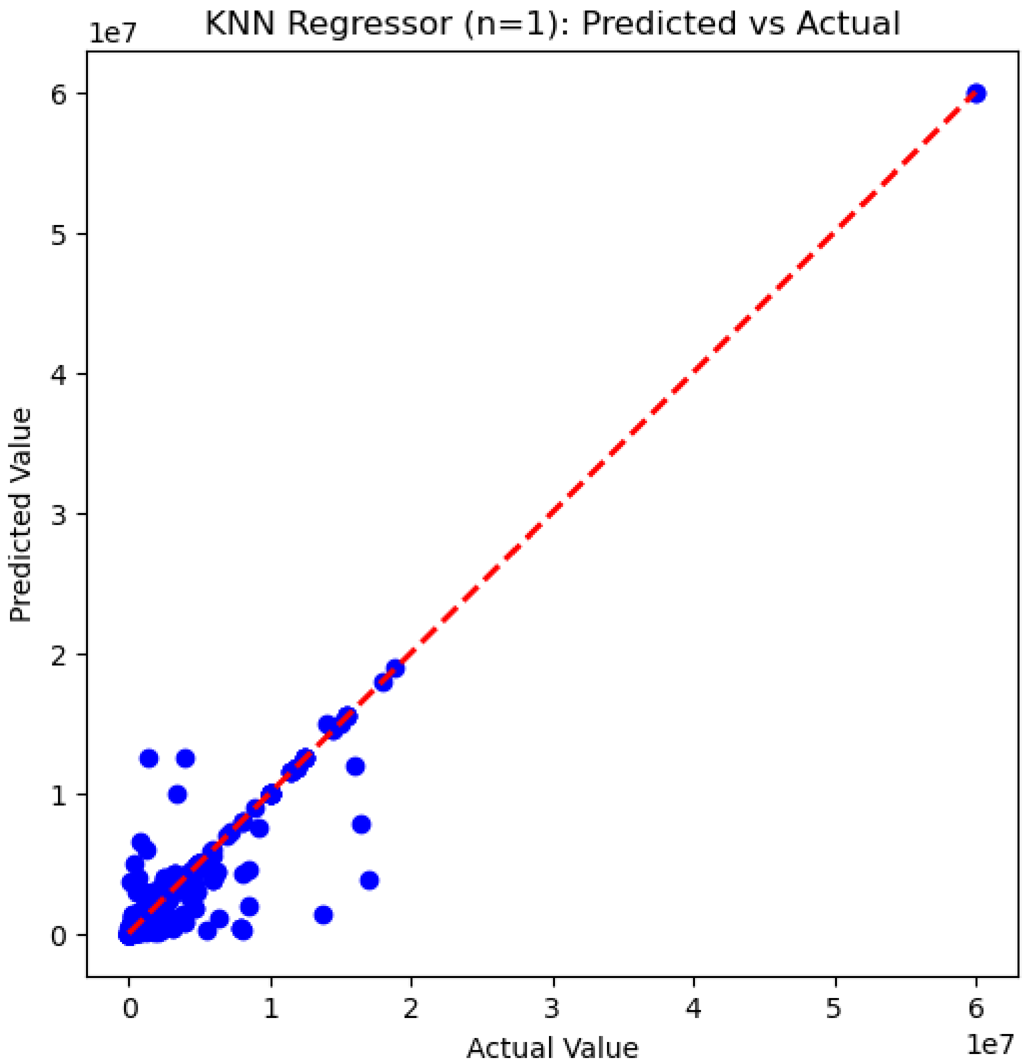
```
Mean Squared Error for decision tree is: 141943209922.19012
Root Mean Squared Error for decision tree is: 376753.5134835376
Mean Absolute Error for decision tree is: 30359.651683723034
R-squared for decision tree is: 0.8664297396592684
```

KNN model

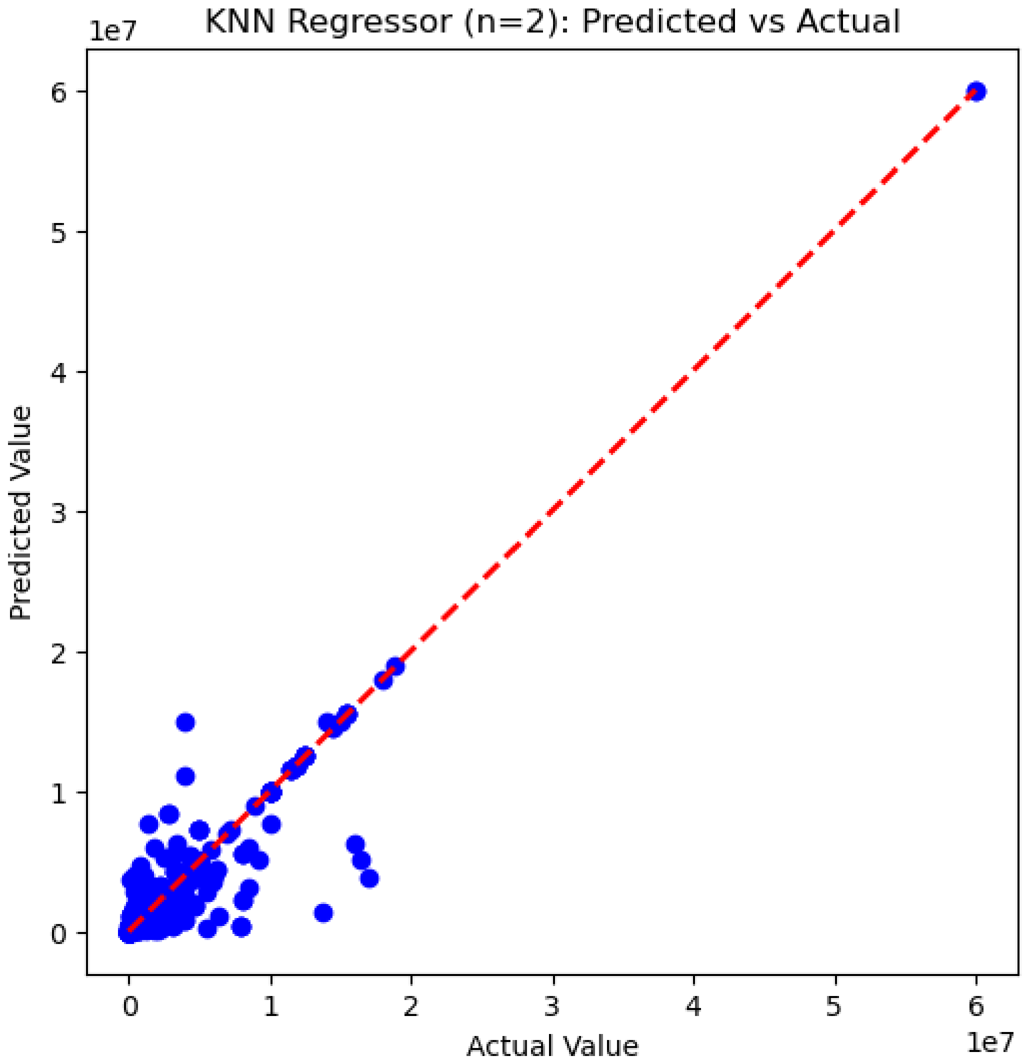
In [35]:
```
from sklearn.neighbors import KNeighborsRegressor

# Define neighbor values
neighbor_values = [1, 2, 3, 5, 15]
```

In [36]:
```
for n in neighbor_values:
    # Create the KNN regressor
    model3 = KNeighborsRegressor(n_neighbors=n)

    # Fit the model to the training data
    model3.fit(X_train, y_train)

    # Predict house prices using the trained model
    predictions3 = model3.predict(X_test)
    predictions3_df = pd.DataFrame(predictions3)

    #mean square error
    mse_knn = mean_squared_error(y_test, predictions3_df)
    print(f'Mean Squared Error for KNN model with (n={n})is :', mse_knn)

    #root mean square error
    rmse_knn = np.sqrt(mse_knn)
    print('Root Mean Squared Error for decision tree is:', rmse_knn)

    # R-squared score
    r2_knn = r2_score(y_test, predictions3_df)
    print(f'R-squared for KNN model with (n={n}):', r2_knn)



    #  Plot of predicted values in  linear plot
    plt.figure(figsize=(6, 6))
    plt.scatter(y_test, predictions3_df, color='blue')
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color=
    plt.xlabel('Actual Value')
    plt.ylabel('Predicted Value')
    plt.title(f'KNN Regressor (n={n}): Predicted vs Actual')
    plt.show()
```

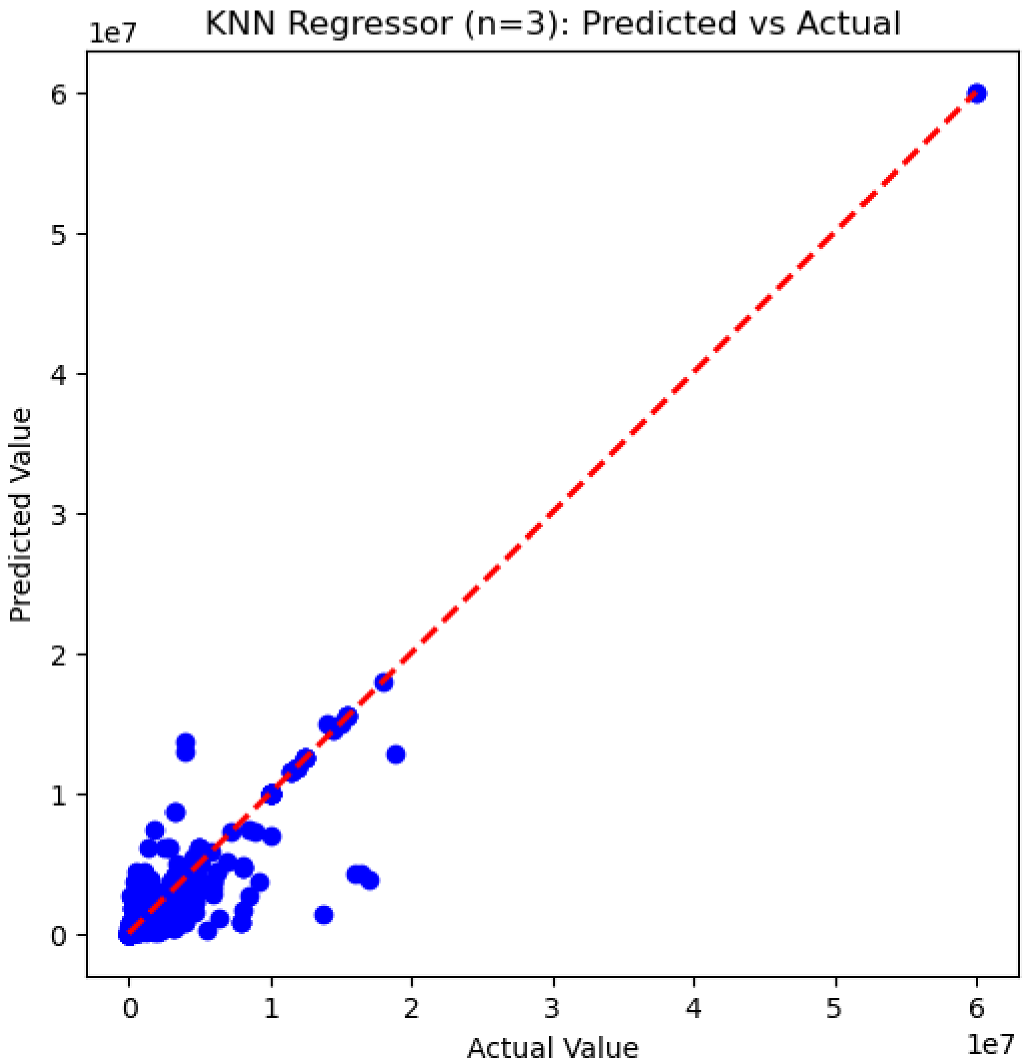
```
Mean Squared Error for KNN model with (n=1)is : 75370744704.4625
Root Mean Squared Error for decision tree is: 274537.3284354288
R-squared for KNN model with (n=1): 0.9290752266503729
```



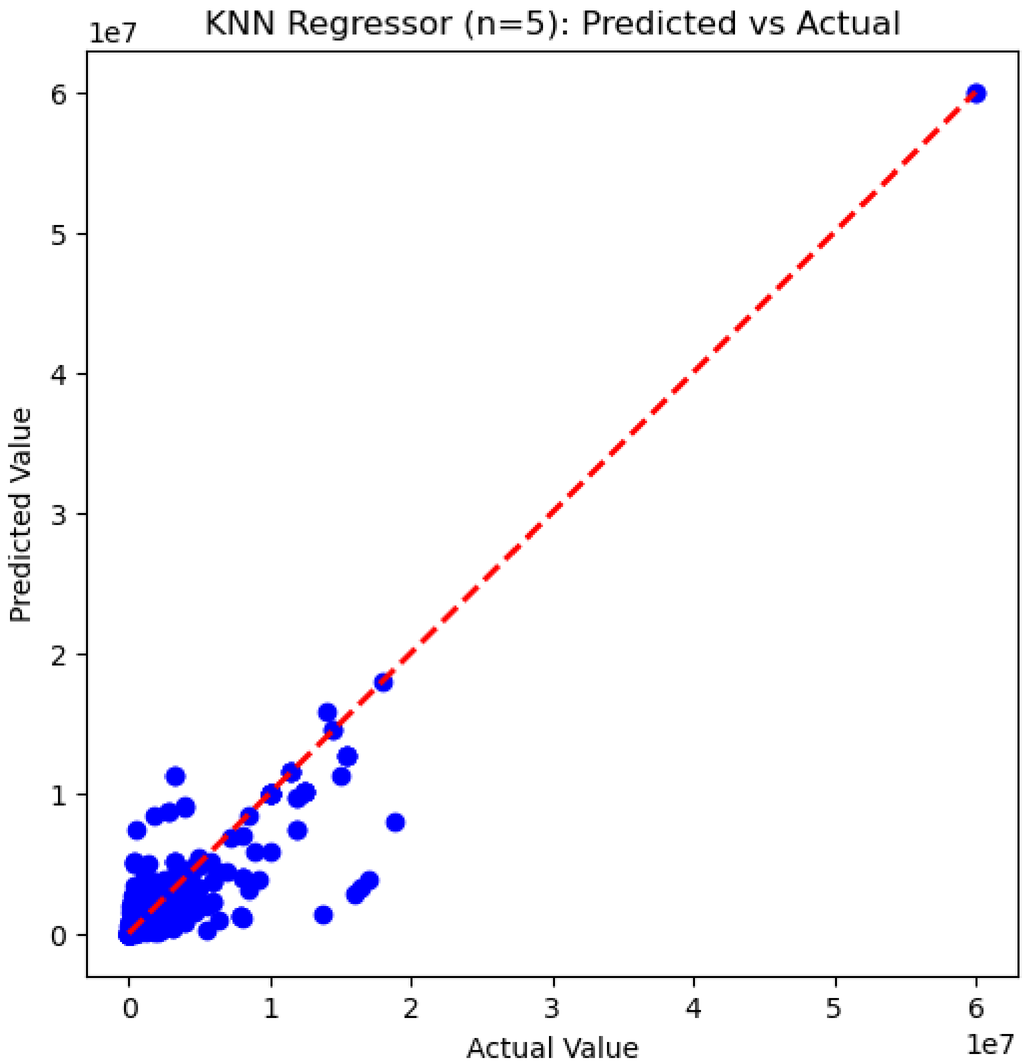

```
Mean Squared Error for KNN model with (n=2)is : 89311111374.74884
Root Mean Squared Error for decision tree is: 298849.64677032636
R-squared for KNN model with (n=2): 0.9159571746744023
```



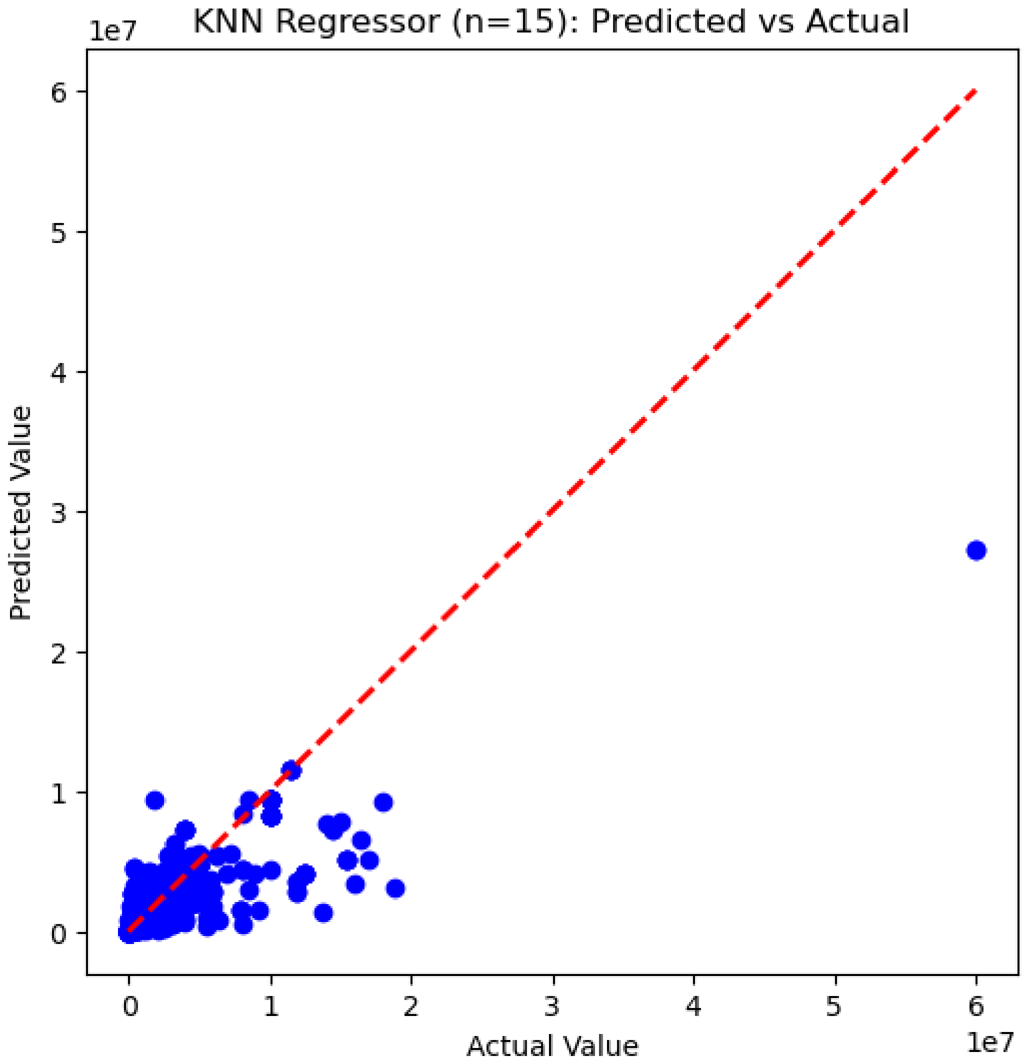

```
Mean Squared Error for KNN model with (n=3)is : 96781034553.86751
Root Mean Squared Error for decision tree is: 311096.50360276876
R-squared for KNN model with (n=3): 0.9089278875087317
```




```
Mean Squared Error for KNN model with (n=5)is : 125690546458.12044
Root Mean Squared Error for decision tree is: 354528.62572452513
R-squared for KNN model with (n=5): 0.8817236906084974
```




```
Mean Squared Error for KNN model with (n=15)is : 333567218468.54047
Root Mean Squared Error for decision tree is: 577552.7841405844
R-squared for KNN model with (n=15): 0.6861092528737343
```



Predicting the price of new house data samples created randomly

In [37]:
```
#Create new random data sample
sampleDF_test = pd.DataFrame(columns=['status', 'bed', 'bath', 'acre_lot', 'hou
states_test = ['Connecticut', 'Massachusetts', 'New Hampshire','New York','Puer

#'New Jersey','Tennessee', 'Virgin' excluding these states as they have less nu

empty_df = []
for i in range(50):
    row = {
        'status': np.random.choice([0, 1]),
        'bed': np.random.randint(1, 5),
        'bath': np.random.randint(1, 4),
        'acre_lot': np.random.randint(0.15,10.67),
        'state': np.random.choice(states_test),
        'house_size': np.random.randint(181,9999)

    }
    empty_df.append(pd.DataFrame(row, index=[i]))
```

```
sampleDF_test = pd.concat(empty_df, ignore_index=True)
```

In [38]:
```
states_test
```

Out[38]:
```
['Connecticut',
 'Massachusetts',
 'New Hampshire',
 'New York',
 'Puerto Rico',
 'Rhode Island',
 'South Carolina',
 'Vermont',
 'Virgin Islands']
```

In [39]:
```
#sampleDF_test
```

In [40]:
```
# Preprocess the new data sample
sampleDF_test_preprocessed = preprocessor.transform(sampleDF_test)
```

For KNN module

In [41]:
```
knn_test_pred=model3.predict(sampleDF_test_preprocessed)
```

In [42]:
```
#knn_test_pred=knn.predict(df_test)
df_test_knn11= sampleDF_test.copy()
df_test_knn11['price']= knn_test_pred
df_test_knn11.head(10)
```

Out[42]:

	status	bed	bath	acre_lot	state	house_size	price
0	1	2	2	1	Puerto Rico	4601	1.544333e+06
1	0	3	3	6	New York	5186	9.285933e+05
2	1	1	2	1	South Carolina	1262	5.238400e+05
3	0	2	1	8	Massachusetts	5117	4.378667e+05
4	1	3	1	4	New Hampshire	2077	1.870800e+05
5	1	4	3	1	Rhode Island	9820	5.994725e+05
6	0	4	1	1	Vermont	7054	8.718600e+05
7	1	3	2	4	Rhode Island	3649	4.885533e+05
8	0	4	3	6	Massachusetts	7760	1.682800e+06
9	0	2	3	0	Virgin Islands	2144	1.105200e+06

decision_tree

In [43]:
```
#prediction
decision_tree_test_pred=model4.predict(sampleDF_test_preprocessed)
```

```
#copy the predivted values to sample data frame
df_test_decision_tree=sampleDF_test .copy()
df_test_decision_tree['price']= decision_tree_test_pred
df_test_decision_tree.head(10)
```

Out[43]:

	status	bed	bath	acre_lot	state	house_size	price
0	1	2	2	1	Puerto Rico	4601	575000.0
1	0	3	3	6	New York	5186	745000.0
2	1	1	2	1	South Carolina	1262	399900.0
3	0	2	1	8	Massachusetts	5117	388000.0
4	1	3	1	4	New Hampshire	2077	380000.0
5	1	4	3	1	Rhode Island	9820	550000.0
6	0	4	1	1	Vermont	7054	699900.0
7	1	3	2	4	Rhode Island	3649	950000.0
8	0	4	3	6	Massachusetts	7760	550000.0
9	0	2	3	0	Virgin Islands	2144	153000.0

RandomForest

In [44]:

```
#prediction
RandomForest_test_pred=model.predict(sampleDF_test_preprocessed)

#copy the predivted values to sample data frame
df_test_RandomForest= sampleDF_test.copy()
df_test_RandomForest['price']= RandomForest_test_pred
df_test_RandomForest.head()
```

Out[44]:

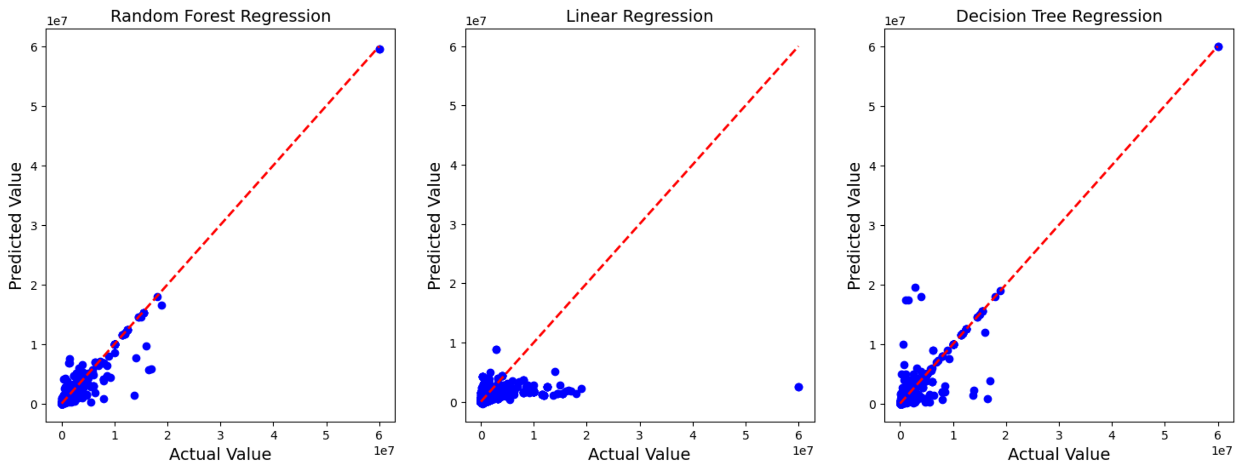
	status	bed	bath	acre_lot	state	house_size	price
0	1	2	2	1	Puerto Rico	4601	561232.00
1	0	3	3	6	New York	5186	622555.00
2	1	1	2	1	South Carolina	1262	379811.98
3	0	2	1	8	Massachusetts	5117	642184.96
4	1	3	1	4	New Hampshire	2077	355106.75

result comparison

In [45]:

```
# Create a figure with subplots
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

# Plot for Random Forest
axs[0].scatter(y_test, predictions_df, color='blue')
axs[0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='
axs[0].set_xlabel('Actual Value',fontsize=14)
```

```
axs[0].set_ylabel('Predicted Value',fontsize=14)
axs[0].set_title('Random Forest Regression',fontsize=14)

# Plot for Linear Regression
axs[1].scatter(y_test, predictions2_df, color='blue')
axs[1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='
axs[1].set_xlabel('Actual Value',fontsize=14)
axs[1].set_ylabel('Predicted Value',fontsize=14)
axs[1].set_title('Linear Regression',fontsize=14)

# Plot for Decision Tree
axs[2].scatter(y_test, predictions4_df, color='blue')
axs[2].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='
axs[2].set_xlabel('Actual Value',fontsize=14)
axs[2].set_ylabel('Predicted Value',fontsize=14)
axs[2].set_title('Decision Tree Regression',fontsize=14)

plt.show()
```



In [46]:
```
# Initialize an empty DataFrame
combined_results_df = pd.DataFrame(columns=['Model', 'MSE', 'RMSE', 'R-squared'

# Add results for each model
combined_results_df.loc[0] = ['Linear Regression', mse2, rmse2, r2_2]
combined_results_df.loc[1] = ['Random Forest', mse, rmse, r2]
combined_results_df.loc[2] = ['Decision Tree', mse4, rmse4, r2_4]


# Display the results table
print("The result of three different models:")
combined_results_df
```

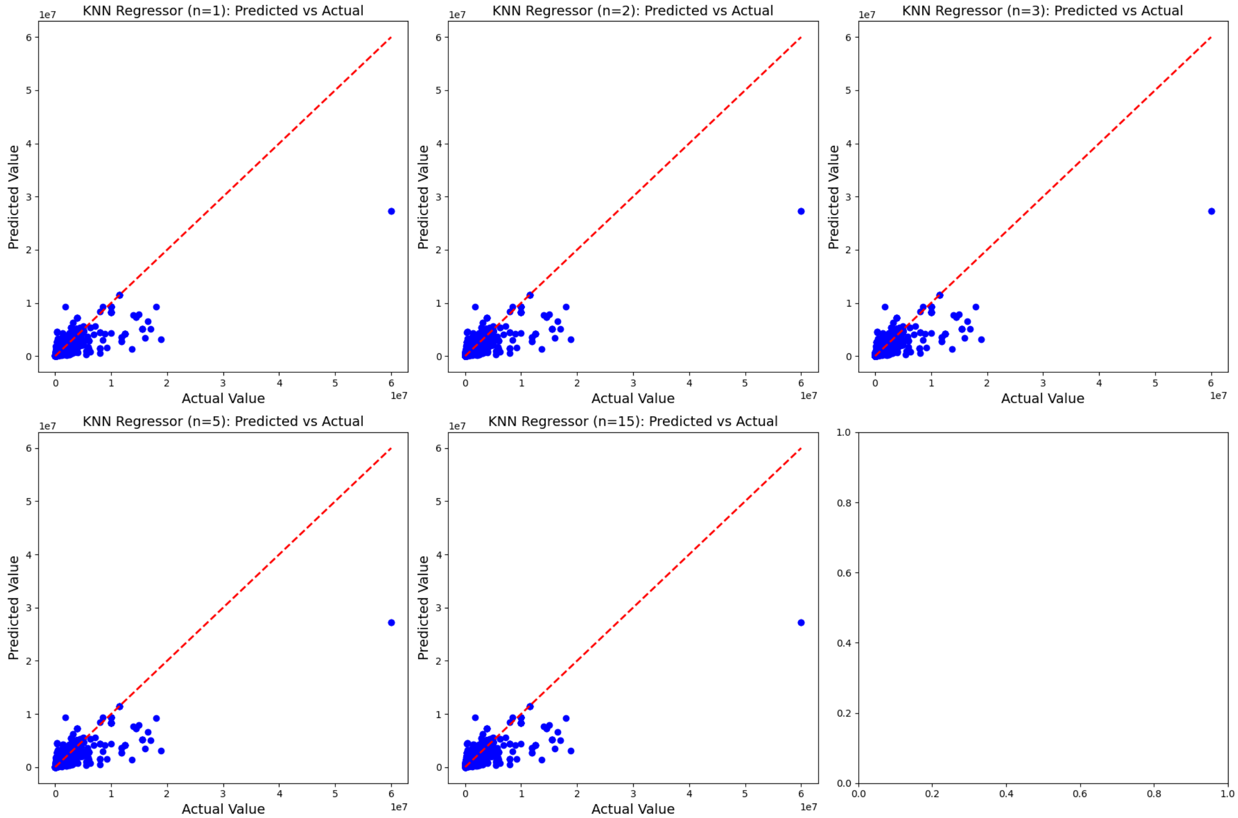
```
The result of three different models:
```

Out[46]:

	Model	MSE	RMSE	R-squared
0	Linear Regression	8.702921e+11	932894.468660	0.181045
1	Random Forest	6.094295e+10	246866.255030	0.942652
2	Decision Tree	1.419432e+11	376753.513484	0.866430

In [47]:
```
fig, axs = plt.subplots(2, 3, figsize=(18, 12))

# Dataframe to store results for knn module
```

```
results_df2 = pd.DataFrame(columns=['n', 'MSE', 'RMSE', 'R-squared'])


for i, n in enumerate(neighbor_values):
    # Create the KNN regressor
    model33 = KNeighborsRegressor(n_neighbors=n)

    # Fit the model to the training data
    model33.fit(X_train, y_train)

    # Predict house prices using the trained model
    predictions33 = model33.predict(X_test)
    predictions33_df = pd.DataFrame(predictions33)

    # Calculate metrics
    mse_knn = mean_squared_error(y_test, predictions33_df)
    rmse_knn = np.sqrt(mse_knn)
    r2_knn = r2_score(y_test, predictions33_df)

    # Print metrics
    #print(f'Mean Squared Error (n={n}):', mse_knn)
    #print(f'Root Mean Squared Error (n={n}):', rmse_knn)
    #print(f'R-squared (n={n}):', r2_knn)

    # Store results in the dataframe
    results_df2.loc[i] = [n, mse_knn, rmse_knn, r2_knn]

    # Plot the predicted values in a linear plot
    row = i // 3
    col = i % 3
    axs[row, col].scatter(y_test, predictions3_df, color='blue')
    axs[row, col].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max(
    axs[row, col].set_xlabel('Actual Value',fontsize=14)
    axs[row, col].set_ylabel('Predicted Value',fontsize=14)
    axs[row, col].set_title(f'KNN Regressor (n={n}): Predicted vs Actual',fonts

# Adjust the spacing between subplots
plt.tight_layout()

# Show the plots
plt.show()

print("The Reasult for KNN-model with different values of n:")
results_df2
```




```
The Reasult for KNN-model with different values of n:
```

Out[47]:

	n	MSE	RMSE	R-squared
0	1.0	7.537074e+10	274537.328435	0.929075
1	2.0	8.931111e+10	298849.646770	0.915957
2	3.0	9.678103e+10	311096.503603	0.908928
3	5.0	1.256905e+11	354528.625725	0.881724
4	15.0	3.335672e+11	577552.784141	0.686109

In [ ]:

In [ ]: