In [49]:
```
import pandas as pd
data= pd.read_csv("House_Rent_Dataset.csv")
data
```

Out[49]:

	Posted On	BHK	Rent	Size	Floor	Area Type	Area Locality	City	Furnishing Status	Tenant Preferred	Bathroom	Point of Contact
0	2022-05-18	2	10000	1100	Ground out of 2	Super Area	Bandel	Kolkata	Unfurnished	Bachelors/Family	2	Contact Owner
1	2022-05-13	2	20000	800	1 out of 3	Super Area	Phool Bagan, Kankurgachi	Kolkata	Semi-Furnished	Bachelors/Family	1	Contact Owner
2	2022-05-16	2	17000	1000	1 out of 3	Super Area	Salt Lake City Sector 2	Kolkata	Semi-Furnished	Bachelors/Family	1	Contact Owner
3	2022-07-04	2	10000	800	1 out of 2	Super Area	Dumdum Park	Kolkata	Unfurnished	Bachelors/Family	1	Contact Owner
4	2022-05-09	2	7500	850	1 out of 2	Carpet Area	South Dum Dum	Kolkata	Unfurnished	Bachelors	1	Contact Owner
...	...	...	...	...	...	...	...	...	...	...	...	...
4741	2022-05-18	2	15000	1000	3 out of 5	Carpet Area	Bandam Kommu	Hyderabad	Semi-Furnished	Bachelors/Family	2	Contact Owner
4742	2022-05-15	3	29000	2000	1 out of 4	Super Area	Manikonda, Hyderabad	Hyderabad	Semi-Furnished	Bachelors/Family	3	Contact Owner
4743	2022-07-10	3	35000	1750	3 out of 5	Carpet Area	Himayath Nagar, NH 7	Hyderabad	Semi-Furnished	Bachelors/Family	3	Contact Agent
4744	2022-07-06	3	45000	1500	23 out of 34	Carpet Area	Gachibowli	Hyderabad	Semi-Furnished	Family	2	Contact Agent
4745	2022-05-04	2	15000	1000	4 out of 5	Carpet Area	Suchitra Circle	Hyderabad	Unfurnished	Bachelors	2	Contact Owner

4746 rows × 12 columns

import necessary libraries

In [50]:
```
# Load libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.impute import SimpleImputer
```

Data preparation

In [51]:
```
# Check for missing values
print(data.isnull().sum())

# Simple strategy to fill missing values
imputer = SimpleImputer(strategy='most_frequent')
data = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)
```

```
Posted On           0
BHK                 0
Rent                0
Size                0
Floor               0
Area Type           0
Area Locality       0
City                0
Furnishing Status   0
Tenant Preferred    0
Bathroom            0
Point of Contact    0
dtype: int64
```

Convert Categorical Data The Floor column contains strings like "Ground out of 2" or "1 out of 3". We'll need to extract the numeric value for modeling.

In [52]:
```
# Extract floor level from 'Floor' column
data['Floor_Level'] = data['Floor'].apply(lambda x: int(str(x).split()[0]) if x.split()[0].isdigit() else 0)

# Drop the original 'Floor' column
data.drop(columns=['Floor'], inplace=True)

# Define features and target
X = data.drop(columns=['Rent'])  # Features
y = data['Rent']  # Target
```

Split the Dataset

In [53]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Preprocess the Data

In [54]:
```
# Define categorical columns
categorical_cols = ['Posted On', 'Area Type', 'Area Locality', 'City', 'Furnishing Status', 'Tenant Preferred', 'Poi

# Define numerical columns
numerical_cols = ['BHK', 'Size', 'Bathroom', 'Floor_Level']

# Preprocessing pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
    ])
```

```
# Apply preprocessing
X_train = preprocessor.fit_transform(X_train)
X_test = preprocessor.transform(X_test)
```

Model Building

Let's start with a simple model like Linear Regression.

In [55]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Define the model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)
```

Evaluate the Model

In [56]:
```
# Calculate evaluation metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"MAE: {mae}")
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
print(f"R2 Score: {r2}")
```

```
MAE: 25708.219896251707
MSE: 4638921321.096881
RMSE: 68109.62722770461
R2 Score: -0.16398430963369992
```

Now lets understand the project together

Mean Absolute Error (MAE): 25,708.22 Definition: MAE is the average absolute difference between the predicted values and the actual values. It gives an idea of how much, on average, your predictions are off by. Interpretation: On average, your model's predictions are off by approximately 25,708.22 units (likely the currency value for rent). For a rental price prediction, this is quite a high error, suggesting that the model is not very accurate.

Mean Squared Error (MSE): 4,638,921,321.10 Definition: MSE is the average of the squared differences between the predicted and actual values. Squaring the errors emphasizes larger errors more, which makes this metric sensitive to outliers. Interpretation: The MSE value is extremely high, which indicates that there are some predictions that are significantly off from the actual values. This large error is further emphasized due to the squaring.

Root Mean Squared Error (RMSE): 68,109.63 Definition: RMSE is the square root of the MSE. It gives an idea of the magnitude of error in the same units as the target variable. Interpretation: The RMSE of 68,109.63 suggests that, on average, the error in your predictions is around 68,109.63 units. This is even higher than the MAE, which again indicates significant errors in the predictions.

R^2 Score: -0.1639 Definition: The R^2 score (coefficient of determination) indicates the proportion of the variance in the dependent variable that is predictable from the independent variables. It ranges from 0 to 1, where 1 indicates perfect prediction and 0 indicates that the model does not explain any of the variance. A negative R^2 indicates that the model performs worse than a simple mean-based prediction. Interpretation: An R^2 score of -0.1639 is a strong indication that your model is performing poorly. It means that the model is worse than simply predicting the mean rental price for every instance. Essentially, your model fails to capture the relationship between the features and the target variable.

In [57]:
```
from sklearn.ensemble import RandomForestRegressor

# Define and train the model
model_rf = RandomForestRegressor(random_state=42)
model_rf.fit(X_train, y_train)

# Predict and evaluate
y_pred_rf = model_rf.predict(X_test)
print(f"R2 Score (Random Forest): {r2_score(y_test, y_pred_rf)}")
```

```
R2 Score (Random Forest): 0.5701194011034492
```

lets understand this together

An R^2 score of 0.5701 means that approximately 57% of the variance in the target variable (Rent in your case) is explained by the features used in the Random Forest model. This indicates a moderate level of predictive power:

Strengths: The model captures a significant portion of the underlying patterns in the data, making it reasonably effective at predicting rental prices.

Limitations: However, 43% of the variance in rental prices is still unexplained by the model, suggesting that there might be other factors or features not included in the model that could further improve prediction accuracy.

How to Interpret This Model Performance: With an R^2 of 0.5701, your model is doing a decent job, but there's room for improvement. This could involve adding more relevant features, feature engineering, or trying more advanced models.

lets improve the model together

In [71]:
```
print(f"Number of features: {len(features)}")
print(f"Number of coefficients: {len(coefficients)}")
```

```
Number of features: 11
Number of coefficients: 2020
```

In [72]:
```
# Assuming 'preprocessor' is your ColumnTransformer
numerical_features = numerical_cols
categorical_features = preprocessor.named_transformers_['cat'].get_feature_names_out()
all_features = numerical_features + list(categorical_features)
```

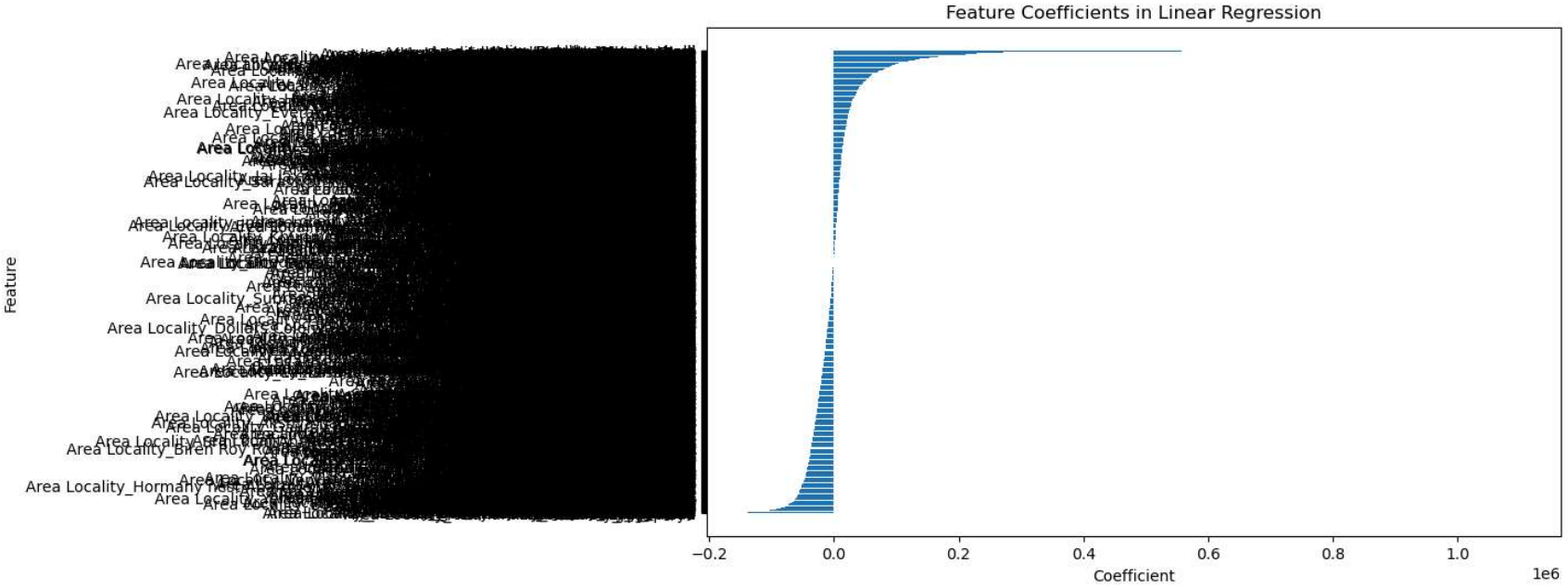
In [69]:
```
import pandas as pd

# Assuming 'model' is your trained Linear Regression model
coefficients = model.coef_

# Create a DataFrame for better visualization
coef_df = pd.DataFrame({
    'Feature': all_features,
    'Coefficient': coefficients
}).sort_values(by='Coefficient', ascending=False)

# Plotting the coefficients
```

```
plt.figure(figsize=(10, 6))
plt.barh(coef_df['Feature'], coef_df['Coefficient'])
plt.xlabel('Coefficient')
plt.ylabel('Feature')
plt.title('Feature Coefficients in Linear Regression')
plt.gca().invert_yaxis()
plt.show()
```



In [ ]:
```
lets explain this
```

Features with the highest positive or negative coefficients will appear at the top of the chart, making it easy to identify which features have the most influence.

Interpretation Positive Coefficients: Features with positive coefficients contribute positively to the target variable. For example, a feature with a high positive coefficient increases the predicted value as its value increases. Negative Coefficients: Features with negative coefficients contribute negatively to the target variable. For instance, a feature with a high negative coefficient decreases the predicted value as its value increases. Magnitude: The length of the bars shows the magnitude of the coefficients, which indicates the strength of the feature's impact. Larger magnitudes (positive or negative) mean that the feature has a stronger effect on the model's predictions

In [ ]: