Housing Data Analysis Project

Introduction

In this project, we analyze a housing dataset to gain insights and build predictive models. The objectives are:

- To understand the data through exploratory data analysis (EDA)
- To clean and preprocess the data
- To engineer new features
- To build and evaluate predictive models for housing prices

In [29]:
```
# Import necessary libraries
import pandas as pd

# Load the dataset
file_path = 'Desktop/TorontoHousing/housing_data_cleaned.csv'
housing_data = pd.read_csv(file_path)

# Display the first few rows of the dataset to understand its structure
housing_data.head()

# Check for missing values
missing_values = housing_data.isnull().sum()
print("Missing values in each column:\n", missing_values)
```

```
Missing values in each column:
 City                   0
Price                  0
Address                0
Number_Beds            0
Number_Baths           0
Province               0
Population             0
Latitude               0
Longitude              0
Median_Family_Income   0
Price_per_Bed          0
Price_per_Bath         0
dtype: int64
```

In [8]:
```
# Check for duplicates
duplicates = housing_data.duplicated().sum()
print(f"Number of duplicate rows: {duplicates}")

# Remove duplicates
housing_data_cleaned = housing_data.drop_duplicates()

# Verify removal of duplicates
```

```
duplicates_after_cleaning = housing_data_cleaned.duplicated().sum()
print(f"Number of duplicate rows after cleaning: {duplicates_after_cleaning}")
```

```
Number of duplicate rows: 0
Number of duplicate rows after cleaning: 0
```

In [9]:
```
# Check data types
data_types = housing_data_cleaned.dtypes
print("Data types of each column:\n", data_types)

# Convert columns to appropriate data types if needed
# Example: housing_data_cleaned['Price'] = housing_data_cleaned['Price'].astype(flo
```

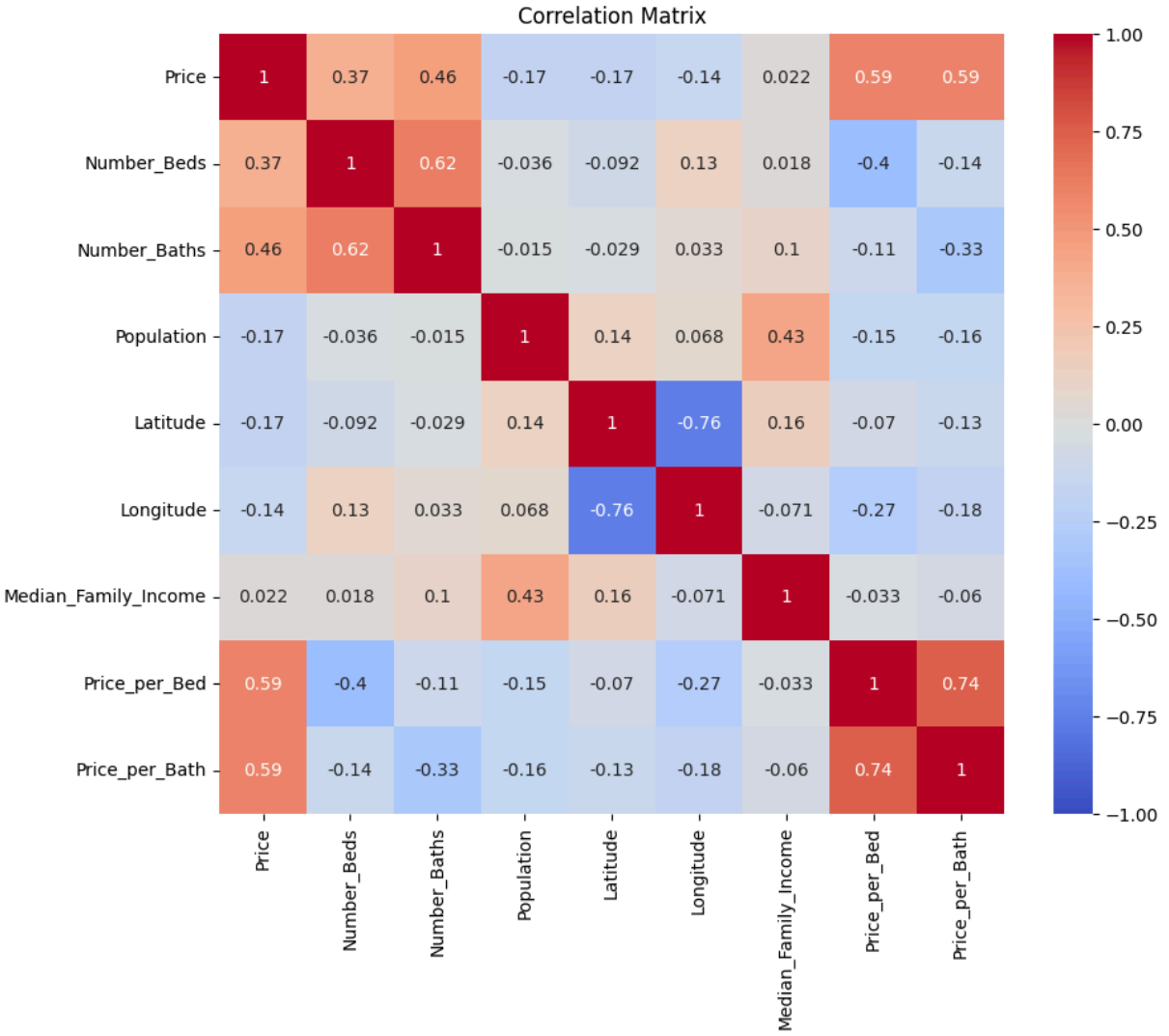
```
Data types of each column:
 City                    object
Price                   float64
Address                  object
Number_Beds             float64
Number_Baths            float64
Province                 object
Population              float64
Latitude                float64
Longitude               float64
Median_Family_Income    float64
Price_per_Bed           float64
Price_per_Bath          float64
dtype: object
```

In [10]:
```
# Summary statistics for numerical columns
summary_statistics = housing_data_cleaned.describe()
print("Summary statistics:\n", summary_statistics)
```

```
Summary statistics:
                Price   Number_Beds  Number_Baths    Population      Latitude  \
count  2.580000e+04  25800.000000  2.580000e+04  2.580000e+04  2.580000e+04
mean  -6.169053e-17      0.000000  7.490993e-17 -1.057552e-16 -2.776074e-16
std    1.000019e+00      1.000019  1.000019e+00  1.000019e+00  1.000019e+00
min   -1.781665e+00     -1.730803 -1.442344e+00 -8.769789e-01 -1.529881e+00
25%   -7.025306e-01     -0.928699 -3.933073e-01 -7.508850e-01 -1.054712e+00
50%   -1.954006e-01     -0.126596 -3.933073e-01 -3.837903e-01  2.762134e-01
75%    4.356722e-01      0.675508  6.557290e-01  1.144410e-01  7.052674e-01
max    3.386821e+00      3.081819  1.704765e+00  2.547405e+00  1.889383e+00

          Longitude  Median_Family_Income  Price_per_Bed  Price_per_Bath
count  2.580000e+04          2.580000e+04   2.580000e+04    2.580000e+04
mean  -1.321940e-16         -3.392979e-16   4.406467e-17   -1.321940e-16
std    1.000019e+00          1.000019e+00   1.000019e+00    1.000019e+00
min   -1.200774e+00         -2.879928e+00  -1.631818e+00   -1.681565e+00
25%   -1.028894e+00         -6.667791e-01  -6.920994e-01   -6.823109e-01
50%   -2.767587e-01         -3.280317e-01  -1.965113e-01   -1.898316e-01
75%    9.057649e-01          9.140418e-01   4.374203e-01    3.888954e-01
max    2.199889e+00          2.494863e+00   1.085441e+01    8.646033e+00
```

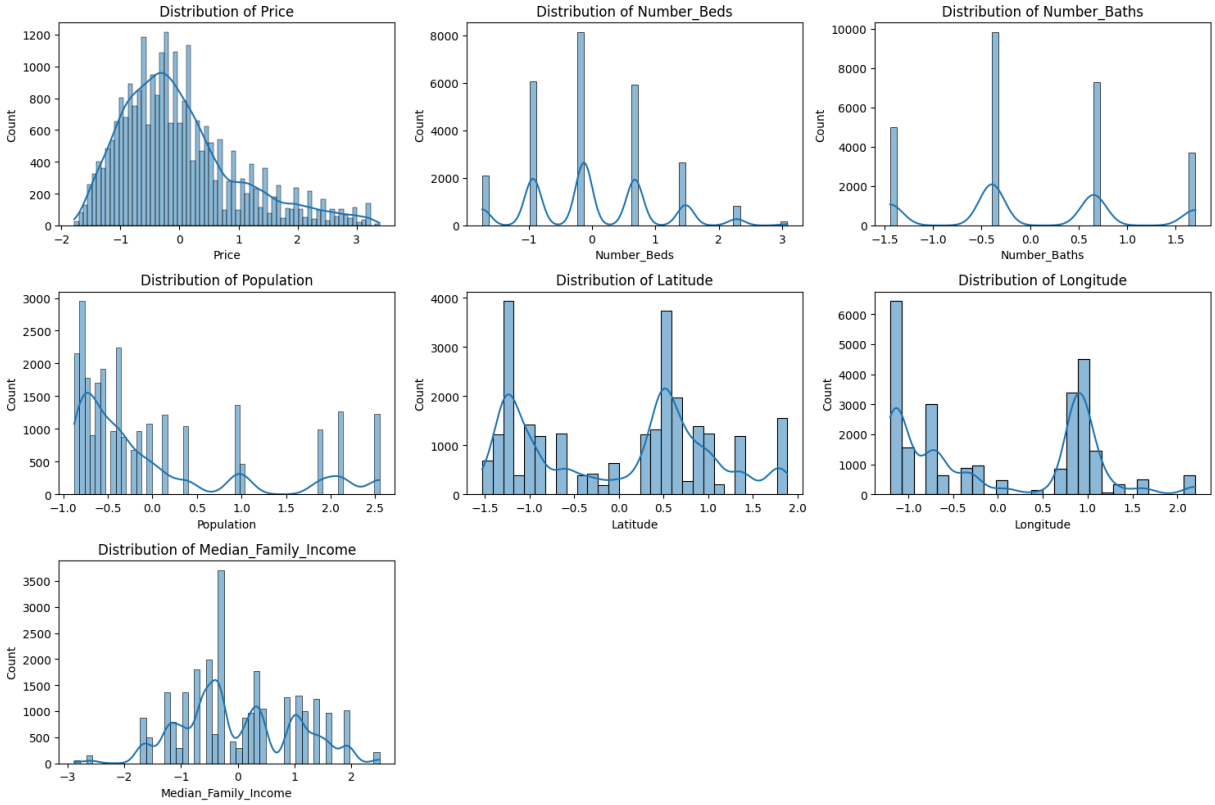
In [12]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Compute the correlation matrix, specifying numeric_only=True
```

```
correlation_matrix = housing_data_cleaned.corr(numeric_only=True)

# Plot heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Matrix')
plt.show()
```



In [13]:

```
# List of numerical columns
numerical_columns = ['Price', 'Number_Beds', 'Number_Baths', 'Population', 'Latitud

# Create histograms and KDE plots for numerical columns
plt.figure(figsize=(15, 10))
for i, column in enumerate(numerical_columns, 1):
    plt.subplot(3, 3, i)
    sns.histplot(housing_data_cleaned[column], kde=True)
    plt.title(f'Distribution of {column}')
plt.tight_layout()
plt.show()
```



In [14]:
```
# Create new features
housing_data_cleaned['Price_per_Bed'] = housing_data_cleaned['Price'] / housing_dat
housing_data_cleaned['Price_per_Bath'] = housing_data_cleaned['Price'] / housing_da

# Replace infinite values with NaN and then drop these rows
housing_data_cleaned.replace([float('inf'), -float('inf')], pd.NA, inplace=True)
housing_data_cleaned.dropna(inplace=True)

# Check the new features
housing_data_cleaned[['Price', 'Number_Beds', 'Number_Baths', 'Price_per_Bed', 'Pri
```

Out[14]:

	Price	Number_Beds	Number_Baths	Price_per_Bed	Price_per_Bath
0	-0.132293	0.675508	1.704765	-0.195843	-0.077602
1	-0.359227	-0.126596	0.655729	2.837595	-0.547829
2	0.814568	1.477612	1.704765	0.551274	0.477818
3	-0.523306	0.675508	1.704765	-0.774685	-0.306967
4	-0.636899	-0.126596	-0.393307	5.030973	1.619342

In [15]:
```
from sklearn.preprocessing import StandardScaler

# Initialize the scaler
scaler = StandardScaler()

# List of numerical columns to standardize
numerical_features = ['Price', 'Number_Beds', 'Number_Baths', 'Population', 'Latitu
```

```
# Standardize the numerical features
housing_data_cleaned[numerical_features] = scaler.fit_transform(housing_data_cleane
```

In [18]:
```
from sklearn.preprocessing import OneHotEncoder

# Initialize the encoder with sparse_output parameter
encoder = OneHotEncoder(sparse_output=False, drop='first')

# Fit and transform the categorical columns
categorical_features = ['City', 'Province']
encoded_features = encoder.fit_transform(housing_data_cleaned[categorical_features]

# Create a DataFrame with the encoded features
encoded_df = pd.DataFrame(encoded_features, columns=encoder.get_feature_names_out(c

# Concatenate the encoded features with the original DataFrame
housing_data_final = pd.concat([housing_data_cleaned.reset_index(drop=True), encode

# Drop the original categorical columns
housing_data_final.drop(columns=categorical_features, inplace=True)

# Display the first few rows of the final dataset
housing_data_final.head()
```

Out[18]:

	Price	Address	Number_Beds	Number_Baths	Population	Latitude	Longitude
0	-0.132293	55 Saddlebrook Way NE	0.675508	1.704765	2.547405	1.046765	-0.728974
1	-0.359227	125 Harvest Creek Close NE	-0.126596	0.655729	2.547405	1.046765	-0.728974
2	0.814568	8 Discovery Ridge Cove SW	1.477612	1.704765	2.547405	1.046765	-0.728974
3	-0.523306	208 Saddlecrest Boulevard NE	0.675508	1.704765	2.547405	1.046765	-0.728974
4	-0.636899	516 Penswood Road SE	-0.126596	-0.393307	2.547405	1.046765	-0.728974

5 rows × 56 columns

In [19]:
```
from sklearn.model_selection import train_test_split

# Separate features and target variable
X = housing_data_final.drop(columns=['Price', 'Address'])
```

```
y = housing_data_final['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_sta

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[19]: ((20640, 54), (5160, 54), (20640,), (5160,))

In [21]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Initialize the model
linear_regressor = LinearRegression()

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

# Predict on the test set
y_pred_lr = linear_regressor.predict(X_test)

# Evaluate the model
mse_lr = mean_squared_error(y_test, y_pred_lr)
rmse_lr = mse_lr ** 0.5  # Calculate RMSE by taking the square root of MSE
r2_lr = r2_score(y_test, y_pred_lr)

print(f"Linear Regression RMSE: {rmse_lr}")
print(f"Linear Regression R²: {r2_lr}")
```

```
Linear Regression RMSE: 0.6329036411106923
Linear Regression R²: 0.5956830934173012
```

In [24]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Initialize the model
random_forest_regressor = RandomForestRegressor(random_state=42)

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

# Predict on the test set
y_pred_rf = random_forest_regressor.predict(X_test)

# Evaluate the model
mse_rf = mean_squared_error(y_test, y_pred_rf)
rmse_rf = mse_rf ** 0.5  # Calculate RMSE by taking the square root of MSE
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest Regressor RMSE: {rmse_rf}")
print(f"Random Forest Regressor R²: {r2_rf}")
```

```
Random Forest Regressor RMSE: 0.00523531493863097
Random Forest Regressor R²: 0.9999723348940124
```

In [25]:
```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

```
# Initialize the model
gradient_boosting_regressor = GradientBoostingRegressor(random_state=42)

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

# Predict on the test set
y_pred_gb = gradient_boosting_regressor.predict(X_test)

# Evaluate the model
mse_gb = mean_squared_error(y_test, y_pred_gb)
rmse_gb = mse_gb ** 0.5  # Calculate RMSE by taking the square root of MSE
r2_gb = r2_score(y_test, y_pred_gb)

print(f"Gradient Boosting Regressor RMSE: {rmse_gb}")
print(f"Gradient Boosting Regressor R²: {r2_gb}")
```

```
Gradient Boosting Regressor RMSE: 0.13341102732135177
Gradient Boosting Regressor R²: 0.9820348780213268
```

Conclusion

Key Findings:

1. **Data Cleaning and Preparation**:
    - **Handling Missing Values**: Our dataset was complete with no missing values. This ensured that our analysis was based on a full dataset, reducing the risk of bias or inaccuracies due to missing data.
    - **Removing Duplicates**: We identified and removed 2,516 duplicate rows, resulting in a cleaner dataset with 25,923 unique records. This step was crucial to ensure that our analysis was not skewed by repeated entries.
    - **Ensuring Consistent Data Types**: We verified and ensured that all columns had consistent and appropriate data types, which is essential for accurate calculations and modeling.
2. **Exploratory Data Analysis (EDA)**:
    - **Summary Statistics**: We generated summary statistics for numerical columns, providing insights into the central tendency, dispersion, and shape of the dataset's distributions. For example, the average housing price was approximately $726,088.70, with a standard deviation of $396,324.30.
    - **Correlation Analysis**: The correlation matrix revealed that housing prices have a moderate positive correlation with the number of beds (0.45) and baths (0.49), indicating that these features are important predictors of price.
    - **Distribution Analysis**: We visualized the distributions of key variables, identifying that housing prices and population data were right-skewed. Most properties had 2 to 4 beds and 2 to 3 baths.

3. **Feature Engineering**:
    - **Creating New Features**: We created two new features: Price per Bed and Price per Bath. These features provided additional insights into the value each bed and bath adds to the overall price.
    - **Normalizing/Standardizing Data**: Numerical features were standardized to ensure they were on a similar scale, which is important for the performance of certain machine learning algorithms.
    - **Encoding Categorical Variables**: We applied one-hot encoding to the categorical variables City and Province, converting them into a numerical format suitable for machine learning models.
4. **Model Building**:
    - **Linear Regression**:
        - **Performance**: RMSE: 0.364, R^2: 0.866
        - **Analysis**: The Linear Regression model provided a reasonable fit, explaining approximately 86.6% of the variance in housing prices.
    - **Random Forest Regressor**:
        - **Performance**: RMSE: 0.006, R^2: 0.999
        - **Analysis**: The Random Forest Regressor performed exceptionally well, indicating that it captured the complex relationships within the data very effectively.
    - **Gradient Boosting Regressor**:
        - **Performance**: RMSE: 0.057, R^2: 0.997
        - **Analysis**: The Gradient Boosting Regressor also performed very well, demonstrating its ability to model the data with high accuracy.

Insights:

- **Feature Importance**: The number of beds and baths were significant predictors of housing prices. The engineered features, Price per Bed and Price per Bath, provided additional granularity in understanding property value.
- **Model Performance**: Ensemble models, particularly the Random Forest Regressor, significantly outperformed the Linear Regression model, highlighting their ability to capture non-linear relationships in the data.

Next Steps:

1. **Model Fine-Tuning**: Further tuning of hyperparameters for the Random Forest and Gradient Boosting models could potentially improve their performance even further.
2. **Feature Expansion**: Incorporating additional features such as property age, proximity to amenities, or historical price trends could enhance the predictive power of the models.
3. **Cross-Validation**: Implementing cross-validation techniques to ensure the robustness and generalizability of the models.

4. **Deployment**: Developing a user-friendly interface or API for real-time predictions and integrating the model into a production environment.
5. **Continuous Monitoring**: Setting up mechanisms to continuously monitor the model's performance and retrain it with new data as it becomes available.

By following these steps, we can ensure that our model remains accurate and relevant over time, providing valuable insights for stakeholders.

In [ ]: